Package 'IncDTW'

October 9, 2020

Type Package

Title Incremental Calculation of Dynamic Time Warping

Version 1.1.4.2

Author Maximilian Leodolter

Maintainer Maximilian Leodolter <maximilian.leodolter@gmail.com>

Description The Dynamic Time Warping (DTW) distance measure for time series allows non-linear alignments of time series to match similar patterns in time series of different lengths and or different speeds. IncDTW is characterized by (1) the incremental calculation of DTW (reduces runtime complexity to a linear level for updating the DTW distance) - especially for life data streams or subsequence matching, (2) the vector based implementation of DTW which is faster because no matrices are allocated (reduces the space complexity from a quadratic to a linear level in the number of observations) - for all runtime intensive DTW computations, (3) the subsequence matching algorithm runDTW, that efficiently finds the k-NN to a query pattern in a long time series, and (4) C++ in the heart. For details about DTW see the original paper ``Dynamic programming algorithm optimization for spoken word recognition" by Sakoe and Chiba (1978) <DOI:10.1109/TASSP.1978.1163055>.

License GPL (>= 2)

Encoding UTF-8

LazyData true

Depends R (>= 2.10)

Imports Rcpp (>= 0.12.8), RcppParallel, ggplot2, scales, parallel, stats, data.table

LinkingTo Rcpp, RcppParallel, RcppArmadillo

NeedsCompilation yes

RoxygenNote 6.1.1

Suggests knitr, dtw, rmarkdown, gridExtra, testthat, dtwclust, parallelDist, microbenchmark, rucrdtw, proxy, R.rsp, dendextend, reshape2, colorspace, fastcluster

VignetteBuilder knitr, R.rsp

SystemRequirements GNU make

Repository CRAN

Date/Publication 2020-10-09 08:30:02 UTC

R topics documented:

`IncDTW-package` *Incremental Dynamic Time Warping*

Description

The Dynamic Time Warping (DTW) distance for time series allows non-linear alignments of time series to match similar patterns in time series of different lengths and or different speeds. Beside the traditional implementation of the DTW algorithm, the specialties of this package are, (1) the incremental calculation, which is specifically useful for life data streams due to computationally efficiency, (2) the vector based implementation of the traditional DTW algorithm which is faster because no matrices are allocated and is especially useful for computing distance matrices of pair-wise DTW distances for many time series and (3) the combination of incremental and vector-based calculation.

Details

Main features:

- Incremental Calculation, `idtw`, `idtw2vec` and `increment`
- Detect k-nearest subsequences in longer time series, `rundtw`
- Matrix-based `dtw` and Vector-based `dtw2vec` implementation of the DTW algorithm
- Sakoe Chiba warping window
- Early abandoning and lower bounding

- Support for multivariate time series
- Fast calculation of a distance matrix of pairwise DTW distances for clustering or classification of many multivariate time series, dtw_dismat
- Aggregate cluster members with dba or get the centroid with centroid
- C++ in the heart thanks to Rcpp

Author(s)

Maximilian Leodolter

Maintainer: Maximilian Leodolter <maximilian.leodolter@gmail.com>

References

Dynamic programming algorithm optimization for spoken word recognition by Sakoe and Chiba published in 1978 (DOI:10.1109/TASSP.1978.1163055)

See Also

https://ieeexplore.ieee.org/document/1163055/

https://en.wikipedia.org/wiki/Dynamic_time_warping

https://github.com/maxar/IncDTW

---

dba *Dynamic Time Warping Barycenter Averaging*

Description

Average multiple time series that are non-linearly aligned by Dynamic Time Warping. Find the centroid of a list of time series.

Usage

```
dba(lot, m0 = NULL, iterMax = 10, eps = NULL,
                dist_method = c("norm1", "norm2", "norm2_square"),
                step_pattern = c("symmetric2", "symmetric1"),
                ws = NULL,
                iter_dist_method = c("dtw_norm1", "dtw_norm2",
                                     "norm1","norm2", "max", "min"),
                plotit = FALSE)

# deprecated
DBA(lot, m0 = NULL, iterMax = 10, eps = NULL,
                dist_method = c("norm1", "norm2", "norm2_square"),
                step_pattern = c("symmetric2", "symmetric1"),
                ws = NULL,
                iter_dist_method = c("dtw_norm1", "dtw_norm2",
```

```
                                        "norm1","norm2", "max", "min"),
                 plotit = FALSE)

centroid(lot, dist_method = c("norm1", "norm2", "norm2_square"),
              step_pattern = c("symmetric2", "symmetric1"),
              normalize = TRUE, ws = NULL, ncores = NULL,
              useRcppParallel = TRUE)



## S3 method for class 'dba'
print(x, digits = getOption("digits"),  ...)

## S3 method for class 'dba'
summary(object, ...)

is.dba(x)
```

Arguments

lot	List of time series. Each entry of the list is a time series as described in dtw2vec.
m0	time series as vector or matrix. If m0 is NULL, the initial time series m0 is determined by centroid as the centroid of lot, which is the one time series of lot with the minimum average DTW distance to all other time series of lot.
iterMax	integer, number of maximum iterations
eps	numeric, threshold parameter that causes the algorithm to break if the distance of two consecutive barycenters are closer than eps
dist_method	character, describes the method of distance measure. See also dtw.
step_pattern	character, describes the step pattern. See also dtw.
ws	integer, describes the window size for the sakoe chiba window. If NULL, then no window is applied. (default = NULL)
iter_dist_method	character, that describes how the distance between two consecutive barycenter iterations are defined (default = "dtw")
plotit	logical, if the iterations should be plotted or not (only possible for univariate time series)
normalize	logical, default is TRUE, passed to dtw_dismat
ncores	integer, default = NULL, passed to dtw_dismat
useRcppParallel	logical, default is TRUE, passed to dtw_dismat
x	output from dba
object	any R object
digits	passed to round and print
...	additional arguments, e.g. passed to print or summary

Details

The parameter `iter_dist_method` describes the method to measure the progress between two iterations. For two consecutive centroid candidates `m1` and `m2` the following methods are implemented:

'dtw_norm1': `dtw2vec(m1,m2,dist_method = "norm1",step_pattern = "symmetric2")$normalized_distance`

'idm_dtw2': `dtw2vec(m1,m2,dist_method = "norm2",step_pattern = "symmetric2")$normalized_distance`

'idm_norm1': `sum(abs(m1-m2))/(ncol(m1) * 2 * nrow(m1))`

'idm_norm2': `sqrt(sum((m1-m2)^2))/(ncol(m1) * 2 * nrow(m1))`

'idm_max': `max(abs(m1-m2))`

'idm_min': `min(abs(m1-m2))`

Value

`call`	function call
`m1`	new centroid/ bary center of the list of time series
`iterations`	list of time series that are the best centroid of the respective iteration
`iterDist_m2lot`	list of distances of the iterations to lot
`iterDist_m2lot_norm`	list of normalized distances of the iterations to lot
`iterDist_m2m`	vector of distances of the iterations to their ancestors
`centroid_index`	integer giving the index of the centroid time series of `lot`
`dismat_result`	list of results of `dtw_dismat` called by `centroid`

References

Sakoe, H.; Chiba, S., Dynamic programming algorithm optimization for spoken word recognition, Acoustics, Speech, and Signal Processing [see also IEEE Transactions on Signal Processing], IEEE Transactions on , vol.26, no.1, pp. 43-49, Feb 1978. https://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1163055

Petitjean, F; Ketterlin, A; Gancarski, P, A global averaging method for dynamic time warping, with applications to clustering, Pattern Recognition, Volume 44, Issue 3, 2011, Pages 678-693, ISSN 0031-3203

Examples

```
## Not run:
data("drink_glass")
# initialize with any time series
m1 <- dba(lot = drink_glass[1:10], m0 = drink_glass[[1]],
          dist_method = "norm2", iterMax = 20)

# initialize with the centroid

tmp <- centroid(drink_glass)
cent <- drink_glass[[tmp$centroid_index]]
m1 <- dba(lot = drink_glass[1:10], m0 = cent,
```

```
          dist_method = "norm2", iterMax = 20)

# plot all dimensions of the barycenters m_n per iteration:
plot(m1)

# plot the distances of the barycenter of one iteration m_n
# to the barycenter of the previous iteration m_n-1:
plot(m1, type = "m2m")

# plot the average distances of the barycenter m_n
# to the list of time series:
plot(m1, type = "m2lot")


## End(Not run)
```

---

dec_dm *Decrement the Warping Path*

Description

Update the warping path to omit observations of the alignment of two time series.

Usage

```
dec_dm(dm, Ndec, diffM = NULL)
```

Arguments

dm	direction matrix, output from dtw(Q=Q, C=C, ws=ws)
Ndec	integer, number of observations (columns) to be reduced
diffM	matrix of differences

Value

wp	warping path
ii	indices of Q of the optimal path
jj	indices of C of the optimal path
diffp	path of differences (only returned if diffM is not NULL)

References

Sakoe, H.; Chiba, S., Dynamic programming algorithm optimization for spoken word recognition, Acoustics, Speech, and Signal Processing [see also IEEE Transactions on Signal Processing], IEEE Transactions on , vol.26, no.1, pp. 43-49, Feb 1978. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1163055

Examples

```
Q <- cos(1:100)
C <- cumsum(rnorm(80))
# the ordinary calculation
result_base <- dtw(Q=Q, C=C, return_wp = TRUE)

# the ordinary calculation without the last 4 observations
result_decr <- dtw(Q=Q, C=C[1:(length(C) - 4)], return_wp = TRUE)
# the decremental step: reduce C for 4 observation
result_decr2 <- dec_dm(result_base$dm, Ndec = 4)

# compare ii, jj and wp of result_decr and those of
result_decr$ii
result_decr2$ii
identical(result_decr$ii, result_decr2$ii)

result_decr$jj
result_decr2$jj
identical(result_decr$jj, result_decr2$jj)

result_decr$wp
result_decr2$wp
identical(result_decr$wp, result_decr2$wp)
```

---

drink_glass *Accelerometer: drink a glass, walk, brush teeth.*

Description

3-dimensional acceleration time series recorded during the activities of walking, drinking a glass or brushing teeth.

Usage

```
data("drink_glass")
```

Format

A list of matrices, where each matrix has 3 columns (x, y, and z axis of the accelerometer). The number of rows differ.

Details

list of 3-dimensional time series stored as matrix. The data is recorded with 32Hz. The data is z-scaled (z-normalized).

Source

UCI Machine Learning Repository https://archive.ics.uci.edu/ml/datasets/Dataset+for+ADL+Recognition+with+Wrist-worn+Accelerometer

Examples

```
## Not run:
data(drink_glass)
class(drink_glass)
length(drink_glass)
dim(drink_glass[[1]])
matplot(drink_glass[[1]], type="l")

data(walk)
class(walk)
length(walk)
dim(walk[[1]])
matplot(walk[[1]], type="l")

data(brush_teeth)
class(brush_teeth)
length(brush_teeth)
dim(brush_teeth[[1]])
matplot(brush_teeth[[1]], type="l")

## End(Not run)
```

---

dtw *Dynamic Time Warping*

Description

Calculate the DTW distance, cost matrices and direction matrices including the warping path two multivariate time series.

Usage

```
dtw(Q, C, dist_method = c("norm1", "norm2", "norm2_square"),
    step_pattern = c("symmetric2", "symmetric1"), ws = NULL,
    return_cm = FALSE,
    return_diffM = FALSE,
    return_wp = FALSE,
    return_diffp = FALSE,
    return_QC = FALSE)

cm(Q, C, dist_method = c("norm1", "norm2", "norm2_square"),
   ws = NULL, ...)
```

```
## S3 method for class 'idtw'
print(x, digits = getOption("digits"), ...)

## S3 method for class 'idtw'
summary(object, ...)

is.idtw(x)
```

Arguments

Q	Query time series. Q needs to be one of the following: (1) a one dimensional vector, (2) a matrix where each row is one observations and each column is one dimension of the time series, or (3) a matrix of differences/ costs (diffM, cm). If Q and C are matrices they need to have the same number of columns.
C	Candidate time series. C needs to be one of the following: (1) a one dimensional vector, (2) a matrix where each row is one observations and each column is one dimension of the time series, or (3) if Q is a matrix of differences or costs C needs to be the respective character string 'diffM' or 'cm'.
dist_method	character, describes the method of distance measure for multivariate time series (this parameter is ignored for univariate time series). Currently supported methods are 'norm1' (default, is the Manhattan distance), 'norm2' (is the Euclidean distance) and 'norm2_square'. For the function `cm()` the parameter `dist_method` can also be a user defined distance function (see details and examples).
step_pattern	character, describes the step pattern. Currently implemented are the patterns `symmetric1` and `symmetric2`, see details.
ws	integer, describes the window size for the sakoe chiba window. If NULL, then no window is applied. (default = NULL)
return_cm	logical, if TRUE then the Matrix of costs (the absolute value) is returned. (default = FALSE)
return_diffM	logical, if TRUE then the Matrix of differences (not the absolute value) is returned. (default = FALSE)
return_wp	logical, if TRUE then the warping path is returned. (default = FALSE) If return_diffp == TRUE, then return_wp is set to TRUE as well.
return_diffp	logical, if TRUE then the path of differences (not the absolute value) is returned. (default = FALSE)
return_QC	logical, if TRUE then the input vectors Q and C are appended to the returned list. This is useful for the `plot.idtw` function. (default = FALSE)
x	output from `dtw` or `idtw`.
object	any R object
...	additional arguments, e.g. passed to `print`, `summary`, or a user defined distance function for `cm()`
digits	passed to `round` and `print`

Details

The dynamic time warping distance is the element in the last row and last column of the global cost matrix.

For the multivariate case where Q is a matrix of n rows and k columns and C is a matrix of m rows and k columns the dist_method parameter defines the following distance measures:

norm1:

$$dist(Q_{i,.}, C_{j,.}) = \sum l = 1 : k|Q_{i,l} - C_{j,l}|$$

norm2:

$$dist(Q_{i,.}, C_{j,.}) = (\sum l = 1 : k(Q_{i,l} - C_{j,l})^2)^0.5$$

norm2_square:

$$dist(Q_{i,.}, C_{j,.}) = \sum l = 1 : k(Q_{i,l} - C_{j,l})^2$$

The parameter `step_pattern` describes how the two time series are aligned. If `step_pattern == "symmetric1"` then

$$gcm_{i,j} = cmi, j + min(gcm_{i-1,j}, gcmi - 1, j - 1, gcmi, i - 1$$

.

If `step_pattern == "symmetric2"` then

$$gcm_{i,j} = cmi, j + min(gcm_{i-1,j}, cmi, j + gcmi - 1, j - 1, gcmi, i - 1$$

.

To make DTW distances comparable for many time series of different lengths use the `normlized_distance` with the setting `step_method = 'symmetric2'`. Please find a more detailed discussion and further references here: http://dtw.r-forge.r-project.org/.

User defined distance function: To calculate the DTW distance measure of two time series a distance function for the local distance of two observations `Q[i,]` and `C[j,]` of the time series `Q` and `C` has to be selected. The predefined distance function are 'norm1', 'norm2' and 'norm2-square'. It is also possible to define a customized distance function and use the cost matrix `cm` as input for the DTW algorithm, also for the incremental functions. In the following experiment we apply the cosine distance as local distance function:

$$d_cos(C_i, Q_j) = 1 - (\sum o = 1 : OQ_{io} * C_{jo})/((\sum o = 1 : OQ_{io}^2)^0.5 * (\sum o = 1 : OC_{jo}^2)^0.5).$$

Value

`distance`	the DTW distance, that is the element of the last row and last column of gcm
`normalized_distance`	the normalized DTW distance, that is the distance divided by `N+M`, where N and M are the lengths of the time series Q and C, respectively. If `step_pattern == 'symmetric1'` no normalization is performed and `NA` is returned (see details).
`gcm`	global cost matrix
`dm`	direction matrix (3=up, 1=diagonal, 2=left)

wp	warping path
ii	indices of Q of the optimal path
jj	indices of C of the optimal path
cm	Matrix of costs
diffM	Matrix of differences
diffp	path of differences
Q	input Q
C	input C

References

Sakoe, H.; Chiba, S., Dynamic programming algorithm optimization for spoken word recognition, Acoustics, Speech, and Signal Processing [see also IEEE Transactions on Signal Processing], IEEE Transactions on , vol.26, no.1, pp. 43-49, Feb 1978. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1163055

Examples

```
#--- univariate
Q <- cumsum(rnorm(100))
C <- Q[11:100] + rnorm(90, 0, 0.5)
tmp <- dtw(Q = Q, C = C, ws = 15, return_diffM = FALSE,
           return_QC = TRUE, return_wp = TRUE)
names(tmp)
print(tmp, digits = 3)
plot(tmp)
plot(tmp, type = "warp")


#--- compare different input variations
dtw_base  <- dtw(Q = Q, C = C, ws = 15, return_diffM = TRUE)
dtw_diffM <- dtw(Q = dtw_base$diffM, C = "diffM", ws = 15,
                 return_diffM = TRUE)
dtw_cm    <- dtw(Q = abs(dtw_base$diffM), C = "cm", ws = 15,
                 return_diffM = TRUE)

identical(dtw_base$gcm, dtw_cm$gcm)
identical(dtw_base$gcm, dtw_diffM$gcm)

# of course no diffM is returned in the 'cm'-case
dtw_cm$diffM


#--- multivariate case
Q <- matrix(rnorm(100), ncol=2)
C <- matrix(rnorm(80), ncol=2)
dtw(Q = Q, C = C, ws = 30, dist_method = "norm2")


#--- user defined distance function
```

```
# We define the distance function d_cos and use it as input for the cost matrix function cm.
# We can pass the output from cm() to dtw2vec(), and also to idtw2vec() for the
# incrermental calculation:

d_cos <- function(x, y){
  1 - sum(x * y)/(sqrt(sum(x^2)) * sqrt(sum(y^2)))
}

Q <- matrix(rnorm(100), ncol=5, nrow=20)
C <- matrix(rnorm(150), ncol=5, nrow=30)
cm1 <- cm(Q, C, dist_method = d_cos)
dtw2vec(Q = cm1, C = "cm")$distance

res0 <- idtw2vec(Q = cm1[ ,1:20], newObs =  "cm")
idtw2vec(Q = cm1[ ,21:30], newObs =  "cm", gcm_lc = res0$gcm_lc_new)$distance

# The DTW distances -- based on the customized distance function -- of the
# incremental calculation and the one from scratch are identical.
```

`dtw2vec` *Fast vector-based Dynamic Time Warping*

Description

Calculates the Dynamic Time Warping distance by hand of a vector-based implementation and is much faster than the traditional method `dtw()`. Also allows early abandoning and sakoe chiba warping window, both for univariate and multivariate time series.

Usage

```
dtw2vec(Q, C, dist_method = c("norm1", "norm2", "norm2_square"),
         step_pattern = c("symmetric2", "symmetric1"),
         ws = NULL, threshold = NULL)
```

Arguments

`Q`	Either Q is (a) a time series (vector or matrix for multivariate time series) or (b) Q is a cost matrix, so a matrix storing the local distances of the time series Q and C. If Q and C are matrices, they need to have the same number of columns. If Q is a cost matrix, C needs to be equal the character string "cm".
`C`	time series as vector or matrix, or for case (b) C equals "cm"
`dist_method`	character, describes the method of distance measure. See also `dtw`. If Q is a cost matrix, the dist_method parameter is not necessary.
`step_pattern`	character, describes the step pattern. See also `dtw`.
`ws`	integer, describes the window size for the sakoe chiba window. If NULL, then no window is applied. (default = NULL)

`threshold` numeric, the threshold for early abandoning. In the calculation of the global cost matrix a possible path stops as soon as the threshold is reached. Facilitates faster calculations in case of low threshold. The threshold relates to the non-normalized distance measure. (default = NULL, no early abandoning)

Details

no matrices are allocated, no matrices are returned

Value

`distance` the DTW distance

`normalized_distance`
the normalized DTW distance, see also `dtw`

References

Sakoe, H.; Chiba, S., Dynamic programming algorithm optimization for spoken word recognition, Acoustics, Speech, and Signal Processing [see also IEEE Transactions on Signal Processing], IEEE Transactions on , vol.26, no.1, pp. 43-49, Feb 1978. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1163055

Examples

```
Q <- cumsum(rnorm(100))
C <- Q[11:100] + rnorm(90, 0, 0.5)
dtw2vec(Q = Q, C = C)
dtw2vec(Q = Q, C = C, ws = 30)
dtw2vec(Q = Q, C = C, threshold = 100)
dtw2vec(Q = Q, C = C, ws = 30, threshold = 100)

cm0 <- cm(Q, C)
dtw2vec(Q = cm0, C = "cm", ws = 30, threshold = 100)
```

---

dtw_dismat *DTW Distance Matrix/ Distance Vector*

Description

Calculate a matrix of pairwise DTW distances for a set of univariate or multivariate time series. The output matrix (or dist object) of DTW distances can easily be applied for clustering the set of time series. Or calculate a vector of DTW distances of a set of time series all relative to one query time series. Parallel computations are possible.

Usage

```
dtw_dismat(lot, dist_method = c("norm1", "norm2", "norm2_square"),
         step_pattern = c("symmetric2", "symmetric1"), normalize = TRUE,
         ws = NULL, threshold = NULL,
         return_matrix = FALSE, ncores = NULL, useRcppParallel = TRUE)

dtw_disvec(Q, lot, dist_method = c("norm1", "norm2", "norm2_square"),
         step_pattern = c("symmetric2", "symmetric1"), normalize = TRUE,
         ws = NULL, threshold = NULL, ncores = NULL)
```

Arguments

Q	time series, vector (univariate) or matrix (multivariate)
lot	List of time series. Each entry of the list is a time series as described in dtw2vec.
dist_method	character, describes the method of distance measure. See also dtw.
step_pattern	character, describes the step pattern. See also dtw.
normalize	logical, whether to return normalized pairwise distances or not. If step_pattern == 'symmetric1' only non-normalized distances can be returned (default = TRUE)
ws	integer, describes the window size for the sakoe chiba window. If NULL, then no window is applied. (default = NULL)
threshold	numeric, the threshold for early abandoning. In the calculation of the global cost matrix a possible path stops as soon as the threshold is reached. Facilitates faster calculations in case of low threshold. (default = FALSE)
return_matrix	logical, If FALSE (default) the distance matrix is returned as dist object. If TRUE a symmetric matrix of differences is returned.
ncores	integer, number of cores to be used for parallel computation of the distance matrix. If ncores = NULL (default) then ncores is set to the number of available cores minus 1. If ncores = 0 then no parallel computation is performed and standard sapply instead of parallel::parSapply is applied.
useRcppParallel	logical, if the package RcppParallel (TRUE, default) or parallel (FALSE) is used for parallel computation

Details

By setting the parameter return_matrix = FALSE (default) the output value dismat of dtw_dismat is a dist object and can easily be passed to standard clustering functions (see examples).

No matrices are allocated for calculating the pairwise distances.

Value

input	the function input parameters
dismat	the matrix of pairwise DTW distances, either as matrix or dist object
disvec	the vector DTW distances

Examples

```
## Not run:

#--- Example for clustering a set of time series by feeding well known
# clustering methods with DTW-distance objects. First we simulate
# two prototype random walks and some cluster members. The cluster
# members are simulated by adding noise and randomly stretching and
# comressing the time series, to get time warped time series of
# varying lengths. The built clusters are 1:6 and 7:12.
set.seed(123)
N <- 100
rw_a <- cumsum(rnorm(N))
rw_b <- cumsum(rnorm(N))
sth <- sample(seq(0, 0.2, 0.01), size = 10)
cmp <- sample(seq(0, 0.2, 0.01), size = 10)
lot <- c(list(rw_a),
         lapply(1:5, function(i){
           simulate_timewarp(rw_a + rnorm(N), sth[i], cmp[i])
         }),
         list(rw_b),
         lapply(6:10, function(i){
           simulate_timewarp(rw_b + rnorm(N), sth[i], cmp[i])
         }))


# Next get the distance matrix, as dist object. Per default all
# minus 1 available cores are used:
result <- dtw_dismat(lot = lot, dist_method = "norm2", ws = 50,
                     return_matrix = FALSE)
class(result$dismat)


# Finally you can cluster the result with the following
#  well known methods:
require(cluster)
myclus <- hclust(result$dismat)
plot(myclus)
summary(myclus)

myclus <- agnes(result$dismat)
plot(myclus)
summary(myclus)

myclus <- pam(result$dismat, k=2)
plot(myclus)
summary(myclus)
myclus$medoids



## End(Not run)
```

dtw_partial *Partial Dynamic Time Warping*

Description

Get the cheapest partial open end alignment of two time series

Usage

```
dtw_partial(x, partial_Q = TRUE, partial_C = TRUE, reverse = FALSE)
```

Arguments

x	result object of either dtw() or idtw2vec()
partial_Q	logical (default = TRUE), whether Q is aligned completely to C or open ended.
partial_C	logical (default = TRUE), whether C is aligned completely to Q or open ended.
reverse	logical (default = FALSE), whether Q and C are in original or reverse order.

Details

Q is the time series that describes the vertical dimension of the global cost matrix, so length(Q) is equal to nrow(x$gcm). So C describes the horizontal dimension of the global cost matrix, length(C) is equal to ncol(x$gcm).

dtw_partial() returns the open-end alignment of Q and C with the minimal normalized distance. If partial_Q and partial_C both are TRUE the partial alignment with the smaller normalized distance is returned.

If Q and C are in reverse order, then the optimal solution for the reverse problem is found, that is the alignment with minimal normalized distance allowing and open-start alignment.

Value

rangeQ	Vector of initial and ending index for best alignment
rangeC	Vector of initial and ending index for best alignment
normalized_distance	the normalized DTW distance (see details in dtw).

Examples

```
#--- Open-end alignment for multivariate time series.
# First simulate a 2-dim time series Q
Q <- matrix(cumsum(rnorm(50 * 2)), ncol = 2)

# Then simulate C as warped version of Q,
C <- simulate_timewarp(Q, stretch = 0.2, compress = 0.2,
                       preserve_length = TRUE)
```

```
# add some noise
C <- C + rnorm(prod(dim(C)))

# and append noise at the end
C <- rbind(C, matrix(rnorm(30), ncol = 2))

tmp <- dtw(Q = Q, C = C, ws = 50, return_QC = TRUE, return_wp = TRUE)
par <- dtw_partial(tmp, partial_C = TRUE)
par
plot(tmp, partial = par, type = "QC")
plot(tmp, partial = par, type = "warp")
plot(tmp, partial = par, type = "QC", selDim = 2)


#--- Open-start is possible as well:
Q <- sin(1:100)
C <- c(rnorm(50), Q)
tmp <- dtw(Q = rev(Q), C = rev(C))
dtw_partial(tmp, reverse = TRUE)
```

`find_peaks` *find_peaks*

Description

Find negative or positive peaks of a vector in a predefined neighborhood w

Usage

```
find_peaks(x, w, get_min = TRUE, strict = TRUE)
```

Arguments

x	vector
w	window, at least w-many values need to be in-between two consecutive peaks to find both, otherwise only the bigger one is returned
get_min	logical (default TRUE) if TRUE, then minima are returned, else maxima
strict	logical, if TRUE (default) then a local minimum needs to be smaller then all neighbors. If FALSE, then a local minimum needs to be smaller or equal all neighbors.

Value

integer vector of indices where x has local extreme values

Examples

```
#--- Find the peaks (local minima and maxima),
# and also the border peak at index 29. First the local maxima:
x <- c(1:10, 9:1, 2:11)
peak_indices <- find_peaks(x, w=3, get_min=FALSE)
peak_indices
x[peak_indices]

# and now the local minima
peak_indices <- find_peaks(x, w=3, get_min=TRUE)
peak_indices
x[peak_indices]


#--- What exactly does the neigbohood parameter 'w' mean?
# At least w-many values need to be inbetween two consecutive peaks:
x <- -c(1:10, 9, 9, 11, 9:8, 7)
peak_indices <- find_peaks(x, w=3)
peak_indices
x[peak_indices]

x <- -c(1:10, 9, 9,9, 11, 9:8, 7)
peak_indices <- find_peaks(x, w=3)
peak_indices
x[peak_indices]


#--- What does the parameter 'strict' mean?
# If strict = TRUE, then the peak must be '<' (or '>')
# then the neighbors, other wise '<=' (or '>=')
x <- c(10:1,  1:10)
peak_indices <-  find_peaks(x, w=3, strict = TRUE)
peak_indices
x[peak_indices]

peak_indices <-  find_peaks(x, w=3, strict = FALSE)
peak_indices
x[peak_indices]
```

`idtw` *Incremental DTW*

Description

Update the DTW distance, cost matrices and direction matrices including the warping path for new observations of two time series.

Usage

```
idtw(Q, C, newObs, gcm, dm,
      dist_method = c("norm1", "norm2", "norm2_square"),
      step_pattern = c("symmetric2", "symmetric1"),
      diffM = NULL, ws = NULL,
      return_cm = FALSE,
      return_diffM = FALSE,
      return_wp = FALSE,
      return_diffp = FALSE,
      return_QC = FALSE)
```

Arguments

Q	numeric vector, or matrix (see also dtw)
C	numeric vector, or matrix
newObs	vector or matrix of new observations to be appended to C
gcm	global cost matrix, output from dtw(Q,C,...)
dm	direction matrix, output from dtw(Q,C,...)
dist_method	character, describes the method of distance measure. See also dtw.
step_pattern	character, describes the step pattern. See also dtw.
diffM	differences matrix, output from dtw(Q,C,...). This matrix is an optional input parameter (default = NULL) that is necessary to return the path of differences. Only for univariate time series Q and C.
ws	integer, describes the window size for the sakoe chiba window. If NULL, then no window is applied. (default = NULL)
return_cm	logical, if TRUE then the Matrix of costs (the absolute value) is returned. (default = FALSE)
return_diffM	logical, if TRUE then the Matrix of differences (not the absolute value) is returned. (default = FALSE)
return_wp	logical, if TRUE then the warping path is returned. (default = FALSE) If return_diffp == TRUE, then return_wp is set to TRUE as well.
return_diffp	logical, if TRUE then the path of differences (not the absolute value) is returned. (default = FALSE)
return_QC	logical, if TRUE then the input vectors Q and C are appended to the returned list. This is useful for the plot.idtw function. (default = FALSE)

Details

The dynamic time warping distance is the element in the last row and last column of the global cost matrix.

Value

distance	the DTW distance, that is the element of the last row and last column of gcm
gcm	global cost matrix
dm	direction matrix (3=up, 1=diagonal, 2=left)
wp	warping path
ii	indices of Q of the optimal path
jj	indices of C of the optimal path
cm	Matrix of costs
diffM	Matrix of differences
diffp	path of differences
Q	input Q
C	input C
normalized_distance	the normalized DTW distance, see also link{dtw}

References

Sakoe, H.; Chiba, S., Dynamic programming algorithm optimization for spoken word recognition, Acoustics, Speech, and Signal Processing [see also IEEE Transactions on Signal Processing], IEEE Transactions on , vol.26, no.1, pp. 43-49, Feb 1978. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1163055

Examples

```
#--- Compare the incremental calculation with the basic
# calculation from scratch.
Q <- cumsum(rnorm(100))
C <- Q[11:100] + rnorm(90, 0, 0.5)
newObs <-  c(2, 3)# new observation
base <- dtw(Q = Q, C = C, ws = 15, return_diffM = TRUE)
base

# recalculation from scratch with new observations
result0 <- dtw(Q = Q, C = c(C, newObs), ws = 15,  return_diffM = TRUE)

# the incremental step with new observations
result1 <- idtw(Q, C, ws = 15, newO = newObs, gcm = base$gcm,
                dm = base$dm, diffM = base$diffM, return_diffp = TRUE,
                return_diffM = TRUE, return_QC = TRUE)
print(result1, digits = 2)
plot(result1)


#--- Compare the incremental calculation with external calculated
# costMatrix cm_add with the basic calculation from scratch.
cm_add <- cm(Q, newObs)
result2 <- idtw(Q = cm_add, C = "cm_add", ws = 15, newO = newObs,
                gcm = base$gcm, dm = base$dm)
```

```
c(result0$distance, result1$distance, result2$distance)
```

---

idtw2vec *Incremental vector-based DTW*

Description

Update the DTW distance for new observations of two time series.

Usage

```
idtw2vec(Q, newObs, dist_method = c("norm1", "norm2", "norm2_square"),
         step_pattern = c("symmetric2", "symmetric1"),
         gcm_lc = NULL, gcm_lr = NULL, nC = NULL, ws = NULL)
```

Arguments

Q	Either Q is (a) a time series (vector or matrix for multivariate time series) or (b) Q is a cost matrix, so a matrix storing the local distances of the time series Q and newObs. If Q and newObs are matrices, they need to have the same number of columns. If Q is a cost matrix, see details...
newObs	time series as vector or matrix, or if Q is a cost matrix newObs must equals "cm". If newObs is a time series, see details...
dist_method	character, describes the method of distance measure. See also dtw.
step_pattern	character, describes the step pattern. See also dtw.
gcm_lc	vector, last column of global cost matrix of previous calculation. If NULL (necessary for the initial calculation), then DTW is calculated and the last column and last row are returned to start upcoming incremental calculations. (default = NULL)
gcm_lr	vector, last row of global cost matrix of previous calculation (default = NULL).
nC	integer, is the length of the original time series C, of which newObs are the new observations. Length of time series C exclusive new observations, such that length(c(C,newObs)) = nC + length(newObs). Necessary if ws is not NULL. (default = NULL)
ws	integer, describes the window size for the sakoe chiba window. If NULL, then no window is applied. (default = NULL)

Details

If new observations are recorded only for C and the only interest is a fast update of the DTW distance, the last row is not required, neither for the current nor for future incremental calculations.

If `Q` is a cost matrix, it needs to store either the distances of `Q` and new observations of `C` (running calculations, in that case gcm_lc != NULL), or it stores the distances of `Q` and the entire time series `C` (initial calculation, in that case gcm_lc = NULL).

If `newObs` is a time series, it stores either new Observations of `C` (running calculations) or the complete time series `C` (initial calculation).

no matrices are allocated, no matrices are returned

Value

`distance`	the DTW distance
`gcm_lc_new`	the last column of the new global cost matrix
`gcm_lr_new`	the last row of the new global cost matrix. Only if the input vector `gcm_lr` is not NUll and represents the last row of the previous global cost matrix, `gcm_lr_new` actually is the last row of the updated global cost matrix. Otherwise, if `gcm_lr` is NULL then `gcm_lr_new` is only the last row of the new part (concerning the new observations) of the global cost matrix.
`normalized_distance`	the normalized DTW distance, see also `dtw`

References

Sakoe, H.; Chiba, S., Dynamic programming algorithm optimization for spoken word recognition, Acoustics, Speech, and Signal Processing [see also IEEE Transactions on Signal Processing], IEEE Transactions on , vol.26, no.1, pp. 43-49, Feb 1978. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1163055

Examples

```
#--- Do the vector-based incremental DTW
# calculation and compare it with the basic
Q <- cumsum(rnorm(100))
C <- Q[11:100] + rnorm(90, 0, 0.5)

# initial calculation
res0 <- idtw2vec(Q = Q, newObs = C, gcm_lc = NULL)

# incremental calculation for new observations
nobs <- rnorm(10)
res1 <- idtw2vec(Q, newObs = nobs, gcm_lc = res0$gcm_lc_new)

# compare with result from scratch
res2 <- dtw2vec(Q, c(C, nobs))
res1$distance - res2$distance
```

```
#--- Perform an incremental DTW calculation with a
#  customized distance function.
d_cos <- function(x, y){
   1 - sum(x * y)/(sqrt(sum(x^2)) * sqrt(sum(y^2)))
}

x <- matrix(rnorm(100), ncol = 5, nrow = 20)
y <- matrix(rnorm(150), ncol = 5, nrow = 30)
cm1 <- cm(x, y, dist_method = d_cos)

# initial calculation
res0 <- idtw2vec(Q = cm(x, y[1:20,], dist_method = d_cos),
                 newObs =  "cm")

# incremental calculation for new observations
res1 <- idtw2vec(Q = cm(x, y[21:30,], d_cos), newObs =  "cm",
         gcm_lc = res0$gcm_lc_new)$distance

# compare with result from scratch
res2 <- dtw2vec(Q = cm1, C = "cm")$distance
res1 - res2
```

`initialize_plane` *Initialize and navigate in the plane of possible fits*

Description

Initialize and navigate in the plane of possible fits to detect subsequences (of different lengths) in a long time series that are similar (in terms of DTW distance) to a query pattern: Initialize the plane of possible fits as `.planedtw` object. Increment and decrement the time series observations and respective DTW calculation. Reverse the time order to increment or decrement observations at the other end of the time horizon. Refresh the DTW calculation without changing the time series.

Usage

```
initialize_plane(Q, C, dist_method = c("norm1", "norm2", "norm2_square"),
                 step_pattern = c("symmetric2", "symmetric1"), ws = NULL)


## S3 method for class 'planedtw'
increment(x, newObs, direction = c("C", "Q"), ...)

## S3 method for class 'planedtw'
decrement(x, direction = c("C", "Q", "both"),
          refresh_dtw = FALSE, nC = NULL, nQ = NULL, ...)

## S3 method for class 'planedtw'
```

```
refresh(x, ...)

## S3 method for class 'planedtw'
reverse(x, ...)

is.planedtw(x)
```

Arguments

Q	a time series (vector or matrix for multivariate time series)
C	a time series (vector or matrix for multivariate time series)
dist_method	character, describes the method of distance measure. See also dtw.
step_pattern	character, describes the step pattern. See also dtw.
ws	integer, describes the window size for the sakoe chiba window. If NULL, then no window is applied. (default = NULL)
x	object of class planedtw (output from initialize_plane)
newObs	a time series (vector or matrix for multivariate time series). If Q and C are vectors, newObs must be a vector. If Q and C are matrices with nc columns, then newObs must also have nc columns. See details for the correct time order of newObs.
direction	character, gives the direction of increment or decrement. decrement() is a wrapper for dtw_partial and the direction parameter is translated to the respective partial_Q and partial_C parameters.
refresh_dtw	logical (default = FALSE), after decrementing the time series, should the DTW calculation be refreshed, or not.
nC	integer, default = NULL, if not NULL, then decrement subsets the time series C to the range of 1:nC, drops invalid interim calculation results, and refreshes if refresh_dtw = TRUE.
nQ	analog to nC
...	additional arguments (currently not used)

Details

All functions are wrapper functions for idtw2vec and dtw_partial.

- initialize_plane calculates the DTW distance between Q and C and saves the last column and row of the global cost matrix. It returns an object of class planedtw that contains all necessary information to incrementally update the DTW calculation with new observations. Also for decrementing the calcultions for skipping some observations at the end.
- increment updates the DTW calculation by appending new observations to C or Q (depends on the parameter direction) and calculating DTW by recycling previous results represented by gcm_lc_new and gcm_lr_new. A wrapper for idtw2vec
- decrement is a wrapper for dtw_partial and also returns a planedtw object.
- refresh serves to recalculate the gcm_lc_new and gcm_lr_new from scratch, if these objects are NULL (e.g. after decrementing with refresh_dtw = FALSE).

- `reverse` reverses the order of Q and C, and refreshes the calculation for the new order. This is useful for appending observations to Q or C at the other end, the beginning. For incrementing in the reverse order also apply the function `increment`. Then the time series in the parameter `newObs` also needs to be in reverse order. Assent et al. (2009) proved that the DTW distance is reversible for the step pattern "symmetric1", so `dtw(Q,C)` = `dtw(rev(Q),rev(C))`. Also see examples. For the step pattern "symmetric2" DTW is not exactly reversible, but empirical studies showed that the difference is realtive small. For further details please see the appendix A of the vignette "IncDTW: An R Package for Incremental Calculation of Dynamic Time Warping" on CRAN.

Value

`distance`	the DTW distance
`normalized_distance`	the DTW distance devided by the sum of the lengths of Q and C (see also `dtw`).
`gcm_lc_new`	the last column of the new global cost matrix
`gcm_lr_new`	the last row of the new global cost matrix
`Q`	the time series
`C`	the time series
`control`	list of input parameters and the lengths of the time series

References

Assent, Ira, et al. "Anticipatory DTW for efficient similarity search in time series databases." Proceedings of the VLDB Endowment 2.1 (2009): 826-837.

Examples

```
## Not run:

#--- 1. example: Increment too far and take a step back:
rw <- function(nn) cumsum(rnorm(nn))
Q <- sin(1:100)
C <- Q[1:90] + rnorm(90, 0, 0.1)
WS <- 40


# start with the initial calculation
x <- initialize_plane(Q, C, ws = WS)

# Then the incremental calculation for new observations
y1 <- Q[91:95] + rnorm(5, 0, 0.1)# new observations
x <- increment(x, newObs = y1)

# Again new observations -> just increment x
y2 <- c(Q[96:100] + rnorm(5, 0, 0.1), rw(10))# new observations
x <- increment(x, newObs = y2)

# Compare the results with the calculation from scratch
```

```
from_scratch <- dtw2vec(Q, c(C, y1, y2) , ws = WS)$normalized_distance
x$normalized_distance - from_scratch
plot(x)

# The plot shows alignments of high costs at the end
# => attempt a decremtal step to find better partial matching
x <- decrement(x, direction = "C", refresh_dtw = TRUE)
x
plot(x)


#--- 2. example: First increment, then reverse increment
rw <- function(nn) cumsum(rnorm(nn))
Q <- rw(100)
C <- Q[11:90] + rnorm(80, 0, 0.1)
WS <- 40

# initial calculation
x <- initialize_plane(Q, C, ws = WS)
plot(x)

# incremental calculation for new observations that
# are appened at the end of C
y1 <- Q[91:100] + rnorm(10, 0, 0.1)
x <- increment(x, newObs = y1)

# reverse the order of Q and C
x <- reverse(x)

# append new observations at the beginning: the new
# obervations must be in the same order as Q and C
# => so newObs must be in reverse order, so y2 is
# defined as Q from 10 to 6 (plus noise).
y2 <- Q[10:6] + rnorm(5, 0, 0.1)
x <- increment(x, newObs = y2)

# another incremental step in the reverse direction
y3 <- Q[5:1] + rnorm(5, 0, 0.1)
x <- increment(x, newObs = y3)

# compare with calculations from scratch, and plot x
from_scratch <- dtw2vec(rev(Q), rev(c(rev(y3), rev(y2), C, y1)),
                        ws = WS)$distance
x$distance - from_scratch
print(x)
plot(x)


## End(Not run)
```

---

lowerbound *lowerbound*

Description

Calculate the lowerbound for the DTW distance measure in linear time.

Usage

```
lowerbound(C, ws, scale = c("z", "01", "none"),
           dist_method = c("norm1", "norm2", "norm2_square"),
           Q = NULL, tube = NULL)

lowerbound_tube(Q, ws, scale = c("z", "01", "none"))
```

Arguments

Q	vector or matrix, the query time series
C	vector or matrix, the query time series
dist_method	distance method, one of ("norm1", "norm2", "norm2_square")
scale	either "none", so no scaling is performed, or one of ("z", "01") to scale both Q and C. Also see dtw
ws	see dtw
tube	tube for lower bounding. "tube" can be the output from lowerbound_tube(). If tube = NULL, then Q must not be NULL, so that tube can be defined. If the tube is passed as argument to lowerbound(), then it is necessary that the scale parameter in the lowerbound() call is identical to the scaling method applied on Q before calculating the tube.

Details

Lower Bounding: The following methods are implemented:

- LB_Keogh for univariate time series (Keogh et al. 2005)
- LB_MV for multivariate time series with the dist_method = "norm2_square", (Rath et al. 2002)
- Adjusted for different distance methods "norm1" and "norm2", inspired by (Rath et al. 2002).

Value

lowerbound distance measure that is proven to be smaller than the DTW distance measure

References

- Keogh, Eamonn, and Chotirat Ann Ratanamahatana. "Exact indexing of dynamic time warping." Knowledge and information systems 7.3 (2005): 358-386.
- Rath, Toni M., and R. Manmatha. "Lower-bounding of dynamic time warping distances for multivariate time series." University of Massachusetts Amherst Technical Report MM 40 (2002).

- Sakurai, Yasushi, Christos Faloutsos, and Masashi Yamamuro. "Stream monitoring under the time warping distance." Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on. IEEE, 2007.

Examples

```
## Not run:

#--- Univariate time series Q and C
ws <- sample(2:40, size = 1)
dist_method <- "norm1"
N <- 50
N <- 50
Q <- cumsum(rnorm(N))
C <- cumsum(rnorm(N))
Q.z <- IncDTW::scale(Q, "z")
C.z <- IncDTW::scale(C, "z")

lb.z <- lowerbound(C = C.z, ws = ws, scale ="none", dist_method = dist_method, Q = Q.z)
lb <- lowerbound(C = C, ws = ws, scale ="z", dist_method = dist_method, Q = Q)
d1 <- dtw2vec(Q = Q.z, C = C.z, step_pattern = "symmetric1",
  dist_method = dist_method, ws = ws)$distance
d2 <- dtw2vec(Q = Q.z, C = C.z, step_pattern = "symmetric2",
  dist_method = dist_method, ws = ws)$distance

c(lb, lb.z, d1, d2)


#--- with pre-calculated tube
ws <- sample(2:40, size = 1)
dist_method <- "norm1"
N <- 50
N <- 50
Q <- cumsum(rnorm(N))
C <- cumsum(rnorm(N))
Q.z <- IncDTW::scale(Q, "z")
C.z <- IncDTW::scale(C, "z")

tube <- lowerbound_tube(Q, ws, scale = "z")

lb.z <- lowerbound(C = C.z, ws = ws, scale ="none", dist_method = dist_method, tube = tube)
lb <- lowerbound(C = C, ws = ws, scale ="z", dist_method = dist_method, tube = tube)
d1 <- dtw2vec(Q = Q.z, C = C.z, step_pattern = "symmetric1",
  dist_method = dist_method, ws = ws)$distance
d2 <- dtw2vec(Q = Q.z, C = C.z, step_pattern = "symmetric2",
  dist_method = dist_method, ws = ws)$distance

c(lb, lb.z, d1, d2)



#--- Multivariate time series Q and C
```

```
ws <- sample(2:40, size = 1)
dist_method <- sample(c("norm1", "norm2", "norm2_square"), size = 1)
N <- 50
Q <- matrix(cumsum(rnorm(N * 3)), ncol = 3)
C <- matrix(cumsum(rnorm(N * 3)), ncol = 3)
Q.z <- IncDTW::scale(Q, "z")
C.z <- IncDTW::scale(C, "z")

lb.z <- lowerbound(C = C.z, ws = ws, scale ="none", dist_method = dist_method, Q = Q.z)
lb <- lowerbound(C = C, ws = ws, scale ="z", dist_method = dist_method, Q = Q)
d1 <- dtw2vec(Q = Q.z, C = C.z, step_pattern = "symmetric1",
  dist_method = dist_method, ws = ws)$distance
d2 <- dtw2vec(Q = Q.z, C = C.z, step_pattern = "symmetric2",
  dist_method = dist_method, ws = ws)$distance

c(lb, lb.z, d1, d2)


## End(Not run)
```

`plot.dba` *Plot the results from Dynamic Time Warping Barycenter Averaging*

Description

Plot function for objects of type dba, the output of `dba()`.

Usage

```
## S3 method for class 'dba'
plot(x, type = c("barycenter", "m2m", "m2lot"), ...)
# an alias for plot_dba
plot_dba(x, type = c("barycenter", "m2m", "m2lot"), ...)

plotBary(x, ...)

plotM2m(x, ...)

plotM2lot(x, ...)
```

Arguments

x	output from `dba()`
type	character, one of c('barycenter', 'm2m', 'm2lot')
...	Other arguments passed on to methods. Currently not used.

Details

- 'barycenter' plots the iterations of the barycenter per dimension.
- 'm2m' plots the distances (distance method set by iter_dist_method, see dba) of one barycenter-iteration to the former iteration step.
- 'm2lot' plots the distances (if step_pattern == 'symmetric2' the normalized distances are plotted) of the barycenter to the list of time series per iteration.

See Also

dba

Examples

```
# see examples of dba()
```

---

plot.idtw *Plot the results from Dynamic Time Warping*

Description

Plot function for objects of type idtw, the output of dtw() and idtw() respectively.

Usage

```
## S3 method for class 'idtw'
plot(x, type = c("QC", "warp"), partial = NULL, selDim = 1, ...)

# an alias for plot_idtw
plot_idtw(x, type = c("QC", "warp"), partial = NULL, selDim = 1, ...)

## S3 method for class 'planedtw'
plot(x, type = c("QC", "warp"), partial = NULL, selDim = 1, ...)

# an alias for plot_planedtw
plot_planedtw(x, type = c("QC", "warp"), partial = NULL, selDim = 1, ...)

plotQC(x, Q, C, partial = NULL, selDim = 1, ...)

plotWarp(x, Q, C, partial = NULL, selDim = 1, ...)
```

Arguments

x	output from dtw(Q,C)
Q	one dimensional numeric vector
C	one dimensional numeric vector
type	character, one of c('QC', 'warp')

partial	list, the return value of dtw_partial(). Default = NULL, see dtw_partial() for details.
selDim	integer, gives the column index of the multivariate time series (matrices) to be plotted. (default = 1) If Q and C are univariate time series (vectors) then selDim is neglected.
...	Other arguments passed on to methods.

Details

The plot function visualizes the time warp and the alignment of the two time series. Also for partial alignments see dtw_partial()

Examples

```
Q <- cumsum(rnorm(100))
C <- Q[11:100] + rnorm(90, 0, 0.5)
tmp <- dtw(Q = Q, C = C, ws = 15, return_wp = TRUE, return_QC = TRUE)

plot(tmp, type = 'QC')
plotQC(tmp)
plot(tmp, type = 'warp')
plotWarp(tmp)
```

---

plot.rundtw *Plot*

Description

Plot the results from rundtw.

Usage

```
## S3 method for class 'rundtw'
plot(x, knn = TRUE, minima = TRUE,
     scale = c("none", "01", "z"),
    selDim = 1, lix = 1, Q = NULL, C = NULL, normalize = c("none", "01", "z"),  ...)
```

Arguments

x	output from rundtw
knn	logical, if TRUE (= default) and the k nearest neighbors were found by rundtw, then they are plotted. See details.

minima	logical, if TRUE (= default) and Q is either passed or also returned by rundtw then the local (with window size of the lengh of Q) minima of the vector of distances is plotted. See details.
scale	character, one of c("none", "01", "z"). If "01" or "z" then the detected minima and knn are normed and plotted.
selDim	integer vector, default = 1. Set the dimensions to be plotted for multivariate time series Q and C.
lix	list index, integer, default = 1. If C is a list of time series, set with lix the list entry of C to be plotted.
Q	time series, default = NULL, either passed as list entry of x (when the parameter return_QC of rundtw is set to TRUE) or passed manually. Necessary for plotting the minima.
C	time series, default = NULL, either passed as list entry of x (when the parameter return_QC or rundtw is set to TRUE) or passed manually. Necessary for plotting the minima and knn.
normalize	deprecated, use scale instead. If normalize is set, then scale is overwritten by normalize for compatibility.
...	additional arguments passed to ggplot()

Details

Only for those subsequences for which the calculations were finished by rundtw, the distances are plotted (see the parameters threshold, k and early_abandon of rundtw).

See Also

rundtw

Examples

```
#--- Simulate a query pattern Q and a longer time series C,
# and detect rescaled versions of Q in C
set.seed(123)
Q <- sin(seq(0, 2*pi, length.out = 20))
Q_rescaled <- Q * abs(rnorm(1)) + rnorm(1)
C <- c(rnorm(20), Q_rescaled , rnorm(20))

# Force rundtw to finish all calculations and plot the vector of DTW distances
ret <- rundtw(Q, C, threshold = NULL, lower_bound = FALSE)
ret
plot(ret)

# Allow early abandoning and lower bounding, and also plot C
ret <- rundtw(Q, C, return_QC = TRUE, ws = 5)
ret
plot(ret)

# Get 1 nearest neighbor -> allow early abandon and lower bounding,
```

```
# and plot C and also plot the scaled detected nearest neighbors
ret <- rundtw(Q, C, ws = 5, k = 1, return_QC = TRUE)
ret
plot(ret, scale = "01")


#--- See the help page of rundtw() for further examples.
```

---

rundtw *rundtw*

Description

Detect recurring patterns similar to given query pattern by measuring the distance with DTW. A window of the length of the query pattern slides along the longer time series and calculates computation-time-efficiently the DTW distance for each point of time. The function incrementally updates the scaling of the sliding window, incrementally updates the cost matrix, applies the vector-based implementation of the DTW algorithm, early abandons and applies lower bounding methods to decrease the calculation time.

Usage

```
rundtw(Q, C, dist_method = c("norm1", "norm2", "norm2_square"),
      step_pattern = c("symmetric1", "symmetric2"), k = NULL,
      scale = c("01", "z", "none"), ws = NULL, threshold = NULL,
      lower_bound = TRUE, overlap_tol = 0, return_QC = FALSE,
      normalize = c("01", "z", "none"))

## S3 method for class 'rundtw'
print(x, ...)

## S3 method for class 'rundtw'
summary(object, ...)

is.rundtw(x)
```

Arguments

Q	vector or matrix, the query time series
C	vector or matrix (equal number of columns as Q), the longer time series which is scanned for multiple fits of the query time series. C can also be a list of time series (either all vectors, or all matrices of equal number of columns) with varying lengths. See Details.
dist_method	see dtw
step_pattern	see dtw, only for "symmetric1" the lower bounding is implemented yet

k	integer >= 0. If k > 0, then the k-nearest neighbors to the query pattern that are found in all possible sub-sequences of the long time series C are returned. Per default the found fits don't overlap, except the overlap_tol parameter is adjusted (this should be done with care!). If k > 0 then lowerbound is set to TRUE.
scale	character, one of c("01", "z", "none") (default = "01"), if not "none" then Q (once at the start) and C (running scaling) are scaled. Either min-max ("01") or the z-scaling ("z") is applied. TRUE (identical to '01') and FALSE (identical to 'none') are deprecated and will be dropped in the next package version.
ws	see dtw
threshold	numeric >= 0, global threshold for early abandoning DTW calculation if this threshold is hit. (also see dtw). If NULL (default) no early abandoning is applied.
lower_bound	logical, (default = TRUE) If TRUE (default) then lower bounding is applied (see Details).
overlap_tol	integer between 0 and length of Q, (default = 0) gives the number of observations that two consecutive fits are accepted to overlap.
return_QC	logical, default = FALSE. If TRUE then Q and C are appended to the return list.
normalize	deprecated, use scale instead. If normalize is set, then scale is overwritten by normalize for compatibility.
x	the output object from rundtw.
object	any R object
...	further arguments passed to print or summary.

Details

This function and algorithm was inspired by the work of Sakurai et al. (2007) and refined for running min-max scaling and lower bounding.

Lower Bounding: The following methods are implemented:

- LB_Keogh for univariate time series (Keogh et al. 2005)
- LB_MV for multivariate time series with the dist_method = "norm2_square", (Rath et al. 2002)
- Adjusted for different distance methods "norm1" and "norm2", inspired by (Rath et al. 2002).

Counter vector:

- "scale_reset" counts how many times the min and max of the sliding window and the scaling need to be reset completely
- "scale_new_extreme" how many times the min or max of the sliding window are adjusted incrementally and the scaling need to be reset completely
- "scale_1step" how many times only the new observation in the sliding window needs to be scaled based on the current min and max
- "cm_reset" how many times the cost matrix for the sliding window needs to be recalculated completely

- "cm_1step" how many times only the front running column of the cost matrix is calculated
- "early_abandon" how many times the early abandon method aborts the DTW calculation before finishing
- "lower_bound" how many times the lower bounding stops the initialization of the DTW calculation
- "completed" for how many subsequences the DTW calculation finished

C is a list of time series: If C is a list of time series, the algorithm concatenates the list to one long time series to apply the logic of early abandoning, lower bounding, and finding the kNN. Finally the results are split to match the input. The

Value

`dist`	vector of DTW distances
`counter`	named vector of counters. Gives information how the algorithm proceeded. see Details
`knn_indices`	indices of the found kNN
`knn_values`	DTW distances of the found kNN
`knn_list_indices`	indices of list entries of C, where to find the kNN. Only returned if C is a list of time series. See examples.
`Q, C`	input time series

References

- Keogh, Eamonn, and Chotirat Ann Ratanamahatana. "Exact indexing of dynamic time warping." Knowledge and information systems 7.3 (2005): 358-386.
- Rath, Toni M., and R. Manmatha. "Lower-bounding of dynamic time warping distances for multivariate time series." University of Massachusetts Amherst Technical Report MM 40 (2002).
- Sakurai, Yasushi, Christos Faloutsos, and Masashi Yamamuro. "Stream monitoring under the time warping distance." Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on. IEEE, 2007.

Examples

```
## Not run:
#--- Simulate a query pattern Q and a longer time series C with
# distorted instances of Q within C. Apply rundtw() do detect
# these instances of C.

rw <- function(nr) cumsum(rnorm(nr))
noise <- function(nr) rnorm(nr)
set.seed(1234)
nC <- 500
nQ <- 40
nfits <- 5
```

```
nn <- nC - nfits * nQ # length of noise
nn <- nn/nfits + 1

Q <- sin(seq(from = 1, to = 4 * pi, length.out = nQ))
Qscale <- IncDTW::scale(Q, type = "01")
C <- rw(0)
for(i in 1:nfits){
   C <- c(C, rw(nn)  ,
          Q * abs(rnorm(1, 10, 10)) +
            rnorm(1, 0, 10) + noise(nQ))
}

# Apply running min-max scaling and allow lower
# bounding to find the 3 NN
x <- rundtw(Q, C, scale = '01', ws = 10, k = 3,
            lower_bound = TRUE, return_QC = TRUE)

# Have a look at the result and get the best indices
# with lowest distance.
x
summary(x)
find_peaks(x$dist, nQ)
plot(x, scale = "01")

# The fourth and fifth simuated fits are not returned,
# since the DTW distances are higher than the other found 3 NN.
# The algorithm early abandons and returns NA for these
# indices. Get all distances by the following command:
x_all <- rundtw(Q, C, scale = '01', ws = 10,
                k = 0, lower_bound = FALSE)
plot(x_all)

# Do min-max-scaling and lower bound
rundtw(Q, C, scale = '01', ws = 10, lower_bound = TRUE)

# Do z-scaling and lower bound
rundtw(Q, C, scale = 'z', ws = 10, lower_bound = TRUE)

# Don't scaling and don't lower bound
rundtw(Q, C, scale = 'none', ws = 10, lower_bound = FALSE)

# kNN: Do z-scaling and lower bound
rundtw(Q, C, scale = 'z', ws = 10, k = 3)



#--- For multivariate time series
rw <- function(nr, nco) {
        matrix(cumsum(rnorm(nr * nco)), nrow = nr, ncol = nco)
}

nC <- 500
```

```
nQ <- 50
nco <- 2
nfits <- 5

nn <- nC - nfits * nQ# length of noise
nn <- nn/nfits

Q <- rw(nQ, nco)
Qscale <- IncDTW::scale(Q, type="01")
C <- matrix(numeric(), ncol=nco)
for(i in 1:nfits){
   C <- rbind(C, rw(nn, nco), Q)
}

# Do min-max-scaling and lower bound
rundtw(Q, C, scale = '01', ws = 10, threshold = Inf,
       lower_bound = TRUE)

# Do z-scaling and lower bound
rundtw(Q, C, scale = 'z', ws = 10, threshold = NULL,
       lower_bound = TRUE)

# Don't scale and don't lower bound
rundtw(Q, C, scale = 'none', ws = 10, threshold = NULL,
       lower_bound = FALSE)



#--- C can also be a list of (multivariate) time series.
# So rundtw() detects the closest fits of a query pattern
# across all time series in C.
nC <- 500
nQ <- 50
nco <- 2
rw <- function(nr, nco){
        matrix(cumsum(rnorm(nr * nco)), nrow = nr, ncol = nco)
}

Q <- rw(nQ, nco)
C1 <- rbind(rw(100, nco), Q, rw(20, nco))
C2 <- rbind(rw(10, nco), Q, rw(50, nco))
C3 <- rbind(rw(200, nco), Q, rw(30, nco))
C_list <- list(C1, C2, C3)

# Do min-max-scaling and lower bound
x <- rundtw(Q, C_list, scale = '01', ws = 10, threshold = Inf,
            lower_bound = TRUE, k = 3, return_QC = TRUE)
x
# Plot the kNN fit of the 2nd or 3rd list entry of C
plot(x, lix = 2)
plot(x, lix = 3)

# Do z-scaling and lower bound
```

```
rundtw(Q, C_list, scale = 'z', ws = 10, threshold = Inf,
       lower_bound = TRUE, k = 3)

# Don't scale and don't lower bound
x <- rundtw(Q, C_list, scale = 'none', ws = 10,
            lower_bound = FALSE, k = 0, return_QC = TRUE)
x
plot(x)

## End(Not run)
```

`scale` *Time Series Scaling*

Description

scales a time series per dimension/column.

Usage

```
scale(x, type = c('z', '01'), threshold = 1e-5,
     xmean = NULL, xsd = NULL, xmin = NULL, xmax = NULL)

# deprecated
norm(x, type = c('z', '01'), threshold = 1e-5,
     xmean = NULL, xsd = NULL, xmin = NULL, xmax = NULL)
```

Arguments

`x`	time series as vector or matrix
`type`	character, describes the method how to scale (or normalize) the time series (per column if x is multivariate). The parameter type is either 'z' for z-scaling or '01' for max-min scaling.
`threshold`	double, defines the minimum value of the standard deviation, or difference of minimum and maximum. If this value is smaller than the threshold, then no scaling is performed, only shifting by the mean or minimum, respectively. Default value = `1e-5`.
`xmean`	mean used for z-scaling. See details.
`xsd`	standard deviation used for z-scaling. See details.
`xmin`	minimum used for 0-1 scaling. See details.
`xmax`	maximum used for 0-1 scaling. See details.

Details

For a vector `x` the z-scaling subtracts the mean and devides by the standard deviation: of `(x-mean(x))/sd(x)`. The min-max scaling performs `(x-min(x))/(max(x)-min(x))`.

The parameters `xmean`, `xsd`, `xmin`, can be set `xmax` or passed as `NULL` (= default value). If these values are NULL, they are calculated based on `x`.

Value

x the scaled vector or matrix

Examples

```
# min-max scaling or z-scaling for a vector
x <- cumsum(rnorm(100, 10, 5))
y <- scale(x, "01")
z <- scale(x, "z")
par(mfrow = c(1, 3))
plot(x, type="l")
plot(y, type="l")
plot(z, type="l")
par(mfrow = c(1, 1))


# columnwise for matrices
x <- matrix(c(1:10, sin(1:10)), ncol = 2)
y <- scale(x, "01")
z <- scale(x, "z")
par(mfrow = c(1, 3))
matplot(x, type="l")
matplot(y, type="l")
matplot(z, type="l")
par(mfrow = c(1, 1))


# IncDTW::scale() and base::scale() perform same z-scaling
x <- cumsum(rnorm(100))
xi <- IncDTW::scale(x, type = "z")
xb <- base::scale(x, TRUE, TRUE)
sum(abs(xi-xb))
```

---

simulate_timewarp *Simulate time warp*

Description

Simulate a time warp for a given time series.

Usage

```
simulate_timewarp(x, stretch = 0, compress = 0,
                  stretch_method = insert_linear_interp,
                  p_index = "rnorm", p_number = "rlnorm",
                  p_index_list = NULL, p_number_list = NULL,
                  preserve_length = FALSE, seed = NULL, ...)
```

```
insert_const(x, ix, N, const = NULL)

insert_linear_interp(x, ix, N)

insert_norm(x, ix, N, mean = 0, sd = 1)

insert_linear_norm(x, ix, N, mean = 0, sd = 1)
```

Arguments

x	time series, vector or matrix
stretch	numeric parameter for stretching the time series. stretch >= 0, see details
compress	numeric parameter for compressing the time series. compress >= 0 & compress < 1, see details
stretch_method	function, either one of (insert_const, insert_linear_interp, insert_norm, insert_linear_norm), or any user defined function that needs the parameters x (univariate time series as vector), ix (index where to insert), N (number of observations to insert) and any other arguments required for that function. See Details.
p_index	string, distribution for simulating the indices where to insert simulated observations, e.g. "rnorm", "runif", etc.
p_number	string, distribution for simulating the number of observations to insert per index, e.g. "rnorm", "runif", etc.
p_index_list	list of named parameters for the distribution p_index
p_number_list	list of named parameters for the distribution p_number
preserve_length	logical, if TRUE (default = FALSE) then the length of the return time series is the same as before the warping, so the compression and stretching do not change the length of the time series, nevertheless perform local warpings
seed	set a seed for reproducible results
...	named parameters for the stretch_method
ix	index of x where after which to insert
N	number of simulated observations to insert at index ix
const	the constant to be inserted, if NULL (default), then const <-x[ix]
mean	mean for rnorm
sd	sd for rnorm

Details

The different distributions p_index and p_number also determine the behavior of the warp. A uniform distribution for p_number more likely draws high number than e.g. log-normal distributions with appropriate parameters. So, a uniform distribution more likely simulates fewer, but longer warps, that is points of time where the algorithm inserts simulations.

The algorithm stretches by randomly selecting an index between 1 and the length of the time series. Then a number of observations to be inserted is drawn out of the range 1 to the remaining number of observations to be inserted. These observations are inserted. Then the algorithm starts again with drawing an index, drawing a number of observations to be inserted, and proceeds until the requested time series length is achieved.

The algorithm for compressing works analogous, except it simply omits observations instead of linear interpolation.

The parameter `stretch` describes the ratio how much the time series `x` is stretched: e.g. if `compress = 0` and ...

- `stretch = 0` then `length(x_new) = length(x)`, or
- `stretch = 0.1` then `length(x_new) = length(x) * 1.1`, or
- `stretch = 1` then `length(x_new) = length(x) * 2`

The parameter `compress` describes the ratio how much the time series `x` is compressed: e.g. if `stretch = 0` and ...

- `compress = 0` then `length(x_new) = length(x)`, or
- `compress = 0.2` then `length(x_new) = length(x) * 0.8`

There are four functions to chose from to insert new simulated observations. You can also define your own function and apply this one. The four functions to chose from are:

- insert a constant, either a constant defined by the user via the input parameter `const`, or if `const = NULL`, then the last observation of the time series where the insertion starts is set as `const`
- insert linear interpolated observations (default)
- insert a constant with gaussian noise
- insert linear interpolated observations and add gaussian noise.

For the methods with Gaussian noise the parameters `mean` and `sd` for `rnorm` can be set at the function call of `simulate_timewarp()`.

Value

time warped time series

Examples

```
## Not run:
#--- Simulate a time warped version of a time series x
set.seed(123)
x <- cumsum(rnorm(100))
x_warp <- simulate_timewarp(x, stretch = 0.1, compress = 0.2, seed = 123)
plot(x, type = "l")
lines(x_warp, col = "red")


#--- Simulate a time warp of a multivariate time series
```

```
y <- matrix(cumsum(rnorm(10^3)), ncol = 2)
y_warp <- simulate_timewarp(y, stretch = 0.1, compress = 0.2, seed = 123)
plot(y[,1], type = "l")
lines(y_warp[,1], col = "red")


#--- Stretchings means to insert at new values at randomly
# selected points of time. Next the new values are set as constant NA,
# and the points of time simulated uniformly:
y_warp <- simulate_timewarp(y, stretch = 0.2, p_number = "runif", p_index = "runif",
                            stretch_method = insert_const,
                            const = NA)
matplot(y_warp, type = "l")


# insert NA and simulate the points of time by log normal
y_warp <- simulate_timewarp(y, stretch = 0.2, p_number = "rlnorm",
                            p_number_list = list(meanlog = 0, sdlog = 1),
                            stretch_method = insert_const,
                            const = NA)
matplot(y_warp, type = "l")


# insert linear interpolation
y_warp <- simulate_timewarp(y, stretch = 0.2, p_number = "rlnorm",
                            stretch_method = insert_linear_interp)
matplot(y_warp, type = "l")


# insert random walk with gaussian noise
y_warp <- simulate_timewarp(y, stretch = 0.2, p_number = "rlnorm",
                            stretch_method = insert_norm,
                            sd = 1, mean = 0)
matplot(y_warp, type = "l")


# insert constant, only 1 observation per random index
y_warp <- simulate_timewarp(y, stretch = 0.2, p_number = "runif", p_index = "runif",
                            p_number_list = list(min = 1, max = 1),
                            stretch_method = insert_const)
matplot(y_warp, type = "l")


# insert by customized insert function
my_stretch_method <- function(x, ix, N, from, to){
   c(x[1:ix],
     sin(seq(from = from, to = to, length.out = N)) + x[ix],
     x[(ix + 1):length(x)])
}
y_warp <- simulate_timewarp(y, stretch = 0.5, p_number = "rlnorm",
                            stretch_method = my_stretch_method,
                            from = 0, to = 4 * pi)
matplot(y_warp, type = "l")
```

```
## End(Not run)
```

Index