Package 'samc'

October 14, 2022

Type Package

Title Spatial Absorbing Markov Chains

Version 2.0.1

Description Implements functions for working with absorbing Markov chains. The implementation is based on the framework described in ``Toward a unified framework for connectivity that disentangles movement and mortality in space and time'' by Fletcher et al. (2019) <doi:10.1111/ele.13333>, which applies them to spatial ecology. This framework incorporates both resistance and absorption with spatial absorbing Markov chains (SAMC) to provide several short-term and long-term predictions for metrics related to connectivity in landscapes. Despite the ecological context of the framework, this package can be used in any application of absorbing Markov chains.

License GPL (>= 3)

URL https://andrewmarx.github.io/samc/

BugReports https://github.com/andrewmarx/samc/issues/

Encoding UTF-8

LazyData true

Depends R (>= 3.6.0)

Imports methods, gdistance, Matrix, raster

RoxygenNote 7.1.1

Suggests knitr, rmarkdown, testthat, viridisLite

VignetteBuilder knitr

Collate 'RcppExports.R' 'location-class.R' 'internal-classes.R' 'samc-class.R' 'absorption.R' 'check.R' 'cond_passage.R' 'data.R' 'visitation.R' 'dispersal.R' 'distribution.R' 'generics.R' 'internal-functions.R' 'locate.R' 'map.R' 'mortality.R' 'pairwise.R' 'samc.R' 'survival.R' 'zzz.R'

LinkingTo Rcpp (>= 1.0.5), RcppEigen (>= 0.3.3.9.1), RcppThread (>= 2.1.3)

NeedsCompilation yes

Author Andrew Marx [aut, cre, cph] (<https://orcid.org/0000-0002-7456-1631>),
Robert Fletcher [ctb] (<https://orcid.org/0000-0003-1717-5707>),
Miguel Acevedo [ctb] (<https://orcid.org/0000-0002-8289-1497>),
Jorge Sefair [ctb] (<https://orcid.org/0000-0002-5887-8938>),
Chao Wang [ctb] (<https://orcid.org/0000-0003-2706-5122>)

Maintainer Andrew Marx <ajm.rpackages@gmail.com>

Repository CRAN

Date/Publication 2022-06-11 07:20:02 UTC

R topics documented:

`absorption` *Calculate absorption metrics*

Description

Calculates the probability of absorption for absorbing states rather than individual transient states. This is distint from, yet very closely linked to, the mortality() metric, which calculates the probability of absorption at individual transient states. If the results of the mortality() metric are decomposed into individual results for each absorbing state, then the sums of the individual results for every transient state are equivalent to the results of the absorption() metric.

Usage

```
absorption(samc, occ, origin)

## S4 method for signature 'samc,missing,missing'
absorption(samc)

## S4 method for signature 'samc,missing,location'
absorption(samc, origin)

## S4 method for signature 'samc,RasterLayer,missing'
absorption(samc, occ)

## S4 method for signature 'samc,matrix,missing'
absorption(samc, occ)
```

Arguments

samc	A samc-class object created using the samc function.
occ	The initial state ψ of the Markov chain. If the samc-class objects was constructed using map inputs, then occ must be the same type of input (either RasterLayer-class or matrix), and must have the same properties as initial map inputs (see the check function).
origin	A positive integer or character name representing transient state i. Corresponds to row i of matrix $\mathbf{P}$ in the samc-class object. When paired with the dest parameter, multiple values may be provided as a vector.

Details

$A = FR$

- **absorption(samc)**

  The result is a matrix M where $M_{i,k}$ is the probability of absorption due to absorbing state k if starting at transient state i.

- **absorption(samc, origin)**

  The result is a vector $\mathbf{v}$ where $\mathbf{v}_k$ is the probability of absorption due to absorbing state k if starting at transient state i.

$\psi^T A$

- **absorption(samc, occ)**

  The result is a vector $\mathbf{v}$ where $\mathbf{v}_k$ is the probability of absorption due to absorbing state k given an initial state ψ.

Value

See Details

Performance

Any relevant performance information about this function can be found in the performance vignette: `vignette("performance", package = "samc")`

Examples

```
# "Load" the data. In this case we are using data built into the package.
# In practice, users will likely load raster data using the raster() function
# from the raster package.
res_data <- samc::ex_res_data
abs_data <- samc::ex_abs_data
occ_data <- samc::ex_occ_data


# Make sure our data meets the basic input requirements of the package using
# the check() function.
check(res_data, abs_data)
check(res_data, occ_data)

# Setup the details for our transition function
tr <- list(fun = function(x) 1/mean(x), # Function for calculating transition probabilities
           dir = 8, # Directions of the transitions. Either 4 or 8.
           sym = TRUE) # Is the function symmetric?


# Create a `samc-class` object with the resistance and absorption data using
# the samc() function. We use the recipricol of the arithmetic mean for
# calculating the transition matrix. Note, the input data here are matrices,
# not RasterLayers.
samc_obj <- samc(res_data, abs_data, tr_args = tr)


# Convert the occupancy data to probability of occurrence
occ_prob_data <- occ_data / sum(occ_data, na.rm = TRUE)


# Calculate short- and long-term metrics using the analytical functions
short_mort <- mortality(samc_obj, occ_prob_data, time = 50)
short_dist <- distribution(samc_obj, origin = 3, time = 50)
long_disp <- dispersal(samc_obj, occ_prob_data)
visit <- visitation(samc_obj, dest = 4)
surv <- survival(samc_obj)


# Use the map() function to turn vector results into RasterLayer objects.
short_mort_map <- map(samc_obj, short_mort)
short_dist_map <- map(samc_obj, short_dist)
long_disp_map <- map(samc_obj, long_disp)
visit_map <- map(samc_obj, visit)
surv_map <- map(samc_obj, surv)
```

check *Check landscape data*

Description

Check that landscape inputs have valid values and matching properties.

Usage

```
check(a, b)

## S4 method for signature 'Raster,missing'
check(a)

## S4 method for signature 'matrix,missing'
check(a)

## S4 method for signature 'Raster,Raster'
check(a, b)

## S4 method for signature 'matrix,matrix'
check(a, b)

## S4 method for signature 'samc,Raster'
check(a, b)

## S4 method for signature 'samc,matrix'
check(a, b)
```

Arguments

a	A samc-class, matrix, or RasterLayer-class object
b	A matrix or RasterLayer-class object

Details

This function is used to ensure that landscape inputs (resistance, absorption, fidelity, and occupancy) have valid values and the same properties. This includes checking the CRS (if using RasterLayer inputs), dimensions, and locations of cells with NA data. It can be used to directly compare two matrices or two RasterLayers, or it can be used to check a samc-class object against a matrix or RasterLayer.

The function returns TRUE if the inputs have matching properties. Otherwise, it will stop execution and print the error message generated by the compareRaster() function from the raster package. This error will provide some details about the difference between the two inputs.

Note that the package assumes the different landscape inputs will be the same type, either matrices or RasterLayers. Mixing RasterLayer data and matrix data is not supported.

Value

See *Details* section.

Examples

```
# "Load" the data. In this case we are using data built into the package.
# In practice, users will likely load raster data using the raster() function
# from the raster package.
res_data <- samc::ex_res_data
abs_data <- samc::ex_abs_data
occ_data <- samc::ex_occ_data


# Make sure our data meets the basic input requirements of the package using
# the check() function.
check(res_data, abs_data)
check(res_data, occ_data)

# Setup the details for our transition function
tr <- list(fun = function(x) 1/mean(x), # Function for calculating transition probabilities
           dir = 8, # Directions of the transitions. Either 4 or 8.
           sym = TRUE) # Is the function symmetric?


# Create a `samc-class` object with the resistance and absorption data using
# the samc() function. We use the recipricol of the arithmetic mean for
# calculating the transition matrix. Note, the input data here are matrices,
# not RasterLayers.
samc_obj <- samc(res_data, abs_data, tr_args = tr)


# Convert the occupancy data to probability of occurrence
occ_prob_data <- occ_data / sum(occ_data, na.rm = TRUE)


# Calculate short- and long-term metrics using the analytical functions
short_mort <- mortality(samc_obj, occ_prob_data, time = 50)
short_dist <- distribution(samc_obj, origin = 3, time = 50)
long_disp <- dispersal(samc_obj, occ_prob_data)
visit <- visitation(samc_obj, dest = 4)
surv <- survival(samc_obj)


# Use the map() function to turn vector results into RasterLayer objects.
short_mort_map <- map(samc_obj, short_mort)
short_dist_map <- map(samc_obj, short_dist)
long_disp_map <- map(samc_obj, long_disp)
visit_map <- map(samc_obj, visit)
surv_map <- map(samc_obj, surv)
```

cond_passage *Conditional Mean First Passage Time*

Description

Calculate the mean number of steps to first passage

Usage

```
cond_passage(samc, origin, dest)

## S4 method for signature 'samc,missing,location'
cond_passage(samc, dest)

## S4 method for signature 'samc,location,location'
cond_passage(samc, origin, dest)
```

Arguments

samc	A samc-class object created using the samc function.
origin	A positive integer or character name representing transient state i. Corresponds to row i of matrix $\mathbf{P}$ in the samc-class object. When paired with the dest parameter, multiple values may be provided as a vector.
dest	A positive integer or character name representing transient state j. Corresponds to column j of matrix $\mathbf{P}$ in the samc-class object. When paired with the origin parameter, multiple values may be provided as a vector.

Details

$\tilde{t} = \tilde{B}_j^{-1}\tilde{F}\tilde{B}_j \cdot 1$

- **cond_passage(samc, dest)**

  The result is a vector where each element corresponds to a cell in the landscape, and can be mapped back to the landscape using the map function. Element *i* is the mean number of steps before absorption starting from location *i* conditional on absorption into *j*

  Note that mathematically, the formula actually does not return a value for when *i* is equal to *j*. This leads to a situation where the resultant vector is actually one element short and the index for some of the elements may be shifted. The **cond_passage()** function fills inserts a 0 value for vector indices corresponding to *i* == *j*. This corrects the final result so that vector indices work as expected, and allows the vector to be properly used in the map function.

- **cond_passage(samc, origin, dest)**

  The result is a numeric value representing the mean number of steps before absorption starting from a given origin conditional on absorption into *j*.

  As described above, mathematically the formula does not return a result for when the origin and dest inputs are equal, so the function simply returns a 0 in this case.

WARNING: This function is not compatible when used with data where there are states with total absorption present. When present, states representing total absorption leads to unsolvable linear equations. The only exception to this is when there is a single total absorption state that corresponds to input to the `dest` parameter. In this case, the total absorption is effectively ignored when the linear equations are solved.

WARNING: This function will crash when used with data representing a disconnected graph. This includes, for example, isolated pixels or islands in raster data. This is a result of the transition matrix for disconnected graphs leading to some equations being unsolvable. Different options are being explored for how to best identify these situations in data and handle them accordingly.

Value

See Details

Performance

Any relevant performance information about this function can be found in the performance vignette: `vignette("performance", package = "samc")`

Examples

```
# "Load" the data. In this case we are using data built into the package.
# In practice, users will likely load raster data using the raster() function
# from the raster package.
res_data <- samc::ex_res_data
abs_data <- samc::ex_abs_data
occ_data <- samc::ex_occ_data


# Make sure our data meets the basic input requirements of the package using
# the check() function.
check(res_data, abs_data)
check(res_data, occ_data)

# Setup the details for our transition function
tr <- list(fun = function(x) 1/mean(x), # Function for calculating transition probabilities
           dir = 8, # Directions of the transitions. Either 4 or 8.
           sym = TRUE) # Is the function symmetric?


# Create a `samc-class` object with the resistance and absorption data using
# the samc() function. We use the recipricol of the arithmetic mean for
# calculating the transition matrix. Note, the input data here are matrices,
# not RasterLayers.
samc_obj <- samc(res_data, abs_data, tr_args = tr)


# Convert the occupancy data to probability of occurrence
occ_prob_data <- occ_data / sum(occ_data, na.rm = TRUE)
```

```
# Calculate short- and long-term metrics using the analytical functions
short_mort <- mortality(samc_obj, occ_prob_data, time = 50)
short_dist <- distribution(samc_obj, origin = 3, time = 50)
long_disp <- dispersal(samc_obj, occ_prob_data)
visit <- visitation(samc_obj, dest = 4)
surv <- survival(samc_obj)


# Use the map() function to turn vector results into RasterLayer objects.
short_mort_map <- map(samc_obj, short_mort)
short_dist_map <- map(samc_obj, short_dist)
long_disp_map <- map(samc_obj, long_disp)
visit_map <- map(samc_obj, visit)
surv_map <- map(samc_obj, surv)
```

`dispersal` *Calculate dispersal metrics*

Description

Calculates the probability of individuals visiting locations

Usage

```
dispersal(samc, occ, origin, dest, time)

## S4 method for signature 'samc,missing,missing,location,numeric'
dispersal(samc, dest, time)

## S4 method for signature 'samc,ANY,missing,location,numeric'
dispersal(samc, occ, dest, time)

## S4 method for signature 'samc,missing,missing,missing,missing'
dispersal(samc)

## S4 method for signature 'samc,missing,location,missing,missing'
dispersal(samc, origin)

## S4 method for signature 'samc,missing,missing,location,missing'
dispersal(samc, dest)

## S4 method for signature 'samc,missing,location,location,missing'
dispersal(samc, origin, dest)

## S4 method for signature 'samc,ANY,missing,missing,missing'
dispersal(samc, occ)

## S4 method for signature 'samc,ANY,missing,location,missing'
dispersal(samc, occ, dest)
```

Arguments

samc	A samc-class object created using the samc function.
occ	The initial state ψ of the Markov chain. If the samc-class objects was constructed using map inputs, then occ must be the same type of input (either RasterLayer-class or matrix), and must have the same properties as initial map inputs (see the check function).
origin	A positive integer or character name representing transient state i. Corresponds to row i of matrix $\mathbf{P}$ in the samc-class object. When paired with the dest parameter, multiple values may be provided as a vector.
dest	A positive integer or character name representing transient state j. Corresponds to column j of matrix $\mathbf{P}$ in the samc-class object. When paired with the origin parameter, multiple values may be provided as a vector.
time	A positive integer or a vector of positive integers representing t time steps. Vectors must be ordered and contain no duplicates. Vectors may not be used for metrics that return dense matrices. The maximum time step value is capped at 10,000 due to numerical precision issues.

Details

$\tilde{D}_{jt} = (\sum_{n=0}^{t-1} \tilde{Q}^n)\tilde{q}_j$

- **dispersal(samc, dest, time)**

  The result is a vector $\mathbf{v}$ where $\mathbf{v}_i$ is the probability of visiting transient state j within t or fewer time steps if starting at transient state i.

  Note: Given the current derivation, when $i = j$, then $\mathbf{v}_i$ is unknown and has been set to NA.

  If multiple time steps were provided as a vector, then the result will be an ordered named list containing a vector for each time step.

  If the samc-class object was created using matrix or RasterLayer maps, then vector $\mathbf{v}$ can be mapped to a RasterLayer using the map function.

$\psi^T \tilde{D}_{jt}$

- **dispersal(samc, occ, dest, time)**

  The result is a numeric that is the probability of visiting transient state j within t or fewer time steps given an initial state ψ

  If multiple time steps were provided as a vector, then the result will be an ordered named list containing a vector for each time step.

$D = (F - I)diag(F)^{-1}$

- **dispersal(samc)**

  The result is a matrix M where $M_{i,j}$ is the probability of visiting transient state j if starting at transient state i.

  The returned matrix will always be dense and cannot be optimized. Must enable override to use (see samc-class).

- **dispersal(samc, origin)**
  The result is a vector $\mathbf{v}$ where $\mathbf{v}_j$ is the probability of visiting transient state j if starting at transient state i.
  If the samc-class object was created using matrix or RasterLayer maps, then vector $\mathbf{v}$ can be mapped to a RasterLayer using the `map` function.
- **dispersal(samc, dest)**
  The result is a vector $\mathbf{v}$ where $\mathbf{v}_i$ is the probability of visiting transient state j if starting at transient state i.
  If the samc-class object was created using matrix or RasterLayer maps, then vector $\mathbf{v}$ can be mapped to a RasterLayer using the `map` function.
- **dispersal(samc, origin, dest)**
  The result is a numeric value that is the probability of visiting transient state j if starting at transient state i.

$\psi^T D$

- **dispersal(samc, occ)**
  The result is a vector $\mathbf{v}$ where $\mathbf{v}_j$ is the probability of visiting transient state j given an initial state ψ.
  If the samc-class object was created using matrix or RasterLayer maps, then vector $\mathbf{v}$ can be mapped to a RasterLayer using the `map` function.
- **dispersal(samc, occ, dest)**
  The result is a numeric value that is the probability of visiting transient state j given an initial state ψ.

Value

See Details

Performance

Any relevant performance information about this function can be found in the performance vignette:
`vignette("performance", package = "samc")`

Examples

```
# "Load" the data. In this case we are using data built into the package.
# In practice, users will likely load raster data using the raster() function
# from the raster package.
res_data <- samc::ex_res_data
abs_data <- samc::ex_abs_data
occ_data <- samc::ex_occ_data


# Make sure our data meets the basic input requirements of the package using
# the check() function.
check(res_data, abs_data)
check(res_data, occ_data)
```

```
# Setup the details for our transition function
tr <- list(fun = function(x) 1/mean(x), # Function for calculating transition probabilities
           dir = 8, # Directions of the transitions. Either 4 or 8.
           sym = TRUE) # Is the function symmetric?


# Create a `samc-class` object with the resistance and absorption data using
# the samc() function. We use the recipricol of the arithmetic mean for
# calculating the transition matrix. Note, the input data here are matrices,
# not RasterLayers.
samc_obj <- samc(res_data, abs_data, tr_args = tr)


# Convert the occupancy data to probability of occurrence
occ_prob_data <- occ_data / sum(occ_data, na.rm = TRUE)


# Calculate short- and long-term metrics using the analytical functions
short_mort <- mortality(samc_obj, occ_prob_data, time = 50)
short_dist <- distribution(samc_obj, origin = 3, time = 50)
long_disp <- dispersal(samc_obj, occ_prob_data)
visit <- visitation(samc_obj, dest = 4)
surv <- survival(samc_obj)


# Use the map() function to turn vector results into RasterLayer objects.
short_mort_map <- map(samc_obj, short_mort)
short_dist_map <- map(samc_obj, short_dist)
long_disp_map <- map(samc_obj, long_disp)
visit_map <- map(samc_obj, visit)
surv_map <- map(samc_obj, surv)
```

`distribution` *Calculate distribution metrics*

Description

Calculate the probability of being at a transient state at a specific time.

Usage

```
distribution(samc, occ, origin, dest, time)

## S4 method for signature 'samc,missing,missing,missing,numeric'
distribution(samc, time)

## S4 method for signature 'samc,missing,location,missing,numeric'
distribution(samc, origin, time)

## S4 method for signature 'samc,missing,missing,location,numeric'
```

```
distribution(samc, dest, time)

## S4 method for signature 'samc,missing,location,location,numeric'
distribution(samc, origin, dest, time)

## S4 method for signature 'samc,ANY,missing,missing,numeric'
distribution(samc, occ, time)
```

Arguments

samc	A samc-class object created using the samc function.
occ	The initial state ψ of the Markov chain. If the samc-class objects was constructed using map inputs, then occ must be the same type of input (either RasterLayer-class or matrix), and must have the same properties as initial map inputs (see the check function).
origin	A positive integer or character name representing transient state i. Corresponds to row i of matrix $\mathbf{P}$ in the samc-class object. When paired with the dest parameter, multiple values may be provided as a vector.
dest	A positive integer or character name representing transient state j. Corresponds to column j of matrix $\mathbf{P}$ in the samc-class object. When paired with the origin parameter, multiple values may be provided as a vector.
time	A positive integer or a vector of positive integers representing t time steps. Vectors must be ordered and contain no duplicates. Vectors may not be used for metrics that return dense matrices. The maximum time step value is capped at 10,000 due to numerical precision issues.

Details

Q^t

- **distribution(samc, time)**

  The result is a matrix M where $M_{i,j}$ is the probability of being at transient state j after t time steps if starting at transient state i.

  The returned matrix will always be dense and cannot be optimized. Must enable override to use (see samc-class).

- **distribution(samc, origin, time)**

  The result is a vector $\mathbf{v}$ where $\mathbf{v}_j$ is the probability of being at transient state j after t time steps if starting at transient state i.

  If multiple time steps were provided as a vector, then the result will be an ordered named list containing a vector for each time step.

  If the samc-class object was created using matrix or RasterLayer maps, then vector $\mathbf{v}$ can be mapped to a RasterLayer using the map function.

- **distribution(samc, dest, time)**

  The result is a vector $\mathbf{v}$ where $\mathbf{v}_i$ is the probability of being at transient state j after t time steps if starting at transient state i.

  If multiple time steps were provided as a vector, then the result will be an ordered named list containing a vector for each time step.

If the samc-class object was created using matrix or RasterLayer maps, then vector $\mathbf{v}$ can be mapped to a RasterLayer using the map function.

- **distribution(samc, origin, dest, time)**

  The result is a numeric value that is the probability of being at a transient state j after t time steps if starting at transient state i.

  If multiple time steps were provided as a vector, then the result will be an ordered named list containing a vector for each time step.

$\psi^T Q^t$

- **distribution(samc, occ, time)**

  The result is a vector $\mathbf{v}$ where $\mathbf{v}_j$ is the probability of being at transient state i after t time steps given an initial state ψ.

  If multiple time steps were provided as a vector, then the result will be an ordered named list containing a vector for each time step.

  If the samc-class object was created using matrix or RasterLayer maps, then vector $\mathbf{v}$ can be mapped to a RasterLayer using the map function.

Value

See Details

Performance

Any relevant performance information about this function can be found in the performance vignette: `vignette("performance", package = "samc")`

Examples

```
# "Load" the data. In this case we are using data built into the package.
# In practice, users will likely load raster data using the raster() function
# from the raster package.
res_data <- samc::ex_res_data
abs_data <- samc::ex_abs_data
occ_data <- samc::ex_occ_data


# Make sure our data meets the basic input requirements of the package using
# the check() function.
check(res_data, abs_data)
check(res_data, occ_data)

# Setup the details for our transition function
tr <- list(fun = function(x) 1/mean(x), # Function for calculating transition probabilities
           dir = 8, # Directions of the transitions. Either 4 or 8.
           sym = TRUE) # Is the function symmetric?


# Create a `samc-class` object with the resistance and absorption data using
# the samc() function. We use the recipricol of the arithmetic mean for
# calculating the transition matrix. Note, the input data here are matrices,
```

```
# not RasterLayers.
samc_obj <- samc(res_data, abs_data, tr_args = tr)


# Convert the occupancy data to probability of occurrence
occ_prob_data <- occ_data / sum(occ_data, na.rm = TRUE)


# Calculate short- and long-term metrics using the analytical functions
short_mort <- mortality(samc_obj, occ_prob_data, time = 50)
short_dist <- distribution(samc_obj, origin = 3, time = 50)
long_disp <- dispersal(samc_obj, occ_prob_data)
visit <- visitation(samc_obj, dest = 4)
surv <- survival(samc_obj)


# Use the map() function to turn vector results into RasterLayer objects.
short_mort_map <- map(samc_obj, short_mort)
short_dist_map <- map(samc_obj, short_dist)
long_disp_map <- map(samc_obj, long_disp)
visit_map <- map(samc_obj, visit)
surv_map <- map(samc_obj, surv)
```

---

ex_abs_data *Example absorption data*

Description

A fabricated dataset containing landscape absorption probability values.

Usage

```
ex_abs_data
```

Format

A matrix with 34 rows and 202 columns.

Source

Fletcher et al (2019) <doi:10.1111/ele.13333>

ex_occ_data *Example occupancy data*

Description

A fabricated dataset containing landscape occupancy values.

Usage

```
ex_occ_data
```

Format

A matrix with 34 rows and 202 columns.

Source

Fletcher et al (2019) <doi:10.1111/ele.13333>

ex_res_data *Example resistance data*

Description

A fabricated dataset containing landscape resistance values.

Usage

```
ex_res_data
```

Format

A matrix with 34 rows and 202 columns.

Source

Fletcher et al (2019) <doi:10.1111/ele.13333>

`locate` *Get cell numbers*

Description

Get cell numbers from raster data

Usage

```
locate(samc, xy)

## S4 method for signature 'samc,missing'
locate(samc)

## S4 method for signature 'samc,ANY'
locate(samc, xy)
```

Arguments

`samc`	A `samc-class` object
`xy`	Any valid input to the y argument of the `extract` function in the raster package.

Details

This function is used to get cell numbers from raster data. The numbers used for origin and destination values in many samc metrics refer to column/row numbers of the P matrix. For a P matrix derived from raster data, these numbers would normally line up with the cell numbers of the raster, but this is not always true. This is the case when the raster contains NA data; the cells associated with this data are excluded from the P matrix. This causes issues trying to determine the cell numbers that should be used in analyses.

The `locate` function operates more-or-less like the `cellFromXY` function in the raster package, but unlike `cellFromXY`, locate properly accounts for NA cells in identifying cell numbers from coordinate data.

This function can also be used if the samc object was created from matrix inputs for the resistance, absorption, and fidelity parameters. In this case, the values in the xy coordinate parameter can be column-row values with the caveat that (1,1) is the bottom left corner.

The xy parameter can also be excluded. In this case, the function returns a raster where the values of the cells contains the cell number.

Internally, this function relies on the `extract` function from the raster package, and any valid input for the y argument of that function is valid here.

Value

A RasterLayer or a vector

Examples

```
library(samc)
library(raster)


# Load example data
res_data <- samc::ex_res_data
abs_data <- samc::ex_abs_data
occ_data <- samc::ex_occ_data


# Create samc-class object
samc_obj <- samc(res_data, abs_data,
                 tr_args = list(fun = function(x) 1/mean(x), dir = 8, sym = TRUE))


# We can use locate() to return a raster with the cell numbers encoded as data
# in the cells
cell_raster <- locate(samc_obj)
plot(cell_raster)


# We can use a variety of spatial inputs to get cell numbers using locate()
# The simplest is a two-column data.frame
coords <- data.frame(x = c(50, 79, 22),
                     y = c(25, 11, 19))
print(coords)
locate(samc_obj, coords)

# You will get an error if you input a coordinate that does not correspond
# to a non-NA cell
coords <- data.frame(x = c(1),
                     y = c(1))
print(coords)
try(locate(samc_obj, coords))
```

---

location-class *location class*

Description

Union class for location inputs

Details

The location class is a union class of the "numeric" and "character" classes. Users generally do not need to worry about it except to know that any method parameter with "location" as the type can have either an integer or a character name provided as input.

map *Map vector data*

Description

Map vector data to a RasterLayer

Usage

```
map(samc, vec)

## S4 method for signature 'samc,numeric'
map(samc, vec)

## S4 method for signature 'samc,list'
map(samc, vec)
```

Arguments

samc	Spatial absorbing Markov chain object. This should be output from the samc() function.
vec	Vector data to fill into the map.

Details

This is a convenience function to ensure that vector data is properly mapped back to the original landscape data. The reason this is needed is that the package supports both matrices and RasterLayers, which differ in the order that data is read and written (R matrices are column-major order, whereas the raster package uses row-major order). Internally, the package uses only a single order, regardless of the original data. This can cause issues with mapping vector results if care is not taken, and this function is provided to simplify the process. It also correctly maps results for landscape data that has NA cells, which are another potential source of error if not careful.

The only requirement of the vec input is that the number of elements in it matches the number of non-NA cells in the landscape data that was used to create the samc object.

Value

A RasterLayer object

Examples

```
# "Load" the data. In this case we are using data built into the package.
# In practice, users will likely load raster data using the raster() function
# from the raster package.
res_data <- samc::ex_res_data
abs_data <- samc::ex_abs_data
occ_data <- samc::ex_occ_data
```

```
# Make sure our data meets the basic input requirements of the package using
# the check() function.
check(res_data, abs_data)
check(res_data, occ_data)

# Setup the details for our transition function
tr <- list(fun = function(x) 1/mean(x), # Function for calculating transition probabilities
           dir = 8, # Directions of the transitions. Either 4 or 8.
           sym = TRUE) # Is the function symmetric?


# Create a `samc-class` object with the resistance and absorption data using
# the samc() function. We use the recipricol of the arithmetic mean for
# calculating the transition matrix. Note, the input data here are matrices,
# not RasterLayers.
samc_obj <- samc(res_data, abs_data, tr_args = tr)


# Convert the occupancy data to probability of occurrence
occ_prob_data <- occ_data / sum(occ_data, na.rm = TRUE)


# Calculate short- and long-term metrics using the analytical functions
short_mort <- mortality(samc_obj, occ_prob_data, time = 50)
short_dist <- distribution(samc_obj, origin = 3, time = 50)
long_disp <- dispersal(samc_obj, occ_prob_data)
visit <- visitation(samc_obj, dest = 4)
surv <- survival(samc_obj)


# Use the map() function to turn vector results into RasterLayer objects.
short_mort_map <- map(samc_obj, short_mort)
short_dist_map <- map(samc_obj, short_dist)
long_disp_map <- map(samc_obj, long_disp)
visit_map <- map(samc_obj, visit)
surv_map <- map(samc_obj, surv)
```

`mortality` *Calculate mortality metrics*

Description

Calculates the probability of absorption at individual transient states.

Usage

```
mortality(samc, occ, origin, dest, time)
```

```
## S4 method for signature 'samc,missing,missing,missing,numeric'
mortality(samc, time)

## S4 method for signature 'samc,missing,location,missing,numeric'
mortality(samc, origin, time)

## S4 method for signature 'samc,missing,missing,location,numeric'
mortality(samc, dest, time)

## S4 method for signature 'samc,missing,location,location,numeric'
mortality(samc, origin, dest, time)

## S4 method for signature 'samc,ANY,missing,missing,numeric'
mortality(samc, occ, time)

## S4 method for signature 'samc,missing,missing,missing,missing'
mortality(samc)

## S4 method for signature 'samc,missing,location,missing,missing'
mortality(samc, origin)

## S4 method for signature 'samc,missing,missing,location,missing'
mortality(samc, dest)

## S4 method for signature 'samc,missing,location,location,missing'
mortality(samc, origin, dest)

## S4 method for signature 'samc,ANY,missing,missing,missing'
mortality(samc, occ)
```

Arguments

samc	A samc-class object created using the samc function.
occ	The initial state ψ of the Markov chain. If the samc-class objects was constructed using map inputs, then occ must be the same type of input (either RasterLayer-class or matrix), and must have the same properties as initial map inputs (see the check function).
origin	A positive integer or character name representing transient state i. Corresponds to row i of matrix $\mathbf{P}$ in the samc-class object. When paired with the dest parameter, multiple values may be provided as a vector.
dest	A positive integer or character name representing transient state j. Corresponds to column j of matrix $\mathbf{P}$ in the samc-class object. When paired with the origin parameter, multiple values may be provided as a vector.
time	A positive integer or a vector of positive integers representing t time steps. Vectors must be ordered and contain no duplicates. Vectors may not be used for metrics that return dense matrices. The maximum time step value is capped at 10,000 due to numerical precision issues.

Details

$\tilde{B}_t = (\sum_{n=0}^{t-1} Q^n)\tilde{R}$

- **mortality(samc, time)**

  The result is a matrix M where $M_{i,j}$ is the probability of absorption at transient state j within t or fewer steps if starting at transient state i.

  The returned matrix will always be dense and cannot be optimized. Must enable override to use (see samc-class).

- **mortality(samc, origin, time)**

  The result is a vector $\mathbf{v}$ where $\mathbf{v}_j$ is the probability of absorption at transient state j within t or fewer steps if starting at transient state i.

  If multiple time steps were provided as a vector, then the result will be an ordered named list containing a vector for each time step.

  If the samc-class object was created using matrix or RasterLayer maps, then vector $\mathbf{v}$ can be mapped to a RasterLayer using the map function.

- **mortality(samc, dest, time)**

  The result is a vector $\mathbf{v}$ where $\mathbf{v}_i$ is the probability of absorption at transient state j within t or fewer steps if starting at transient state i.

  If multiple time steps were provided as a vector, then the result will be an ordered named list containing a vector for each time step.

  If the samc-class object was created using matrix or RasterLayer maps, then vector $\mathbf{v}$ can be mapped to a RasterLayer using the map function.

- **mortality(samc, origin, dest, time)**

  The result is a numeric value that is the probability of absorption at transient state j within t or fewer time steps if starting at transient state i.

  If multiple time steps were provided as a vector, then the result will be an ordered named list containing a numeric value for each time step.

$\psi^T \tilde{B}_t$

- **mortality(samc, occ, time)**

  The result is a vector $\mathbf{v}$ where $\mathbf{v}_j$ is the unconditional probability of absorption at transient state j within t or fewer steps given an initial state ψ.

  If multiple time steps were provided as a vector, then the result will be an ordered named list containing a vector for each time step.

  If the samc-class object was created using matrix or RasterLayer maps, then vector $\mathbf{v}$ can be mapped to a RasterLayer using the map function.

$B = F\tilde{R}$

- **mortality(samc)**

  The result is a matrix M where $M_{i,j}$ is the probability of absorption at transient state j if starting at transient state i.

  The returned matrix will always be dense and cannot be optimized. Must enable override to use (see samc-class).

- **mortality(samc, origin)**
  The result is a vector $\mathbf{v}$ where $\mathbf{v}_j$ is the probability of absorption at transient state j if starting at transient state i.
  If the samc-class object was created using matrix or RasterLayer maps, then vector $\mathbf{v}$ can be mapped to a RasterLayer using the map function.
- **mortality(samc, dest)**
  The result is a vector $\mathbf{v}$ where $\mathbf{v}_i$ is the probability of absorption at transient state j if starting at transient state i.
  If the samc-class object was created using matrix or RasterLayer maps, then vector $\mathbf{v}$ can be mapped to a RasterLayer using the map function.
- **mortality(samc, origin, dest)**
  The result is a numeric value that is the probability of absorption at transient state j if starting at transient state i.

$\psi^T B$

- **mortality(samc, occ)**
  The result is a vector $\mathbf{v}$ where $\mathbf{v}_j$ is the unconditional probability of absorption at transient state j given an initial state ψ.
  If the samc-class object was created using matrix or RasterLayer maps, then vector $\mathbf{v}$ can be mapped to a RasterLayer using the map function.

Value

See Details

Performance

Any relevant performance information about this function can be found in the performance vignette:
`vignette("performance", package = "samc")`

Examples

```
# "Load" the data. In this case we are using data built into the package.
# In practice, users will likely load raster data using the raster() function
# from the raster package.
res_data <- samc::ex_res_data
abs_data <- samc::ex_abs_data
occ_data <- samc::ex_occ_data


# Make sure our data meets the basic input requirements of the package using
# the check() function.
check(res_data, abs_data)
check(res_data, occ_data)

# Setup the details for our transition function
tr <- list(fun = function(x) 1/mean(x), # Function for calculating transition probabilities
           dir = 8, # Directions of the transitions. Either 4 or 8.
           sym = TRUE) # Is the function symmetric?
```

```
# Create a `samc-class` object with the resistance and absorption data using
# the samc() function. We use the recipricol of the arithmetic mean for
# calculating the transition matrix. Note, the input data here are matrices,
# not RasterLayers.
samc_obj <- samc(res_data, abs_data, tr_args = tr)


# Convert the occupancy data to probability of occurrence
occ_prob_data <- occ_data / sum(occ_data, na.rm = TRUE)


# Calculate short- and long-term metrics using the analytical functions
short_mort <- mortality(samc_obj, occ_prob_data, time = 50)
short_dist <- distribution(samc_obj, origin = 3, time = 50)
long_disp <- dispersal(samc_obj, occ_prob_data)
visit <- visitation(samc_obj, dest = 4)
surv <- survival(samc_obj)


# Use the map() function to turn vector results into RasterLayer objects.
short_mort_map <- map(samc_obj, short_mort)
short_dist_map <- map(samc_obj, short_dist)
long_disp_map <- map(samc_obj, long_disp)
visit_map <- map(samc_obj, visit)
surv_map <- map(samc_obj, surv)
```

`pairwise` *Pairwise analyses*

Description

Analysis for pairwise combinations locations

Usage

```
pairwise(fun, samc, origin, dest)

## S4 method for signature '`function`,samc,location,location'
pairwise(fun, samc, origin, dest)

## S4 method for signature '`function`,samc,location,missing'
pairwise(fun, samc, origin)
```

Arguments

`fun`	A samc analytical function with signature fun(samc, origin, dest)
`samc`	A `samc-class` object

origin	A vector of locations
dest	A vector of locations. Can be excluded to reuse the origin parameter

Details

When providing vector inputs for the 'origin' and 'dest' parameters to analytical functions, the package assumes that users are providing pairs of 'origin' and 'dest'. That is, 'origin[1]' is paired with 'dest[1]', 'origin[2]' is paired 'dest[2]', etc. Another way to think about it is that these two vector inputs can be treated as columns in the same dataframe. The result of the analytical function then is a vector of the same length as the input. This behavior works for any situation, so it is the default for the package.

However, some users may wish to run an analytical function for all the pairwise combinations of the values in the input vectors. That is, 'origin[1]' is paired with 'dest[1]','dest[2]', 'dest[3]', etc, before moving on to the next elements in 'origin'. This approach has the advantage of potentially reducing the amount of code needed for an analysis, and the results can be represented as a pairwise matrix, but it is not suitable for all situations. To enable this second approach more easily, the 'pairwise()' function runs all the combinations of the 'origin' and 'dest' parameters for an analytical function and returns the results in a 'long' format data.frame. This data.frame can then be reshaped into a pairwise matrix or 'wide' format data.frame using tools like the reshape2 or tidyr packages.

This function is not intended to be used with other inputs such as 'occ' or 'time'

Value

A 'long' format data.frame

Examples

```
library(samc)

# Load example data
res_data <- samc::ex_res_data
abs_data <- samc::ex_abs_data
occ_data <- samc::ex_occ_data


# Create samc-class object
samc_obj <- samc(res_data, abs_data,
                 tr_args = list(fun = function(x) 1/mean(x), dir = 8, sym = TRUE))

# pairwise() example
pw <- pairwise(cond_passage, samc_obj, origin = 1:4, dest = 5)
print(pw)

# pairwise() example without dest
pw <- pairwise(dispersal, samc_obj, origin = c(2, 7))
print(pw)
```

samc *Create an samc object*

Description

Create an samc object that contains the absorbing Markov chain data

Usage

```
samc(data, absorption, fidelity, tr_args)

## S4 method for signature 'TransitionLayer,RasterLayer,RasterLayer,missing'
samc(data, absorption, fidelity)

## S4 method for signature 'TransitionLayer,RasterLayer,missing,missing'
samc(data, absorption)

## S4 method for signature 'RasterLayer,RasterLayer,RasterLayer,list'
samc(data, absorption, fidelity, tr_args)

## S4 method for signature 'RasterLayer,RasterLayer,missing,list'
samc(data, absorption, tr_args)

## S4 method for signature 'matrix,matrix,matrix,list'
samc(data, absorption, fidelity, tr_args)

## S4 method for signature 'matrix,matrix,missing,list'
samc(data, absorption, tr_args)

## S4 method for signature 'dgCMatrix,missing,missing,missing'
samc(data)

## S4 method for signature 'matrix,missing,missing,missing'
samc(data)
```

Arguments

data	A RasterLayer-class or matrix or Matrix package dgCMatrix sparse matrix.
absorption	A RasterLayer-class or matrix
fidelity	A RasterLayer-class or matrix
tr_args	A list with args for constructing a transition matrix.

Details

This function is used to create a samc-class object. There are multiple options for creating this object.

Option 1: Raster or Matrix Maps

`samc(data = matrix, absorption = matrix, fidelity = matrix, tr_args = list())`

`samc(data = RasterLayer, absorption = RasterLayer, fidelity = RasterLayer, tr_args = list())`

The samc-class object can be created from a combination of resistance (or conductance), absorption, and fidelity data. These different landscape data inputs must be the same type (a matrix or RasterLayer), and have identical properties, including dimensions, location of NA cells, and CRS (if using RasterLayers).

The `data` and `absorption` inputs are always mandatory for this approach. The `fidelity` input is optional. If the `fidelity` input is not provided, then it it is assumed that there is no site fidelity (i.e., individuals will always move to an adjacent cell each time step).

The `tr_args` parameter is mandatory. It used when calculating the values for the transition matrix. Internally, is used in the transition function in the gdistance package to create the transition matrix. `tr_args` must be constructed as list with a transition function, the number of directions (4 or 8), and if the transition function is symmetric (TRUE or FALSE). Here is the template: `list(fun = \`function\`, dir = \`numeric\`, sym = \`logical\`)`

Option 2: TransitionLayer

`samc(data = TransitionLayer, absorption = RasterLayer, fidelity = RasterLayer)`

The `data` parameter can be a `TransitionLayer` object created using the gdistance package. In this case the `absorption` parameter is mandatory and should be a RasterLayer that has identical properties to the RasterLayer used to create the TransitionLayer object. The `fidelity` parameter is optional and has the same requirements as the `absorption` parameter. The check function can be used to verify these requirements.

The advantage of this approach is that it offers slightly more flexibility than the first option. Namely, it's useful if the TransitionLayer needs additional modifications before it is normalized with the absorption and fidelity inputs. The disadvantage compared to the first option is that samc() cannot detect certain issues when the TransitionLayer is manually created and modified. So if users do not need to manually modify the TransitionLayer, then the first option for creating a samc object is recommended.

Option 3: P Matrix

`samc(data = matrix)`

`samc(data = dgCMatrix)`

The `data` parameter can be used alone to create a samc-class object directly from a preconstructed P matrix. This matrix must be either a base R matrix, or a sparse matrix (dgCMatrix format) from the Matrix package. It must meet the following requirements:

- The number of rows must equal the number of columns (a square matrix)
- Total absorption must be a single column on the right-hand side of the matrix
- At the bottom of the matrix, there must be a row filled with 0's except for last element (bottom-right of the matrix diagonal), which must be set to 1
- Every disconnected region of the matrix must have at least one non-zero absorbing value
- Each row must sum to 1
- All values must be in the range of 0-1

Additionally, the columns and rows of the P matrix may be named (e.g., using dimnames(), rowname(), colnames(), etc). When specifying `origin` or `dest` inputs to metrics, these names may be used instead of cell numbers. This has the advantage of making the code for an analysis easier to read and interpret, which may also help to eliminate unintentional mistakes. There are two requirements for naming the rows/cols of a P matrix. First, since the P matrix represents a pairwise matrix, the row and column names must be the same. Second, there must be no duplicate names. The exception to these rules is the very last column and the very last row of the P matrix. Since these are not part of the pairwise transition matrix, they may have whatever names the user prefers.

Additional Information

Depending on the data used to construct the samc-class object, some metrics may cause crashes. This is a result of the underlying P matrix having specific properties that make some equations unsolvable. One known case is a P matrix that represents a disconnected graph, which can lead to the `cond_passage()` function crashing. In terms of raster/matrix inputs, a disconnected graph occurs when one or more pixels/cells are unreachable from other pixels/cells due to the presence of a full barrier made up of NA values. In a raster, these may be obvious as islands, but can be as inconspicuous as a rogue isolated pixel. There may be other currently unknown situations that lead to unsolvable metrics.

Future work is planned towards identifying these issues during creation of the samc-class object and handling them appropriately to prevent inadvertent crashes.

Version 2 Changes

Version 1.5.0 officially removed support for the deprecated `resistance`, `tr_fun`, `directions`, `p_mat`, `latlon`, and `override` arguments. Old code will have to be updated to the new samc() function structure in order to work.

Value

A `samc-class` object

Examples

```
# "Load" the data. In this case we are using data built into the package.
# In practice, users will likely load raster data using the raster() function
# from the raster package.
res_data <- samc::ex_res_data
abs_data <- samc::ex_abs_data
occ_data <- samc::ex_occ_data


# Make sure our data meets the basic input requirements of the package using
# the check() function.
check(res_data, abs_data)
check(res_data, occ_data)

# Setup the details for our transition function
tr <- list(fun = function(x) 1/mean(x), # Function for calculating transition probabilities
           dir = 8, # Directions of the transitions. Either 4 or 8.
           sym = TRUE) # Is the function symmetric?
```

```
# Create a `samc-class` object with the resistance and absorption data using
# the samc() function. We use the recipricol of the arithmetic mean for
# calculating the transition matrix. Note, the input data here are matrices,
# not RasterLayers.
samc_obj <- samc(res_data, abs_data, tr_args = tr)


# Convert the occupancy data to probability of occurrence
occ_prob_data <- occ_data / sum(occ_data, na.rm = TRUE)


# Calculate short- and long-term metrics using the analytical functions
short_mort <- mortality(samc_obj, occ_prob_data, time = 50)
short_dist <- distribution(samc_obj, origin = 3, time = 50)
long_disp <- dispersal(samc_obj, occ_prob_data)
visit <- visitation(samc_obj, dest = 4)
surv <- survival(samc_obj)


# Use the map() function to turn vector results into RasterLayer objects.
short_mort_map <- map(samc_obj, short_mort)
short_dist_map <- map(samc_obj, short_dist)
long_disp_map <- map(samc_obj, long_disp)
visit_map <- map(samc_obj, visit)
surv_map <- map(samc_obj, surv)
```

`samc-class` *samc class*

Description

S4 class to manage SAMC data.

Details

The samc class is used to help ensure that the package is used correctly and to minimize the possibility for users to accidentally produce nonsensical results that may not be obviously incorrect. This class contains the p matrix necessary for all the calculations in the package, and enforces its type so that users are less likely to inadvertently alter it in a way that will cause issues in calculations.

The `samc()` function is used to create `samc-class` objects.

The samc-class slots are subject to change, so users should not be using the `@` operator to access or change them. Doing so leads to the risk of broken code in the future. Instead, where relevant, the `$` operator can be used to get and set components of the class safely. This is a current list of what can be accessed and modified in the class:

- **override**
  Some analyses are memory intensive and have the potential to make a user's system non-responsive or crash. By default, a samc-class object cannot be used in these analyses to prevent

unintentional loss of work. In some cases, users may wish to use these particular analyses, in which case this behavior can be overridden. To get the current state of the override, use `samc_obj$override`. To enable the use of the analyses, the override can be set to `TRUE` using `samc_obj$override <- TRUE`. Before enabling the override, users should familiarize themselves with the Performance vignette.

- **q_matrix**

  Advanced users may wish to have direct access to the Q matrix for developing custom calculations/analyses. Assumptions should not be made about the internal storage and management of the P and Q matrices in the samc-class; these things are subject to change in the future. To safely access the Q matrix, use `samc_obj$q_matrix`. The Q matrix inside of the samc-class cannot be modified.

- **p_matrix**

  `samc_obj$p_matrix` can be used to get a copy of the P matrix.

- **abs_states**

  Used to attach additional absorbing states to an samc object. This does not cause P/Q matrices to be updated. Instead, it is intended to provide decomposed results from the `mortality()` and `absorption()` metrics for different sources of absorption that might be contributing to the total absorption values that were used to create the samc object.

  The input must be in the same form as the absorption inputs used in `samc()`. Matrices are passed in as a `list`, and rasters are passed in as a `RasterStack`. Using `NA` as the input will reset it.

- **threads**

  `samc_obj$threads` can be used to get or set the number of threads used for parallel computations. Details can be found in the Parallel Computing vignette.

Slots

`data` Data associated with different components of the P matrix

`source` Information about the data source for the P matrix

`map` Used to verify landscape inputs and mapping of vector data

`clumps` Number of discontinuous regions in data

`override` Used to prevent accidental use of memory intensive functions

`threads` Used for multi-threading

`.cache` Cached data for performance boosts

`survival` *Calculate survival metrics*

Description

Calculates the expected time to absorption

Usage

```
survival(samc, occ)

## S4 method for signature 'samc,missing'
survival(samc)

## S4 method for signature 'samc,ANY'
survival(samc, occ)
```

Arguments

samc	A samc-class object created using the samc function.
occ	The initial state ψ of the Markov chain. If the samc-class objects was constructed using map inputs, then occ must be the same type of input (either RasterLayer-class or matrix), and must have the same properties as initial map inputs (see the check function).

Details

$z = (I - Q)^{-1} \cdot 1 = F \cdot 1$

- **survival(samc)**
  The result is a vector $\mathbf{v}$ where $\mathbf{v}_i$ is the expected time to absorption if starting at transient state i.
  If the samc-class object was created using matrix or RasterLayer maps, then vector $\mathbf{v}$ can be mapped to a RasterLayer using the map function.

$\psi^T z$

- **survival(samc, occ)**
  The result is a numeric that is the expected time to absorption given an initial state ψ.

Value

See Details

Performance

Any relevant performance information about this function can be found in the performance vignette:
`vignette("performance", package = "samc")`

Examples

```
# "Load" the data. In this case we are using data built into the package.
# In practice, users will likely load raster data using the raster() function
# from the raster package.
res_data <- samc::ex_res_data
abs_data <- samc::ex_abs_data
occ_data <- samc::ex_occ_data
```

```
# Make sure our data meets the basic input requirements of the package using
# the check() function.
check(res_data, abs_data)
check(res_data, occ_data)

# Setup the details for our transition function
tr <- list(fun = function(x) 1/mean(x), # Function for calculating transition probabilities
           dir = 8, # Directions of the transitions. Either 4 or 8.
           sym = TRUE) # Is the function symmetric?


# Create a `samc-class` object with the resistance and absorption data using
# the samc() function. We use the recipricol of the arithmetic mean for
# calculating the transition matrix. Note, the input data here are matrices,
# not RasterLayers.
samc_obj <- samc(res_data, abs_data, tr_args = tr)


# Convert the occupancy data to probability of occurrence
occ_prob_data <- occ_data / sum(occ_data, na.rm = TRUE)


# Calculate short- and long-term metrics using the analytical functions
short_mort <- mortality(samc_obj, occ_prob_data, time = 50)
short_dist <- distribution(samc_obj, origin = 3, time = 50)
long_disp <- dispersal(samc_obj, occ_prob_data)
visit <- visitation(samc_obj, dest = 4)
surv <- survival(samc_obj)


# Use the map() function to turn vector results into RasterLayer objects.
short_mort_map <- map(samc_obj, short_mort)
short_dist_map <- map(samc_obj, short_dist)
long_disp_map <- map(samc_obj, long_disp)
visit_map <- map(samc_obj, visit)
surv_map <- map(samc_obj, surv)
```

`visitation` *Calculate visitation metrics*

Description

Calculates the number of times that transient states are visited before absorption.

Usage

```
visitation(samc, origin, dest)

## S4 method for signature 'samc,missing,missing'
```

```
visitation(samc)

## S4 method for signature 'samc,location,missing'
visitation(samc, origin)

## S4 method for signature 'samc,missing,location'
visitation(samc, dest)

## S4 method for signature 'samc,location,location'
visitation(samc, origin, dest)
```

Arguments

samc	A samc-class object created using the samc function.
origin	A positive integer or character name representing transient state i. Corresponds to row i of matrix $\mathbf{P}$ in the samc-class object. When paired with the dest parameter, multiple values may be provided as a vector.
dest	A positive integer or character name representing transient state j. Corresponds to column j of matrix $\mathbf{P}$ in the samc-class object. When paired with the origin parameter, multiple values may be provided as a vector.

Details

$F = (I - Q)^{-1}$

- **visitation(samc)**

  The result is a matrix M where $M_{i,j}$ is the number of times that transient state j is visited before absorption if starting at transient state i.

  The returned matrix will always be dense and cannot be optimized. Must enable override to use (see samc-class).

- **visitation(samc, origin)**

  The result is a vector $\mathbf{v}$ where $\mathbf{v}_j$ is the number of times that transient state j is visited before absorption if starting at transient state i.

  If the samc-class object was created using matrix or RasterLayer maps, then vector $\mathbf{v}$ can be mapped to a RasterLayer using the map function.

- **visitation(samc, dest)**

  The result is a vector $\mathbf{v}$ where $\mathbf{v}_i$ is the number of times that transient state j is visited before absorption if starting at transient state i.

  If the samc-class object was created using matrix or RasterLayer maps, then vector $\mathbf{v}$ can be mapped to a RasterLayer using the map function.

- **visitation(samc, origin, dest)**

  The result is a numeric value that is the number of times transient state j is visited before absorption if starting at transient state i.

Value

See Details

Performance

Any relevant performance information about this function can be found in the performance vignette: `vignette("performance", package = "samc")`

Examples

```
# "Load" the data. In this case we are using data built into the package.
# In practice, users will likely load raster data using the raster() function
# from the raster package.
res_data <- samc::ex_res_data
abs_data <- samc::ex_abs_data
occ_data <- samc::ex_occ_data


# Make sure our data meets the basic input requirements of the package using
# the check() function.
check(res_data, abs_data)
check(res_data, occ_data)

# Setup the details for our transition function
tr <- list(fun = function(x) 1/mean(x), # Function for calculating transition probabilities
           dir = 8, # Directions of the transitions. Either 4 or 8.
           sym = TRUE) # Is the function symmetric?


# Create a `samc-class` object with the resistance and absorption data using
# the samc() function. We use the recipricol of the arithmetic mean for
# calculating the transition matrix. Note, the input data here are matrices,
# not RasterLayers.
samc_obj <- samc(res_data, abs_data, tr_args = tr)


# Convert the occupancy data to probability of occurrence
occ_prob_data <- occ_data / sum(occ_data, na.rm = TRUE)


# Calculate short- and long-term metrics using the analytical functions
short_mort <- mortality(samc_obj, occ_prob_data, time = 50)
short_dist <- distribution(samc_obj, origin = 3, time = 50)
long_disp <- dispersal(samc_obj, occ_prob_data)
visit <- visitation(samc_obj, dest = 4)
surv <- survival(samc_obj)


# Use the map() function to turn vector results into RasterLayer objects.
short_mort_map <- map(samc_obj, short_mort)
short_dist_map <- map(samc_obj, short_dist)
long_disp_map <- map(samc_obj, long_disp)
visit_map <- map(samc_obj, visit)
surv_map <- map(samc_obj, surv)
```

Index