Package 'sarp.snowprofile.alignment'

December 3, 2020

Title Snow Profile Alignment, Aggregation, and Clustering

Version 1.0.2

Date 2020-11-30

Description Snow profiles describe the vertical (1D) stratigraphy of layered snow with different layer characteristics, such as grain type, hardness, deposition date, and many more. Hence, they represent a data format similar to multivariate time series containing categorical, ordinal, and numerical data types. Use this package to align snow profiles by matching their individual layers based on Dynamic Time Warping (DTW). The aligned profiles can then be assessed with an independent, global similarity measure that is geared towards avalanche hazard assessment. Finally, through exploiting data aggregation and clustering methods, the similarity measure provides the foundation for grouping and summarizing snow profiles according to similar hazard conditions. For more background information refer to Herla, Horton, and Haegeli (accepted) <doi:10.5194/gmd-2020-171>.

URL http://www.avalancheresearch.ca/

License GPL (>= 3)

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

Language en-US

Imports dtw, grid, shiny

Depends R (>= 2.10), sarp.snowprofile

Suggests knitr, rmarkdown, dendextend, smacof

VignetteBuilder knitr

NeedsCompilation no

Author Florian Herla [aut, cre],
Pascal Haegeli [aut],
Simon Horton [aut],
SFU Avalanche Research Program [fnd]

Maintainer Florian Herla <fherla@sfu.ca>

Repository CRAN

Date/Publication 2020-12-03 08:30:03 UTC

R topics documented:

`ddateDistance` *Deposition Date Distance*

Description

Calculate the distance (i.e. dissimilarity) between two deposition dates

Usage

```
ddateDistance(
  ddate1,
  ddate2,
  normalizeBy = 5,
  clipWindow = FALSE,
```

```
  na.dist = 0.5
)
```

Arguments

ddate1	1D array of POSIX dates
ddate2	same format and length as ddate1
normalizeBy	Numeric scalar to be used for normalization, i.e. the number of days, that defines the distance value of 1
clipWindow	Should differences larger than 'normalizeBy' number of days be set to distance 'Infinity'? default FALSE.
na.dist	replace NA values with that distance

Value

An array of length(ddate1) containing the distances according to the configurations.

Author(s)

fherla

Examples

```
## create ddate arrays..
ddate <- as.POSIXct("2019/04/20 12:00", tz = "UTC")
ddate1 <- rep(ddate, 5)
ddate2 <- as.POSIXct(c("2019/04/12 08:00", "2019/04/16 10:00", "2019/04/20 12:00",
                       "2019/04/21 16:00", "2019/04/22 20:00"), tz = "UTC")

## .. and calculate distance:
ddateDistance(ddate1, ddate2, normalizeBy = 5)
```

---

distanceSP *Wrapper for dtwSP and simSP*

Description

Calculate the distance between two snowprofile objects by

Usage

```
distanceSP(query, ref, ...)
```

Arguments

query	The query snowprofile object (will be warped onto ref)
ref	The reference snowprofile object (will *not* be warped)
...	passed on to dtwSP

Details

1. Matching their layers and aligning them (i.e., warp one profile onto the other one)
2. Assessing the similarity of the aligned profiles based on avalanche hazard relevant characteristics
3. Convert the similarity score into a distance value between [0, 1]

This procedure is useful for clustering and aggregating tasks, given a set of multiple profiles.

Author(s)

fherla

See Also

dtwSP, simSP, medoidSP

`distMatSP` *Calculate a multidimensional distance matrix between two profiles*

Description

This routine calculates a distance matrix for two given profiles (`query` and `ref`). Analogously to other DTW routines, the query is arranged along the matrix rows, the ref along the columns. Every cell of the matrix represents the distance between the corresponding profile layers. The distance is calculated based on the specified layer properties (e.g., `hardness`, `gtype`, `ddate`). The routine calls subroutines to calculate the distance for each property and combines the normalized distances by weighted averaging.

Usage

```
distMatSP(
  query,
  ref,
  dims = c("hardness", "gtype"),
  weights = c(0.2, 0.8),
  gtype_distMat = sim2dist(grainSimilarity_align(FALSE)),
  prefLayerWeights = layerWeightingMat(FALSE),
  ddateNorm = 5,
  windowFunction = warpWindowSP,
  ...
)
```

Arguments

`query`	The query snowprofile object
`ref`	The ref snowprofile object
`dims`	Character vector containing the layer properties to calculate the distance over. Currently implemented are the properties `hardness`, `gtype`, `ddate`.
`weights`	Numeric vector of the same length as `dims` specifying the averaging weights to each element of dims.
`gtype_distMat`	A symmetric **distance** scoring matrix provided as data.frame that stores information about the distances between the encountered grain types of the provided profiles. Default is the corresponding distance matrix of grainSimilarity_align, cf. sim2dist.
`prefLayerWeights`	A matrix similar to `gtype_distMat`, but storing weights for preferential layer matching, e.g. defaults to layerWeightingMat; the higher the values for a given grain type pair, the more the algorithm will try to match those layers above others. To turn weighting scheme off, set `prefLayerWeights = NA`
`ddateNorm`	Normalize the deposition date distance by `ddateNorm` number of days. Numeric, default 5.
`windowFunction`	a window function analogous to warpWindowSP (*Note!* designed for a quadratic distance matrix, i.e. two profiles with identical numbers of layers. To resample the profiles see resampleSPpairs and examples below. Other compatible window functions can be found in dtw::dtwWindowingFunctions.)
`...`	arguments to the window function, e.g. `window.size`, `ddate.window.size`, ...

Value

A distance matrix of dimension (n x m), where n, m are the number of layers in the query and ref, respectively.

Note

For package developers: dot inputs to the function (i.e., `...`) also necessary to keep dtwSP highly flexible and customizable. Dot inputs may contain arguments that remain unused in this function.

Author(s)

fherla

See Also

resampleSPpairs

Examples

```
## call function with two snow profiles of unequal lengths, without using a window function:
dMat_noWindow <- distMatSP(SPpairs$A_modeled, SPpairs$A_manual, windowFunction = NA)
```

```
graphics::image(dMat_noWindow, main = "Default distance matrix without a warping window")


## compute distance based on grain type alone,
## and additionally disable preferential layer matching:
dMat <- distMatSP(SPpairs$A_modeled, SPpairs$A_manual, windowFunction = NA,
                  dims = "gtype", weights = 1, prefLayerWeights = NA)
graphics::image(dMat,
             main = "... only based grain type, and without preferential layer matching")


## to use a warping window, the profiles need to have equal numbers of layers:
## i.e., resample them first, then compute dMat with a warping window
plist <- resampleSPpairs(SPpairs$A_modeled, SPpairs$A_manual)
dMat <- distMatSP(plist$query, plist$ref)
graphics::image(dMat, main = "Default with warping window")
```

---

dtwSP *Calculate DTW alignment of two snow profiles*

Description

This is the core function of the package and allows to match layers between pairs of snow profiles to align them. It provides a variety of options, where the default values represent a good starting point to the alignment of most generic profiles.

Usage

```
dtwSP(
  query,
  ref,
  open.end = TRUE,
  checkGlobalAlignment = TRUE,
  keep.internals = TRUE,
  step.pattern = symmetricP1,
  resamplingRate = 0.5,
  rescale2refHS = TRUE,
  bottom.up = TRUE,
  top.down = TRUE,
  ...
)
```

Arguments

query	The query snow profile to be warped
ref	The reference snow profile to be warped against
open.end	Is an open end alignment desired?

`checkGlobalAlignment`
: If `open.end = TRUE`, do you want to check whether a global alignment performs better (i.e., `open.end = FALSE`), and use the optimal one of the computed alignments?

`keep.internals`
: Append resampled and aligned snow profiles as well as internal parameters to the output object?

`step.pattern`
: The local slope constraint of the warping path, defaults to Sakoe-Chiba's symmetric pattern described by a slope factor of P = 1, see dtw::stepPattern

`resamplingRate`
: Resampling rate for a regular depth grid; can also be a vector that provides the depth grid. Set to `NA` to keep the (original, likely irregular) depth grid (see Details, bullet point 2.2).

`rescale2refHS`
: Rescale the query snow height to match the ref snow height?

`bottom.up`
: Compute an open.end alignment from the ground upwards?

`top.down`
: Compute an open.end alignment from the snow surface downwards?

`...`
: Arguments passed to `distMatSP` and `dtw`, e.g.
  - `dims`, `weights` (defaults specified in `distMatSP`)
  - `ddateNorm`, numeric, normalize deposition date (default specified in `distMatSP`)
  - `windowFunction`, default `warpWindowSP`
  - `window.size`, `ddate.window.size` (defaults specified in `warpWindowSP`)
  - `gtype_distMat`, (default specified in `distMatSP`), cf. e.g. grainSimilarity_align
  - `prefLayerWeights`, weighting matrix for preferential layer matching, e.g. layerWeightingMat

Details

The individual steps of aligning snow profiles:

1. (optional) **Rescale** the profiles to the same height (cf., scaleSnowHeight)
2. **Resample** the profiles onto the same depth grid. 2 different approaches:
   - regular grid with a sampling rate that is provided by the user (recommended, cf., resampleSP). This approach requires to rescale the profiles.
   - irregular grid that includes all layer interfaces within the two profiles (i.e., set `resamplingRate = NA`) (cf., resampleSPpairs)
3. Compute a weighted **local cost matrix** from multiple layer characteristics (cf., distMatSP)
4. **Match the layers** of the profiles with a call to dtw (eponymous R package)
5. Align the profiles by **warping** the query profile onto the reference profile (cf., warpSP)
6. (optional) If the function has been called with multiple different boundary conditions (global, top-down, or bottom-up alignments), the optimal alignment as determined by simSP will be returned.

Value

An alignment object of class 'dtwSP' is returned. This is essentially a list with various information about the alignment. If `keep.internals = TRUE`, the resampled snow profiles 'query', 'reference' and 'queryWarped', as well as the 'costMatrix' and 'directionMatrix' are elements of the returned object.

Note

While DTW can be applied to sequences of different lengths, it is necessary that the two snow profiles have the same number of layers for optimally warping one profile onto the other one and thereby aligning them. As long as the profiles are rescaled and resampled this requirement is satisfied. This requirement is inconvenient only if the profiles are not supposed to be rescaled (e.g. in a layer tracking application). In these cases, a workaround is to rescale the profiles anyway, but increase the window size of the warping window proportionally to the snow height offset. In these cases also consider including layer date information to reduce the likelihood of bad alignments.

Furthermore, the alignment based on grain type information is currently only possible for specific grain types. These grain types require a pre-defined distance or similarity, such as given by grainSimilarity_align. If your profile contains other grain types, you are required to define your custom `grainSimilarity` matrix.

Author(s)

fherla

See Also

plotSPalignment, simSP

Examples

```
dtwAlignment <- dtwSP(SPpairs$A_modeled, SPpairs$A_manual, open.end = FALSE)

## dtwSP object:
summary(dtwAlignment)
plot(dtwAlignment$queryWarped)
plotSPalignment(dtwAlignment = dtwAlignment)
```

`extractFromScoringMatrix` *Extract from Scoring matrix*

Description

Vectorized function to efficiently extract elements from scoring matrix of type data.frame

Usage

```
extractFromScoringMatrix(
  ScoringFrame,
  grainType1,
  grainType2,
  profile_handle = NULL
)
```

Arguments

ScoringFrame	Scoring matrix of type data.frame (needs to be of symmetric, matrix like format)
grainType1	character vector (yes, vector!) of grain type contained in ScoringFrame
grainType2	same as grainType1
profile_handle	character or numeric handle that links a potential warning message to the set of grain types, if an unknown grain type is encountered (must be of length = 1)

Value

numeric vector of length grainType1 with the elements of ScoringFrame that are defined by grainType1 and grainType2

Author(s)

fherla

---

flipLayers *Flip snow profile layers top down*

Description

Flip snow profile layers top down

Usage

```
flipLayers(x)
```

Arguments

x	snowprofile or snowprofileLayers object with layers to be flipped

Value

same object with layers dataframe flipped upside down

Note

only do that with a specific reason (better, don"t do it!), as all functions with snowprofile objects are designed to have the layers increase in height.

`grainSimilarity_align` *Grain Type similarity matrix for DTW alignments*

Description

Get the relative similarity matrix of grain types as used for snow profile alignments. This similarity matrix considers the formation and metamorphosis of grain types, as well as quirks of the SNOWPACK model.
grainSimilarity_evaluate is an analogous matrix designed for assessing the similarity between two profiles, which requires considering the resulting avalanche hazard implications of grain types.
The domain is [0, 1] — 1 representing identical grain types. The column 'NA' can be used for unknown grain types.

Usage

```
grainSimilarity_align(triag = TRUE)
```

Arguments

`triag`	Return a triangular matrix (TRUE, default) or a symmetric matrix (FALSE)

Value

data.frame, either triangular or symmetric

Author(s)

fherla

See Also

grainSimilarity_evaluate, layerWeightingMat

Examples

```
## "similarity" matrix:
simMat <- grainSimilarity_align()
print(simMat)

## equivalent "distance" matrix:
distMat <- sim2dist(grainSimilarity_align())
print(distMat)
```

grainSimilarity_evaluate *Grain type similarity matrix for evaluation purposes*

Description

Similar to grainSimilarity_align, but designed for assessing the similarity between snow profiles based on avalanche hazard relevant characteristics. To be used in combination with simSP.

Usage

```
grainSimilarity_evaluate(triag = TRUE)
```

Arguments

triag	Return a triangular matrix (TRUE, default) or a symmetric matrix (FALSE)

Value

data.frame, either triangular or symmetric

Author(s)

fherla

Examples

```
simMat <- grainSimilarity_evaluate()
print(simMat)
```

hardnessDistance *Difference in Hand Hardness*

Description

Calculate the difference (i.e. distance) in hand hardness

Usage

```
hardnessDistance(hardness1, hardness2, normalize = FALSE, absDist = TRUE)
```

Arguments

hardness1	character or numeric hand hardness value (1D array)
hardness2	character or numeric hand hardness value (1D array)
normalize	Should result be normalized? boolean, default False.
absDist	Interested in absolute distance? default True.

Value

numeric Hand Hardness Distance

Author(s)

fherla

---

interactiveAlignment *Run interactive alignment app*

Description

This app allows to interactively explore the alignment of two snowprofiles, which are either given as input to this function, or are uploaded to the app interactively as caaml files. Example profiles are also provided in the app.

Usage

```
interactiveAlignment(query = NaN, ref = NaN)
```

Arguments

query	an optional query snowprofile
ref	an optional reference snowprofile

Value

An interactive session will be started

Author(s)

fherla

Examples

```
if (FALSE){  # this example won't be started in package tests.

## start app and choose profiles from within the app:
interactiveAlignment()

## start app with package internal profile data (from `sarp.snowprofile`):
interactiveAlignment(query = SPpairs$A_modeled, ref = SPpairs$A_manual)

}
```

`layerWeightingMat` *Weighting scheme for preferential layer matching*

Description

A matrix, of the same form as grainSimilarity_align, but containing weighting coefficients for preferential layer matching based on the grain types of the layers.

Usage

```
layerWeightingMat(triag = TRUE)
```

Arguments

`triag`	Return a triangular matrix (TRUE, default) or a symmetric matrix (FALSE)

Value

data.frame, either triangular or symmetric

Author(s)

fherla

Examples

```
weightsMat <- layerWeightingMat()
print(weightsMat)
```

`medoidSP` *Find the medoid snow profile among a group of profiles*

Description

Find the medoid snowprofile among a group of profiles, based on their pairwise dissimilarity. Either provide a list of `snowprofile` objects, or a precomputed distance matrix.
If you provide a list of profiles the profiles can optionally be rescaled and resampled before the distance matrix for the medoid calculation is computed. When computing the distance matrix this routine calls distanceSP for *every possible pair* of profiles among the group. During that call the profile pair is aligned by dtwSP and the aligned pair is evaluated by simSP. Note that the number of possible profile pairs grows exponentially with the number of profiles in the group (i.e., O(n^2) calls, where n is the number of profiles in the group).

Usage

```
medoidSP(
  profileList = NULL,
  rescale_resample = TRUE,
  retDistmat = FALSE,
  distmat = NULL,
  verbose = FALSE,
  resamplingRate = 0.5,
  ...
)
```

Arguments

`profileList`	List of snowprofile objects
`rescale_resample`	Do you want to uniformly rescale and resample the set of profiles prior to calculating the distance matrix?
`retDistmat`	Do you want to *return* the pairwise distance matrix?
`distmat`	If you have a precalculated distance matrix, provide it here to compute the medoid on it.
`verbose`	print pairwise distance matrix? default FALSE
`resamplingRate`	The resampling rate that is used for the whole set if `rescale_resample = TRUE`
`...`	arguments passed to distanceSP

Value

If `retDistmat = FALSE` return the (named) index of the medoid snow profile, otherwise return a list with the elements `iMedoid` and `distmat`.

Author(s)

fherla

See Also

reScaleSampleSPx

Examples

```
this_example_runs_about_5s <- TRUE
if (!this_example_runs_about_5s) {  # exclude from cran checks

  ## take a list of profiles
  grouplist <- SPgroup[1:4]
  plot(grouplist, SortMethod = 'unsorted', labelOriginalIndices = TRUE)

  ## calulate medoid profile
  idxMedoid <- medoidSP(grouplist)
  representativeProfile <- grouplist[[idxMedoid]]
  plot(representativeProfile, main = paste0("medoid (i.e., profile ", idxMedoid, ")"))

}
```

---

mergeIdentLayers *Merge layers with identical properties*

Description

Merge adjacent layers that have identical properties, such as grain type, hardness etc..

Usage

```
mergeIdentLayers(x, properties = c("hardness", "gtype"))
```

Arguments

x	a snowprofile or snowprofileLayers object with *height* grid information
properties	a character array of layer properties that are considered when searching for identical layers (e.g., hardness, gtype, ...)

Value

A new snowprofileLayers object will be returned with the dimensions height, hardness, gtype and any other properties given in 'properties'. Depth and thickness information will be autocalculated. For snowprofile objects, the field 'changes' will be initialized or extended.

Author(s)

fherla

Examples

```
## Merge identical layers based on hardness and grain type:
fewerLayers <- mergeIdentLayers(x = SPpairs$A_modeled, properties = c("hardness", "gtype"))
summary(SPpairs$A_modeled)[, c("hs", "nLayers")]
summary(fewerLayers)[, c("hs", "nLayers")]

## compare profile plots before and after merging (i.e., appear identical!)
opar <- par(no.readonly =TRUE)
par(mfrow = c(1, 2))
plot(SPpairs$A_modeled, main = "original", ylab = "Snow height")
plot(fewerLayers, main = "merged layers", ylab = "Snow height")
par(opar)
```

`plotCostDensitySP` *Plot alignment cost density and warping path*

Description

Plot alignment cost density and warping path, optionally with the two snow profiles plotted in the margins along the axes.

Usage

```
plotCostDensitySP(
  alignment,
  localCost = TRUE,
  labelHeight = FALSE,
  marginalPros = TRUE,
  pathCol = "black",
  target = FALSE,
  movingTarget = FALSE,
  tlty = "dotted",
  tlwd = 1.5,
  tcol = "black",
  tcex = 1.5,
  cex.lab = 1,
  xlab = NULL,
  ylab = NULL,
  ...
)
```

Arguments

`alignment`	object from dtwSP
`localCost`	plot *local* cost matrix, otherwise plot accumulated global cost.

labelHeight	plot axes in units of height (cm) or in unitless (i.e., layer index).
marginalPros	plot profiles in margins along the axes. default TRUE
pathCol	color of warping path
target	draw horizontal & vertical lines from matrix cells to corresponding layers in the (marginal) profiles. Provide either a vector of length 1 (i.e., index of warping path) or length 2 (i.e., x, y coordinates in terms of layer indices), or a matrix with 2 columns, specifying (x, y) if you desire multiple 'targets'
movingTarget	Do you want to draw the warping path only partially, from the origin to the target cross? Only possible if target cross is given as a scalar! default = FALSE (Useful to create GIF animations of a moving path)
tlty	target lty
tlwd	target lwd
tcol	target col
tcex	target cex
cex.lab	cex of axis labels (cf. par)
xlab	x-axis label to change default labeling
ylab	y-axis label to change default labeling
...	forwarded to par

Note

If you can't see the axis labels, try e.g., `par(oma = c(3,3,0,0))` before calling the function. Note, there seems to be a problem (only sometimes) with the left-hand labels that are for some reason not plotted parallel to the axis. Also, the routine is not bulletproof with respect to drawing 'targets'. Apologies for any inconveniences!

Author(s)

fherla

Examples

```
## first align profiles:
dtwAlignment <- dtwSP(SPpairs$A_modeled, SPpairs$A_manual, open.end = FALSE)

## then plot cost density:
plotCostDensitySP(dtwAlignment)

## label height instead of layer index, and don't show warping path:
plotCostDensitySP(dtwAlignment, labelHeight = TRUE, pathCol = "transparent")

## draw lines to the cell that corresponds to the DH and SH layers
plotCostDensitySP(dtwAlignment, target = c(191, 208))

## "moving target", i.e., draw warping path only from origin to target:
plotCostDensitySP(dtwAlignment, target = 200, movingTarget = TRUE)
```

```
plotCostDensitySP(dtwAlignment, target = 266, movingTarget = TRUE)


## A cool GIF can be created from frames like those
create_GIF <- FALSE
if (create_GIF){
  nPath <- length(dtwAlignment$index1)
  resolution <- 100  # i.e. super low, make value smaller for smoother GIF
  for (k in seq(1, nPath, by = resolution)) {
    plotCostDensitySP(dtwAlignment, target = k, movingTarget = TRUE)
  }
}
```

`plotSPalignment` *Align and plot two snow profiles using DTW*

Description

This is a plotting routine for the DTW alignment of two snow profiles. Either provide two snow profiles or a `dtwSP` alignment object. Don't resize the figure, otherwise the plotted alignment segments will not be in correct place anymore! If you need a specific figure size, use `grDevices::png` with a width/height aspect ratio of about 5/3.

Usage

```
plotSPalignment(
  query,
  ref,
  dtwAlignment = NULL,
  open.end = TRUE,
  keep.alignment = FALSE,
  plot.costDensity = FALSE,
  plot.warpedQuery = TRUE,
  label.ddate = FALSE,
  segCol = "gray70",
  segLty = "dotted",
  segLwd = 1,
  segTidy = FALSE,
  segInd = TRUE,
  segEmph = NA,
  cex = 1,
  mainQu = "query",
  mainRef = "reference",
  mainQwarped = "warped query",
  emphasizeLayers_qu = FALSE,
  emphasizeLayers_ref = FALSE,
  ...
)
```

Arguments

query	The query snowprofile to be warped
ref	The reference snowprofile to be warped against
dtwAlignment	dtwSP object (optional)
open.end	Is an open end alignment desired? boolean
keep.alignment	Return dtwSP object with resampled query, ref and warped query? boolean
plot.costDensity	First graph, plotCostDensitySP with warping path? boolean, default = FALSE
plot.warpedQuery	plot warped query additionally to query, ref and alignment segments? (i.e. three pane plot) boolean, default = TRUE
label.ddate	Label deposition date in profiles? (Only possible if ddate is given in 'dims', cf distMatSP)
segCol	Color of alignment segments. Passed to gpar, default = "gray70"
segLty	Linestyle of alignment segments. Passed to gpar, default = "dotted"
segLwd	Linewidth of alignment segments, default = 1
segTidy	Tidy up alignment segments, if profiles have not been resampled? boolean, default FALSE i.e. one segment line per (synthetic) layer interface -> supports visual understanding of alignment, but is also often confusing (segTidy currently only implemented for tidying up to gtype and hardness interfaces)
segInd	Index vector of query layers that will get alignment segments drawn. Note, that the profiles might get resampled, so pre-calculate your correct indices!
segEmph	Index vector of query layers, the alignment segments of which will be emphasized (thick and red). Note, that the profiles might get resampled, so pre-calculate your correct indices!
cex	font size, cf. par
mainQu	subtitle for query subfigure
mainRef	subtitle for reference subfigure
mainQwarped	subtitle for warped query subfigure
emphasizeLayers_qu	emphasize Layers in query, see plot.snowprofile
emphasizeLayers_ref	emphasize Layers in reference, see plot.snowprofile
...	Arguments passed to distMatSP and dtwSP

Value

dtw object with the resampled '$query' and '$reference', as well as the warped query '$queryWarped' (only if keep.alignment is TRUE)

Author(s)

fherla

Examples

```
plotSPalignment(SPpairs$B_modeled1, SPpairs$B_modeled2)

plotSPalignment(SPpairs$B_modeled1, SPpairs$B_modeled2, dims = c("gtype"), weights = c(1))

## alternatively keep alignment:
alignment <- plotSPalignment(SPpairs$B_modeled1, SPpairs$B_modeled2, keep.alignment = TRUE)
print(paste("Similarity between profiles:", alignment$sim))

## alternatively, with precomputed alignment and emphasized layer matches:
dtwAlignment <- dtwSP(SPpairs$A_modeled, SPpairs$A_manual, open.end = FALSE)
plotSPalignment(dtwAlignment = dtwAlignment, segEmph = c(190, 192))

## directly after plotting, add text to figure:
grid::grid.text("Profiles SPpairs$A (modeled/manual)", x = 0.5, y = 0.8,
                gp = grid::gpar(fontsize=12, col="grey"))
```

---

resampleSP *Resample snowprofile*

Description

Resample an individual snow profile onto a new depth-grid (i.e., height-grid).

Usage

```
resampleSP(x, h = 0.5, n = NULL)
```

Arguments

x	snowprofile (or snowprofileLayers) object
h	Sampling rate (i.e. constant depth increment) in centimeters, if given as scalar (default is 0.5 cm). Layers smaller than the scalar h will not be resolved in the resampled profile. Can also be a vector specifying the desired *height* grid of the resampled profile (useful for non-constant increments). But, be WARNED, that a meaningless grid will produce colorful but senseless output!
n	Number of layers in resampled profile (optional). *A given n will overrule a conflicting h!*

Details

This routine alters how the layer information of snow profiles is *stored* without changing how the profiles appear. Note, however, that only layer properties that are constant within the individual layers will be resampled: i.e., `height`, `hardness`, `gtype`, `ddate` will be resampled. However, `temperature`, for example, will not be resampled, because it is not constant within layers.

Value

resampled snowprofile with the same metadata as x, but resampled "layers". **Note** that only the following layer properties will be resampled: `height`, `hardness`, `gtype`, `ddate`. If input was a snowprofileLayers object, the output will be, too.

Author(s)

fherla

See Also

resampleSPpairs, mergeIdentLayers

Examples

```
## (1) constant sampling rate of 1 cm:
profileResampled <- resampleSP(SPpairs$A_modeled, h = 1.0)

## compare profile summary before and after resampling:
summary(SPpairs$A_modeled)[, c("hs", "nLayers")]
summary(profileResampled)[, c("hs", "nLayers", "changes")]
head(profileResampled$layers)

## compare profile plots before and after resampling (i.e., appear identical!)
opar <- par(no.readonly=TRUE)
par(mfrow = c(1, 2))
plot(SPpairs$A_modeled, main = "original", ylab = "Snow height")
plot(profileResampled, main = "resampled", ylab = "Snow height")
par(opar)

## (2) resample to 150 layers:
profileResampled <- resampleSP(SPpairs$A_manual, n = 150)
summary(profileResampled)[, c("hs", "nLayers", "changes")]
head(profileResampled$layers)


## (3) resample onto arbitrarily specified grid
## (issues a warning when the new-grid HS deviates too much from the original HS)
irregularGrid <- c(2 + cumsum(c(0, c(10, 15, 5, 1, 3, 30, 50))), 120)
profileResampled <- resampleSP(SPpairs$A_manual, h = irregularGrid)
```

`resampleSPpairs` *Resample a pair of profiles*

Description

Resample a pair of (irregularly layered) profiles onto the smallest common height grid. To reduce data storage this routine can be used to merge layers based on specified layer properties, if the input profiles have been resampled earlier, or if due to other reasons existing layers in the individual profiles can be merged. In summary, this routine alters how the layer information of snow profiles is *stored* without changing how the profiles appear.

Usage

```
resampleSPpairs(
  query,
  ref,
  mergeBeforeResampling = FALSE,
  dims = c("gtype", "hardness")
)
```

Arguments

`query`	query snowprofile or snowprofileLayers object
`ref`	reference snowprofile or snowprofileLayers object
`mergeBeforeResampling`	shall adjacent layers with identical layer properties be merged? (boolean)
`dims`	layer properties to consider for a potential merging

Details

The smallest common height grid is found by

1. extract all unique layer interfaces in both profiles
2. resample each profile with the above height grid,
   (!) but set all height values that exceed each's max snowheight to that max snowheight!

Value

a list with the resampled input objects under the entries `query` and `ref`.

Author(s)

fherla

See Also

resampleSP, mergeIdentLayers

Examples

```
## initial situation before mutual resampling:
## two profiles with different snow heights and different numbers of layers
summary(SPpairs$A_manual)[, c("hs", "nLayers")]
summary(SPpairs$A_modeled)[, c("hs", "nLayers")]
opar <- par(no.readonly=TRUE)
par(mfrow = c(1, 2))
plot(SPpairs$A_manual, main = "Initial profiles before resampling",
     ylab = "Snow height", ylim = c(0, 272))
plot(SPpairs$A_modeled, ylab = "Snow height", ylim = c(0, 272))

## resampling:
resampledSPlist <- resampleSPpairs(SPpairs$A_manual, SPpairs$A_modeled,
                                   mergeBeforeResampling = TRUE)

## two profiles with different snow heights and IDENTICAL numbers of layers
summary(resampledSPlist$query)[, c("hs", "nLayers")]
summary(resampledSPlist$ref)[, c("hs", "nLayers")]
plot(resampledSPlist$query, main = "Profiles after resampling",
     ylab = "Snow height", ylim = c(0, 272))
plot(resampledSPlist$ref, ylab = "Snow height", ylim = c(0, 272))
par(opar)
```

---

reScaleSampleSPx *Rescale and resample a snow profile list*

Description

Rescale and resample all snow profiles provided in a list to an identical snow height and resampling rate.

Usage

```
reScaleSampleSPx(SPx, resamplingRate = 0.5, scHeight = median, ...)
```

Arguments

SPx	list of snowprofile objects
resamplingRate	resampling rate, units in centimeters
scHeight	a function that calculates the resulting height from the profiles, default median
...	arguments passed on to the function provided in scHeight

Value

A list with the first entry $set storing the rescaled and resampled profile list, the second entry $maxHS stores the maximum snow height found among the profiles

Author(s)

fherla

Examples

```
## let's take the 'SPgroup' object as profile list
SPrr <- reScaleSampleSPx(SPgroup)
print(paste0("max height before rescaling: ", SPrr$maxHS, " cm"))
print(paste0("rescaled height: ", SPrr$set[[1]]$hs, " cm"))
plot(SPrr$set, SortMethod = 'unsorted')
```

---

rmZeroThicknessLayers *Remove layers with a thickness of 'zero cm'*

Description

Find layers in a snow profile that are zero cm thick (i.e. height vector stays constant for some layers, even though grain types or hardness may change). Then, either remove those layers, or reset them with the layer characteristics of the lower adjacent (non-zero-thickness) layer. In the latter case (i.e., reset), the number of layers won't change, but those non-zero thickness layers will be made ineffective. This procedure is particularly necessary for warping snow profiles (cf., dtwSP, warpSP).

Usage

```
rmZeroThicknessLayers(x, rm.zero.thickness = TRUE)
```

Arguments

`x`	A `snowprofile` or `snowprofileLayers` object
`rm.zero.thickness`	Want to remove zero-thickness layers from profile? boolean, default TRUE. If FALSE, those zero-thickness layers will be reset to the lower adjacent (non-zero-thickness) layer; thus, the number of layers won't be changed.

Value

A modified copy of the input object. For snowprofile objects, the field $changes will be initialized or extended.

Author(s)

fherla

scaleSnowHeight *Scale total height of a snow profile*

Description

Scale the snow height of a snow profile either (1) based on another profile, or (2) based on a provided (predetermined) snow height. This function can therefore be used to scale two snow profiles to an identical snow height by scaling the height vector of the (query) profile against the height vector of the (reference) profile.

Usage

```
scaleSnowHeight(query, ref = NA, height = NA)
```

Arguments

query	the query snow profile (whose height vector will be scaled)
ref	the reference snow profile (whose total snow height will be used as the reference height for the scaling)
height	an optional reference height that can be given instead of the query profile

Value

query profile with scaled height vector

Author(s)

fherla

sim2dist *Convert 'similarity' matrix to 'distance' matrix*

Description

Convert a 'similarity' matrix to 'distance' matrix. *Note* that the similarity must be normalized (i.e. within [0, 1])

Usage

```
sim2dist(SimMat)
```

Arguments

SimMat	similarity matrix of type data.frame with ranges [0, 1]

Value

copy of input data.frame with similarities inverted to distances (i.e. dist = 1 - sim)

Author(s)

fherla

Examples

```
## the 'swissSimilarityMatrix' as similarity and as distance
graphics::image(as.matrix(swissSimilarityMatrix))
graphics::image(as.matrix(sim2dist(swissSimilarityMatrix)))
```

---

simSP *Similarity measure between snow profile pairs*

Description

A simple similarity score –based on avalanche hazard relevant characteristics– is calculated for two snow profiles that have been aligned onto the same height grid (either through DTW or resampling). If one profile contains more layers than the other one, the layers with a non-matched height represent missing layers and will be treated accordingly. The similarity score is weighted according to grain type classes to accredit the importance of weak layers, new snow and crusts against less important bulk layers. The similarity measure is compatible with top-down alignments and is symmetric with respect to its inputs, i.e. `simSP(P1,P2) == simSP(P2,P1)`.

Usage

```
simSP(
  ref,
  qw,
  gtype_distMat = sim2dist(grainSimilarity_evaluate(triag = FALSE)),
  verbose = FALSE,
  returnDF = FALSE
)
```

Arguments

`ref`	snowprofile object 1
`qw`	snowprofile object 2 (matched layers need to be on the same height grid of ref)
`gtype_distMat`	a distance matrix that stores **distance** information of grain types (*Be careful* to convert similarities, as in grainSimilarity_evaluate, into dissimilarities with sim2dist.)
`verbose`	print similarities of different grain classes to console? default FALSE
`returnDF`	additionally return the similarities of the grain classes as data.frame (analogously to verbose); the return object then has the fields $sim and $simDF

Details

The grain classes contain the following grain types:

- weak layers (wl): SH and DH
- new snow (pp): PP and DF
- crusts (cr): MFcr and IF
- bulk: the rest (i.e., predominantly RG, FC, FCxr — MF falls also in here, will maybe be adjusted in future.)

Value

Either a scalar similarity between [0, 1] with 1 referring to the two profiles being identical, or (if `returnDF` is TRUE) a list with the elements $sim and $simDF.

Examples

```
## first align two profiles
alignment <- dtwSP(SPpairs$A_modeled, SPpairs$A_manual)

## then assess the similarity of the aligned profiles
SIM <- simSP(alignment$queryWarped, alignment$reference, verbose = TRUE)
```

`swissSimilarityMatrix` *Similarity Matrix of Snow Grain Types*

Description

as defined by Lehning et al (2001). A similarity of 1 represents identity, 0 represents total dissimilarity.

Usage

```
swissSimilarityMatrix
```

Format

A data.frame

Examples

```
print(swissSimilarityMatrix)
```

`warpSP` *Warp one snow profile onto another one*

Description

After the DTW alignment of two profiles, the maps between the two profiles can be used to warp one profile onto the other profile. In other words, the layer thicknesses of the warped profile are adjusted to optimally align with the corresponding layers of the other profile.

Usage

```
warpSP(alignment, whom = NA)
```

Arguments

`alignment`	DTW alignment object from dtwSP containing the two profiles (i.e., called `dtwSP(...,keep.internals = TRUE)`)
`whom`	whom to warp? "query" (= "jmin"), "imin", "queryTopDown" (= "jminTopDown"), "iminTopDown", "ref"; if 'NA' the routine determines that itself from the structure of the alignment object. (see Details)

Details

After this procedure, the thickness of some layers can be zero, which leads to the layers disappearing.

This function is automatically called in `dtwSP(...,keep.internals = TRUE)` to warp the query profile onto the reference profile.

Whom to warp: There exist 8 different options, 4 for warping the query onto the ref and 4 for vice versa. The 4 options for warping the query onto the ref are:

- global alignment / partial alignment where entire query is matched to subsequence of ref ("jmin")
- partial alignment where entire ref is matched to subsequence of query ("imin")
- partial top down alignment where entire query is matched to subsequence of ref ("jminTopDown")
- partial top down alignment where entire ref is matched to subsequence of query ("iminTopDown")

For the other case, warping the ref onto the query, only the equivalent of the first option is implemented.

Value

Returns the input alignment object including the element alignment$queryWarped (or $referenceWarped), which are the warped snow profiles. The class of the alignment object is altered to "dtwSP", but still inherits "dtw".

Author(s)

fherla

Examples

```
## first align profiles
alignment <- dtwSP(SPpairs$A_modeled, SPpairs$A_manual, open.end = FALSE)

## warp reference profile onto query profile:
refWarped <- warpSP(alignment, whom = "ref")$referenceWarped
opar <- par(no.readonly =TRUE)
par(mfrow = c(1, 2))
plot(alignment$query, main = "query")
plot(refWarped, main = "warped reference")
par(opar)
```

---

warpWindowSP *Restrict the DTW warping window for snow profiles alignment*

Description

Given a *quadratic* matrix, this function sets all elements of the matrix that are outside the so-called warping window to NA. The warping window is a slanted band of constant width around the main diagonal (i.e., *Sakoe-Chiba*-band), and it's size can be controlled with function arguments.

Usage

```
warpWindowSP(
  iw,
  jw,
  iheight,
  jheight,
  iddate,
  jddate,
  profile.size,
  profile.height,
  window.size = 0.3,
  ddate.window.size = Inf,
  ...
)
```

Arguments

iw	matrix of integers indicating their row number (cf., ?row)
jw	matrix of integers indicating their column number (cf., ?col)

iheight	matrix of query height filled into the columns of the matrix
jheight	matrix of ref height filled into the rows of the matrix
iddate	same as iheight, but containing deposition date information
jddate	same as jheight, but containing deposition date information
profile.size	number of layers in each of the profiles (scalar)
profile.height	snow height of the profiles (scalar)
window.size	percentage of profile.size or profile.height defining the size of the warping window (i.e., the most restrictive of the two will be applied)
ddate.window.size	number of days that exclude layers from the warping window if their deposition dates differ by more than these days
...	unused—but important to be able to provide other warping functions to distMatSP

Details

Note that the function is designed for cost matrices derived from pairs of snow profiles that have equal numbers of layers (cf., resampleSPpairs). The function takes many matrix-like inputs, all of which need to be of the size (profile.size x profile.size), i.e., square matrices of the size 'number of layers'.

See Also

dtw::dtwWindowingFunctions

Index