Package ‘sarp.snowprofile.alignment’

February 8, 2024

Title Snow Profile Alignment, Aggregation, and Clustering

Version 1.2.2

Date 2024-02-07

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. In particular, this package allows for averaging large numbers of snow profiles with DTW Barycenter Averaging and thereby facilitates the computation of individual layer distributions and summary statistics that are relevant for avalanche forecasting purposes.
For more background information refer to Herla, Horton, Mair, and Haegeli (2021) <doi:10.5194/gmd-14-239-2021>, and Herla, Mair, and Haegeli (2022) <doi:10.5194/tc-16-3149-2022>.

URL `https://avalancheresearch.ca/`

License GPL (>= 3)

Encoding UTF-8

LazyData true

RoxygenNote 7.2.2

Language en-US

Imports dtw, grid, data.table

Depends R (>= 2.10), sarp.snowprofile (>= 1.2.1)

Suggests knitr, rmarkdown, shiny, dendextend, smacof, testthat, progress

VignetteBuilder knitr

NeedsCompilation no

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

Maintainer Florian Herla <fherla@sfu.ca>

Repository CRAN

Date/Publication 2024-02-08 10:20:02 UTC

R topics documented:

averageSP *Average a group of snow profiles*

Description

The functions dbaSP and averageSP implement Dynamic Time Warping Barycenter Averaging of snow profiles. The convenient wrapper averageSP takes care of choosing several appropriate initial conditions and picking the optimal end result (by minimizing the mean squared error between the average profile and the profile set). To pay appropriate attention to (thin) weak layers, weak layers need to be labeled in the profiles. You can either do that manually before calling this routine to suit your personal needs, or you can provide specific properties (in `classifyPWLs`) so that weak layers be labeled according to these properties by sarp.snowprofile::labelPWL. For more details, refer to the reference paper.

Usage

```
averageSP(
  SPx,
  n = 5,
  sm = summary(SPx),
  progressbar = requireNamespace("progress", quietly = TRUE),
  progressbar_pretext = NULL,
  classifyPWLs = list(pwl_gtype = c("SH", "DH")),
  classifyCRs = list(pwl_gtype = c("MFcr", "IF", "IFsc", "IFrc")),
  proportionPWL = 0.5,
  breakAtSim = 0.9,
  breakAfter = 2,
  verbose = FALSE,
  tz = "auto",
  ...
)

dbaSP(
  SPx,
  Avg,
  sm = summary(SPx),
  resamplingRate = 0.5,
  proportionPWL = 0.3,
  maxiter = 10,
  breakAtSim = 0.99,
  breakAfter = 1,
  plotChanges = FALSE,
  verbose = TRUE,
  tz = "auto",
  ...
)
```

Arguments

SPx	SPx a snowprofileSet object. Note that the profile layers need to contain a column called `$layerOfInterest` which classifies weak layers. While averageSP will label weak layers automatically if not done by the user beforehand, dbaSP won't do that but fail instead!; consider thinking about how you want to label weak layers, see Description, `classifyPWLs` below, and the references. Also note, that if you wish to average the *rescaled* profile set, do so manually before calling this function (see examples).
n	the number of initial conditions that will be used to run dbaSP; see also chooseICavg.
sm	a summary of `SPx` metadata
progressbar	should a progressbar be displayed (the larger n, the more meaningful the progressbar)
progressbar_pretext	a character string to be prepended to the progressbar (mainly used by higher level cluster function)
classifyPWLs	an argument list for a function call to sarp.snowprofile::findPWL which returns relevant PWLs for identifying initial conditions. **Importantly**, these arguments will also be used to label weak layers in the profiles, if these labels do not yet exist in the layers objects as column `$layerOfInterest`. Check out the documentation of `findPWL` to familiarize yourself with your manifold options!
classifyCRs	an argument list for a function call to sarp.snowprofile::findPWL which returns relevant crusts for identifying initial conditions.
proportionPWL	decimal number that specifies the proportion required to average an ensemble of grain types as weak layer type. A value of 0.3, for example, means that layers will get averaged to a PWL type if 30% of the layers are of PWL type. Meaningful range is between `[0.1, 0.5]`. Values larger than 0.5 get set to 0.5.
breakAtSim	stop iterations when simSP between the last average profiles is beyond that value. Can range between `[0, 1]`. Default values differ between dbaSP and averageSP.
breakAfter	integer specifying how many values of simSP need to be above `breakAtSim` to stop iterating. Default values differ between dbaSP and averageSP.
verbose	print similarities between old and new average in between iterations?
tz	timezone of profiles; necessary for assigning the correct timezone to the average profile's ddate/bdate. Either `'auto'` or a timezone known to `[as.POSIXct]`.
...	alignment configurations which are passed on to dbaSP and then further to dtwSP. Note, that you can't provide `rescale2refHS`, which is always set to FALSE. If you wish to rescale the profiles, read the description of the `SPx` parameter and the examples.
Avg	the initial average snow profile: either a snowprofile object or an index to an initial average profile in SPx
resamplingRate	Resampling rate for a regular depth grid among the profiles
maxiter	maximum number of iterations
plotChanges	specify whether and how you want to plot the dba process: either `FALSE`, 'TRUE=='iterations'`, or` 'averages+last'`

Details

Technical note: Since the layer characteristics of the average profile represent the median characteristics of the individual profiles, it can happen that ddates of the averaged layers are not in a monotonical order. That is, of course unphysical, but we specifically decided not to override these values to highlight these slight inconsistencies to users, so that they can decide how to deal with them. As a consequence, the function sarp.snowprofile::deriveDatetag does not work for these average profiles with ddate inconsistencies, but throws an error. The suggested workaround for this issue is to apply that function to all individual profiles *before* computing the average profile. This ensures that bdates or datetags are also included in the average profile.

For developers: Including new variables into the averaging/dba routines can be done easily by following commit #9f9e6f9

Value

A list of class `avgSP` that contains the fields

- `$avg`: the resulting average profile
- `$set`: the corresponding resampled profiles of the group
- `$call`: (only with `averageSP`) the function call
- `$prelabeledPWLs`: (only with `averageSP`) boolean scalar whether PWLs (or any other layers of interest) were prelabeled before this routine (`TRUE`) or labeled by this routine with the defaults specified in `classifyPWLs` (`FALSE`)

The profile layers of the average profile refer to the median properties of the predominant layers. For example, if you labeled all SH/DH layers as your 'layersOfInterest', and you find a SH or DH layer in the average profile, then it means that the predominant grain type is SH/DH (i.e., more profiles than specified in `proportionPWL` have that layer) and layer properties like hardness, p_unstable, etc refer to the median properties of these SH/DH layers. If you find a RG layer in your average profile, it means that most profiles have that RG layer and the layer properties refer to the median properties of all these RG layers. There are two exceptions to this rule, one for `height/depth`, and one for layer properties with the ending `_all`, such as `ppu_all`:

- `height` and `depth` provide the vertical grid of the average profile, and for algorithmic reasons, this grid is not always equal to the actual median height or depth of the predominant layers. To account for that, two layer columns exist called `medianPredominantHeight` and `medianPredominantDepth`.
- Properties ending with `_all`: For example, while `ppu` refers to the proportion of profiles, whose *predominant* layers are unstable (i.e., `p_unstable` >= 0.77), `ppu_all` refers to the the proportion of profiles, whose layers are unstable while taking into account *all* individual layers matched to this average layer (i.e., despite grain type, etc).
- Other layer properties specific to the average profile: `distribution` ranges between `[0, 1]` and specifies the proportion of profiles that contain the predominant layer described in the other properties.

Functions

- `averageSP()`: convenient wrapper function
- `dbaSP()`: DTW barycenter averaging of snow profiles (low level worker function)

Author(s)

fherla

References

Herla, F., Haegeli, P., and Mair, P. (2022). A data exploration tool for averaging and accessing large data sets of snow stratigraphy profiles useful for avalanche forecasting, The Cryosphere, 16(8), 3149–3162, https://doi.org/10.5194/tc-16-3149-2022

See Also

averageSPalongSeason

Examples

```
## EXAMPLES OF averageSP
this_example_runs_about_10s <- TRUE
if (!this_example_runs_about_10s) {  # exclude from cran checks

## compute the average profile of the demo object 'SPgroup'
## * by labeling SH/DH layers as weak layers,
##   - choosing 3 initial conditions with an above average number of weak layers
##   - in as many depth ranges as possible
## * and neglecting crusts for initial conditions

  avgList <- averageSP(SPgroup, n = 3,
                       classifyPWLs = list(pwl_gtype = c("SH", "DH")),
                       classifyCRs = NULL)

  opar <- par(mfrow = c(1, 2))
  plot(avgList$avg, ymax = max(summary(avgList$set)$hs))
  plot(avgList$set, SortMethod = "unsorted", xticklabels = "originalIndices")
  par(opar)


## compute the average profile of the demo object 'SPgroup'
## * by labeling SH/DH/FC/FCxr layers with an RTA threshold of 0.65 as weak layers,
## * otherwise as above

  SPx <- computeRTA(SPgroup)
  avgList <- averageSP(SPx, n = 3,
                       classifyPWLs = list(pwl_gtype = c("SH", "DH", "FC", "FCxr"),
                                           threshold_RTA = 0.65),
                       classifyCRs = NULL)

  opar <- par(mfrow = c(1, 2))
  plot(avgList$avg, ymax = max(summary(avgList$set)$hs))
  plot(avgList$set, SortMethod = "unsorted", xticklabels = "originalIndices")
  par(opar)

## compute the average profile of the other demo object 'SPgroup2', which
## contains more stability indices, such as SK38 or p_unstable
```

```
## * by labeling SH/DH/FC/FCxr layers that either
##   - have an SK38 below 0.95, *or*
##   - have a p_unstable above 0.77

  SPx <- snowprofileSet(SPgroup2)
  avgList <- averageSP(SPx,
                       classifyPWLs = list(pwl_gtype = c("SH", "DH", "FC", "FCxr"),
                                           threshold_SK38 = 0.95, threshold_PU = 0.77))

  opar <- par(mfrow = c(1, 2))
  plot(avgList$avg, ymax = max(summary(avgList$set)$hs))
  plot(avgList$set, SortMethod = "unsorted", xticklabels = "originalIndices")
  par(opar)

}



## EXAMPLES OF dbaSP
## either rescale profiles beforehand...
if (FALSE) {  # don't run in package check to save time
  SPx <- reScaleSampleSPx(SPgroup)$set          # rescale profiles
  SPx <- snowprofileSet(lapply(SPx, labelPWL))  # label PWLs
  DBA <- dbaSP(SPx, 5, plotChanges = TRUE)      # average profiles
}

## or use unscaled snow heights:
if (FALSE) {  # don't run in package check to save time
  SPx <- snowprofileSet(lapply(SPgroup, labelPWL))  # label PWLs
  DBA <- dbaSP(SPx, 5, plotChanges = TRUE)          # average profiles
}
```

---

averageSPalongSeason *Compute a seasonal timeseries of an average snowprofile*

Description

This routine computes the seasonal timeseries of the average snow profile for a given region/set of profiles. The total snow height of the seasonal average profile closely follows the *median snow height* represented by the group of profiles each day. Also the new snow amounts represent the *median new snow amounts* within the group (i.e., PP and DF grains). The routine maintains temporal consistency by using the previous day average profile as initial condition to derive the next day's. This creates the need for re-scaling the layer thicknesses each day to account for snow settlement and melting. Two different re-scaling approaches have been implemented, which both aim to re-scale the old snow part of the column (i.e., the snow which was on the ground already at the previous day). See parameter description for more details. Also note, that the routine can be started at any day of the season by providing an average profile from the previous day. The routine modifies several parameters, which are passed on to dtwSP. These parameters differ from the defaults specified in dtwSP, which are held very generic, whereas the application in this function is much more specific to certain requirements and algorithm behavior. For more details, refer to the reference paper.

Usage

```
averageSPalongSeason(
  SPx,
  sm = summary(SPx),
  AvgDayBefore = NULL,
  DateEnd = max(sm$date),
  keep.profiles = TRUE,
  progressbar = requireNamespace("progress", quietly = TRUE),
  dailyRescaling = c("settleTopOldSnow", "settleEntireOldSnow")[1],
  proportionPWL = 0.3,
  breakAtSim = 0.9,
  breakAfter = 2,
  verbose = FALSE,
  resamplingRate = 0.5,
  top.down = FALSE,
  checkGlobalAlignment = FALSE,
  prefLayerWeights = NA,
  dims = c("gtype", "hardness", "ddate"),
  weights = c(0.375, 0.125, 0.5),
  ...
)
```

Arguments

`SPx`	a snowprofileSet that contains all profiles from the region to be averaged at all days of the season for which you want to compute the average profile. Identically to dbaSP, weak layers need to be labeled prior to this function call, see dbaSP and sarp.snowprofile::labelPWL. Note that only daily sampling is allowed at this point (i.e., one profile per grid point per day).
`sm`	a summary of `SPx` containing meta-data
`AvgDayBefore`	an average snowprofile from the previous day. This is only necessary if you want to resume the computation mid season.
`DateEnd`	an end date character string (`"YYYY-MM-DD"`) if you only want to compute the timeseries up to a certain point in time. Defaults to the future-most date contained in the meta-data object `sm`.
`keep.profiles`	Do you want to keep the (resampled) individual snow profiles from `SPx` in your return object? **Note** that this must be `TRUE` if you plan to backtrackLayers to derive any kind of summary statistics for the averaged layers. See Notes below, and examples of how to conveniently backtrackLayers.
`progressbar`	display a progress bar during computation?
`dailyRescaling`	choose between two settlement rescaling approaches. `settleEntireOldSnow` re-scales the entire old snow column so that the average snow height represents the median snow height from the profile set. `settleTopOldSnow` (the default)

re-scales the upper part of the old snow column to achieve the same goal. While the former mostly leads to buried layers being settled to too deep snow depths, the default approach aims to leave those buried layers unchanged, which are located at depths that represent the median depths of their aligned layers.

`proportionPWL` decimal number that specifies the proportion required to average an ensemble of grain types as weak layer type. A value of 0.3, for example, means that layers will get averaged to a PWL type if 30% of the layers are of PWL type. Meaningful range is between `[0.1, 0.5]`. Values larger than 0.5 get set to 0.5.

`breakAtSim` stop iterations when simSP between the last average profiles is beyond that value. Can range between `[0, 1]`. Default values differ between dbaSP and averageSP.

`breakAfter` integer specifying how many values of simSP need to be above `breakAtSim` to stop iterating. Default values differ between dbaSP and averageSP.

`verbose` print similarities between old and new average in between iterations?

`resamplingRate` Resampling rate for a regular depth grid among the profiles

`top.down` a dtwSP parameter, which needs to be set to `FALSE` to ensure correct growing of the snowpack during snowfall.

`checkGlobalAlignment`
a dtwSP parameter, which needs to be set to `FALSE` analogous to `top.down`

`prefLayerWeights`
a dtwSP parameter. Might be best to set this to `NA`, but can potentially be set to `layerWeightingMat(FALSE)` *in case of* averaging a very large geographic region with temporal lags between weather events.

`dims` a dtwSP parameter, which is modified to include deposition date alignments per default

`weights` a dtwSP parameter that sets the according weights to the `dims` specified above.

`...` any other parameters passed on to dbaSP and then dtwSP.

Details

Computing the seasonal average profile for an entire season and about 100 grid points (with a max of 150 cm snow depth) takes roughly 60 mins.

Value

A list of class `avgSP_timeseries` containing the fields `$avgs` with a snowprofileSet of the average profiles at each day. If `keep.profiles == TRUE` a field `$sets` with the according profiles informing the average profile at each day (which can be used to backtrackLayers to compute summary statistics of the averaged layers). And two fields `$call` and `$meta`. The latter contains several useful meta-information such as `...$date`, `...$hs`, `...$hs_median`, `...$thicknessPPDF_median`, or `...$rmse`, which gauges the representativity of the average profile (the closer to `0`, the better; the closer to `1`, the worse).

Note

- If you don't provide an AvgDayBefore, it will be computed with averageSP and *default* parameters (dots won't be passed to initializing the first average profile)!
- Even though backtrackLayers allows for backtracking layers based on height, it is not recommended to try and backtrack layers if `keep.profiles = FALSE`, since profiles that can't be aligned to the average profile (`$avgs[[i]]`) are being discarded from the profile set at that day (`$sets[[i]]`), which changes `queryIDs` in the backtrackingTable. Conclusion: If you want to backtrack layers from the seasonal average profile, you *must* `keep.profiles = TRUE`. See examples!

Author(s)

fherla

References

Herla, F., Haegeli, P., and Mair, P. (2022). A data exploration tool for averaging and accessing large data sets of snow stratigraphy profiles useful for avalanche forecasting, The Cryosphere, 16(8), 3149–3162, https://doi.org/10.5194/tc-16-3149-2022

See Also

dbaSP, averageSP, sarp.snowprofile::labelPWL

Examples

```
run_the_examples <- FALSE  # exclude long-running examples
if (run_the_examples) {

## compute average timeseries for simplistic example data set 'SPspacetime'
## first: label weak layers (you can choose your own rules and thresholds!)
SPspacetime <- snowprofileSet(lapply(SPspacetime, function(sp) {
 labelPWL(sp, pwl_gtype = c("SH", "DH", "FC", "FCxr"), threshold_RTA = 0.8)
}))  # label weak layers in each profile of the profile set 'SPspacetime'

## second: average along several days
avgTS <- averageSPalongSeason(SPspacetime)

## explore resulting object
names(avgTS)

# timeseries figure
plot(avgTS$avgs, main = "average time series")
# add line representing median snow height
lines(avgTS$meta$date, avgTS$meta$hs_median)
# add line representing median new snow amounts
lines(avgTS$meta$date, avgTS$meta$hs - avgTS$meta$thicknessPPDF_median, lty = 'dashed')

# individual profile sets from one day
plot(avgTS$sets[[1]], SortMethod = "hs", main = "individual profiles from first day")
```

```
## backtrack individual layers of the average profile...
individualLayers <- backtrackLayers(avgProfile = avgTS$avgs[[1]],
                      profileSet = avgTS$sets[[1]],
                      layer = findPWL(avgTS$avgs[[1]], pwl_gtype = c("SH", "DH"),
                                      pwl_date = "2018-10-17", threshold_RTA = 0.8))
## ... to retrieve summary statistics or distributions, e.g. stability distribution
hist(individualLayers[[1]]$rta)
hist(individualLayers[[1]]$depth)

## see the Vignette about averaging profiles for more examples!


}
```

---

backtrackLayers *Backtrack layers from average or summary profile*

Description

An average profile as computed by dbaSP summarizes the prevalent layer properties of the entire profile set. To better understand the distribution of layer properties within the set, use this function to retrieve layers of interest from the individual profiles of the original profile set.

Usage

```
backtrackLayers(
  avgProfile,
  layer = NA,
  profileSet = NULL,
  layer_units = "row#",
  condition = NULL,
  computationByHeight = FALSE
)
```

Arguments

`avgProfile`	an average profile as per dbaSP
`layer`	the height or row number of the layer to retrieve the distribution for (given as height or row number of the average profile). If layer is `NA`, all layers from the avgProfile are considered.
`profileSet`	the profile set that is averaged by `avgProfile`. Optimally, it is the resampled profile set as returned by dbaSP or averageSP, see parameter `computationByHeight` if that resampled profile set is not available anymore.
`layer_units`	either `"row#"` or `"cm"`

`condition` a condition that subsets which layers are returned. E.g., only layers with a specific grain type, etc.. Note that the condition needs to be substituted in the function call, e.g. `condition = substitute(gtype == "SH")`. In most cases, it's best to subset the data.frame manually after this function has been called. A *secret* and *dangerous* trick is to use `condition = substitute(gtype %in% return_conceptually_similar_gtypes(as.character(avgProfile$layers$gtype[lidx])))` to get the very same layers that have been used to compute the median layer properties which are included in the avgProfile$layers.

`computationByHeight`
There are two ways of how to backtrack layers that were aligned to `avgProfile$layers`. The first and safest approach is by index, which requires the resampled `profileSet` as returned by dbaSP or averageSP. The second approach is by layer height, which should yield the same results (beta phase: still bugs possible, check yourself!) and allows to backtrack the layers even if the resampled profileSet is not available anymore, but only the original unmodified set which was used to create the average profile.

Value

This function returns a list of data.frames with the backtracked layers. Each (named) list item corresponds to a specific layer height (cm).

Author(s)

fherla

Examples

```
## See Vignette for examples.
```

`chooseICavg` *Get index of appropriate initial condition average profile*

Description

To average a set of snow profiles, dbaSP requires a snow profile as initial condition (IC) to start the algorithm. To prevent persistent weak layers (PWLs) and crusts from being averaged-out during the call to `dbaSP`, it is advised to start the algorithm with a best-guess IC. This best guess IC contains a large number of PWLs and crusts to ensure that the most prevalent ones actually make their way into the final average profile. This function helps to choose meaningful IC profiles. See Details or (better) the source code for how this function picks the profiles.

Usage

```
chooseICavg(
  set,
  n,
  classifyPWLs,
  classifyCRs,
  nPWL = round((2 * n/3) + 0.001),
  sm = summary(set)
)
```

Arguments

`set`	a snowprofileSet
`n`	number of profile indices to be picked (i.e., returned)
`classifyPWLs`	an argument list for a function call to sarp.snowprofile::findPWL which returns relevant PWLs for identifying initial conditions
`classifyCRs`	an argument list for a function call to sarp.snowprofile::findPWL which returns relevant CR(ust)s for identifying initial conditions
`nPWL`	number of profile indices to be picked from profiles that have many PWLs in many different vertical levels; an analogous `nCR` will be the difference `n - nPWL`.
`sm`	a (precomputed) summary of the `set`

Details

This function first computes how many PWLs and how many crusts are in the profiles that have a close to median total snow height HS. Each of these profile is then divided into several vertical levels (by numberOfPWLsPerVerticalLevel). nPWL and nCR profiles are then randomly picked from the profiles that have PWLs or CR in most vertical levels and additionally have a rather large number of PWLs/CR overall. The larger `n`, the more profiles with decreasing number of PWLs/CR in different levels are also returned. Note that this function is best applied to large profile sets to obtain semi-random results. For small sets, the indices returned can actually be deterministic since the pool of relevant profiles is too small.

Value

`n` number of indices that correspond to profiles in the `set`

Author(s)

fherla

See Also

sarp.snowprofile::findPWL, averageSP

Examples

```
plot(SPgroup, SortMethod = "unsorted", TopDown = TRUE,
     xticklabels = "originalIndices", main = "entire profile set")
IC_ids_pwl <- chooseICavg(SPgroup, n = 4, nPWL = 4,
                          classifyPWLs = list(pwl_gtype = c("SH", "DH")),
                          classifyCRs = NULL)
plot(SPgroup[IC_ids_pwl], SortMethod = "unsorted", hardnessResidual = 0, TopDown = TRUE,
   xticklabels = IC_ids_pwl, main = "sample of profiles with rather many and distributed PWLs")
```

---

concat_avgSP_timeseries

Concatenate time series of average profiles

Description

This is useful in operations to update a time series that was computed in the past with a newly computed average time series. The routine merges all entries with duplicated entries (read dates) being taken from avgSP2.

Usage

```
concat_avgSP_timeseries(avgSP1, avgSP2)
```

Arguments

avgSP1	old time series of average profiles as returned by averageSPalongSeason
avgSP2	new time series of average profiles as returned by averageSPalongSeason

Author(s)

fherla

See Also

averageSPalongSeason

`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)
```

`densityDistance` *Difference in layer density*

Description

Calculate the difference (i.e. distance) in layer density

Usage

```
densityDistance(density1, density2, normalize = FALSE, absDist = TRUE)
```

Arguments

`density1`	numeric density values (1D array)
`density2`	numeric density values (1D array)
`normalize`	Should result be normalized? boolean, default False.
`absDist`	Interested in absolute distance? default True.

Value

numeric density distance

Author(s)

pbillecocq

`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,
  top.down.mirroring = FALSE,
  warn.if.na.in.distance.calc = FALSE,
  ...
)
```

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`, `density`, `ogs`.
- `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 (Other compatible window functions can be found in dtw::dtwWindowingFunctions.)
- `top.down.mirroring` Will the resulting distance matrix be used for top down alignments? i.e., do you want to mirror the matrix about its anti-diagonal (top-left/bottom-right diagonal)?
- `warn.if.na.in.distance.calc` most dependent functions in this package should be able to deal with NA values encountered in distance calculations. Set this argument to `TRUE` if you want to be warned anyways.
- `...` arguments to the window function, e.g. `window.size`, `window.size.abs`, `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 on grain type, and without preferential layer matching")

## enable preferential layer matching:
dMat <- distMatSP(SPpairs$A_modeled, SPpairs$A_manual, windowFunction = NA)
graphics::image(dMat,
                main = "... with preferential layer matching")


## using a warping window:
dMat <- distMatSP(SPpairs$A_modeled, SPpairs$A_manual, window.size.abs = 50)
graphics::image(dMat, main = "... and superimposing an absolute warping window of 50 cm")
```

---

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 = "auto",
  keep.internals = TRUE,
  step.pattern = symmetricP1,
  resamplingRate = 0.5,
  rescale2refHS = FALSE,
```

```
  bottom.up = TRUE,
  top.down = TRUE,
  nonMatchedSim = 0,
  nonMatchedThickness = 10,
  simType = "HerlaEtAl2021",
  apply_scalingFactor = FALSE,
  ...
)
```

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? Recommended if profiles will not be rescaled.

`checkGlobalAlignment`
: 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? `'auto'` sets to `TRUE` if `simType == "HerlaEtAl2021"` and to `FALSE` otherwise.

`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` Scalar, numeric resampling rate for a regular depth grid. If the profiles have been rescaled prior to calling this routine, set to `NA`. To resample onto the smallest possible mutual (original, likely irregular) depth grid (see Details, bullet point 2.2), set to `'irregularInterfaces'`.

`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?

`nonMatchedSim` Similarity value `[0, 1]` for non-matched layers, see simSP. indifference = 0.5, penalty < 0.5

`nonMatchedThickness`
: How strongly should the thicknesses of non-matched layers influence the resulting similarity of the profiles? The smaller this (positive!) value, the more influence; and vice versa. See simSP for more details.

`simType` the similarity between two profiles can be computed with different approaches, see simSP

`apply_scalingFactor`
: Setting for simSP in case `simType == "layerwise`.

`...` 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`, `window.size.abs`, `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 (which can all be managed from this function):

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).
   - irregular grid that includes all layer interfaces within the two profiles (i.e., set `resamplingRate = 'irregularInterfaces'`) (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 or by the DTW distance will be returned.
7. (optional) Compute a **similarity score** for the two profiles with simSP

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

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.

The package used to require re-scaling of the profiles to identical snow heights. This requirement has been removed in v1.1.0. Profiles therefore can be resampled onto a regular grid, whilst keeping their original total snow heights. The alignment can then be carried out bottom.up or top.down with a relative or absolute window size. If the profiles have different snow heights and a relative window size is provided, the window size is computed using the larger snow height of the two profiles (e.g., Profile A HS 100 cm, Profile B HS 80 cm; window.size = 0.3 translates to an effective window size of +/- 33 cm). See examples for alignments without prior re-scaling.

Author(s)

fherla

References

Herla, F., Horton, S., Mair, P., & Haegeli, P. (2021). Snow profile alignment and similarity assessment for aggregating, clustering, and evaluating of snowpack model output for avalanche forecasting. Geoscientific Model Development, 14(1), 239–258. https://doi.org/10.5194/gmd-14-239-2021

See Also

plotSPalignment, simSP

Examples

```
## Align a modeled and a manual snow profile, primarily based on default settings:
dtwAlignment <- dtwSP(SPpairs$A_modeled, SPpairs$A_manual, open.end = FALSE)

## check out the resulting dtwSP alignment object:
summary(dtwAlignment)
plotSPalignment(dtwAlignment = dtwAlignment)
plotCostDensitySP(dtwAlignment)


## Align profiles from subsequent days without re-scaling them:
dtwAlignment <- dtwSP(SPpairs$C_day3, SPpairs$C_day1, resamplingRate = 0.5, rescale2refHS = FALSE,
                      window.size.abs = 30)
## Note, per default both bottom.up and top.down alignments have been considered,
#  let's check out which one was suited better:
dtwAlignment$direction  # i.e., bottom up
## Check it out visually:
plotSPalignment(dtwAlignment = dtwAlignment,
                mainQu = "3 Days after...", mainRef = "...the reference profile.")
plotCostDensitySP(dtwAlignment, labelHeight = TRUE)
```

---

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)
```

match_with_tolerance *Match with numeric tolerance*

Description

Match with numeric tolerance

Usage

```
match_with_tolerance(x, y, d = 2)
```

Arguments

x	numeric vector
y	numeric vector
d	numeric tolerance in form of digits

Value

boolean vector equivalently to match

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,
  progressbar = requireNamespace("progress", quietly = TRUE),
  ...
)
```

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`
`progressbar`	Do you want to print a progress bar with recommended package "progress"?
`...`	arguments passed to distanceSP and then further to dtwSP

Details

Note that the pairwise distance matrix is modified within the function call to represent a symmetric distance matrix. That is,, however, not originally the case, since `dtwSP(A, B) != dtwSP(B, A)`. The matrix is therefore made symmetric by setting the similarity between the profiles A and B to `max({dtwSP(A, B), dtwSP(B, A)})`.

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', xticklabels = "originalIndices")

  ## 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 auto-calculated. 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)
```

ogsDistance *Difference in layer ogs*

Description

Calculate the difference (i.e. distance) in layer ogs

Usage

```
ogsDistance(ogs1, ogs2, normalize = FALSE, absDist = TRUE)
```

Arguments

`ogs1`	numeric ogs values (1D array)
`ogs2`	numeric ogs values (1D array)
`normalize`	Should result be normalized? boolean, default False.
`absDist`	Interested in absolute distance? default True.

Value

numeric ogs distance

Author(s)

pbillecocq

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,
  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,
  failureLayers_qu = FALSE,
  failureLayers_qu_col = "red",
  ...
)
```

Arguments

query	The query snowprofile to be warped
ref	The reference snowprofile to be warped against
dtwAlignment	dtwSP object (optional)
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

`failureLayers_qu`
draw arrow to failure layers (see plot.snowprofile)? provide height vector.

`failureLayers_qu_col`
color of arrow(s) (individual color string or vector, see plot.snowprofile)

`...` Arguments passed to `distMatSP` and `dtwSP`

Value

dtw object with the resampled '$query' and '$reference', as well as the warped query '$query-Warped' (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 snow height to that max snow height!

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$A_manual, main = "Initial profiles before resampling",
     ylab = "Snow height", ymax = 272)
plot(SPpairs$A_modeled, ylab = "Snow height", ymax = 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$query, main = "Profiles after resampling",
     ylab = "Snow height", ymax = 272)
plot(resampledSPlist$ref, ylab = "Snow height", ymax = 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')
```

return_conceptually_similar_gtypes *Return conceptually similar grain types*

Description

Note, use this function with care. It's a brief helper function for specific usage, not generally applicable! It is, however, sometimes useful for backtracking layers, see backtrackLayers.

Usage

```
return_conceptually_similar_gtypes(gt)
```

Arguments

gt	a single gtype

Value

a character vector of similar gtypes

Examples

```
return_conceptually_similar_gtypes("SH")
return_conceptually_similar_gtypes("MFcr")
return_conceptually_similar_gtypes("RG")
```

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

This function calculates a similarity measure 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 measure is compatible with top-down alignments and is symmetric with respect to its inputs, i.e. `simSP(P1, P2) == simSP(P2, P1)`. **Several different approaches of computing the measure have been implemented by now, see Details below.**

Usage

```
simSP(
  ref,
  qw,
  gtype_distMat = sim2dist(grainSimilarity_evaluate(triag = FALSE)),
  type = "HerlaEtAl2021",
  nonMatchedSim = 0,
  nonMatchedThickness = 10,
  verbose = FALSE,
  returnDF = FALSE,
  apply_scalingFactor = 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.)

`type` the similarity measure can be computed in several different ways (of sophistication). See Details section. Possible choices

- `simple`
- `HerlaEtAl2021` (= `simple2`)
- `tsa_WLdetection` & `rta_WLdetection`
- `layerwise` & `rta_scaling`
- `remotesensing`

`nonMatchedSim` sets the similarity value of non-matched layers `[0, 1]`. "indifference" = 0.5, penalty < 0.5. Note that dtwSP sets the same value and overrides the default value in this function!

`nonMatchedThickness` If `NA`, every unique non-matched layer (i.e., contiguous resampled layers with identical properties) contributes to the overall similarity by 1 x `nonMatchedSim`. In that case, 5cm of non-matched new snow has the same effect on the overall similarity as 50cm of non-matched new snow. To make the effect of non-matched layers dependent on the layer thickness, provide a positive number to `nonMatchedThickness`. For `nonMatchedThickness = 10`, every 10cm of a unique non-matched layer contribute to the overall similarity by 1 x `nonMatchedSim`. So, 50cm of non-matched new snow would contribute 5 times stronger than 5cm of non-matched new snow. Note that dtwSP sets the same value and overrides the default value in this function!

`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`

`apply_scalingFactor` Only applicable to `type = layerwise`: `TRUE` or `FALSE`, see Details.

Details

The first several implementation types (**simple**, **HerlaEtAl2021**, **tsa_WLdetection**, **rta_WLdetection**) represent different flavors of the approach detailed in Herla et al (2021). In essence, they are a simple approach to incorporate avalanche hazard relevant characteristics into the score by computing the score as arithmetic mean of 4 different grain type classes:

- 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.)

Additionally, for classes wl and cr, vertical windows are computed to weigh layers more heavily that have no other wl or cr grain types in their neighborhood.

Type **simple** deviates from *simple2* (= *HerlaEtAl2021*) by computing the aforementioned vertical windows based on heuristic depth ranges (i.e., Surface–30cm depth–80cm depth–150cm depth–Ground). It is otherwise identical to the **simple2** type, which computes as many numbers of equidistant vertical windows as number of wl or cr are present in the profile.

Type **tsa_WLdetection** employs a similar approach as *simple*, but it identifies weak layers (wl) based on the Threshold Sum Approach (>= 5 TSA, lemons, German 'Nieten'). Therefore, the original profiles need to contain grain size information, which allows you to pre-compute the lemons for all layers (additionally to the otherwise necessary gain type and hardness information). It is thus more targeted to simulated profiles or detailed manual profiles of very high quality. While the former two types neglect hardness information of wl and cr classes, this type does not. Type **rta_WLdetection** works analogous, but uses RTA instead of TSA and a threshold of >= 0.8.

Unlike the former types, **layerwise** applies no weighting at all if used as per default. That means that the similarity of each individual layer contributes equally to the overall similarity measure. It is, however, very flexible in that any custom scaling factor can be applied to each layer. The resulting similarity score is then computed by

- simSP = sum(sim * scalingFactor) / sum(scalingFactor),

where the denominator ensures that the resulting score will be within `[0, 1]`. If you want to explore your own scaling approach, both input snow profiles need to contain a column called `$layers$scalingFactor` that store the desired factor. Type **rta_scaling** is a special case of `layerwise`, where the scaling is determined by the relative lemons of each layer (RTA, see Monti et al 2013). Type **remotesensing** makes use of the layerwise algorithm, but triggers an alternative similarity computation beforehand. Similarity is first computed from density and Optical Grain Size (ogs), and then the layerwise similarity is called upon to compute the global sim score.

NOTE that for all types that include TSA/RTA values, these values need to be computed *prior to aligning* the profiles (and therefore need to be present in the profiles provided to this function!)

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`.

References

Herla, F., Horton, S., Mair, P., & Haegeli, P. (2021). Snow profile alignment and similarity assessment for aggregating, clustering, and evaluating of snowpack model output for avalanche forecasting. Geoscientific Model Development, 14(1), 239–258. https://doi.org/10.5194/gmd-14-239-2021

Monti, F., & Schweizer, J. (2013). A relative difference approach to detect potential weak layers within a snow profile. Proceedings of the 2013 International Snow Science Workshop, Grenoble, France, 339–343. Retrieved from https://arc.lib.montana.edu/snow-science/item.php?id=1861

Examples

```
## first align two profiles, then assess the similarity of the aligned profiles
alignment <- dtwSP(SPpairs$A_modeled, SPpairs$A_manual)
SIM <- simSP(alignment$queryWarped, alignment$reference, verbose = TRUE)

## similarity of identical profiles
SIM <- simSP(alignment$queryWarped, alignment$queryWarped, verbose = TRUE)

## non-matched layers become apparent here:
alignment <- plotSPalignment(SPpairs$C_day1, SPpairs$C_day2, keep.alignment = TRUE,
                             rescale2refHS = FALSE, checkGlobalAlignment = FALSE)
simSP(alignment$queryWarped, alignment$reference, nonMatchedSim = 0.5)
## smaller similarity score due to 'penalty' of non-matched layers:
simSP(alignment$queryWarped, alignment$reference, nonMatchedSim = 0)
## even smaller similarity score due to higher impact of non-matched layer thickness:
simSP(alignment$queryWarped, alignment$reference, nonMatchedSim = 0, nonMatchedThickness = 1)

## detect WL based on lemons (instead of grain type alone):
P1 <- computeTSA(SPpairs$D_generalAlignment1)
P2 <- computeTSA(SPpairs$D_generalAlignment2)
alignment <- dtwSP(P1, P2, simType = "tsa_wldetection")
# sim based on WL-detection with TSA:
simSP(alignment$queryWarped, alignment$reference, type = "tsa_wldetection", verbose = TRUE)
# sim solely based on grain type, neglecting TSA information
simSP(alignment$queryWarped, alignment$reference, type = "simple", verbose = TRUE)

## RTA scaling type
P1 <- computeRTA(P1)
P2 <- computeRTA(P2)
alignment <- dtwSP(P1, P2, simType = "rta_scaling")
# sim based on scaling with RTA
simSP(alignment$queryWarped, alignment$reference, type = "rta_scaling")
# sim based on WL-detection with RTA
simSP(alignment$queryWarped, alignment$reference, type = "rta_wldetection")
# sim based on WL-detection with TSA
simSP(alignment$queryWarped, alignment$reference, type = "tsa_wldetection")

## layerwise similarity (i) unscaled...
simSP(alignment$queryWarped, alignment$reference, type = "layerwise", verbose = TRUE)

##... or (ii) with custom scaling factor (example only illustrative)
```

```
alignment$queryWarped$layers$scalingFactor <- 0.1
alignment$queryWarped$layers$scalingFactor[findPWL(alignment$queryWarped)] <- 1
alignment$reference$layers$scalingFactor <- 0.1
alignment$reference$layers$scalingFactor[findPWL(alignment$reference)] <- 1
simSP(alignment$queryWarped, alignment$reference, type = "layerwise",
      apply_scalingFactor = TRUE, verbose = TRUE)
```

SPgroup2 *Additional example set of snow profiles*

Description

Additional example set of snow profiles. The main difference to the example data set SPgroup is that SPgroup2 contains various different stability indices.

Usage

```
SPgroup2
```

Format

A snowprofileSet

See Also

SPgroup

Examples

```
plot(SPgroup2, SortMethod = "unsorted")
```

SPspacetime *Additional example set of snow profiles*

Description

Additional example set of 4 spatially distributed snow profiles for 5 consecutive days, also containing different stability indices.

Usage

```
SPspacetime
```

Format

A snowprofileSet

See Also

SPgroup2

Examples

```
plot(SPspacetime, SortMethod = "elev")
```

---

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" (= "jminTop-Down"), "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 ("jminTop-Down")
- partial top down alignment where entire ref is matched to subsequence of query ("iminTop-Down")

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

For developers: Including new variables in the output of warped profiles can easily be done by inserting a respective command at the end of this function. There are many example variables added already.

Value

Returns the input alignment object including the element alignment$queryWarped (or $reference-Warped), 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 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,
  window.size.abs = NA,
  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 (i.e., POSIXct data converted to numeric through matrix call!)
jddate	same as jheight, but containing deposition date information (i.e., POSIXct data converted to numeric through matrix call!)
profile.size	number of layers in the longer one of the two profiles (scalar)
profile.height	snow height of the deeper one of the two 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)

`window.size.abs`
: Instead of a `window.size` percentage, an absolute value (in *cm*!) can be provided

`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

See Also

dtw::dtwWindowingFunctions

Index