Package 'rsMove'

June 3, 2020

Type Package

Title Guidelines for the use of Remote Sensing in Movement Ecology

Version 0.2.8

Date 2020-06-01

URL https://github.com/RRemelgado/rsMove/tree/master/

BugReports https://github.com/RRemelgado/rsMove/issues/

Maintainer Ruben Remelgado <remelgado.ruben@gmail.com>

Description Tools for the guided selection of satellite data and environmental predictors, the combination of remote sensing and animal movement data and the mapping of resource suitability. Based on the paper by Remelgado et al. (2015) <doi:10.1111/2041-210X.13199>.

LazyData TRUE

Imports raster, sp, caret, ggplot2, grDevices, pryr, plyr, Rcpp, lubridate, RCurl

RoxygenNote 7.1.0

License GPL (>= 3)

Suggests knitr, rmarkdown, kableExtra, imager, devtools, lattice, rgdal, igraph, randomForest, e1071

VignetteBuilder knitr

Encoding UTF-8

LinkingTo Rcpp

NeedsCompilation yes

Author Ruben Remelgado [aut, cre]

Depends R (>= 3.5.0)

Repository CRAN

Date/Publication 2020-06-03 19:30:02 UTC

R topics documented:

`backSample` *backSample*

Description

Background sample selection.

Usage

```
backSample(x, y, z, sampling.method = "random", nr.samples = NULL)
```

Arguments

x	Object of class *SpatialPoints* of *SpatialPointsDataFrame*.
y	Object of class *RasterLayer*, *RasterStack* or *RasterBrick*.
z	Vector of region identifiers for each sample.
sampling.method	One of *random* or *pca*. Default is *random*.
nr.samples	Number of random background samples.

Details

First, the function determines the unique pixel coordinates for *x* based on the dimensions of *y* and retrieves *n*, random background samples where *n* is determined by *nr.samples*. If *sampling.method* is set to *"random"*, the function will return the selected samples as a *SpatialPoints* object. However, if *sampling.method* is set to *"pca"*, the function performs a Principal Components Analysis (PCA) over *y* to evaluate the similarity between the samples associated to *x* and the initial set of random samples. To achieve this, the function selects the most important Principal Components (PC's) using the kaiser rule (i.e. PC's with eigenvalues greater than 1) and, for each PC, estimates the median and the Median Absolute Deviation (MAD) based on the samples of related ot each unique identifier in *z*). Based on this data, the function selects background samples where the difference between their variance and the variance of the region samples exceeds the absolute difference between the median and the MAD. Finally, the algorithm filteres out all the background samples that were not selected by all sample regions. The ouptut is a *SpatialPointsDataFrame* containing the selected samples and the corresponding *y* values. If *nr.samples* is not provided all background pixels are considered.

Value

A *SpatialPoints* or a *SpatialPointsDataFrame*.

See Also

labelSample hotMove dataQuery

Examples

```
{

 require(raster)

 # read raster data
file <- list.files(system.file('extdata', '', package="rsMove"), 'ndvi.tif', full.names=TRUE)
 r.stk <- stack(file)

 # read movement data
 data(shortMove)

 # find sample regions
 label <- labelSample(shortMove, 30, agg.radius=30, nr.pixels=2)

 # select background samples (pca)
```

```
 bSamples <- backSample(shortMove, r.stk, label, sampling.method='pca')

 # select background samples (random)
 bSamples <- backSample(shortMove, r.stk, sampling.method='random')

}
```

---

dataQuery *dataQuery*

Description

Query environmental data for coordinate pairs using the nearest non NA value in time.

Usage

```
dataQuery(
  x,
  y,
  x.dates,
  y.dates,
  time.buffer,
  spatial.buffer = NULL,
  smooth.fun = NULL
)
```

Arguments

x	Object of class *RasterStack*, *RasterBrick*.
y	Object of class *SpatialPoints* or *SpatialPointsDataFrame*.
x.dates	Object of class *Date* with *x* observation dates.
y.dates	Object of class *Date* with *y* observation dates.
time.buffer	Two element vector with a temporal search buffer (expressed in days).
spatial.buffer	Spatial buffer size used to smooth the returned values. The unit depends on the spatial projection.
smooth.fun	Smoothing function applied with *spatial.buffer*.

Details

Returns environmental variables from a multi-layer raster object *x* for a given set of coordinates (*y*) depending on the temporal distance between the observation dates (*y.dates*) and the date on which each layer in the environmental data was collected (*x.dates*). *time.buffer* controls the search for non-NA values in time and is adjusted to the observation date of each element in *y*. The user may also provide *spatial.buffer* to spatially smooth the selected environmental information. In this case, for each sample, the function will consider the neighboring pixels within the selected acquisition and apply a smoothing function defined by *smooth.fun*. If *smooth.fun* is not specified, a weighted mean will be returned by default.

Value

A *data.frame* with the selected values and their corresponding dates.

See Also

sampleMove backSample

Examples

```
{

 require(raster)

 # read raster data
 file <- list.files(system.file('extdata', '', package="rsMove"), 'ndvi.tif', full.names=TRUE)
 r.stk <- stack(file)
 r.stk <- stack(r.stk, r.stk, r.stk) # dummy files for the example

 # read movement data
 data(shortMove)

 # raster dates
 file.name <- names(r.stk)
 x.dates <- as.Date(paste0(substr(file.name, 2, 5), '-',
 substr(file.name, 7, 8), '-', substr(file.name, 10, 11)))

 # sample dates
 y.dates <- as.Date(shortMove@data$date)

 # retrieve remote sensing data for samples
 rsQuery <- dataQuery(r.stk, shortMove, x.dates, y.dates, c(10,10))

}
```

---

hotMove *hotMove*

Description

Detection of geographic regions of samples using a pixel based approach.

Usage

```
hotMove(x, y, return.shp = FALSE)
```

Arguments

x	Object of class *SpatialPoints* of *SpatialPointsDataFrame*.
y	Grid resolution. Unit depends on *x* projection.
return.shp	Logical. Should the function return polygons of the regions? Default is FALSE.

Details

First, the function builds a raster with a resolution equal to *y* and the spatial extent of *x*. Then, each point in *x* is converted into pixel coordinates. Based on the unique pixel coordinates, the function then evaluates the spatial connectivity of these pixels using a 8-neighbor connected component labeling algorithm to detect regions. Finally, the ID's are related back to each individual data point in *x* based on their pixel coordinates and - if *return.shp* is TRUE - a polygon is derived from the convex hull of the points within each region. The output of the function consists of:

- *region.id* - Vector reporting on the region each element in *x* belongs to.
- *polygons* - Polygons for each unique region in *region.id* specified by the data column *region*.
- *plot* - A plot with the output of *polygons* accessible if *return.shp* is TRUE.

Value

A List containing a vector of region ID's per data point (*$indices*) and region polygons (*$polygons*).

See Also

sampleMove hotMoveStats

Examples

```
{

require(raster)

# reference data
data(longMove)

# extract regions
hm <- hotMove(longMove, 0.1)

}
```

hotMoveStats *hotMoveStats*

Description

Segmentation and statistical analysis of the time spent by an animal within a geographical region.

Usage

```
hotMoveStats(x, y, z)
```

Arguments

x	Region unique identifiers. Vector of class *numeric*.
y	Observation time. Object of class *Date*.
z	Individual identifier. Vector of class *character*.

Details

For each unique region defined by *x*, the function identifies unique temporal segments defined as periods of consecutive days with observations. Then, for each region, the function uses the identified segments to report on the minimum, maximum and mean time spent as well as the total amount of time spent within the region. Moreover, the function provides a detailed report of each segment and informs on the corresponding sample indices. If *z* is specified, the function will in addition count the number of individuals found within each region and within each temporal segment. The final output consists of:

- *region.stats* - *data.frame* with the distribution of samples per region.
- *segment.stats* - *data.frame* with all temporal segments assigned to each region.
- region.plot - Plot describing the distribution of samples and recorded time per region.

Value

A list.

See Also

hotMove

Examples

```
{

require(raster)

# reference data
data(longMove)
```

```
# extract regions
hm <- hotMove(longMove, 0.1)

# add new information to original shapefile
longMove@data <- cbind(longMove@data, hm$region.id)

# derive statistics
hm.region.stats <- hotMoveStats(hm$region.id, as.Date(longMove@data$timestamp))

}
```

---

imgInt *imgInt*

Description

Temporal linear interpolation of environmental data using a *raster*, *SpatialPointsDataFrames* or *matrix/data.frame*.

Usage

```
imgInt(
  x,
  x.dates,
  y,
  time.buffer,
  smooth = TRUE,
  smooth.fun = function(j) {     runmean2(as.numeric(j), 1) }
)
```

Arguments

x	Object of class *RasterStack*, *RasterBrick* or *data.frame*.
x.dates	Object of class *Date* with dates of *x*.
y	Object of class *Date* with target dates. Alternatively, a *RasterStack* or *RasterBrick* with julian days for each pixel.
time.buffer	A two-element vector with temporal search buffer (expressed in days).
smooth	Logical argument. Default is TRUE.
smooth.fun	Smoothing function. uses runmean2 by default.

Details

Wrapper for the function intime that performs a time-sensitive, linear interpolation of a multi-band raster. The output dates are specified by *y* and can differ from the dates of the input, specified by *x.dates*. *time.buffer* controls the search for dates to interpolate from specifying the maximum number of days that the selected data points can differ from the target date(s) in *y*. *time.buffer* is provided as a two element vector which limits the search in the past and future. If *smooth* is TRUE, the function will also smooth the interpolated time series. *fun* determines which function to use. By default, runmean2 is used whcih is an NA-sensitive, c++ implementation of a simple running mean.

Value

A *RasterBrick* or a *data frame*. If a *RasterBrick*, each layer represents a date in *y*. If a *data.frame/matrix*, columns represent dates and rows represent samples.

See Also

dataQuery timeDir spaceDir moveSeg

Examples

```
{

 require(raster)

 #'  # read raster data
 file <- list.files(system.file('extdata', '', package="rsMove"), 'ndvi.tif', full.names=TRUE)
 r.stk <- stack(file)
 r.stk <- stack(r.stk, r.stk, r.stk) # dummy files for the example

 # read movement data
 data(shortMove)

 # raster dates
 file.name <- names(r.stk)
 x.dates <- as.Date(paste0(substr(file.name, 2, 5), '-',
 substr(file.name, 7, 8), '-', substr(file.name, 10, 11)))

 # interpolate raster data to target dates
 out <- imgInt(r.stk[1:50,1:50,drop=FALSE], x.dates, as.Date("2013-08-10"), c(60,60))

}
```

`intime` *intime*

Description

Time-adjusted, linear interpolation.

Usage

```
intime(x, ij, oj, z)
```

Arguments

`x`	matrix with data to interpolate.
`ij`	Numeric vector with julian days of input data.
`oj`	Numeric vector with julian days of the output data.
`z`	Two-element, numeric vector with temporal buffer.

Details

For each row in x, the function finds the two nearest, non NA values to the target dates in ij retrieved before and after this day. Then, the function linearly interpolates the missing value linearly. This function is most suitable when all observation have the same date of recording.

Value

A matrix.

`intime2` *intime2*

Description

Time-adjusted, linear interpolation.

Usage

```
intime2(x, ij, oj, z)
```

Arguments

`x`	matrix with data to interpolate.
`ij`	Numeric Matrix with julian days of input data.
`oj`	Numeric vector with julian days of the output data.
`z`	Two-element, numeric vector with temporal buffer.

Details

For each row in x, the function finds the two nearest, non NA values to the target dates in ij retrieved before and after this day. Then, the function linearly interpolates the missing value linearly. This function is most suitable when all observation have the same date of recording. The function assumes that each observation has its own recording date.

Value

A matrix.

labelSample *labelSample*

Description

Pixel-based labeling of spatially connected groups of points in a *SpatialPoints* object.

Usage

```
labelSample(x, y, nr.points = 1, nr.pixels = NULL, agg.radius = NULL)
```

Arguments

x	Object of class *SpatialPoints* of *SpatialPointsDataFrame*.
y	Pixel resolution or a valid raster layer.
nr.points	Minimum number of observations per pixel.
nr.pixels	Minimum number of pixels per region.
agg.radius	Minimum radius for pixel aggregation. Unit depends on the projection of the data.

Details

First, the observations are converted to pixel coordinates and pixels with a corresponding number of observations greater than *nr.points* are filtered out. Then, if *nr.pixels* is set, the function evaluates the spatial connectivity of the pixels and regions with a pixel count smaller than *nr.pixels* are filtered out. Then, the algorithm aggregates nearby regions within the distance specified by *agg.radius*. The final region identifiers are then assigned back to the original observations in *x* based on their corresponding pixel coordinates. This analysis is based on the spatial extent of *x* and a given pixel resolution (*y*). Alternatively, the user may assign a raster object as *y* assuring that the final output is aligned with it.

Value

A numeric *vector* with region identifiers for each observation in *x* to their correspondent pixel region. Filtered observations are returned as *NA*.

See Also

sampleMove hotMove

Examples

```
{

 require(raster)

 # read raster data
```

```
 r <- raster(system.file('extdata', '2013-07-16_ndvi.tif', package="rsMove"))

 # read movement data
 data(shortMove)

 # derive region labels
 labels <- labelSample(shortMove, r, agg.radius=60)

}
```

---

longMove *Example data of animal movements during a migration.*

Description

Movement data for one White Stork collected during it migration between Germany and Spain.

Usage

```
data(longMove)
```

Format

A SpatialPointsDataFrame

Details

- timestampobservation timestamp.
- longlongitude.
- latlatitude.

---

moveCloud *moveCloud*

Description

Provides historical information on cloud cover for a set of coordinate pairs. The temporal information is adjusted to the sample observation date.

Usage

```
moveCloud(x, y, data.path = NULL, buffer.size = NULL, remove.file = FALSE)
```

Arguments

x	Object of class *Date* with observation dates of *y*.
y	Object of class *SpatialPoints* or *SpatialPointsDataFrame*.
data.path	Output data path for downloaded data.
buffer.size	Two element vector with temporal buffer size (expressed in days).
remove.file	Logical. Should the files be deleted after usage?

Details

This function uses daily cloud fraction data from NASA's NEO service. For each observation date in *obs.dates*, the function downloads the correspondent image and extracts the percent cloud cover for the corresponding samples in *y*. Before downloading any data, the function will look within *data.path* for previoulsy acquired data. If they exist, they won't be downloaded reducing the processing time required by the function. Moreover, if *buffer.size* is specified, for each date, the function will download all images that are within the specified temporal buffer. *buffer.size* requires a twoelement vector which specifies the buffer size before and after the target dates. These additional images will be used to report on the closest time step with the lowest possible cloud cover. The final output provides a *data.frame* ($report) with information on:

- *cloud cover % (day)*: cloud cover for the observation dates.
- *best date (after)*: dates before the observation dates with the lowest cloud cover.
- *best date cloud cover % (before)*: cloud cover for best before dates.
- *best date (after)*: dates after the observation dates with the lowest cloud cover.
- *best date cloud cover % (after)*: cloud cover best after dates.

Finally, the function generates a plot ($plot) reporting on the variability of cloud cover and the number of observation registered in *y* for each date.

Value

A *list* object reporting on the variability of cloud cover within and around each observation dates.

References

https://cneos.jpl.nasa.gov/

See Also

sMoveRes tMoveRes

Examples

```
## Not run:

 require(raster)

 # read movement data
 data(shortMove)
```

```
 # test function for 30 day buffer
 od <- as.Date(shortMove@data$date)
 c.cover <- moveCloud(shortMove, od, data.path=".", buffer.size=c(30,30))


## End(Not run)
```

---

moveReduce *moveReduce*

Description

Pixel based summary of movement data that preserves periodic movements.

Usage

```
moveReduce(x, y, z, preserve.revisits = TRUE, derive.raster = FALSE)
```

Arguments

x	Object of class *SpatialPoints* or *SpatialPointsDataFrame*.
y	Object of class *RasterLayer*, *RasterStack* or *RasterBrick*.
z	Object of class *Date*, *POSIXlt* or *POSIXct* with the observation time of *x*.
preserve.revisits	Logical. Should the function preserve revisit patterns?
derive.raster	Should a *RasterLayer* with the total time per pixel be provided?

Details

Translates (*x*) into pixel coordinates within a reference raster (*y*). The function identifies temporal segments corresponding to groups of consecutive observations within the same pixel. In this process, revisits to recorded pixels are preserved. Once the segments are identified, the function derives mean x and y coordinates for each of them and evaluates the time spent within each pixel. The function reports on the start and end timestamps and the elapsed time. If *preserve.revisits* is FALSE, the function will then summarize the output on a pixel level summing the time spent at each pixel. Additionally, if *derive.raster* is TRUE, the function will derive a *RasterLayer* with the same configuration as *y* depicting the the total amount of time spent per pixel. The output of the function consists of:

- *points* - *SpatialPointsDataFrame* with the reduced sample set.
- *total.time* - *RasterLayer* depicting the total time spent at each pixel.

Value

A *list* object.

See Also

sampleMove moveSeg

Examples

```
{

 require(raster)

 # read raster data
 r <- raster(system.file('extdata', '2013-07-16_ndvi.tif', package="rsMove"))

 # read movement data
 data(shortMove)

 # observation time
 z <- strptime(paste0(shortMove@data$date, ' ', shortMove@data$time),
 format="%Y/%m/%d %H:%M:%S")

 # reduce amount of samples
 move.reduce <- moveReduce(shortMove, r, z, derive.raster=TRUE)

}
```

---

moveSeg *moveSeg*

Description

Pixel based segmentation of movement data using environmental data.

Usage

```
moveSeg(
  x,
  z,
  y,
  data.type = "cont",
  threshold = NULL,
  summary.fun = NULL,
  buffer.size = NULL,
  smooth.fun = NULL
)
```

Arguments

x	Object of class *RasterLayer* or *data.frame*.
z	Object of class *Date*, *POSIXlt* or *POSIXct*.

y	Object of class *SpatialPoints* or *SpatialPointsDataFrame*.
data.type	Raster data data.type. One of *cont* (continuous) or *cat* (for categorical).
threshold	Change threshold. Required if *data.type* is set to *cat*.
summary.fun	Summary function used to summarize the values within each segment when *method* is *cont*. Default is mean.
buffer.size	Spatial buffer size applied around each segment (unit depends on spatial projection).
smooth.fun	Smoothing function applied with *buffer.size* when *method* is *cont*. Default is mean.

Details

This function identifies segments of comparable environmental conditions along a movement track given by *y*. Looking at consecutive data points, the function queries *x* and proceeds to identify a new segment if *threshold* is exceeded. Then, for each segment, the function summarizes *x* using *summary.fun* and reports on the amount of points found within it. Moreover, if *z* is set, the function reports on the start and end timestamps and the elapsed time for each segment. If *data.type* is set as *'cont'*, the function assumes *y* is a continuous variable. This will require the user to define *threshold* which indicates when the difference between consecutive points should be considered as a change. If *data.type* is set as *'cat'*, then the function will ignore *threshold* and map a change every time a change in value occurs. The user might choose to smooth the extracted values using *buffer.size*. This will prompt the function to summarize the values arounde ach sample in *y* using a metric define by*smooth.fun*. However, if *data.type* is set to *cat*, *smooth.fun* is ignored. In this case, the function will report on the majority value within the spatial buffer. The output of this function consists of:

- *segment.id* - Vector reporting on the segment identifiers associated to each sample in *y*.
- *segment.stats* - Statistical information for each segment reporting on the corresponding environmental and temporal information.
- *segment.plot* - plot of *stats* showing the variability of environmental conditions and time spent per segment.

Value

A *list*.

See Also

dataQuery imgInt timeDir spaceDir

Examples

```
{

 require(raster)

 # read raster data
 r <- raster(system.file('extdata', 'landCover.tif', package="rsMove"))
```

```
# read movement data
data(shortMove)

# observation time
z <- strptime(paste0(shortMove@data$date, ' ', shortMove@data$time),
format="%Y/%m/%d %H:%M:%S")

# perform directional sampling
seg <- moveSeg(r, z, shortMove, data.type="cat")

}
```

`plausibilityTest` *plausibilityTest*

Description

Quantifies and plots the distribution of pixels within a mask over a reference categorical raster object.

Usage

```
plausibilityTest(x, y, class.labels = NULL)
```

Arguments

`x`	Object of class *RasterLayer* and *RasterStack*.
`y`	Object of class *RasterLayer*.
`class.labels`	Labels of classes in *y* provided as a character vector.

Details

For each lazer in *x*, (e.g. classification mask) the function returns the absolute and relative count of non-NA pixels within each unique value of *y* (e.g. land cover map). Then, the results for each layer are compared in a combined plot. The output of the function is a list consisting of:

- *absolute.count* - Absolute pixel count for each layer of *x* overlapping with each value of *y*.
- *relative.count* - Relative pixel count for each layer of *x* overlapping with each value of *y*.
- *relative.plot* - Plot comparing the relative pixel count of the layers in *x* within each value of *y*.

Value

A *list*.

Examples

```
{

 require(raster)

 # load example probability image
 file <- system.file('extdata', 'probabilities.tif', package="rsMove")
 p <- raster(file) > 0.5

 # land cover map
 lc <- raster(system.file('extdata', 'landCover.tif', package="rsMove"))

 # segment probabilities
 pt <- plausibilityTest(p, lc)

 # show plot
 pt$relative.plot

 # see relative sample count
 head(pt$relative.count)

}
```

---

plotMove *plotMove*

Description

Standardized plotting of environmental and temporal information for a set of coordinate pairs.

Usage

```
plotMove(
  x,
  y,
  size.var = NULL,
  fill.var = NULL,
  var.type = NULL,
  var.names = NULL
)
```

Arguments

x	Vector of x coordinates.
y	Vector of y coordinates.
size.var	Optional. Controls the point size.
fill.var	Optional. Controls the fill color.
var.type	One of 'cont' or 'cat'. Defines the type of *fill.var*.
var.names	Character vector with names for *size.var* and *fill.var* to ve added to the plot.

Details

This function was designed to extent on other functions such as dataQuery, which provides environmental information, and moveReduce, which provides information on e.g. the time spent per sample. Using these two functions as an example, *plotMove* can represent the relation between the elapsed time and the change in environmental conditions.

Value

A *ggplot* object.

See Also

dataQuery moveReduce

Examples

```
{

 require(raster)

 # read raster data
 r <- raster(system.file('extdata', '2013-07-16_ndvi.tif', package="rsMove"))

 # read movement data
 data(shortMove)

 # observation time
 time <- strptime(paste0(shortMove@data$date, ' ', shortMove@data$time), format="%Y/%m/%d %H:%M:%S")

 # reduce amount of samples
 move.reduce <- moveReduce(shortMove, r, time)

 # query data
 ov <- extract(r, move.reduce$points)

 # plot output
 x <- move.reduce$points@data$x
 y <- move.reduce$points@data$y
 et <- move.reduce$points@data$elapsed.time
 op <- plotMove(x, y, size.var=et, fill.var=ov, var.type="cont")

}
```

---

predictResources *predictResources*

Description

Spatially stratified predictive modeling of resource suitability based on presence/absence samples.

Usage

```
predictResources(x, y, z, env.data = NULL)
```

Arguments

x	Object of class *data.frame* with environmental variables for presence samples.
y	Object of class *data.frame* with environmental variables for background samples.
z	*Numeric* or *character* vector with sample region labels. If missing, *x* is assumed as being one region.
env.data	Object of class *RasterStack* or *RasterBrick* with environmental variables in *x* and *y*.

Details

Modeling of resource suitability using animal movement data following the method of Remelgado et al (2017). Each unique label in *z* is kept for validation while the remaining samples are used for training. Then, the function evaluates the performance of this model reporting (internally) on the number of true positives, false positives and the number of cases for both presences and absences. Once all sample regions are used for validation, the reported values are summed and used to derive a F1-measure. The F1-measure is estimated as *2 * (P * R) / (P + R)* where *P* is the Precision (ratio of true positives within the number of predicted values) and *R* is the Recall (ratio of true positives within the number of validation samples). As a consequence, rather than reporting on an average performance, the final performance assessment reported by *predictResources* depicts an objective picture on how the model performed among the different sets sample regions. This metric is provided for presences (*x*) and absences (*y*) separately informing on the stability of the model. This analysis is performed using a Random Forest model as provided within the `train` function of the caret package. The final predictive model is then derived with all samples. The output of the function is a list object consisting of:

- *f1* - *data.frame* with final F1-measure for presences and absences.
- *overal..validation* - *data.frame* with region identifiers and validation sample count at each iteration.
- *sample.validation* -:Logical vector with the validation of each observation in *x*.
- *iteration.models* - List of models estimated at each iteration.
- *final.model* - Final predictive model based on all samples.
- *probabilities* - Predicted probability image. Given if *env.data* is set.

Value

A *list*.

See Also

`sampleMove` `labelSample` `backSample` `train`

Examples

```
## Not run:

 require(raster)

 # read remote sensing data
 file <- list.files(system.file('extdata', '', package="rsMove"), 'ndvi.tif', full.names=TRUE)
 r.stk <- stack(file)

 # read movement data
 data(shortMove)

 # observation time
 obs.time <- strptime(paste0(shortMove@data$date, ' ', shortMove@data$time),
 format="%Y/%m/%d %H:%M:%S")

 # remove redundant samples
 shortMove <- moveReduce(shortMove, r.stk, obs.time)$points

 # retrieve remote sensing data for samples
 rsQuery <- extract(r.stk, shortMove)

 # identify unique sample regions
 label <- labelSample(shortMove, r.stk, agg.radius=30)

 # select background samples
 bSamples <- backSample(shortMove, r.stk, label, sampling.method='pca')

 # derive model predictions
 out <- predictResources(rsQuery, bSamples@data, label, env.data=r.stk)


## End(Not run)
```

`rsComposite` *rsComposite*

Description

Phenological and date driven Pixel Based Compositing (PBC).

Usage

```
rsComposite(
  x,
  x.dates,
  obs.dates,
  comp.method = "closest",
  temporal.buffer = NULL
)
```

Arguments

x	Object of class *RasterStack* or *RasterBrick*.
x.dates	Object of class *Date* with *x* observation dates.
obs.dates	Object of class *Date* with reference dates.
comp.method	One of "closest" or "phenological". The default is "closest".
temporal.buffer	Search buffer (expressed in days). The default is NULL.

Details

The function uses a multi-layer raster object to build a composite for a reference date which corresponds to the median of *obs.dates*. Moreover,the function estimates the Median Absolute Deviation (MAD) of *obs.dates* which determines the temporal buffer that is used to search for usable data. As an alternative, *temporal.buffer* can be specified manually and will be required if *obs.dates* consists of a single value. The user can also specify how the compositing should be done. *comp.method* can be set to: #'

- *closest* - Selects pixels from layers with the closest possible date.
- *phenological* - Selects pixels from layers with the closest Day of the Year (DoY).

The final output of *rsComposite* is a list consisting of: #'

- *composite* - Final image composite
- *dates* - Temporal composition of the composite reporting on the julian day (origin is "1970-01-01").
- *pixel.count* - pixel count of unique values in *dates*. Additionally, it reports on NA values.
- *pixel.count.plot* - Plot with relative frequency of pixels per date extracted from *pixel.count*.
- *target.date* - Reference date used during compositing.
- *temporal.buffer* - Temporal buffer used during compositing.

If *pheno2* is used, for each pixel, the function will estimate a weighted mean of the clear pixels within the temporal buffer. The weights represent the inverse time difference between the target and the available dates giver higher weights to small differences.

Value

A *list*.

See Also

imgInt dataQuery

Examples

```
## Not run:

 require(raster)

 # read raster data
 file <- list.files(system.file('extdata', '', package="rsMove"), 'ndvi.tif', full.names=TRUE)
 r.stk <- stack(file)
 r.stk <- stack(r.stk, r.stk, r.stk) # dummy files for the example

 # raster dates
 file.name <- names(r.stk)
 x.dates <- as.Date(paste0(substr(file.name, 2, 5), '-',
 substr(file.name, 7, 8), '-', substr(file.name, 10, 11)))

 # target date
 obs.dates = as.Date("2013-06-01")

 # build composite
 r.comp <- rsComposite(r.stk, x.dates, obs.dates, comp.method="closest", temporal.buffer=90)


## End(Not run)
```

`rsMove` *rsMove.*

Description

Example datasets of the rsMove package

`runmean2` *runmean2*

Description

NA-sensitive running mean.

Usage

```
runmean2(x, y)
```

Arguments

x	A *Numeric* vector.
y	A *numeric* element.

Details

Applies a running mean over *x* with a window size defined by *y*. Missing values are ignored and/or filled during the compution.

Value

A vector.

`sampleMove` *sampleMove*

Description

Remote sensing oriented sampling of stops along a movement track.

Usage

```
sampleMove(
  xy,
  obs.time,
  search.radius,
  distance.method = "m",
  time.unit = NULL
)
```

Arguments

`xy`	Object of class *SpatialPoints* or *SpatialPointsDataFrame*.
`obs.time`	Object of class *Date*, *POSIXlt* or *POSIXct* with the same length as *xy*.
`search.radius`	Numeric element with search radius (in meters).
`distance.method`	How should the distance be estimated? One of 'm' or 'deg'. Default is 'm'.
`time.unit`	Time unit to estimate elapsed time. See `difftime` for keywords. Default is *mins*.

Details

This function finds location where an animal showed little or no movement based on GPS tracking data. It looks at the distance among consecutive samples and identifies the start of a segment when the distance is bellow *search.radius*. When a segment is started, the function looks at the distance between the starting point and the following observations are assigned to the same segment until the threshold is exceeded. When this occurs, the function summarizes the observations assigned to the segment deriving mean coordinates, the start, end and total time spent and the total number of observations per segment. The user should selected *distance.method* in accordance with the projection system associated to the data. If 'm', the function bases this analysis on the the ecludian distance. However, if 'deg' it set, the function uses the haversine formula. The final output is a *SpatialPointsDataFrame* containing the following information:

- *x* - X coordinate.
- *y* - Y coordinate.
- *start.time* - Start time of segment.
- *end.time* - End time of segment.
- *total.time* - Elapsed time within the segment.
- *nr.samples* - Number of observations.

Value

A *SpatialPointsDataFrame*.

See Also

labelSample backSample dataQuery

Examples

```
{

 require(raster)

# reference data
data(longMove)

 # sampling without reference grid
 obs.time = strptime(longMove$timestamp, "%Y-%m-%d %H:%M:%S")
 output <- sampleMove(longMove, obs.time, 7, distance.method='deg')

 # compare original vs new samples
 plot(longMove, col="black", pch=16)
 points(output$x, output$y, col="red", pch=15)

}
```

---

segRaster *segRaster*

Description

Connected-region based raster segmentation that preserves spatial gradients.

Usage

```
segRaster(x, break.point = 0.1, min.value = 0.5)
```

Arguments

x	Object of class *RasterLayer*.
break.point	Difference threshold. Default is 0.05.
min.value	Minimum value. Default is 0.5.

Details

The function segments *x* using a connected component region labeling approach. For each pixel, the function estimates the difference between it and its immediate neighbors. The pixels where the difference is below *break.point* are aggregated into a single region. Moreover, the user can define a minimum pixel value using *min.value* which will ignore all pixels below it. The output of this function consists of:

- *regions* - Region raster image.
- *stats* - Basic statistics for each pixel region.

Value

A list object.

See Also

predictResources

Examples

```
## Not run:

 require(raster)

 # load example probability image
 file <- system.file('extdata', 'probabilities.tif', package="rsMove")
 r <- raster(file)

 # segment probabilities
 rs <- segRaster(r)


## End(Not run)
```

---

shortMove *Example data of animal movements during the nesting period.*

Description

Movement data for one White Stork collected within its nesting site.

Usage

```
data(shortMove)
```

Format

A SpatialPointsDataFrame

Details

- xx coordinate.
- yy coordinate.
- dateobservation date.
- timeobservation time.

---

sMoveRes *sMoveRes*

Description

Tool to support the selection of an adequate satellite spatial resolution. Evaluates how the change in spatial resolution changes the amount of samples and sample regions based on a set of coordinate pairs.

Usage

```
sMoveRes(x, y)
```

Arguments

x	Object of class *SpatialPoints* or *SpatialPointsDataFrame*.
y	vector of spatial resolutions (unit depends on spatial projection).

Details

Given a vector of pixel resolutions (*y*), the function determines the number of unique pixels and unique pixel regions after their temporal aggregation. For each spatial resolution, the function starts by converting *x* to unique pixel coordinates and labels them based on their spatial aggregation. Then, the function counts the number of samples and sample regions. The output of the function consists of:

- *stats* - Summary statistics reporting on the number of unique samples and sample regions per spatial resolution.
- *plot* - Plot representing the change in number of samples and sample regions per spatial resolution.

If *x* is a *move* or a *moveStack* object, the function will iterate through each unique dataset name and return a nested output where the elements of of the list are named in accordance with the oataset names.

Value

A *list*.

See Also

tMoveRes specVar

Examples

```
{

 require(raster)

 # read movement data
 data(shortMove)

 # test function for 5, 10 20 and 30 m
 a.res <- sMoveRes(shortMove, c(5, 10, 20, 30))

}
```

---

spaceDir *spaceDir*

Description

Analysis of environmental change in space along a movement track.

Usage

```
spaceDir(
  x,
  y,
  sample.direction,
  data.type,
  obs.time = NULL,
  distance.method = "m",
  buffer.size = NULL,
  stat.fun = NULL,
  min.count = 2
)
```

Arguments

x	Object of class *SpatialPoints* or *SpatialPointsDataFrame*.
y	Object of class *RasterLayer*.
sample.direction	One of *forward*, *backward* or *both*. Default is *both*.

`data.type`	One of 'cont' or 'cat'. Defines which type of variable is in use.
`obs.time`	Object of class *Date*, *POSIXlt* or *POSIXct* with *x* observation dates.
`distance.method`	One of 'm' or 'deg' specifying the projection unit. Default is 'm'.
`buffer.size`	Spatial buffer size expressed in the map units.
`stat.fun`	Output statistical metric.
`min.count`	Minimum number of pixels required by *stat.fun*. Default is 2.

Details

This function quantifies environmnetal changes in space along a movement track. For each set of consecutive points, the function will derive coordinates for all unique pixels that overlap with - and are beteween - each point in a spatial moving widow with a shape defined by *sample.direction*. The user can define *buffer.size* to consider all pixels within a predefined spatial distance of the initially selected pixels. Once all pixels are selected, the function will extract the corresponding values in *y* summarize them using a pre-defined statistical function. If *data.type* is *cont*, a statistical function can be provided through *stat.fun*. However, if *data.type* is *cat*, the function will report on the dominant class and on the shannon index for each window. If the number of pixels within the a window is lower than *min.count* the function will return *NA*. On top of this, *spaceDir* will also report on the linear distance traveled between endpoints (in meters) and the travel time (in minutes). The output of the function is a list consisting of:

- *endpoints* - Point shapefile with endpoints of each spatial segment. Reports on a given statistical metric, traveled distance, travel time and the mean timestamp.
- *segments* - Line shapefile with spatial segments. Reports on the same information as *endpoints*. *plot* - Plotting of *segments* where each segment is colored according to the extracted raster value.

Value

A *list* containing shapefiles with information on environmental change and travel distance/time and a plot of the results.

See Also

`timeDir` `dataQuery` `imgInt`

Examples

```
{

 require(raster)

 # read raster data
 r <- raster(system.file('extdata', '2013-07-16_ndvi.tif', package="rsMove"))

 # read movement data
 data(shortMove)
```

```
 # observation time
 obs.time <- strptime(paste0(shortMove@data$date, ' ',shortMove@data$time),
 format="%Y/%m/%d %H:%M:%S")

 # perform directional sampling
 of <- function(i) {lm(i~c(1:length(i)))$coefficients[2]}
 s.sample <- spaceDir(shortMove, r, "backward", "cont", obs.time=obs.time, stat.fun=of)

}
```

---

specVar *specVar*

Description

Tool to support the selection of adequate satellite spatial resolution. Evaluates how the spectral variability within a pixel change with the change in spatial resolution.

Usage

```
specVar(x, y)
```

Arguments

x	Object of class *RasterLayer*.
y	Spatial resolution (unit depends on the spatial projection).

Details

Given a raster object (*x*), the function determines how degrading its spatial resolution impacts our ability to perceive the complexity of the landscape. For the pixel resolution given by *y*, The function resamples *x* and estimates the Mean Absolute Percentage Error (MAPE) for each pixel. The MAPE is estimated as $100/n * sum(abs(O - A/O)$ where *O* are the original value in *x*, *A* the aggregated value in the aggregated image and *n* the number of non-NA pixels in the original image The output of the function consists of:

- *mape* - MAPE raster.
- *plot* - Histogram of *mape*.

Value

A *list*.

See Also

tMoveRes sMoveRes

Examples

```
## Not run:

 require(raster)

 # read raster data
 r <- raster(system.file('extdata', '2013-07-16_ndvi.tif', package="rsMove"))

 # apply function
 s.var <- specVar(r, 60)


## End(Not run)
```

---

timeDir *timeDir*

Description

Analysis of environmental change in time for a set of coordinate pairs.

Usage

```
timeDir(
  env.data,
  env.dates,
  obs.dates,
  temporal.buffer,
  xy = NULL,
  stat.fun = NULL,
  min.count = 2
)
```

Arguments

env.data	Object of class *RasterStack* or *RasterBrick* or *data.frame*.
env.dates	Object of class *Date* with *env.data* observation dates.
obs.dates	Object of class *Date* with *xy* observation dates.
temporal.buffer	two element vector with temporal window size (expressed in days).
xy	Object of class "SpatialPoints" or "SpatialPointsDataFrame".
stat.fun	Output statistical metric.
min.count	Minimum number of samples required by *stat.fun*. Default is 2.

Details

This function quantifies environmental change in time along a movement track. First, for each point in *xy*, the function compares its observation date (*obs.dates*) against the acquisition dates (*env.dates*) of *env.data* to select non *NA* timesteps within a predefined temporal window (*temporal.buffer*). The user can adjust this window to determine which images are the most important. For example, if one wishes to know how the landscape evolved up to the observation date of the target sample, *temporal.buffer* can be define as, e.g., c(30,0) forcing the function to only consider pixels recorded within the previous 30 days. After selecting adequate temporal information for each data point, a statistical metric is estimated. This statistical metric is specified by *stat.fun*. By default, the function reports on the slope between the acquisition dates of *env.data* and their corresponding values. When providing a new function, set x for *env.dates* and y for *env.data*. The final output is a list consisting of:

- *stats* - *data.frame* with the estimated statistical metric for each data point.
- *hist.plot* - Histogram plot of the requested statistical metric. The bin size is the standard deviation of all estimated values.
- *point.plot* - Plot of the *xy* showing the spatial variability of the requested statistical metric.

Value

A *vector* with a requested statistical metric for each point in *xy* and informative plots.

See Also

spaceDir dataQuery imgInt

Examples

```
{

 require(raster)

 # read raster data
file <- list.files(system.file('extdata', '', package="rsMove"), 'ndvi.tif', full.names=TRUE)
 r.stk <- stack(file)
 r.stk <- stack(r.stk, r.stk, r.stk) # dummy files for the example

 # read movement data
 data(shortMove)

 # raster dates
 r.dates <- seq.Date(as.Date("2013-08-01"), as.Date("2013-08-09"), 1)

 # sample dates
 obs.dates <- as.Date(shortMove@data$date)

 # perform directional sampling
 of <- function(x,y) {lm(y~x)$coefficients[2]}
 time.env <- timeDir(r.stk, r.dates, obs.dates, c(30,30), xy=shortMove, stat.fun=of)

}
```

tMoveRes *tMoveRes*

Description

Tool to support the selection of an adequate satellite temporal resolution. It evaluates how the change in temporal resolution changes the amount of samples and sample regions based on a set of coordinate pairs and their observation dates.

Usage

```
tMoveRes(xy, obs.date, time.res, pixel.res)
```

Arguments

`xy`	Object of class *SpatialPoints* or *SpatialPointsDataFrame*.
`obs.date`	Object of class *Date* with *xy* observation dates.
`time.res`	Vector of temporal resolutions (expressed in days).
`pixel.res`	Spatial resolution (unit depends on spatial projection).

Details

Given a base spatial resolution (*pixel.res* and a vector of temporal resolutions (*time.res*), the function determines the number of unique pixels and unique pixel regions after their temporal aggregation. For each temporal resolution, the function starts by converting *xy* to unique pixel coordinates and labels them based on their spatial aggregation. Then, the function counts the number of samples and sample regions. The output of the function consists of:

- *stats* - Summary statistics reporting on the number of temporal widows, unique samples and unique sample regions per temporal resolution.
- *plot* - Plot representing the change in number of samples and sample regions per temporal resolution.

Value

A *list* object reporting on the amount and distribution of unique pixels and connected pixel regions per temporal resolution.

See Also

`sMoveRes` `specVar`

Examples

```
{

 require(raster)

 data(longMove) # access reference data
 longMove <- longMove[c(1:50, 2000:2050,3000:3050),] # subset for testing

 # test function for intervals of 1, 8 and 16 days (e.g. of MODIS data)
 obs.date <- as.Date(longMove@data$timestamp)
 a.res <- tMoveRes(longMove, obs.date, c(1,8,16), 0.1)

}
```

Index