Package 'distantia'

October 29, 2019

Type Package

Title Assessing Dissimilarity Between Multivariate Time Series

Version 1.0.2

Author Blas M. Benito

URL https://blasbenito.github.io/distantia/

Maintainer Blas M. Benito <blasbenito@gmail.com>

Description Provides tools to assess the dissimilarity between multivariate time-series. It is based on the psi measure described by Birks and Gordon (1985 <doi:10.1002/jqs.3390020110>), which computes dissimilarity between irregular time-series constrained by sample order. However, in this package the original idea has been extended to work with any kind of multivariate time-series, no matter whether they are regular, irregular, aligned or unaligned. Furthermore, the package allows to assess the significance of dissimilarity values by applying a restricted permutation test, allows to measure the contribution of individual variables to dissimilarity, and offers tools to transfer attributes (generally time or age, but other are possible) between sequences based on the similarity of their samples.

License GPL (>= 2)

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

Depends R (>= 3.5.0)

Imports plyr, grDevices, foreach, parallel, doParallel, fields, viridis, RColorBrewer, data.table, iterators, arrangements

Suggests devtools, formatR, kableExtra, magrittr, knitr, rmarkdown

NeedsCompilation no

Repository CRAN

Date/Publication 2019-10-28 23:40:11 UTC

R topics documented:

`distantia-package` *distantia: Assessing Dissimilarity Between Multivariate Time Series*

Description

Provides tools to assess the dissimilarity between multivariate time-series. It is based on the psi measure described by Birks and Gordon (1985 <doi:10.1002/jqs.3390020110>), which computes dissimilarity between irregular time-series constrained by sample order. However, in this package the original idea has been extended to work with any kind of multivariate time-series, no matter whether they are regular, irregular, aligned or unaligned. Furthermore, the package allows to assess the significance of dissimilarity values by applying a restricted permutation test, allows to measure the contribution of individual variables to dissimilarity, and offers tools to transfer attributes (generally time or age, but other are possible) between sequences based on the similarity of their samples.

Details

Details

See Also

Useful links:

- https://blasbenito.github.io/distantia/

`autoSum` *Computes sum of distances between consecutive samples in a multivariate time-series.*

Description

Computes the sum of distances between consecutive samples in a multivariate time-series. Required to compute the measure of dissimilarity `psi` (Birks and Gordon 1985). Distances can be computed through the methods "manhattan", "euclidean", "chi", and "hellinger", and are implemented in the function `distance`.

Usage

```
autoSum(
  sequences = NULL,
  least.cost.path = NULL,
  time.column = NULL,
  grouping.column = NULL,
  exclude.columns = NULL,
  method = "manhattan",
  parallel.execution = TRUE
  )
```

Arguments

`sequences`	dataframe with one or several multivariate time-series identified by a grouping column.
`least.cost.path`	a list usually resulting from either `leastCostPath` or `leastCostPathNoBlocks`.
`time.column`	character string, name of the column with time/depth/rank data. The data in this column is not modified.
`grouping.column`	character string, name of the column in `sequences` to be used to identify separates sequences within the file. This argument is ignored if `sequence.A` and `sequence.B` are provided.
`exclude.columns`	character string or character vector with column names in `sequences`, or `squence.A` and `sequence.B` to be excluded from the analysis.

`method` character string naming a distance metric. Valid entries are: "manhattan", "euclidean", "chi", and "hellinger". Invalid entries will throw an error.

`parallel.execution` boolean, if `TRUE` (default), execution is parallelized, and serialized if `FALSE`.

Details

Distances are computed as:

- `manhattan`: `d <-sum(abs(x -y))`
- `euclidean`: `d <-sqrt(sum((x -y)^2))`
- `chi`: `xy <-x + y y. <-y / sum(y) x. <-x / sum(x) d <-sqrt(sum(((x. -y.)^2) / (xy / sum(xy))))`
- `hellinger`: `d <-sqrt(1/2 * sum(sqrt(x) -sqrt(y))^2)`

Note that zeroes are replaced by 0.00001 whem `method` equals "chi" or "hellinger".

Value

A list with slots named according `grouping.column` if there are several sequences in `sequences` or a number if there is only one sequence.

Author(s)

Blas Benito <blasbenito@gmail.com>

- Birks, H.J.B. and Gordon, A.D. (1985) Numerical Methods in Quaternary Pollen Analysis. Academic Press.

See Also

`distance`

Examples

```
#loading data
data(sequenceA)
data(sequenceB)

#preparing datasets
AB.sequences <- prepareSequences(
 sequence.A = sequenceA,
 sequence.A.name = "A",
 sequence.B = sequenceB,
 sequence.B.name = "B",
 merge.mode = "complete",
 if.empty.cases = "zero",
 transformation = "hellinger"
 )
```

```
#computing distance matrix
AB.distance.matrix <- distanceMatrix(
 sequences = AB.sequences,
 grouping.column = "id",
 method = "manhattan",
 parallel.execution = FALSE
 )

#computing least cost matrix
AB.least.cost.matrix <- leastCostMatrix(
 distance.matrix = AB.distance.matrix,
 diagonal = FALSE,
 parallel.execution = FALSE
 )

AB.least.cost.path <- leastCostPath(
 distance.matrix = AB.distance.matrix,
 least.cost.matrix = AB.least.cost.matrix,
 parallel.execution = FALSE
 )

#autosum
AB.autosum <- autoSum(
 sequences = AB.sequences,
 least.cost.path = AB.least.cost.path,
 grouping.column = "id",
 parallel.execution = FALSE
 )
AB.autosum
```

`climate` *Dataframe with palaeoclimatic data.*

Description

A dataframe containing palaeoclimate data at 1 ky temporal resolution with the following columns:

Usage

```
data(climate)
```

Format

dataframe with 6 columns and 800 rows.

Details

- *time* in kiloyears before present (ky BP).
- *sequenceId* numeric identifier of sequences of 200ky within the main sequence, useful to test some functions of the package, such as `distancePairedSamples`
- *temperatureAverage* average annual temperature in Celsius degrees.
- *rainfallAverage* average annual precipitation in milimetres per day (mm/day).
- *temperatureWarmestMonth* average temperature of the warmest month, in Celsius degrees.
- *temperatureColdestMonth* average temperature of the coldest month, in Celsius degrees.

Author(s)

Blas M. Benito <blasbenito@gmail.com>

`climateLong` *Dataframe with palaeoclimatic data.*

Description

A dataframe containing 800 simulated samples of palaeoclimate data at 1 ky temporal resolution with the following columns:

Usage

```
data(climateLong)
```

Format

dataframe with 6 columns and 800 rows.

Details

- *age* in kiloyears before present (ky BP).
- *temperatureAverage* average annual temperature in Celsius degrees.
- *rainfallAverage* average annual precipitation in milimetres per day (mm/day).
- *temperatureWarmestMonth* average temperature of the warmest month, in Celsius degrees.
- *temperatureColdestMonth* average temperature of the coldest month, in Celsius degrees.
- *oxigenIsotope* delta O18, global ratio of stable isotopes in the sea floor, see http://lorraine-lisiecki.com/stack.html for further details.

climateShort *Dataframe with palaeoclimatic data.*

Description

A dataframe containing 11 simulated samples of palaeoclimate data at 1 ky temporal resolution with the following columns:

Usage

```
data(climateShort)
```

Format

dataframe with 5 columns and 11 rows.

Details

- *temperatureAverage* average annual temperature in Celsius degrees.
- *rainfallAverage* average annual precipitation in milimetres per day (mm/day).
- *temperatureWarmestMonth* average temperature of the warmest month, in Celsius degrees.
- *temperatureColdestMonth* average temperature of the coldest month, in Celsius degrees.
- *oxigenIsotope* delta O18, global ratio of stable isotopes in the sea floor, see http://lorraine-lisiecki.com/stack.html for further details.

distance *Computes a multivariate distance between two vectors.*

Description

Computes a multivariate distance (one of: "manhattan", "euclidean", "chi", and "hellinger") between two vectors of the same length. It is used internally by distanceMatrix and autoSum. This function has no buit-in error trapping procedures in order to speed up execution.

Usage

```
distance(x, y, method = "manhattan")
```

Arguments

x	numeric vector.
y	numeric vector of the same length as x.
method	character string naming a distance metric. Valid entries are: "manhattan", "euclidean", "chi", and "hellinger". Invalid entries will throw an error.

Details

Vectors x and y are not checked to speed-up execution time. Distances are computed as:

- manhattan: d <-sum(abs(x -y))
- euclidean: d <-sqrt(sum((x -y)^2))
- chi: xy <-x + y y. <-y / sum(y) x. <-x / sum(x) d <-sqrt(sum(((x. -y.)^2) / (xy / sum(xy))))
- hellinger: d <-sqrt(1/2 * sum(sqrt(x) -sqrt(y))^2)

Note that zeroes are replaced by 0.00001 whem method equals "chi" or "hellinger".

Value

A number representing the distance between both vectors.

Author(s)

Blas Benito <blasbenito@gmail.com>

Examples

```
x <- runif(100)
y <- runif(100)
distance(x, y, method = "manhattan")
```

`distanceMatrix` *Computes distance matrices among the samples of two or more multivariate time-series.*

Description

Computes distance matrices among the samples of two or more multivariate time-series provided in a single dataframe (generally produced by `prepareSequences`), identified by a grouping column (argument `grouping.column`). Distances can be computed with the methods "manhattan", "euclidean", "chi", and "hellinger", and are implemented in the function `distance`. The function uses the packages `parallel`, `foreach`, and `doParallel` to compute distances matrices among different sequences in parallel. It is configured to use all processors available minus one.

Usage

```
distanceMatrix(
  sequences = NULL,
  grouping.column = NULL,
  time.column = NULL,
  exclude.columns = NULL,
  method = "manhattan",
  parallel.execution = TRUE
  )
```

Arguments

sequences	dataframe with multiple sequences identified by a grouping column. Generally the ouput of prepareSequences.
grouping.column	character string, name of the column in sequences to be used to identify separates sequences within the file. This argument is ignored if sequence.A and sequence.B are provided.
time.column	character string, name of the column with time/depth/rank data. The data in this column is not modified.
exclude.columns	character string or character vector with column names in sequences, or squence.A and sequence.B to be excluded from the analysis.
method	character string naming a distance metric. Valid entries are: "manhattan", "euclidean", "chi", and "hellinger". Invalid entries will throw an error.
parallel.execution	boolean, if TRUE (default), execution is parallelized, and serialized if FALSE.

Details

Distances are computed as:

- manhattan: `d <-sum(abs(x -y))`
- euclidean: `d <-sqrt(sum((x -y)^2))`
- chi: `xy <-x + y y. <-y / sum(y) x. <-x / sum(x) d <-sqrt(sum(((x. -y.)^2) / (xy / sum(xy))))`
- hellinger: `d <-sqrt(1/2 * sum(sqrt(x) -sqrt(y))^2)`

Note that zeroes are replaced by 0.00001 whem method equals "chi" or "hellinger".

Value

A list with named slots containing the the distance matrices of every possible combination of sequences according to grouping.column.

Author(s)

Blas Benito <blasbenito@gmail.com>

See Also

distance

Examples

```
#loading data
data(sequenceA)
data(sequenceB)
```

```
#preparing datasets
AB.sequences <- prepareSequences(
 sequence.A = sequenceA,
 sequence.A.name = "A",
 sequence.B = sequenceB,
 sequence.B.name = "B",
 merge.mode = "complete",
 if.empty.cases = "zero",
 transformation = "hellinger"
 )

#computing distance matrix
AB.distance.matrix <- distanceMatrix(
 sequences = AB.sequences,
 grouping.column = "id",
 method = "manhattan",
 parallel.execution = FALSE
 )


#plot
plotMatrix(distance.matrix = AB.distance.matrix)
```

`distancePairedSamples` *Computes distance among pairs of aligned samples in two or more multivariate time-series.*

Description

Computes the distance (one of: "manhattan", "euclidean", "chi", or "hellinger") between pairs of aligned samples (same order/depth/age) in two or more multivariate time-series.

Usage

```
distancePairedSamples(
  sequences = NULL,
  grouping.column = NULL,
  time.column = NULL,
  exclude.columns = NULL,
  same.time = FALSE,
  method = "manhattan",
  sum.distances = FALSE,
  parallel.execution = TRUE
  )
```

Arguments

`sequences`	dataframe with multiple sequences identified by a grouping column. Generally the ouput of `prepareSequences`.

`grouping.column`
: character string, name of the column in `sequences` to be used to identify separates sequences within the file. This argument is ignored if `sequence.A` and `sequence.B` are provided.

`time.column`
: character string, name of the column with time/depth/rank data. The data in this column is not modified.

`exclude.columns`
: character string or character vector with column names in `sequences`, or `squence.A` and `sequence.B` to be excluded from the analysis.

`same.time`
: boolean. If `TRUE`, samples in the sequences to compare will be tested to check if they have the same time/age/depth according to `time.column`. This argument is only useful when the user needs to compare two sequences taken at different sites but same time frames.

`method`
: character string naming a distance metric. Valid entries are: "manhattan", "euclidean", "chi", and "hellinger". Invalid entries will throw an error.

`sum.distances`
: boolean, if `TRUE` (default option), the distances between samples are summed, and the output of the function (now a list with a single number on each slot) can be directly used as input for the argument `least.cost` in the function `psi`.

`parallel.execution`
: boolean, if `TRUE` (default), execution is parallelized, and serialized if `FALSE`.

Details

Distances are computed as:

- `manhattan`: `d <-sum(abs(x -y))`
- `euclidean`: `d <-sqrt(sum((x -y)^2))`
- `chi`: `xy <-x + y y. <-y / sum(y) x. <-x / sum(x) d <-sqrt(sum(((x. -y.)^2) / (xy / sum(xy))))`
- `hellinger`: `d <-sqrt(1/2 * sum(sqrt(x) -sqrt(y))^2)`

Note that zeroes are replaced by 0.00001 whem `method` equals "chi" or "hellinger".

Value

A list with named slots (names of the sequences separated by a vertical line, as in "A|B") containing numeric vectors with the distance between paired samples of every possible combination of sequences according to `grouping.column`.

Author(s)

Blas Benito <blasbenito@gmail.com>

See Also

`distance`

Examples

```
#loading data
data(climate)

#preparing sequences
#notice the argument paired.samples
climate.prepared <- prepareSequences(
  sequences = climate,
  grouping.column = "sequenceId",
  time.column = "time",
  paired.samples = TRUE
  )

#compute pairwise distances between paired samples
climate.prepared.distances <- distancePairedSamples(
  sequences = climate.prepared,
  grouping.column = "sequenceId",
  time.column = "time",
  exclude.columns = NULL,
  method = "manhattan",
  sum.distances = FALSE,
  parallel.execution = FALSE
  )
```

`formatPsi` *Formats the output of* `psi`.

Description

Parses a list produced by `psi` to generate either a dataframe or a matrix. Can also format a psi matrix into a dataframe and viceversa.

Usage

```
formatPsi(
  psi.values = NULL,
  to = "dataframe")
```

Arguments

`psi.values`	list produced by `psi`.
`to`	character, either "dataframe" or "matrix".

Details

The function detects the type of input, and checks that it is different from the value of `to`. If that is the case, it throws a warning, and returns the input object. It uses the helper function .psiToDataframe, only intended for internal use.

Author(s)

Blas Benito <blasbenito@gmail.com>

`handleNA` *Handles emtpy and NA data in a multivariate time series.*

Description

This function is used internally by `prepareSequences`. Handles empty and NA data in a multivariate time-series in two possible ways: 1) deleting rows with NA or empty cases; 2) replacing NA data with zeros.

Usage

```
handleNA(
  sequence = NULL,
  if.empty.cases = "zero"
  )
```

Arguments

`sequence`	Dataframe, a multivariate time-series.
`if.empty.cases`	character, one of: "omit" (default), "zero". When "omit", the function removes every row with at least one empty/NA record. When "zero", empty/NA data is replaced with zeros.

Value

A dataframe with the same columns as `sequence`.

Author(s)

Blas Benito <blasbenito@gmail.com>

Examples

```
data(sequenceB)
B.sequence <- handleNA(
  sequence = sequenceB,
  if.empty.cases = "zero"
  )
```

leastCost *Extracts the least cost from a least-cost path.*

Description

Sums the the distances of the samples in a least-cost path.

Usage

```
leastCost(
  least.cost.path = NULL,
  parallel.execution = TRUE
  )
```

Arguments

`least.cost.path`
: dataframe produced by `leastCostPath`.

`parallel.execution`
: boolean, if `TRUE` (default), execution is parallelized, and serialized if `FALSE`.

Value

A named list with least-cost values.

A named list with least cost values.

leastCostMatrix *Computes a least cost matrix from a distance matrix.*

Description

Computes the constrained (by sample order) minimum sum of distances among samples in two multivariate time-series by finding the least cost path between the first and last samples in a distance matrix computed by `distanceMatrix`. The minimum distance is found trough an efficient dynamic programming algorithm that first solves the local least cost path between adjacent samples, and uses the partial solutions to find the global solution.

The algorithm is based on the sequence slotting algorithm described by Birks and Gordon (1985). In its original version, the algorithm searches for the least cost path between a given sample of one sequence (A) and the samples of the other sequence (B) in orthogonal directions (either one step in the x axis or one step in the y axis), which allows to locate the two samples in B between which the target sample in A "belongs" (has the least distance to). Therefore, the algorithm is in fact ordering the samples in both sequences to virtually create a single sequence (as in `B1,A1,A2,B2,etc`) with the samples ordered in the way that minimizes the global distance among them.

This function provides an additional option that allows to include the diagonals in the search of the least cost path through the `diagonal` argument (which is `FALSE` by default). This modification

allows to find, for each sample in A, the most similar sample in B, and align them together, if the distance among them is lower than the one found in the orthogonal directions. Both options give highly correlated least cost distances for the same matrices, but have different applications.

Usage

```
leastCostMatrix(
  distance.matrix = NULL,
  diagonal = FALSE,
  parallel.execution = TRUE
  )
```

Arguments

distance.matrix	numeric matrix or list of numeric matrices, a distance matrix produced by distanceMatrix.
diagonal	boolean, if TRUE, diagonals are included in the computation of the least cost path. Defaults to FALSE, as the original algorithm did not include diagonals in the computation of the least cost path.
parallel.execution	boolean, if TRUE (default), execution is parallelized, and serialized if FALSE.

Value

A list of matrices with the same dimensions as distance.matrix with the cumulative least cost among samples. The value of the lower-right cell (in the actual data matrix, not in the plotted version!) represents the sum of the least cost path across all samples.

- Birks, H.J.B. and Gordon, A.D. (1985) Numerical Methods in Quaternary Pollen Analysis. Academic Press.
- Clark, R.M., (1985) A FORTRAN program for constrained sequence-slotting based on minimum combined path length. Computers & Geosciences, Volume 11, Issue 5, Pages 605-617. Doi: https://doi.org/10.1016/0098-3004(85)90089-5.
- Thompson, R., Clark, R.M. (1989) Sequence slotting for stratigraphic correlation between cores: theory and practice. Journal of Paleolimnology, Volume 2, Issue 3, pp 173–184

Examples

```
#loading data
data(sequenceA)
data(sequenceB)

#preparing datasets
AB.sequences <- prepareSequences(
 sequence.A = sequenceA,
 sequence.A.name = "A",
 sequence.B = sequenceB,
```

```
 sequence.B.name = "B",
 merge.mode = "complete",
 if.empty.cases = "zero",
 transformation = "hellinger"
 )

#computing distance matrix
AB.distance.matrix <- distanceMatrix(
 sequences = AB.sequences,
 grouping.column = "id",
 method = "manhattan",
 parallel.execution = FALSE
 )

#computing least cost matrix
AB.least.cost.matrix <- leastCostMatrix(
 distance.matrix = AB.distance.matrix,
 diagonal = FALSE,
 parallel.execution = FALSE
 )

#plot
par(mfrow=c(1,2))
plotMatrix(distance.matrix = AB.distance.matrix)
plotMatrix(distance.matrix = AB.least.cost.matrix)
dev.off()
```

`leastCostPath` *Find the least cost path in a least cost matrix.*

Description

Uses the original distance matrix created by `distanceMatrix` and the least cost path matrix created by `leastCostMatrix` to find the least cost path between the first and the last cells of the matrix. If `diagonal` was `TRUE` in `leastCostMatrix`, then it must be `TRUE` when using this function. Otherwise, the default is `FALSE` in both.

Usage

```
leastCostPath(
  distance.matrix = NULL,
  least.cost.matrix = NULL,
  diagonal = FALSE,
  parallel.execution = TRUE
  )
```

Arguments

`distance.matrix`
: numeric matrix or list of numeric matrices, a distance matrix produced by `distanceMatrix`.

`least.cost.matrix`
: numeric matrix or list of numeric matrices produced by `leastCostMatrix`.

`diagonal`
: boolean, if `TRUE`, diagonals are included in the computation of the least cost path. Defaults to `FALSE`, as the original algorithm did not include diagonals in the computation of the least cost path.

`parallel.execution`
: boolean, if `TRUE` (default), execution is parallelized, and serialized if `FALSE`.

Value

Alist of dataframes if `least.cost.matrix` is a list, or a dataframe if `least.cost.matrix` is a matrix. The dataframe/s have the following columns:

- *A* row/sample of one of the sequences.
- *B* row/sample of one the other sequence.
- *distance* distance between both samples, extracted from `distance.matrix`.
- *cumulative.distance* cumulative distance at the samples `A` and `B`.

Examples

```
#loading data
data(sequenceA)
data(sequenceB)

#preparing datasets
AB.sequences <- prepareSequences(
 sequence.A = sequenceA,
 sequence.A.name = "A",
 sequence.B = sequenceB,
 sequence.B.name = "B",
 merge.mode = "complete",
 if.empty.cases = "zero",
 transformation = "hellinger"
 )

#computing distance matrix
AB.distance.matrix <- distanceMatrix(
 sequences = AB.sequences,
 grouping.column = "id",
 method = "manhattan",
 parallel.execution = FALSE
 )
```

```
#computing least cost matrix
AB.least.cost.matrix <- leastCostMatrix(
 distance.matrix = AB.distance.matrix,
 diagonal = FALSE,
 parallel.execution = FALSE
 )

AB.least.cost.path <- leastCostPath(
 distance.matrix = AB.distance.matrix,
 least.cost.matrix = AB.least.cost.matrix,
 parallel.execution = FALSE
 )

#plot
plotMatrix(distance.matrix = AB.distance.matrix,
 least.cost.path = AB.least.cost.path,
 )
```

`leastCostPathNoBlocks` *Extracts the least-cost from a least cost matrix by trimming blocks.*

Description

Extracts the minimum cost of a least-cost path by trimming blocks (straight segments of the path that appear in highly dissimilar regions of the sequences). Blocks inflate psi values when two sequences are similar but have very different numbers of rows. This function is for internal use of other functions in the package.

Usage

```
leastCostPathNoBlocks(
  least.cost.path = NULL,
  parallel.execution = TRUE
  )
```

Arguments

`least.cost.path`
: dataframe produced by `leastCostPath`.

`parallel.execution`
: boolean, if `TRUE` (default), execution is parallelized, and serialized if `FALSE`.

Value

A named list with least cost values.

Examples

```
#'#loading data
data(sequenceA)
data(sequenceB)

#preparing datasets
AB.sequences <- prepareSequences(
 sequence.A = sequenceA,
 sequence.A.name = "A",
 sequence.B = sequenceB,
 sequence.B.name = "B",
 merge.mode = "complete",
 if.empty.cases = "zero",
 transformation = "hellinger"
 )

#computing distance matrix
AB.distance.matrix <- distanceMatrix(
 sequences = AB.sequences,
 grouping.column = "id",
 method = "manhattan",
 parallel.execution = FALSE
 )

#computing least cost matrix
AB.least.cost.matrix <- leastCostMatrix(
 distance.matrix = AB.distance.matrix,
 diagonal = FALSE,
 parallel.execution = FALSE
 )

AB.least.cost.path <- leastCostPath(
 distance.matrix = AB.distance.matrix,
 least.cost.matrix = AB.least.cost.matrix,
 parallel.execution = FALSE
 )

AB.least.cost.path.nb <- leastCostPathNoBlocks(
 least.cost.path = AB.least.cost.path,
 parallel.execution = FALSE
 )
```

---

plotMatrix *Plots distance matrices and least cost paths.*

Description

Plots the output matrices of distanceMatrix and leastCostMatrix, and superimposes the least cost path generated by leastCostPath. This functions relies on image.plot to plot a color scale along with the matrix plot, or image when a color scale is not needed.

Usage

```
plotMatrix(
  distance.matrix = NULL,
  least.cost.path = NULL,
  plot.columns = NULL,
  plot.rows = NULL,
  legend = TRUE,
  color.palette = "divergent",
  path.color = "black",
  path.width = 1,
  margins = c(2,3,2,4),
  pdf.filename = NULL,
  pdf.width = 7,
  pdf.height = 4,
  pdf.pointsize = 12,
  rotate = FALSE
  )
```

Arguments

distance.matrix	numeric matrix or list of numeric matrices either produced by distanceMatrix or leastCostMatrix.
least.cost.path	dataframe or list of fdataframes produced by leastCostPath. If a list, must have the same number of slots as distance.matrix.
plot.columns	number of columns of the output plot if the inputs are lists. If not provided, it is computed automatically by n2mfrow.
plot.rows	number of rows of the output plot if the inputs are lists. If not provided, it is computed automatically by n2mfrow.
legend	boolean. If TRUE, the plot is made with image.plot, and includes a color scale on the right side. If FALSE, the plot is made with image, and the color scale is omitted.
color.palette	string defining the color palette to be used, or a color palette. Accepted strings are "divergent" (default), which uses a red-white-blue divergent palette produced by the code colorRampPalette(rev(RColorBrewer::brewer.pal(9,"RdBu")))(100), and "viridis", which uses the default settings of the viridis function to generate the palette. Both settings are color-blind friendly.
path.color	string, color of the line representing the least cost path if least.cost.path is provided.
path.width	line width (lwd) of the plotted path.

margins	a numeric vector with four positions indicating the margins of each plotted matrix. Order of margins in this vector is: bottom, left, top, right.
pdf.filename	character string with the name, without extension, of the pdf to be written. If NULL, no pdf is written.
pdf.width	with in inches of the output pdf. Default value is 7.
pdf.height	height in inches of the output pdf. Default value is 4.
pdf.pointsize	base font size of the output pdf.
rotate	boolean, if TRUE, the matrix is rotated. Allows the user to plot the matrix axes in the desired direction.

Value

A list of dataframes if least.cost.matrix is a list, or a dataframe if least.cost.matrix is a matrix. The dataframe/s have the following columns:

- *A* row/sample of one of the sequences.
- *B* row/sample of one the other sequence.
- *distance* distance between both samples, extracted from distance.matrix.
- *cumulative.distance* cumulative distance at the samples A and B.

Examples

```
#loading data
data(sequenceA)
data(sequenceB)

#preparing datasets
AB.sequences <- prepareSequences(
 sequence.A = sequenceA,
 sequence.A.name = "A",
 sequence.B = sequenceB,
 sequence.B.name = "B",
 merge.mode = "complete",
 if.empty.cases = "zero",
 transformation = "hellinger"
 )

#computing distance matrix
AB.distance.matrix <- distanceMatrix(
 sequences = AB.sequences,
 grouping.column = "id",
 method = "manhattan",
 parallel.execution = FALSE
 )

#plot
plotMatrix(distance.matrix = AB.distance.matrix)
```

```
#viridis palette
plotMatrix(distance.matrix = AB.distance.matrix,
 color.palette = "viridis")

#custom palette
plotMatrix(distance.matrix = AB.distance.matrix,
color.palette = viridis::viridis(8, option = "B", direction = -1))
```

`pollenGP` *Pollen dataset.*

Description

A subset of the Grande Pile dataset (https://doi.pangaea.de/10.1594/PANGAEA.739275). It contains a depth (cm) and age columns (ky BP), and 40 pollen types.

Usage

```
data(sequenceA)
```

Format

Dataframe with 42 columns and 200 rows

`prepareSequences` *Prepare sequences for a comparison analysis.*

Description

This function prepares two or more multivariate time-series that are to be compared. It can work on two different scenarios:

- *Two dataframes*: The user provides two separated dataframes, each containing a multivariate time series. These time-series can be regular or irregular, aligned or unaligned, but must have at least a few columns with the same names (pay attention to differences in case between column names representing the same entity) and units. This mode uses exclusively the following arguments: `sequence.A`, `sequence.A.name` (optional), `sequence.B`, `sequence.B.name` (optional), and `merge.model`.
- *One long dataframe*: The user provides a single dataframe, through the `sequences` argument, with two or more multivariate time-series identified by a `grouping.column`.

Usage

```
prepareSequences(
  sequence.A = NULL,
  sequence.A.name = "A",
  sequence.B = NULL,
  sequence.B.name = "B",
  merge.mode = "complete",
  sequences = NULL,
  grouping.column = NULL,
  time.column = NULL,
  exclude.columns = NULL,
  if.empty.cases = "zero",
  transformation = "none",
  paired.samples = FALSE,
  same.time = FALSE
  )
```

Arguments

sequence.A	dataframe containing a multivariate time-series.
sequence.A.name	character string with the name of sequence.A. Will be used as identificator in the id column of the output dataframe.
sequence.B	dataframe containing a multivariate time-series. Must have overlapping columns with sequence.A with same column names and units.
sequence.B.name	character string with the name of sequence.B. Will be used as identificator in the id column of the output dataframe.
merge.mode	character string, one of: "overlap", "complete" (default option). If "overlap", sequence.A and sequence.B are merged by their common columns, and non-common columns are dropped If "complete", columns absent in one dataset but present in the other are added, with values equal to 0. This argument is ignored if sequences is provided instead of sequence.A and sequence.B.
sequences	dataframe with multiple sequences identified by a grouping column.
grouping.column	character string, name of the column in sequences to be used to identify separates sequences within the file. If two sequences are provided through the arguments sequence.A and sequence.B, this argument defines the name of the grouping column in the output dataframe. If two or several sequences are provided as a single dataframe through the argument sequences, then grouping.column must be a column in this dataset.
time.column	character string, name of the column with time/depth/rank data. If sequence.A and sequence.B are provided, time.column must have the same name and units in both dataframes.
exclude.columns	character string or character vector with column names in sequences, or squence.A and sequence.B, to be excluded from the transformation.

`if.empty.cases` character string with two possible values: "omit", or "zero". If "zero" (default), NA values are replaced by zeroes. If "omit", rows with NA data are removed.

`transformation` character string. Defines what data transformation is to be applied to the sequences. One of: "none" (default), "percentage", "proportion", "hellinger", and "scale" (the latter centers and scales the data using the `scale` function).

`paired.samples` boolean. If TRUE, the function will test if the datasets have paired samples. This means that each dataset must have the same number of rows/samples, and that, if available, the `time.column` must have the same values in every dataset. This is only mandatory when using the functions `distancePairedSamples` or `workflowPsi` with `paired.samples = TRUE` after preparing the sequences. The default setting is FALSE.

`same.time` boolean. If TRUE, samples in the sequences to compare will be tested to check if they have the same time/age/depth according to `time.column`. This argument is only useful when the user needs to compare two sequences taken at different sites but same time frames.

Value

A dataframe with the multivariate time series. If `squence.A` and `sequence.B` are provided, the column identifying the sequences is named "id". If `sequences` is provided, the time-series are identified by `grouping.column`.

Author(s)

Blas Benito <blasbenito@gmail.com>

Examples

```
#two sequences as inputs
data(sequenceA)
data(sequenceB)

AB.sequences <- prepareSequences(
 sequence.A = sequenceA,
 sequence.A.name = "A",
 sequence.B = sequenceB,
 sequence.B.name = "B",
 merge.mode = "complete",
 if.empty.cases = "zero",
 transformation = "hellinger"
 )


#several sequences in a single dataframe
data(sequencesMIS)
MIS.sequences <- prepareSequences(
 sequences = sequencesMIS,
 grouping.column = "MIS",
 if.empty.cases = "zero",
```

```
transformation = "hellinger"
)
```

---

psi *Computes sum of distances between consecutive samples in a multivariate time-series.*

Description

Computes the sum of distances between consecutive samples in a multivariate time-series. Required to compute the measure of dissimilarity `psi` (Birks and Gordon 1985). Distances can be computed through the methods "manhattan", "euclidean", "chi", and "hellinger", and are implemented in the function `distance`.

Usage

```
psi(
  least.cost = NULL,
  autosum = NULL,
  parallel.execution = TRUE)
```

Arguments

`least.cost`	character string, name of the column with time/depth/rank data. The data in this column is not modified.
`autosum`	dataframe with one or several multivariate time-series identified by a grouping column.
`parallel.execution`	boolean, if `TRUE` (default), execution is parallelized, and serialized if `FALSE`.

Details

The measure of dissimilarity `psi` is computed as: `least.cost -(autosum of sequences)) / autosum of sequences`. It has a lower limit at 0, while there is no upper limit.

Value

A list with named slots, each one with a psi value.

Author(s)

Blas Benito <blasbenito@gmail.com>

- Birks, H.J.B. and Gordon, A.D. (1985) Numerical Methods in Quaternary Pollen Analysis. Academic Press.

Examples

```
#loading data
data(sequenceA)
data(sequenceB)

#preparing datasets
AB.sequences <- prepareSequences(
 sequence.A = sequenceA,
 sequence.A.name = "A",
 sequence.B = sequenceB,
 sequence.B.name = "B",
 merge.mode = "complete",
 if.empty.cases = "zero",
 transformation = "hellinger"
 )

#computing distance matrix
AB.distance.matrix <- distanceMatrix(
 sequences = AB.sequences,
 grouping.column = "id",
 method = "manhattan",
 parallel.execution = FALSE
 )

#computing least cost matrix
AB.least.cost.matrix <- leastCostMatrix(
 distance.matrix = AB.distance.matrix,
 diagonal = FALSE,
 parallel.execution = FALSE
 )

AB.least.cost.path <- leastCostPath(
 least.cost.matrix = AB.least.cost.matrix,
 distance.matrix = AB.distance.matrix,
 parallel.execution = FALSE
 )

#extracting least cost
AB.least.cost <- leastCost(
 least.cost.path = AB.least.cost.path,
 parallel.execution = FALSE
 )

#autosum
AB.autosum <- autoSum(
 sequences = AB.sequences,
 least.cost.path = AB.least.cost.path,
 grouping.column = "id",
 parallel.execution = FALSE
 )
```

```
AB.autosum

AB.psi <- psi(
 least.cost = AB.least.cost,
 autosum = AB.autosum,
 parallel.execution = FALSE
 )
AB.psi
```

---

sequenceA *Multivariate and irregular time series with pollen counts.*

Description

A dataframe with 9 columns representing pollen types (betula, pinus, corylus, empetrum, cypera, artemisia, rumex) and 49 rows representing increasing depths with pollen counts taken from the Abernethy dataset (Birks and Mathewes (1978).

Usage

```
data(sequenceA)
```

Format

Dataframe with 9 columns and 49 rows

References

Birks, H.H. and Mathewes, R.W. (1978) Studies in the vegetational history of Scotland. *New Phytologist* **80**, 455-484.

---

sequenceB *Multivariate and irregular time series with pollen counts.*

Description

A dataframe with 8 columns (the column `empetr` is missing with respect to `sequenceA`) representing pollen types (betula, pinus, corylus, cypera, artemisia, rumex) and 41 rows representing increasing depths with pollen counts taken from the Abernethy dataset (Birks and Mathewes (1978). Several NA values have been introduced in the dataset to demonstrate the data-handling capabilities of `prepareSequences`.

Usage

```
data(sequenceB)
```

Format

Dataframe with 9 columns and 41 rows

References

Birks, H.H. and Mathewes, R.W. (1978) Studies in the vegetational history of Scotland. *New Phytologist* **80**, 455-484.

---

sequencesMIS *Dataframe with pollen counts for different MIS stages.*

Description

A dataframe with 427 rows representing pollen counts for 12 marine isotope stages and 6 pollen types

Usage

```
data(sequencesMIS)
```

Format

dataframe with 7 columns and 427 rows.

---

workflowImportance *Computes the contribution to dissimilarity of each variable.*

Description

This workflow executes the following steps:

- computes `psi` as done by `workflowPsi`.
- computes `psi` as many times as numeric variables in `sequences`, removing one of them each time (jacknife analysis) to compute the relative contribution of each variable to overall dissimilarity.
- Delivers an output of type "list" with two slots:
  - `psi` a dataframe with the columns "A" and "B" with the respective names of the sequences compared, a column named "All variables" with the psi values of each pair of sequences computed by considering all variables, and then one column per variable, indicating the `psi` value when that variable is removed.

- psi.drop a dataframe with the columns "A" and "B", and then one column per numeric variable in sequences indicating the percentage of drop in psi (as indicated by the "All variables" column in the psi dataframe) when the given variable is removed. Positive values indicate that the given variable reduces dissimilarity when removed, making the sequences more similar, while negative values indicate that the variable increases dissimilarity when removed, making the sequences more different.

Usage

```
workflowImportance(
  sequences = NULL,
  grouping.column = NULL,
  time.column = NULL,
  exclude.columns = NULL,
  method = "manhattan",
  diagonal = FALSE,
  paired.samples = FALSE,
  same.time = FALSE,
  ignore.blocks = FALSE,
  parallel.execution = TRUE
  )
```

Arguments

sequences	dataframe with multiple sequences identified by a grouping column generated by prepareSequences.
grouping.column	character string, name of the column in sequences to be used to identify separates sequences within the file.
time.column	character string, name of the column with time/depth/rank data.
exclude.columns	character string or character vector with column names in sequences to be excluded from the analysis.
method	character string naming a distance metric. Valid entries are: "manhattan", "euclidean", "chi", and "hellinger". Invalid entries will throw an error.
diagonal	boolean, if TRUE, diagonals are included in the computation of the least cost path. Defaults to FALSE, as the original algorithm did not include diagonals in the computation of the least cost path.
paired.samples	boolean, if TRUE, the sequences are assumed to be aligned, and distances are computed for paired-samples only (no distance matrix required). Default value is FALSE.
same.time	boolean. If TRUE, samples in the sequences to compare will be tested to check if they have the same time/age/depth according to time.column. This argument is only useful when the user needs to compare two sequences taken at different sites but same time frames.
ignore.blocks	boolean. If TRUE, the function leastCostPathNoBlocks analyzes the least-cost path of the best solution, and removes blocks (straight-orthogonal sections of

the least-cost path), which happen in highly dissimilar sections of the sequences, and inflate output psi values.

`parallel.execution`
: boolean, if `TRUE` (default), execution is parallelized, and serialized if `FALSE`.

Details

If we consider the question "what variable contributes the most to the dissimilarity between two sequences?" the answer "the one dropping dissimilarity the most when excluded from the analysis" sounds like a reasonable answer. This workflow attempts to reach that answer by computing `psi` while removing one variable at a time.

Value

A list with two slots named *psi* and *psi.drop*. The former contains the dissimilarity values when removing each variable, while the latter contains the drop in dissimilarity (as a percentage of psi computed on all variables) that happens when each variable is removed. Positive values indicate that dissimilarity drops when the variable is removed, while negative values indicate that similarity drops when the variable is removed.

Author(s)

Blas Benito <blasbenito@gmail.com>

`workflowImportanceHP` *Computes the contribution to dissimilarity of each variable using workflowPsiHP.*

Description

This workflow executes the following steps:

- computes `psi` as done by `workflowPsi`.
- computes `psi` as many times as numeric variables in `sequences`, removing one of them each time (jacknife analysis) to compute the relative contribution of each variable to overall dissimilarity.
- Delivers an output of type "list" with two slots:
  - `psi` a dataframe with the columns "A" and "B" with the respective names of the sequences compared, a column named "All variables" with the psi values of each pair of sequences computed by considering all variables, and then one column per variable, indicating the `psi` value when that variable is removed.
  - `psi.drop` a dataframe with the columns "A" and "B", and then one column per numeric variable in `sequences` indicating the percentage of drop in `psi` (as indicated by the "All variables" column in the psi dataframe) when the given variable is removed. Positive values indicate that the given variable reduces dissimilarity when removed, making the sequences more similar, while negative values indicate that the variable increases dissimilarity when removed, making the sequences more different.

Usage

```
workflowImportanceHP(
  sequences = NULL,
  grouping.column = NULL,
  time.column = NULL,
  exclude.columns = NULL,
  parallel.execution = TRUE
  )
```

Arguments

sequences	dataframe with multiple sequences identified by a grouping column generated by prepareSequences.
grouping.column	character string, name of the column in sequences to be used to identify separates sequences within the file.
time.column	character string, name of the column with time/depth/rank data.
exclude.columns	character string or character vector with column names in sequences to be excluded from the analysis.
parallel.execution	boolean, if TRUE (default), execution is parallelized, and serialized if FALSE.

Details

If we consider the question "what variable contributes the most to the dissimilarity between two sequences?" the answer "the one dropping dissimilarity the most when excluded from the analysis" sounds like a reasonable answer. This workflow attempts to reach that answer by computing psi while removing one variable at a time.

Value

A list with two slots named *psi* and *psi.drop*. The former contains the dissimilarity values when removing each variable, while the latter contains the drop in dissimilarity (as a percentage of psi computed on all variables) that happens when each variable is removed. Positive values indicate that dissimilarity drops when the variable is removed, while negative values indicate that similarity drops when the variable is removed.

Author(s)

Blas Benito <blasbenito@gmail.com>

workflowNullPsi *Computes the dissimilarity measure* psi *on restricted permutations of two or more sequences.*

Description

The function first computes psi on the observed sequences, and then computes it on permutations of the input sequences by the `repetitions` argument. The data is randomized as follows: within each column, each data-point can be: 1) left as is; 2) replaced by the previous case; 3) replaced by the next case. The action applied to each data-point is selected randomly, and independently from the actions applied to other data-points. This type of randomization generates versions of the dataset that have the same general structure as the original one, but small local and independent changes only ocurring within the immediate neighborhood (one row up or down) of each case in the table. The method should generate very conservative random values of `psi`.

Usage

```
workflowNullPsi(
  sequences = NULL,
  grouping.column = NULL,
  time.column = NULL,
  exclude.columns = NULL,
  method = "manhattan",
  diagonal = FALSE,
  paired.samples = FALSE,
  same.time = FALSE,
  ignore.blocks = FALSE,
  parallel.execution = TRUE,
  repetitions = 9
  )
```

Arguments

`sequences`	dataframe with multiple sequences identified by a grouping column generated by `prepareSequences`.
`grouping.column`	character string, name of the column in `sequences` to be used to identify separates sequences within the file.
`time.column`	character string, name of the column with time/depth/rank data.
`exclude.columns`	character string or character vector with column names in `sequences` to be excluded from the analysis.
`method`	character string naming a distance metric. Valid entries are: "manhattan", "euclidean", "chi", and "hellinger". Invalid entries will throw an error.

diagonal	boolean, if TRUE, diagonals are included in the computation of the least cost path. Defaults to FALSE, as the original algorithm did not include diagonals in the computation of the least cost path. If paired.samples is TRUE, then diagonal is irrelevant.
paired.samples	boolean, if TRUE, the sequences are assumed to be aligned, and distances are computed for paired-samples only (no distance matrix required). Default value is FALSE.
same.time	boolean. If TRUE, samples in the sequences to compare will be tested to check if they have the same time/age/depth according to time.column. This argument is only useful when the user needs to compare two sequences taken at different sites but same time frames.
ignore.blocks	boolean. If TRUE, the function leastCostPathNoBlocks analyzes the least-cost path of the best solution, and removes blocks (straight-orthogonal sections of the least-cost path), which happen in highly dissimilar sections of the sequences, and inflate output psi values.
parallel.execution	boolean, if TRUE (default), execution is parallelized, and serialized if FALSE.
repetitions	integer, number of null psi values to obtain.

Value

A list with two slots:

- *psi*: a dataframe. The first two columns contain the names of the sequences being compared, the third column contains the real psi value, and the rest of the column contain psi values computed on permutated versions of the datasets.
- *p*: a dataframe. The first two columns are as above, the third column contains the probability of obtaining a random psi lower than the real psi by chance.

Author(s)

Blas Benito <blasbenito@gmail.com>

Examples

```
#load data
data("sequencesMIS")

#prepare sequences
MIS.sequences <- prepareSequences(
  sequences = sequencesMIS,
  grouping.column = "MIS",
  transformation = "hellinger"
  )

#execute workflow to compute psi
MIS.null.psi <- workflowNullPsi(
```

```
 sequences = MIS.sequences[MIS.sequences$MIS %in% c("MIS-1", "MIS-2"), ],
 grouping.column = "MIS",
 method = "manhattan",
 repetitions = 3,
 parallel.execution = FALSE
 )

MIS.null.psi
```

`workflowNullPsiHP` *Computes the dissimilarity measure* psi *on restricted permutations of two or more sequences. High performance version with limited options*

Description

The function first computes psi on the observed sequences, and then computes it on permutations of the input sequences by the `repetitions` argument. The data is randomized as follows: within each column, each data-point can be: 1) left as is; 2) replaced by the previous case; 3) replaced by the next case. The action applied to each data-point is selected randomly, and independently from the actions applied to other data-points. This type of randomization generates versions of the dataset that have the same general structure as the original one, but small local and independent changes only ocurring within the immediate neighborhood (one row up or down) of each case in the table. The method should generate very conservative random values of `psi`.

Usage

```
workflowNullPsiHP(
  sequences = NULL,
  grouping.column = NULL,
  time.column = NULL,
  exclude.columns = NULL,
  parallel.execution = TRUE,
  repetitions = 9
  )
```

Arguments

`sequences`	dataframe with multiple sequences identified by a grouping column generated by `prepareSequences`.
`grouping.column`	character string, name of the column in `sequences` to be used to identify separates sequences within the file.
`time.column`	character string, name of the column with time/depth/rank data.

`exclude.columns`
: character string or character vector with column names in `sequences` to be excluded from the analysis.

`parallel.execution`
: boolean, if `TRUE` (default), execution is parallelized, and serialized if `FALSE`.

`repetitions`
: integer, number of null psi values to obtain.

Value

A list with two slots:

- *psi*: a dataframe. The first two columns contain the names of the sequences being compared, the third column contains the real `psi` value, and the rest of the column contain `psi` values computed on permutated versions of the datasets.
- *p*: a dataframe. The first two columns are as above, the third column contains the probability of obtaining a `random` `psi` lower than the real `psi` by chance.

Author(s)

Blas Benito <blasbenito@gmail.com>

Examples

```
#load data
data("sequencesMIS")

#prepare sequences
MIS.sequences <- prepareSequences(
  sequences = sequencesMIS,
  grouping.column = "MIS",
  transformation = "hellinger"
  )

#execute workflow to compute psi
MIS.null.psi <- workflowNullPsiHP(
 sequences = MIS.sequences[MIS.sequences$MIS %in% c("MIS-1", "MIS-2"), ],
 grouping.column = "MIS",
 repetitions = 3,
 parallel.execution = FALSE
 )

MIS.null.psi
```

`workflowPartialMatch` *Finds the section in a long sequence that better matches a short sequence.*

Description

This workflow works under the following scenario: the user has a short sequence, and a long sequence, and has the objective of finding the segment in the long sequence that better matches the short sequence. The function identifies automatically the short and the long sequence, but throws an error if more than two sequences are introduced. The lengths of the segments in the long sequence to be compared with the long sequence are defined through the arguments `min.length` and `max.length`. If left empty, `min.length` and `max.length` equal 0, meaning that the segment to be searched for will have the same number of cases as the short sequence. Note that this is a brute force algorithm, can have a large memory footpring if the interval between `min.length` and `max.length` is too long. It might be convenient to pre-check the number of iterations to be performed by computing `sum(nrow(long.sequence) -min.length:max.length) + 1`. The algorithm is parallelized and optimized as possible, so still, large searches are possible.

Usage

```
workflowPartialMatch(
  sequences = NULL,
  grouping.column = NULL,
  time.column = NULL,
  exclude.columns = NULL,
  method = "manhattan",
  diagonal = FALSE,
  paired.samples = FALSE,
  min.length = NULL,
  max.length = NULL,
  ignore.blocks = FALSE,
  parallel.execution = TRUE
  )
```

Arguments

`sequences`	dataframe with multiple sequences identified by a grouping column generated by `prepareSequences`.
`grouping.column`	character string, name of the column in `sequences` to be used to identify separates sequences within the file.
`time.column`	character string, name of the column with time/depth/rank data.
`exclude.columns`	character string or character vector with column names in `sequences` to be excluded from the analysis.

method	character string naming a distance metric. Valid entries are: "manhattan", "euclidean", "chi", and "hellinger". Invalid entries will throw an error.
diagonal	boolean, if TRUE (default), diagonals are included in the computation of the least cost path. This is the best option if the user suspects that a given segment in the short sequence might be identical to the short sequence.
paired.samples	boolean, if TRUE, the sequences are assumed to be aligned, and distances are computed for paired-samples only (no distance matrix required). Default value is FALSE.
min.length	integer, minimum length (in rows) of the subsets of the long sequence to be matched against the short sequence. If NULL (default), the subset of the long sequence to be matched will thave the same number of samples as the short sequence.
max.length	integer, maximum length (in rows) of the subsets of the long sequence to be matched against the short sequence. If NULL (default), the subset of the long sequence to be matched will thave the same number of samples as the short sequence.
ignore.blocks	boolean. If TRUE, the function leastCostPathNoBlocks analyzes the least-cost path of the best solution, and removes blocks (straight-orthogonal sections of the least-cost path), which happen in highly dissimilar sections of the sequences, and inflate output psi values.
parallel.execution	boolean, if TRUE (default), execution is parallelized, and serialized if FALSE.

Value

A dataframe with three columns:

- *first.row* first row of the segment in the long sequence matched against the short one.
- *last.row* last row of the segment in the long sequence matched against the short one.
- *psi* psi values, ordered from lower (máximum similarity / minimum dissimilarity) to higher.

Author(s)

Blas Benito <blasbenito@gmail.com>

Examples

```
#loading the data
data(sequencesMIS)

#removing grouping column
sequencesMIS$MIS <- NULL

#mock-up short sequence
MIS.short <- sequencesMIS[1:10, ]
```

```
#mock-up long sequence
MIS.long <- sequencesMIS[1:30, ]

#preparing sequences
MIS.sequences <- prepareSequences(
 sequence.A = MIS.short,
 sequence.A.name = "short",
 sequence.B = MIS.long,
 sequence.B.name = "long",
 grouping.column = "id",
 transformation = "hellinger"
 )

#matching sequences
#min.length and max.length are
#minimal to speed up execution
MIS.psi <- workflowPartialMatch(
 sequences = MIS.sequences,
 grouping.column = "id",
 time.column = NULL,
 exclude.columns = NULL,
 method = "manhattan",
 diagonal = FALSE,
 parallel.execution = FALSE
 )

#output dataframe
MIS.psi
```

`workflowPsi` *Computes the dissimilarity measure* psi *on two or more sequences.*

Description

If the sequences are not aligned (`paired.samples = FALSE`), the function executes these steps.

- Computes the autosum of the sequences with `autoSum`.
- Computes the distance matrix with `distanceMatrix`.
- Uses the distance matrix to compute the least cost matrix with `leastCostMatrix`.
- Extracts the cost of the least cost path with `leastCost`.
- Computes the dissimilarity measure *psi* with the function `psi`.
- Delivers an output of type "list" (default), "data.frame" or "matrix", depending on the user input, through `formatPsi`.

If the sequences are aligned (`paired.samples = TRUE`), these steps are executed:

- Computes the autosum of the sequences with autoSum.
- Sums the distances between paired samples with distancePairedSamples.
- Computes the dissimilarity measure *psi* with the function psi.
- Delivers an output of type "list" (default), "data.frame" or "matrix", depending on the user input, through formatPsi.

Usage

```
workflowPsi(
  sequences = NULL,
  grouping.column = NULL,
  time.column = NULL,
  exclude.columns = NULL,
  method = "manhattan",
  diagonal = FALSE,
  format = "dataframe",
  paired.samples = FALSE,
  same.time = FALSE,
  ignore.blocks = FALSE,
  parallel.execution = TRUE
  )
```

Arguments

sequences	dataframe with multiple sequences identified by a grouping column generated by prepareSequences.
grouping.column	character string, name of the column in sequences to be used to identify separates sequences within the file.
time.column	character string, name of the column with time/depth/rank data.
exclude.columns	character string or character vector with column names in sequences to be excluded from the analysis.
method	character string naming a distance metric. Valid entries are: "manhattan", "euclidean", "chi", and "hellinger". Invalid entries will throw an error.
diagonal	boolean, if TRUE, diagonals are included in the computation of the least cost path. Defaults to FALSE, as the original algorithm did not include diagonals in the computation of the least cost path. If paired.samples is TRUE, then diagonal is irrelevant.
format	string, type of output. One of: "data.frame", "matrix". If NULL or empty, a list is returned.
paired.samples	boolean, if TRUE, the sequences are assumed to be aligned, and distances are computed for paired-samples only (no distance matrix required). Default value is FALSE.

same.time boolean. If TRUE, samples in the sequences to compare will be tested to check if they have the same time/age/depth according to time.column. This argument is only useful when the user needs to compare two sequences taken at different sites but same time frames.

ignore.blocks boolean. If TRUE, the function leastCostPathNoBlocks analyzes the least-cost path of the best solution, and removes blocks (straight-orthogonal sections of the least-cost path), which happen in highly dissimilar sections of the sequences, and inflate output psi values.

parallel.execution
boolean, if TRUE (default), execution is parallelized, and serialized if FALSE.

Value

A list, matrix, or dataframe, with sequence names and psi values.

Author(s)

Blas Benito <blasbenito@gmail.com>

Examples

```
data("sequencesMIS")
#prepare sequences
MIS.sequences <- prepareSequences(
  sequences = sequencesMIS,
  grouping.column = "MIS",
  if.empty.cases = "zero",
  transformation = "hellinger"
  )

#execute workflow to compute psi
MIS.psi <- workflowPsi(
 sequences = MIS.sequences[MIS.sequences$MIS %in% c("MIS-1", "MIS-2"), ],
 grouping.column = "MIS",
 time.column = NULL,
 exclude.columns = NULL,
 method = "manhattan",
 diagonal = FALSE,
 parallel.execution = FALSE
 )

MIS.psi
```

workflowPsiHP *A refactored version of* workflowPsi *with a higher performance (hence the suffix HP).*

Description

Ideal for large analyses with hundreds to thousands of sequences. Several options available in workflowPsi have been removed from this function in order to simplify the code as much as possible. Psi is computed with the options diagonal = TRUE, ignore.blocks = TRUE, and method = "euclidean".

Usage

```
workflowPsiHP(
  sequences = NULL,
  grouping.column = NULL,
  time.column = NULL,
  exclude.columns = NULL,
  parallel.execution = TRUE
  )
```

Arguments

sequences	dataframe with multiple sequences identified by a grouping column generated by prepareSequences.
grouping.column	character string, name of the column in sequences to be used to identify separates sequences within the file.
time.column	character string, name of the column with time/depth/rank data.
exclude.columns	character string or character vector with column names in sequences to be excluded from the analysis.
parallel.execution	boolean, if TRUE (default), execution is parallelized, and serialized if FALSE.

Details

Due to limitations of the function permutations, the maximum number of groups (according to grouping.column) is around 30000. Besides, a combinations table of this size takes, roughlyl, 7GB of memory.

Value

A dataframe with sequence names and psi values.

Author(s)

Blas Benito <blasbenito@gmail.com>

Examples

```
data("sequencesMIS")
#prepare sequences
MIS.sequences <- prepareSequences(
  sequences = sequencesMIS[sequencesMIS$MIS %in% c("MIS-1", "MIS-2"), ],
  grouping.column = "MIS",
  if.empty.cases = "zero",
  transformation = "hellinger"
  )

#execute workflow to compute psi
MIS.psi <- workflowPsiHP(
 sequences = MIS.sequences,
 grouping.column = "MIS",
 parallel.execution = FALSE
 )

MIS.psi
```

---

workflowSlotting *Slots two sequences into a single composite sequence.*

Description

Generates a composite sequence, constrained by sample order, from two sequences, by minimizing the dissimilarity between adjacent samples of each input sequence. The algorithm computes the distance matrix, least cost matrix, and least cost path of two sequences, and uses the least cost path file to find the slotting that better minimizes the dissimilarity between adjacent samples. The algorithm assumes that the samples are not aligned or paired.

Usage

```
workflowSlotting(
  sequences = NULL,
  grouping.column = NULL,
  time.column = NULL,
  exclude.columns = NULL,
  method = "manhattan",
  plot = TRUE
  )
```

Arguments

sequences	dataframe with two sequences identified by a grouping column generated by prepareSequences.
grouping.column	character string, name of the column in sequences to be used to identify separates sequences within the file.
time.column	character string, name of the column with time/depth/rank data.
exclude.columns	character string or character vector with column names in sequences to be excluded from the analysis.
method	character string naming a distance metric. Valid entries are: "manhattan", "euclidean", "chi", and "hellinger". Invalid entries will throw an error.
plot	boolean, if TRUE, plots the distance matrix and the least-cost path.

Value

A dataframe with the same number of rows as sequences, ordered according to the best solution found by the least-cost algorithm.

Author(s)

Blas Benito <blasbenito@gmail.com>

Examples

```
#loading the data
data(pollenGP)

#getting first 20 samples
pollenGP <- pollenGP[1:20, ]

#sampling indices
set.seed(10) #to get same result every time
sampling.indices <- sort(sample(1:20, 10))

#subsetting the sequence
A <- pollenGP[sampling.indices, ]
B <- pollenGP[-sampling.indices, ]

#preparing the sequences
AB <- prepareSequences(
  sequence.A = A,
  sequence.A.name = "A",
  sequence.B = B,
  sequence.B.name = "B",
  grouping.column = "id",
  exclude.columns = c("depth", "age"),
```

```
  transformation = "hellinger"
  )

AB.combined <- workflowSlotting(
  sequences = AB,
  grouping.column = "id",
  time.column = "age",
  exclude.columns = "depth",
  method = "manhattan",
  plot = TRUE
  )

AB.combined
```

`workflowTransfer` *Transfers an attribute (time, age, depth) from one sequence to another*

Description

Transfers an attribute (generally time/age, but any others are possible) from one sequence (defined by the argument `transfer.from`) to another (defined by the argument `transfer.to`) lacking it. The transference of the attribute is based on the following assumption: similar samples have similar attributes. This assumption might not hold for noisy multivariate time-series. Attribute transference can be done in two different ways (defined by the `mode` argument):

- *Direct*: transfers the selected attribute between samples with the maximum similarity. This option will likely generate duplicated attribute values in the output.
- *Interpolate*: obtains new attribute values through weighted interpolation, being the weights derived from the distances between samples

Usage

```
workflowTransfer(
  sequences = NULL,
  grouping.column = NULL,
  time.column = NULL,
  exclude.columns = NULL,
  method = "manhattan",
  transfer.what = NULL,
  transfer.from = NULL,
  transfer.to = NULL,
  mode = "direct",
  plot = FALSE
  )
```

Arguments

sequences	dataframe with multiple sequences identified by a grouping column generated by prepareSequences.
grouping.column	character string, name of the column in sequences to be used to identify separates sequences within the file.
time.column	character string, name of the column with time/depth/rank data.
exclude.columns	character string or character vector with column names in sequences to be excluded from the analysis.
method	character string naming a distance metric. Valid entries are: "manhattan", "euclidean", "chi", and "hellinger". Invalid entries will throw an error.
transfer.what	character string, column of sequences with the attribute to be transferred. If empty or ill-defined, time.column is used instead if available.
transfer.from	character string, group available in grouping.column identifying the sequence from which to take the attribute values.
transfer.to	character string, group available in grouping.column identifying the sequence to which transfer the attribute values.
mode	character string, one of: "direct" (default), "interpolate".
plot	boolean, if TRUE, plots the distance matrix and the least-cost path.

Value

A dataframe with the sequence transfer.to, with a column named after transfer.what with the attribute values.

Author(s)

Blas Benito <blasbenito@gmail.com>

Examples

```
#loading sample dataset
data(pollenGP)
#subset pollenGP to make a shorter dataset
pollenGP <- pollenGP[1:50, ]

#generating a subset of pollenGP
set.seed(10)
pollenX <- pollenGP[sort(sample(1:50, 40)), ]

#we separate the age column
pollenX.age <- pollenX$age
```

```
#and remove the age values from pollenX
pollenX$age <- NULL
pollenX$depth <- NULL

#removing some samples from pollenGP
#so pollenX is not a perfect subset of pollenGP
pollenGP <- pollenGP[-sample(1:50, 10), ]

#prepare sequences
GP.X <- prepareSequences(
  sequence.A = pollenGP,
  sequence.A.name = "GP",
  sequence.B = pollenX,
  sequence.B.name = "X",
  grouping.column = "id",
  time.column = "age",
  exclude.columns = "depth",
  transformation = "none"
  )

#transferring age
X.new <- workflowTransfer(
 sequences = GP.X,
 grouping.column = "id",
 time.column = "age",
 method = "manhattan",
 transfer.what = "age",
 transfer.from = "GP",
 transfer.to = "X",
 mode = "interpolated"
 )
```

Index