

Package ‘dtw’

May 18, 2018

Type Package

Title Dynamic Time Warping Algorithms

Description

A comprehensive implementation of dynamic time warping (DTW) algorithms in R. DTW computes the optimal (least cumulative distance) alignment between points of two time series. Common DTW variants covered include local (slope) and global (window) constraints, subsequence matches, arbitrary distance definitions, normalizations, minimum variance matching, and so on. Provides cumulative distances, alignments, specialized plot styles, etc.

Author Toni Giorgino <toni.giorgino@gmail.com>,

Maintainer Toni Giorgino <toni.giorgino@gmail.com>

Version 1.20-1

Date 2018-5-17

Depends R (>= 2.10.0), proxy

Imports graphics, grDevices, stats, utils

License GPL (>= 2)

URL <http://dtw.r-forge.r-project.org/>

NeedsCompilation yes

Repository CRAN

Date/Publication 2018-05-18 05:47:54 UTC

R topics documented:

dtw-package	2
aami	4
countPaths	6
dtw	7
dtwDist	12
dtwPlot	14
dtwPlotThreeWay	17
dtwPlotTwoWay	19
dtwWindowingFunctions	21

mvm	23
stepPattern	24
triangleFixing	29
warp	30
warpArea	32

Index	34
--------------	-----------

dtw-package	<i>Dynamic Time Warping Algorithms</i>
-------------	--

Description

A comprehensive implementation of dynamic time warping (DTW) algorithms in R. DTW computes the optimal (least cumulative distance) alignment between points of two time series. Common DTW variants covered include local (slope) and global (window) constraints, subsequence matches, arbitrary distance definitions, normalizations, minimum variance matching, and so on. Provides cumulative distances, alignments, specialized plot styles, etc.

Details

The DESCRIPTION file:

```

Package:      dtw
Type:         Package
Title:        Dynamic Time Warping Algorithms
Description:  A comprehensive implementation of dynamic time warping (DTW) algorithms in R. DTW computes the optim
Author:       Toni Giorgino <toni.giorgino@gmail.com>,
Maintainer:  Toni Giorgino <toni.giorgino@gmail.com>
Version:      1.20-1
Date:         2018-5-17
Depends:      R (>= 2.10.0), proxy
Imports:      graphics, grDevices, stats, utils
License:      GPL (>=2)
URL:          http://dtw.r-forge.r-project.org/

```

Index of help topics:

aami3a	ANSI/AAMI EC13 Test Waveforms, 3a and 3b
countPaths	Count the number of possible warping paths
dtw	Dynamic Time Warp
dtw-package	Dynamic Time Warping Algorithms
dtwDist	Compute a dissimilarity matrix
dtwPlot	Plotting of dynamic time warp results
dtwPlotThreeWay	Plotting of dynamic time warp results: annotated warping function
dtwPlotTwoWay	Plotting of dynamic time warp results:

	pointwise comparison
dtwWindowingFunctions	Global constraints and windowing functions for DTW
mvm	Minimum Variance Matching algorithm
stepPattern	Step patterns for DTW
triangleFixing	Triangle Fixing algorithm for Metric Nearness
warp	Apply a warping to a given timeseries
warpArea	Compute Warping Path Area

Comprehensive implementation of Dynamic Time Warping (DTW) algorithms in R.

The DTW algorithm computes the stretch of the time axis which optimally maps one given time-series (query) onto whole or part of another (reference). It yields the remaining cumulative distance after the alignment and the point-by-point correspondence (warping function). DTW is widely used e.g. for classification and clustering tasks in econometrics, chemometrics and general timeseries mining.

Please see documentation for function `dtw`, which is the main entry point to the package.

The R implementation in `dtw` provides:

- arbitrary windowing functions (global constraints), eg. the Sakoe-Chiba band; see [dtwWindowingFunctions](#)
- arbitrary transition types (also known as step patterns, slope constraints, local constraints, or DP-recursion rules). This includes dozens of well-known types; see [stepPattern](#):
 - all step patterns classified by Rabiner-Juang, Sakoe-Chiba, and Rabiner-Myers;
 - symmetric and asymmetric;
 - Rabiner’s smoothed variants;
 - arbitrary, user-defined slope constraints
- partial matches: open-begin, open-end, substring matches
- proper, pattern-dependent, normalization (exact average distance per step)
- the Minimum Variance Matching (MVM) algorithm (Latecki et al.)

Multivariate timeseries can be aligned with arbitrary local distance definitions, leveraging the `dist` function of package **proxy**. DTW itself becomes a distance function with the `dist` semantics.

In addition to computing alignments, the package provides:

- methods for plotting alignments and warping functions in several classic styles (see [plot gallery](#));
- graphical representation of step patterns;
- functions for applying a warping function, either direct or inverse; and more.

If you use this software, please cite it according to `citation("dtw")`. The package home page is at <http://dtw.r-forge.r-project.org>.

Author(s)

Toni Giorgino <toni.giorgino@gmail.com>,

Maintainer: Toni Giorgino <toni.giorgino@gmail.com>

References

Toni Giorgino. *Computing and Visualizing Dynamic Time Warping Alignments in R: The dtw Package*. Journal of Statistical Software, 31(7), 1-24. <http://www.jstatsoft.org/v31/i07/>

Tormene, P.; Giorgino, T.; Quaglini, S. & Stefanelli, M. *Matching incomplete time series with dynamic time warping: an algorithm and an application to post-stroke rehabilitation*. Artif Intell Med, 2009, 45, 11-34

Rabiner, L. R., & Juang, B.-H. (1993). Chapter 4 in *Fundamentals of speech recognition*. Englewood Cliffs, NJ: Prentice Hall.

See Also

`dtw` for the main entry point to the package; `dtwWindowingFunctions` for global constraints; `stepPattern` for local constraints; `distance`, `outer` for building a local cost matrix with multivariate timeseries and custom distance functions.

Examples

```
library(dtw);  
## demo(dtw);
```

aami

ANSI/AAMI EC13 Test Waveforms, 3a and 3b

Description

ANSI/AAMI EC13 Test Waveforms 3a and 3b, as obtained from the PhysioBank database.

Usage

```
data(aami3a);  
data(aami3b);
```

Format

```
> str(aami3a) Time-Series [1:43081] from 0 to 59.8: 0.185 0.185 0.169 0.185 0.185 0.185 0.185  
0.185 0.185 0.2 ...
```

```
> str(aami3b) Time-Series [1:43142] from 0 to 59.9: 0.192 0.192 0.192 0.192 0.192 0.2 0.2 0.192  
0.2 0.2 ...
```

Details

The following text is reproduced (abridged) from PhysioBank, page <http://www.physionet.org/physiobank/database/aami-ec13/>. Other recordings belong to the dataset and can be obtained from the same page.

The files in this set can be used for testing a variety of devices that monitor the electrocardiogram. The recordings include both synthetic and real waveforms. For details on these test waveforms and how to use them, please refer to section 5.1.2.1, paragraphs (e) and (g) in the reference below. Each recording contains one ECG signal sampled at 720 Hz with 12-bit resolution.

Note

Timestamps in the datasets have been re-created at the indicated frequency of 720 Hz, whereas the original timestamps in ms (at least in text format) only had three decimal digits' precision, and were therefore affected by substantial jittering.

Source

Cardiac monitors, heart rate meters, and alarms [American National Standard (ANSI/AAMI EC13:2002)]. Arlington, VA: Association for the Advancement of Medical Instrumentation, 2002

References

Goldberger AL, Amaral LAN, Glass L, Hausdorff JM, Ivanov PCh, Mark RG, Mietus JE, Moody GB, Peng CK, Stanley HE. *PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals*. *Circulation* 101(23):e215-e220 [Circulation Electronic Pages; <http://circ.ahajournals.org/cgi/content/full/101/23/e215>]; 2000 (June 13).

Examples

```
data(aami3a);
data(aami3b);

## Plot both as a multivariate TS object
## only extract the first 10 seconds

plot( main="ECG (mV)",
      window(
        cbind(aami3a,aami3b) ,end=10)
    )
```

countPaths	<i>Count the number of possible warping paths</i>
------------	---

Description

Count the number of warping paths compatible with the constraints.

Usage

```
countPaths(d, debug=FALSE)
```

Arguments

d	an object of class dtw
debug	return an intermediate result

Details

Count how many possible warping paths exist in the alignment problem passed as an argument. The object passed as an argument is used to look up the problem parameters such as the used step pattern, windowing, open ends, and so on. The actual alignment is ignored.

Note that the number of paths grows exponentially with problems size. The result may be approximate when windowing functions are used.

If debug is TRUE, a matrix used for the computation is returned instead of the final result.

Value

The number of paths.

Author(s)

Toni Giorgino

Examples

```
ds<-dtw(1:7+2,1:8,keep=TRUE,step=asymmetric);  
countPaths(ds)  
## Result: 126
```

dtw

Dynamic Time Warp

Description

Compute Dynamic Time Warp and find optimal alignment between two time series.

Usage

```
dtw(x, y=NULL,
    dist.method="Euclidean",
    step.pattern=symmetric2,
    window.type="none",
    keep.internals=FALSE,
    distance.only=FALSE,
    open.end=FALSE,
    open.begin=FALSE,
    ... )
```

```
is.dtw(d)
## S3 method for class 'dtw'
print(x,...)
```

Arguments

<code>x</code>	query vector <i>or</i> local cost matrix
<code>y</code>	reference vector, unused if <code>x</code> given as cost matrix
<code>dist.method</code>	pointwise (local) distance function to use. See dist in package proxy
<code>step.pattern</code>	a <code>stepPattern</code> object describing the local warping steps allowed with their cost (see stepPattern)
<code>window.type</code>	windowing function. Character: "none", "itakura", "sakoechiba", "slantedband", or a function (see details).
<code>open.begin</code> , <code>open.end</code>	perform open-ended alignments
<code>keep.internals</code>	preserve the cumulative cost matrix, inputs, and other internal structures
<code>distance.only</code>	only compute distance (no backtrack, faster)
<code>d</code>	an arbitrary R object
<code>...</code>	additional arguments, passed to <code>window.type</code>

Details

The function performs Dynamic Time Warp (DTW) and computes the optimal alignment between two time series x and y , given as numeric vectors. The “optimal” alignment minimizes the sum of distances between aligned elements. Lengths of x and y may differ.

The local distance between elements of x (query) and y (reference) can be computed in one of the following ways:

1. if `dist.method` is a string, x and y are passed to the `dist` function in package **proxy** with the method given;
2. if `dist.method` is a function of two arguments, it invoked repeatedly on all pairs $x[i], y[j]$ to build the local cost matrix;
3. multivariate time series and arbitrary distance metrics can be handled by supplying a local-distance matrix. Element $[i, j]$ of the local-distance matrix is understood as the distance between element $x[i]$ and $y[j]$. The distance matrix has therefore $n=\text{length}(x)$ rows and $m=\text{length}(y)$ columns (see note below).

Several common variants of the DTW recursion are supported via the `step.pattern` argument, which defaults to `symmetric2`. Step patterns are commonly used to *locally* constrain the slope of the alignment function. See [stepPattern](#) for details.

Windowing enforces a *global* constraint on the envelope of the warping path. It is selected by passing a string or function to the `window.type` argument. Commonly used windows are (abbreviations allowed):

- `"none"` No windowing (default)
- `"sakoechiba"` A band around main diagonal
- `"slantedband"` A band around slanted diagonal
- `"itakura"` So-called Itakura parallelogram

`window.type` can also be an user-defined windowing function. See [dtwWindowingFunctions](#) for all available windowing functions, details on user-defined windowing, and a discussion of the (mis)naming of the "Itakura" parallelogram as a global constraint. Some windowing functions may require parameters, such as the `window.size` argument.

Open-ended alignment, i.e. semi-unconstrained alignment, can be selected via the `open.end` switch. Open-end DTW computes the alignment which best matches all of the query with a *leading part* of the reference. This is proposed e.g. by Mori (2006), Sakoe (1979) and others. Similarly, open-begin is enabled via `open.begin`; it makes sense when `open.end` is also enabled (subsequence finding). Subsequence alignments are similar e.g. to UE2-1 algorithm by Rabiner (1978) and others. Please find a review in Tormene et al. (2009).

If the warping function is not required, computation can be sped up enabling the `distance.only=TRUE` switch, which skips the backtracking step. The output object will then lack the `index{1,2,1s,2s}` and `stepsTaken` fields.

`is.dtw` tests whether the argument is of class `dtw`.

Value

An object of class `dtw` with the following items:

<code>distance</code>	the minimum global distance computed, <i>not</i> normalized.
<code>normalizedDistance</code>	distance computed, <i>normalized</i> for path length, if normalization is known for chosen step pattern.
<code>N,M</code>	query and reference length
<code>call</code>	the function call that created the object
<code>index1</code>	matched elements: indices in x
<code>index2</code>	corresponding mapped indices in y
<code>stepPattern</code>	the <code>stepPattern</code> object used for the computation
<code>jmin</code>	last element of reference matched, if <code>open.end=TRUE</code>
<code>directionMatrix</code>	if <code>keep.internals=TRUE</code> , the directions of steps that would be taken at each alignment pair (integers indexing production rules in the chosen step pattern)
<code>stepsTaken</code>	the list of steps taken from the beginning to the end of the alignment (integers indexing chosen step pattern)
<code>index1s, index2s</code>	same as <code>index1/2</code> , excluding intermediate steps for multi-step patterns like asymmetricP05
<code>costMatrix</code>	if <code>keep.internals=TRUE</code> , the cumulative cost matrix
<code>query, reference</code>	if <code>keep.internals=TRUE</code> and passed as the x and y arguments, the query and reference timeseries.

Note

Cost matrices (both input and output) have query elements arranged row-wise (first index), and reference elements column-wise (second index). They print according to the usual convention, with indexes increasing down- and rightwards. Many DTW papers and tutorials show matrices according to plot-like conventions, i.e. reference index growing upwards. This may be confusing.

A fast compiled version of the function is normally used. Should it be unavailable, the interpreted equivalent will be used as a fall-back with a warning.

Author(s)

Toni Giorgino

References

Toni Giorgino. *Computing and Visualizing Dynamic Time Warping Alignments in R: The dtw Package*. Journal of Statistical Software, 31(7), 1-24. <http://www.jstatsoft.org/v31/i07/>

Tormene, P.; Giorgino, T.; Quaglini, S. & Stefanelli, M. *Matching incomplete time series with*

dynamic time warping: an algorithm and an application to post-stroke rehabilitation. Artif Intell Med, 2009, 45, 11-34. <http://dx.doi.org/10.1016/j.artmed.2008.11.007>

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

Mori, A.; Uchida, S.; Kurazume, R.; Taniguchi, R.; Hasegawa, T. & Sakoe, H. *Early Recognition and Prediction of Gestures* Proc. 18th International Conference on Pattern Recognition ICPR 2006, 2006, 3, 560-563

Sakoe, H. *Two-level DP-matching—A dynamic programming-based pattern matching algorithm for connected word recognition* Acoustics, Speech, and Signal Processing [see also IEEE Transactions on Signal Processing], IEEE Transactions on, 1979, 27, 588-595

Rabiner L, Rosenberg A, Levinson S (1978). *Considerations in dynamic time warping algorithms for discrete word recognition*. IEEE Trans. Acoust., Speech, Signal Process., 26(6), 575-582. ISSN 0096-3518.

Muller M. *Dynamic Time Warping in Information Retrieval for Music and Motion*. Springer Berlin Heidelberg; 2007. p. 69-84. http://link.springer.com/chapter/10.1007/978-3-540-74048-3_4

See Also

`dtwDist`, for iterating dtw over a set of timeseries; `dtwWindowingFunctions`, for windowing and global constraints; `stepPattern`, step patterns and local constraints; `plot.dtw`, plot methods for DTW objects. To generate a local distance matrix, the functions `dist` in package `proxy`, `distance` in package `analogue`, `outer` may come handy.

Examples

```
## A noisy sine wave as query
idx<-seq(0,6.28,len=100);
query<-sin(idx)+runif(100)/10;

## A cosine is for reference; sin and cos are offset by 25 samples
reference<-cos(idx)
plot(reference); lines(query,col="blue");

## Find the best match
alignment<-dtw(query,reference);

## Display the mapping, AKA warping function - may be multiple-valued
## Equivalent to: plot(alignment,type="alignment")
plot(alignment$index1,alignment$index2,main="Warping function");
```

```

## Confirm: 25 samples off-diagonal alignment
lines(1:100-25,col="red")

#####
##
## Partial alignments are allowed.
##

alignment0BE <-
  dtw(query[44:88],reference,
      keep=TRUE,step=asymmetric,
      open.end=TRUE,open.begin=TRUE);
plot(alignment0BE,type="two",off=1);

#####
##
## Subsetting allows warping and unwarping of
## timeseries according to the warping curve.
## See first example below.
##

## Most useful: plot the warped query along with reference
plot(reference)
lines(query[alignment$index1]~alignment$index2,col="blue")

## Plot the (unwarped) query and the inverse-warped reference
plot(query,type="l",col="blue")
points(reference[alignment$index2]~alignment$index1)

#####
##
## Contour plots of the cumulative cost matrix
##   similar to: plot(alignment,type="density") or
##               dtwPlotDensity(alignment)
## See more plots in ?plot.dtw
##

## keep = TRUE so we can look into the cost matrix

alignment<-dtw(query,reference,keep=TRUE);

contour(alignment$costMatrix,col=terrain.colors(100),x=1:100,y=1:100,
  xlab="Query (noisy sine)",ylab="Reference (cosine)");

lines(alignment$index1,alignment$index2,col="red",lwd=2);

```

```
#####
##
## An hand-checkable example
##

ldist<-matrix(1,nrow=6,ncol=6); # Matrix of ones
ldist[2,]<-0; ldist[,5]<-0;      # Mark a clear path of zeroes
ldist[2,5]<-0.01; # Forcely cut the corner

ds<-dtw(ldist); # DTW with user-supplied local
                    # cost matrix
da<-dtw(ldist,step=asymmetric); # Also compute the asymmetric
plot(ds$index1,ds$index2,pch=3); # Symmetric: alignment follows
                    # the low-distance marked path
points(da$index1,da$index2,col="red"); # Asymmetric: visiting
                    # 1 is required twice

ds$distance;
da$distance;
```

dtwDist

Compute a dissimilarity matrix

Description

Compute the dissimilarity matrix between a set of single-variate timeseries.

Usage

```
dtwDist(mx,my=mx,...)
# dist(mx,my=mx,method="DTW",...)
```

Arguments

mx	numeric matrix, containing timeseries as rows
my	numeric matrix, containing timeseries as rows (for cross-distance)
...	arguments passed to the <code>dtw</code> call

Details

dtwDist computes a dissimilarity matrix, akin to [dist](#), based on the Dynamic Time Warping definition of a distance between single-variate timeseries.

The dtwDist command is a synonym for the [dist](#) function of package **proxy**; the DTW distance is registered as method="DTW" (see examples below).

The timeseries are stored as rows in the matrix argument *m*. In other words, if *m* is an $N * T$ matrix, dtwDist will build $N*N$ ordered pairs of timeseries, perform the corresponding $N*N$ dtw alignments, and return all of the results in a matrix. Each of the timeseries is *T* elements long.

dtwDist returns a square matrix, whereas the [dist](#) object is lower-triangular. This makes sense because in general the DTW "distance" is not symmetric (see e.g. asymmetric step patterns). To make a square matrix with the [dist](#) function semantics, use the two-arguments call as `dist(m,m)`. This will return a square `crossdist` object.

Value

A square matrix whose element `[i,j]` holds the Dynamic Time Warp distance between row *i* (query) and *j* (reference) of *mx* and *my*, i.e. `dtw(mx[i,],my[j,])$distance`.

Note

To convert a square cross-distance matrix (`crossdist` object) to a symmetric [dist](#) object, use a suitable conversion strategy (see examples).

Author(s)

Toni Giorgino

See Also

Other "distance" functions are: [dist](#), [vegdist](#) in package `vegan`, [distance](#) in package `analogue`, etc.

Examples

```
## Symmetric step pattern => symmetric dissimilarity matrix;
## no problem coercing it to a dist object:

m <- matrix(0,ncol=3,nrow=4)
m <- row(m)
dist(m,method="DTW");

# Old-fashioned call style would be:
# dtwDist(m)
# as.dist(dtwDist(m))

## Find the optimal warping _and_ scale factor at the same time.
```

```

## (There may be a better, analytic way)

# Prepare a query and a reference

query<-sin(seq(0,4*pi,len=100))
reference<-cos(seq(0,4*pi,len=100))

# Make a set of several references, scaled from 0 to 3 in .1 increments.
# Put them in a matrix, in rows

scaleSet <- seq(0.1,3,by=.1)
referenceSet<-outer(1/scaleSet,reference)

# The query has to be made into a 1-row matrix.
# Perform all of the alignments at once, and normalize the result.

dist(t(query),referenceSet,meth="DTW")->distanceSet

# The optimal scale for the reference is 1.0
plot(scaleSet,scaleSet*distanceSet,
      xlab="Reference scale factor (denominator)",
      ylab="DTW distance",type="o",
      main="Sine vs scaled cosine alignment, 0 to 4 pi")

## Asymmetric step pattern: we can either disregard part of the pairs
## (as.dist), or average with the transpose

mm <- matrix(runif(12),ncol=3)
dm <- dist(mm,mm,method="DTW",step=asymmetric); # a crossdist object

# Old-fashioned call style would be:
# dm <- dtwDist(mm,step=asymmetric)
# as.dist(dm)

## Symmetrize by averaging:
(dm+t(dm))/2

## check definition
stopifnot(dm[2,1]==dtw(mm[2,],mm[1,],step=asymmetric)$distance)

```

Description

Methods for plotting dynamic time warp alignment objects returned by [dtw](#).

Usage

```
## S3 method for class 'dtw'
plot(x, type="alignment", ...)

# an alias for dtw.plot
dtwPlot(x, type="alignment", ...)

dtwPlotAlignment(d, xlab="Query index", ylab="Reference index",
                 plot.type="l", ...)
dtwPlotDensity(d, normalize=FALSE,
               xlab="Query index", ylab="Reference index",
               ...)
```

Arguments

<code>x, d</code>	dtw object, usually result of call to dtw
<code>xlab</code>	label for the query axis
<code>ylab</code>	label for the reference axis
<code>type</code>	general style for the alignment plot
<code>plot.type</code>	type of line to be drawn, used as the <code>type</code> argument in the underlying plot call
<code>normalize</code>	show per-step average cost instead of cumulative cost
<code>...</code>	additional arguments, passed to plotting functions

Details

`dtwPlot` displays alignment contained in `dtw` objects.

Various plotting styles are available, passing strings to the `type` argument (may be abbreviated):

- `alignment` plots the warping curve in `d`
- `twoway` plots a point-by-point comparison, with matching lines
- `threewayvis` a-vis inspection of the timeseries and their warping curve
- `density` displays the cumulative cost landscape with the warping path overimposed

For two-way plotting, see documentation for function [dtwPlotTwoWay](#).

For three-way plotting, see documentation for function [dtwPlotThreeWay](#).

If `normalize` is `TRUE`, the *average* cost per step is plotted instead of the cumulative one. Step averaging depends on the [stepPattern](#) used.

Additional parameters are carried on to the plotting functions: use with care.

Warning

These functions are incompatible with mechanisms for arranging plots on a device: `par(mfrow)`, `layout` and `split.screen`.

Note

The density plot is more colorful than useful.

Author(s)

Toni Giorgino

See Also

[dtwPlotTwoWay](#) for details on two-way plotting function. [dtwPlotThreeWay](#) for details on three-way plotting function.

Examples

```
## Same example as in dtw

idx<-seq(0,6.28,len=100);
query<-sin(idx)+runif(100)/10;
reference<-cos(idx)

alignment<-dtw(query,reference,keep=TRUE);

## A profile of the cumulative distance matrix
## Contour plot of the global cost

dtwPlotDensity(alignment,
  main="Sine/cosine: symmetric alignment, no constraints")

#####
##
## A study of the "Itakura" parallelogram
##
## A widely held misconception is that the "Itakura parallelogram" (as
## described in the original article) is a global constraint. Instead,
## it arises from local slope restrictions. Anyway, an "itakuraWindow",
## is provided in this package. A comparison between the two follows.

## The local constraint: three sides of the parallelogram are seen

dtw(query,reference,keep=TRUE,step=typeIIIC)->ita;
dtwPlot(ita,type="density",
  main="Slope-limited asymmetric step (Itakura)")
```



```
## Symmetric step with global parallelogram-shaped constraint. Note how
## long (>2 steps) horizontal stretches are allowed within the window.

dtw(query,reference,keep=TRUE>window=itakuraWindow)->ita;
dtwPlot(ita,type="density",
        main="Symmetric step with Itakura parallelogram window")
```

dtwPlotThreeWay *Plotting of dynamic time warp results: annotated warping function*

Description

Display the query and reference time series and their warping curve, arranged for visual inspection.

Usage

```
dtwPlotThreeWay(d,xts=NULL,yts=NULL,
               type.align="l",type.ts="l",
               match.indices=NULL,
               margin=4, inner.margin=0.2, title.margin=1.5,
               xlab="Query index",ylab="Reference index",main="Timeseries alignment",
               ... )
```

Arguments

d	an alignment result, object of class dtw
xts	query vector
yts	reference vector
xlab	label for the query axis
ylab	label for the reference axis
main	main title
type.align	line style for warping curve plot
type.ts	line style for timeseries plot
match.indices	indices for which to draw a visual guide
margin	outer figure margin
inner.margin	inner figure margin
title.margin	space on the top of figure
...	additional arguments, used for the warping curve

Details

The query time series is plotted in the bottom panel, with indices growing rightwards and values upwards. Reference is in the left panel, indices growing upwards and values leftwards. The warping curve panel matches indices, and therefore element (1,1) will be at the lower left, (N,M) at the upper right.

Argument `match.indices` is used to draw a visual guide to matches; if a vector is given, guides are drawn for the corresponding indices in the warping curve (match lines). If integer, it is used as the number of guides to be plotted. The corresponding style is customized via the `match.col` and `match.lty` arguments.

If `xts` and `yts` are not supplied, they will be recovered from `d`, as long as it was created with the two-argument call of `dtw` with `keep.internals=T`. Only single-variate time series can be plotted.

Warning

The function is incompatible with mechanisms for arranging plots on a device: `par(mfrow)`, `layout` and `split.screen`. Appearance of the match lines and timeseries currently can not be customized.

Author(s)

Toni Giorgino

Examples

```
## A noisy sine wave as query
## A cosine is for reference; sin and cos are offset by 25 samples

idx<-seq(0,6.28,len=100);
query<-sin(idx)+runif(100)/10;
reference<-cos(idx)
dtw(query,reference,keep=TRUE)->alignment;

## Beware of the reference's y axis, may be confusing
## Equivalent to plot(alignment,type="three");
dtwPlotThreeWay(alignment);

## Highlight matches of chosen QUERY indices. We will do some index
## arithmetics to recover the corresponding indices along the warping
## curve

hq <- (0:8)/8
hq <- round(hq*100)      # indices in query for pi/4 .. 7/4 pi

hw <- (alignment$index1 %in% hq)  # where are they on the w. curve?
hi <- (1:length(alignment$index1))[hw];  # get the indices of TRUE elems

dtwPlotThreeWay(alignment,match.indices=hi);
```

Description

Display the query and reference time series and their alignment, arranged for visual inspection.

Usage

```
dtwPlotTwoWay(d,xts=NULL,yts=NULL, offset=0,
  ts.type="l",pch=21,
                    match.indices=NULL,
  match.col="gray70", match.lty=3,
  xlab="Index", ylab="Query value",
  ... )
```

Arguments

<code>d</code>	an alignment result, object of class <code>dtw</code>
<code>xts</code>	query vector
<code>yts</code>	reference vector
<code>xlab,ylab</code>	axis labels
<code>offset</code>	displacement between the timeseries, summed to reference
<code>match.col, match.lty</code>	color and line type of the match guide lines
<code>match.indices</code>	indices for which to draw a visual guide
<code>ts.type,pch</code>	graphical parameters for timeseries plotting, passed to <code>matplot</code>
<code>...</code>	additional arguments, passed to <code>matplot</code>

Details

The two vectors are displayed via the `matplot` functions; their appearance can be customized via the `type` and `pch` arguments (constants or vectors of two elements). If `offset` is set, the reference is shifted vertically by the given amount; this will be reflected by the *right-hand* axis.

Argument `match.indices` is used to draw a visual guide to matches; if a vector is given, guides are drawn for the corresponding indices in the warping curve (match lines). If integer, it is used as the number of guides to be plotted. The corresponding style is customized via the `match.col` and `match.lty` arguments.

If `xts` and `yts` are not supplied, they will be recovered from `d`, as long as it was created with the two-argument call of `dtw` with `keep.internals=T`. Only single-variate time series can be plotted this way.

Warning

The function is incompatible with mechanisms for arranging plots on a device: `par(mfrow)`, `layout` and `split.screen`.

Note

When `offset` is set values on the left axis only apply to the query.

Author(s)

Toni Giorgino

See Also

[dtwPlot](#) for other dtw plotting functions, [matplot](#) for graphical parameters.

Examples

```
## A noisy sine wave as query
## A cosine is for reference; sin and cos are offset by 25 samples

idx<-seq(0,6.28,len=100);
query<-sin(idx)+runif(100)/10;
reference<-cos(idx)
dtw(query,reference,step=asymmetricP1,keep=TRUE)->alignment;

## Equivalent to plot(alignment,type="two");
dtwPlotTwoWay(alignment);

## Highlight matches of chosen QUERY indices. We will do some index
## arithmetics to recover the corresponding indices along the warping
## curve

hq <- (0:8)/8
hq <- round(hq*100)      # indices in query for pi/4 .. 7/4 pi

hw <- (alignment$index1 %in% hq)  # where are they on the w. curve?
hi <- (1:length(alignment$index1))[hw];  # get the indices of TRUE elems

## Beware of the reference's y axis, may be confusing
plot(alignment,offset=-2,type="two", lwd=3, match.col="grey50",
      match.indices=hi,main="Match lines shown every pi/4 on query");

legend("topright",c("Query","Reference (rt. axis)"), pch=21, col=1:6)
```

 dtwWindowingFunctions *Global constraints and windowing functions for DTW*

Description

Various global constraints (windows) which can be applied to the `window.type` argument of `dtw`, including the Sakoe-Chiba band, the Itakura parallelogram, and custom functions.

Usage

```
noWindow(iw, jw, ...);
sakoeChibaWindow(iw, jw, window.size, ...);
slantedBandWindow(iw, jw, query.size, reference.size, window.size, ...);
itakuraWindow(iw, jw, query.size, reference.size, ...);

dtwWindow.plot(fun, query.size=200, reference.size=220, ...);
```

Arguments

<code>iw</code>	index in the query (row) – automatically set
<code>jw</code>	index in the reference (column) – automatically set
<code>query.size</code>	size of the query time series – automatically set
<code>reference.size</code>	size of the reference time series – automatically set
<code>window.size</code>	window size, used by some windowing functions – must be set
<code>fun</code>	a windowing function
<code>...</code>	additional arguments passed to windowing functions

Details

Windowing functions can be passed to the `window.type` argument in `dtw` to put a global constraint to the warping paths allowed. They take two integer arguments (plus optional parameters) and must return a boolean value `TRUE` if the coordinates fall within the allowed region for warping paths, `FALSE` otherwise.

User-defined functions can read variables `reference.size`, `query.size` and `window.size`; these are pre-set upon invocation. Some functions require additional parameters which must be set (e.g. `window.size`). User-defined functions are free to implement any window shape, as long as at least one path is allowed between the initial and final alignment points, i.e., they are compatible with the DTW constraints.

The `sakoeChibaWindow` function implements the Sakoe-Chiba band, i.e. `window.size` elements around the main diagonal. If the window size is too small, i.e. if `reference.size-query.size > window.size`, warping becomes impossible.

An `itakuraWindow` global constraint is still provided with this package. See example below for a demonstration of the difference between a local the two.

The `slantedBandWindow` (package-specific) is a band centered around the (jagged) line segment which joins element `[1, 1]` to element `[query.size, reference.size]`, and will be `window.size` columns wide. In other words, the "diagonal" goes from one corner to the other of the possibly rectangular cost matrix, therefore having a slope of M/N , not 1.

`dtwWindow.plot` visualizes a windowing function. By default it plots a 200 x 220 rectangular region, which can be changed via `reference.size` and `query.size` arguments.

Value

Windowing functions return TRUE if the coordinates passed as arguments fall within the chosen warping window, FALSE otherwise. User-defined functions should do the same.

Note

Although `dtwWindow.plot` resembles object-oriented notation, there is not a such a `dtwWindow` class currently.

A widely held misconception is that the "Itakura parallelogram" (as described in reference [2]) is a *global* constraint, i.e. a window. To the author's knowledge, it instead arises from the local slope restrictions imposed to the warping path, such as the one implemented by the `typeIIIc` step pattern.

Author(s)

Toni Giorgino

References

[1] 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 URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1163055

[2] Itakura, F., *Minimum prediction residual principle applied to speech recognition*, Acoustics, Speech, and Signal Processing [see also IEEE Transactions on Signal Processing], IEEE Transactions on , vol.23, no.1, pp. 67-72, Feb 1975. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1162641

Examples

```
## Display some windowing functions
dtwWindow.plot(itakuraWindow, main="So-called Itakura parallelogram window")
dtwWindow.plot(slantedBandWindow, window.size=2,
  reference=13, query=17, main="The slantedBandWindow at window.size=2")
```

```
## Asymmetric step with Sakoe-Chiba band
```

```
idx<-seq(0,6.28,len=100);
query<-sin(idx)+runif(100)/10;
reference<-cos(idx);
```

```
asyband<-dtw(query,reference,keep=TRUE,
             step=asymmetric,
             window.type=sakoeChibaWindow,
             window.size=30
             );

dtwPlot(asyband,type="density",main="Sine/cosine: asymmetric step, S-C window")
```

mvm

Minimum Variance Matching algorithm

Description

Step patterns to compute the Minimum Variance Matching (MVM) correspondence between time series

Usage

```
mvmStepPattern(elasticity=20);
```

Arguments

`elasticity` integer: maximum consecutive reference elements skippable

Details

The Minimum Variance Matching algorithm [1] finds the non-contiguous parts of reference which best match the query, allowing for arbitrarily long "stretches" of reference to be excluded from the match. All elements of the query have to be matched. First and last elements of the query are anchored at the boundaries of the reference.

The `mvmStepPattern` function creates a `stepPattern` object which implements this behavior, to be used with the usual `dtw` call (see example). MVM is computed as a special case of DTW, with a very large, asymmetric-like step pattern.

The `elasticity` argument limits the maximum run length of reference which can be skipped at once. If no limit is desired, set `elasticity` to an integer at least as large as the reference (computation time grows linearly).

Value

A step pattern object.

Author(s)

Toni Giorgino

References

[1] Latecki, L. J.; Megalooikonomou, V.; Wang, Q. & Yu, D. *An elastic partial shape matching technique* Pattern Recognition, 2007, 40, 3069-3080. <http://dx.doi.org/10.1016/j.patcog.2007.03.004>

See Also

Other objects in [stepPattern](#).

Examples

```
## The hand-checkable example given in Fig. 5, ref. [1] above
diffmx <- matrix( byrow=TRUE, nrow=5, c(
  0, 1, 8, 2, 2, 4, 8,
  1, 0, 7, 1, 1, 3, 7,
  -7, -6, 1, -5, -5, -3, 1,
  -5, -4, 3, -3, -3, -1, 3,
  -7, -6, 1, -5, -5, -3, 1 ) );

## Cost matrix
costmx <- diffmx^2;

## Compute the alignment
al <- dtw(costmx,step.pattern=mvmStepPattern(10))

## Elements 4,5 are skipped
print(al$index2)

plot(al,main="Minimum Variance Matching alignment")
```

stepPattern

Step patterns for DTW

Description

A stepPattern object lists the transitions allowed while searching for the minimum-distance path. DTW variants are implemented by passing one of the objects described in this page to the stepPattern argument of the dtw call.

Usage

```
## Well-known step patterns
symmetric1
symmetric2
asymmetric
```



```

## Step patterns classified according to Rabiner-Juang [Rabiner1993]
  rabinerJuangStepPattern(type,slope.weighting="d",smoothed=FALSE)

## Slope-constrained step patterns from Sakoe-Chiba [Sakoe1978]
  symmetricP0; asymmetricP0
  symmetricP05; asymmetricP05
  symmetricP1; asymmetricP1
  symmetricP2; asymmetricP2

## Step patterns classified according to Rabiner-Myers [Myers1980]
  typeIa; typeIb; typeIc; typeId;
  typeIas; typeIbs; typeIcs; typeIds; # smoothed
  typeIIa; typeIIb; typeIIc; typeIID;
  typeIIIC; typeIVc;

## Miscellaneous
  mori2006;
  rigid;

## S3 method for class 'stepPattern'
print(x,...)
## S3 method for class 'stepPattern'
plot(x,...)
## S3 method for class 'stepPattern'
t(x)

stepPattern(v,norm=NA)
is.stepPattern(x)

```

Arguments

x	a step pattern object
type	path specification, integer 1..7 (see [Rabiner1993], table 4.5)
slope.weighting	slope weighting rule: character "a" to "d" (see [Rabiner1993], sec. 4.7.2.5)
smoothed	logical, whether to use smoothing (see [Rabiner1993], fig. 4.44)
v	a vector defining the stepPattern structure
norm	normalization hint (character)
...	additional arguments to print .

Details

A step pattern characterizes the matching model and slope constraint specific of a DTW variant. They also known as local- or slope-constraints, transition types, production or recursion rules [GiorginoJSS].

`print.stepPattern` prints an user-readable description of the recurrence equation defined by the given pattern.

`plot.stepPattern` graphically displays the step patterns productions which can lead to element (0,0). Weights are shown along the step leading to the corresponding element.

`t.stepPattern` transposes the productions and normalization hint so that roles of query and reference become reversed.

A variety of classifications have been proposed for step patterns, including Sakoe-Chiba [Sakoe1978]; Rabiner-Juang [Rabiner1993]; and Rabiner-Myers [Myers1980]. The `dtw` package implements all of the transition types found in those papers, with the exception of Itakura's and Velichko-Zagoruyko's steps, which require subtly different algorithms (this may be rectified in the future). Itakura recursion is almost, but not quite, equivalent to `typeIIIC`.

For convenience, we shall review pre-defined step patterns grouped by classification. Note that the same pattern may be listed under different names. Refer to paper [GiorginoJSS] for full details.

1. Well-known step patterns

Common DTW implementations are based on one of the following transition types.

`symmetric2` is the normalizable, symmetric, with no local slope constraints. Since one diagonal step costs as much as the two equivalent steps along the sides, it can be normalized dividing by $N+M$ (query+reference lengths). It is widely used and the default.

`asymmetric` is asymmetric, slope constrained between 0 and 2. Matches each element of the query time series exactly once, so the warping path $index2 \sim index1$ is guaranteed to be single-valued. Normalized by N (length of query).

`symmetric1` (or White-Neely) is quasi-symmetric, no local constraint, non-normalizable. It is biased in favor of oblique steps.

2. The Rabiner-Juang set

A comprehensive table of step patterns is proposed in Rabiner-Juang's book [Rabiner1993], tab. 4.5. All of them can be constructed through the `rabinerJuangStepPattern(type, slope.weighting, smoothed)` function.

The classification foresees seven families, labelled with Roman numerals I-VII; here, they are selected through the integer argument `type`. Each family has four slope weighting sub-types, named in sec. 4.7.2.5 as "Type (a)" to "Type (d)"; they are selected passing a character argument `slope.weighting`, as in the table below. Furthermore, each subtype can be either plain or smoothed (figure 4.44); smoothing is enabled setting the logical argument `smoothed`. (Not all combinations of arguments make sense.)

Subtype	Rule	Norm	Unbiased
a	min step	–	NO
b	max step	–	NO
c	Di step	N	YES
d	Di+Dj step	N+M	YES

3. The Sakoe-Chiba set

Sakoe-Chiba [Sakoe1978] discuss a family of slope-constrained patterns; they are implemented as shown in page 47, table I. Here, they are called `symmetricP<x>` and `asymmetricP<x>`, where

<x> corresponds to Sakoe's integer slope parameter P . Values available are accordingly: 0 (no constraint), 1, 05 (one half) and 2. See [Sakoe1978] for details.

4. The Rabiner-Myers set

The type<XX><y> step patterns follow the older Rabiner-Myers' classification proposed in [Myers1980] and [MRR1980]. Note that this is a subset of the Rabiner-Juang set [Rabiner1993], and the latter should be preferred in order to avoid confusion. <XX> is a Roman numeral specifying the shape of the transitions; <y> is a letter in the range a-d specifying the weighting used per step, as above; typeIIx patterns also have a version ending in s, meaning the smoothing is used (which does not permit skipping points). The typeId, typeIID and typeIIDs are unbiased and symmetric.

5. Other

The rigid pattern enforces a fixed unitary slope. It only makes sense in combination with open.begin=T, open.end=T to find gapless subsequences. It may be seen as the $P \rightarrow \infty$ limiting case in Sakoe's classification.

mor i2006 is Mori's asymmetric step-constrained pattern [Mori2006]. It is normalized by the matched reference length.

`mvmStepPattern()` implements Latecki's Minimum Variance Matching algorithm, and it is described in its own page.

Note

Constructing stepPattern objects is tricky and thus undocumented. For a commented example please see source code for symmetricP1.

Author(s)

Toni Giorgino

References

[GiorginoJSS] Toni Giorgino. *Computing and Visualizing Dynamic Time Warping Alignments in R: The dtw Package*. Journal of Statistical Software, 31(7), 1-24. <http://www.jstatsoft.org/v31/i07/>

[Itakura1975] Itakura, F., *Minimum prediction residual principle applied to speech recognition*, Acoustics, Speech, and Signal Processing [see also IEEE Transactions on Signal Processing], IEEE Transactions on , vol.23, no.1, pp. 67-72, Feb 1975. URL: <http://dx.doi.org/10.1109/TASSP.1975.1162641>

[MRR1980] Myers, C.; Rabiner, L. & Rosenberg, A. *Performance tradeoffs in dynamic time warping algorithms for isolated word recognition*, IEEE Trans. Acoust., Speech, Signal Process., 1980, 28, 623-635. URL: <http://dx.doi.org/10.1109/TASSP.1980.1163491>

[Mori2006] Mori, A.; Uchida, S.; Kurazume, R.; Taniguchi, R.; Hasegawa, T. & Sakoe, H. *Early Recognition and Prediction of Gestures Proc. 18th International Conference on Pattern Recognition ICPR 2006*, 2006, 3, 560-563. URL: <http://dx.doi.org/10.1109/ICPR.2006.467>

[Myers1980] Myers, Cory S. *A Comparative Study Of Several Dynamic Time Warping Algorithms For Speech Recognition*, MS and BS thesis, Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, archived Jun 20 1980, <http://hdl.handle.net/1721.1/27909>

[Rabiner1993] Rabiner, L. R., & Juang, B.-H. (1993). *Fundamentals of speech recognition*. Englewood Cliffs, NJ: Prentice Hall.

[Sakoe1978] 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 URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1163055

See Also

[mvmStepPattern](#), implementing Latecki's Minimal Variance Matching algorithm.

Examples

```
#####
##
## The usual (normalizable) symmetric step pattern
## Step pattern recursion, defined as:
## g[i,j] = min(
##     g[i,j-1] + d[i,j] ,
##     g[i-1,j-1] + 2 * d[i,j] ,
##     g[i-1,j] + d[i,j] ,
## )

print(symmetric2) # or just "symmetric2"

#####
##
## The well-known plotting style for step patterns

plot(symmetricP2,main="Sakoe's Symmetric P=2 recursion")

#####
##
## Same example seen in ?dtw , now with asymmetric step pattern

idx<-seq(0,6.28,len=100);
query<-sin(idx)+runif(100)/10;
reference<-cos(idx);
```

```

## Do the computation
asy<-dtw(query,reference,keep=TRUE,step=asymmetric);

dtwPlot(asy,type="density",main="Sine and cosine, asymmetric step")

#####
##
## Hand-checkable example given in [Myers1980] p 61
##

`tm` <-
structure(c(1, 3, 4, 4, 5, 2, 2, 3, 3, 4, 3, 1, 1, 1, 3, 4, 2,
3, 3, 2, 5, 3, 4, 4, 1), .Dim = c(5L, 5L))

```

triangleFixing

Triangle Fixing algorithm for Metric Nearness

Description

Return the matrix satisfying the triangle inequality most similar to a given input matrix.

Usage

```
triangleFixing(D, tolerance=1e-15, max.iter=1e6)
tri.ineq.show(D)
```

Arguments

D	the matrix to be fixed
tolerance	threshold for convergence
max.iter	iteration limit

Details

Given a symmetric matrix, returns the symmetric matrix "most similar" to it that satisfies the triangle inequality. (Similarity is measured in the L2 sense.) This is an implementation of the L2 triangle fixing algorithm for metric nearness (Metric_Nearness_L2) given in [1] and [2].

WARNING: the algorithm does not seem to work in all cases.

The input should be a symmetric matrix (or dist object, which is coerced). Convergence of the iterative algorithm is assumed when total matrix element change is less than tolerance or the max.iter iteration limit is reached.

The tri.ineq.show function checks whether a dissimilarity matrix respects the triangle inequality. Returns NULL if it does, or a Nx3 matrix of indices where it doesn't (unsorted).

Value

The fixed distance matrix (square symmetric).

Author(s)

Toni Giorgino

References

1. Brickell, J., Dhillon, I., Sra, S., Tropp, J. (2008). The Metric Nearness Problem. SIAM. J. Matrix Anal. & Appl. 30, 375-396.
2. S. Sra, "METRIC: The Metric Nearness Problem" code at <http://suvrit.de/work/soft/metricn.html>.

See Also

`tri.ineq` in package `fossil`, which checks whether a distance matrix respects the triangle inequality.

Examples

```
a<-matrix(0.2, 4,4)
a[4,2]<-0.8
a<-as.matrix(as.dist(a))
af<-triangleFixing(a)

tri.ineq.show(a)
stopifnot(tri.ineq.show(af)==NULL)

## Example in http://suvrit.de/work/soft/metricn.html
a<-matrix(c(0, 3, 7,
           3, 0, 1,
           7, 1, 0),3)
af<-triangleFixing(a)

tri.ineq.show(a)
stopifnot(tri.ineq.show(af)==NULL)
```

warp

Apply a warping to a given timeseries

Description

Returns the indexing required to apply the optimal warping curve to a given timeseries (warps either into a query or into a reference).

Usage

```
warp(d, index.reference=FALSE)
```

Arguments

```
d                dtw object specifying the warping curve to apply
index.reference   TRUE to warp a reference, FALSE to warp a query
```

Details

The warping is returned as a set of indices, which can be used to subscript the timeseries to be warped (or rows in a matrix, if one wants to warp a multivariate time series). In other words, warp converts the warping curve, or its inverse, into a function in the explicit form.

Multiple indices that would be mapped to a single point are averaged, with a warning. Gaps in the index sequence are filled by linear interpolation.

Value

A list of indices to subscript the timeseries.

Author(s)

Toni Giorgino

See Also

Examples in [dtw](#) show how to *graphically* apply the warping via parametric plots.

Examples

```
idx<-seq(0,6.28,len=100);
query<-sin(idx)+runif(100)/10;
reference<-cos(idx)

alignment<-dtw(query,reference);

wq<-warp(alignment,index.reference=FALSE);
wt<-warp(alignment,index.reference=TRUE);

old.par <- par(no.readonly = TRUE);
par(mfrow=c(2,1));

plot(reference,main="Warping query");
  lines(query[wq],col="blue");

plot(query,type="l",col="blue",
  main="Warping reference");
  points(reference[wt]);
```

```

par(old.par);

#####
##
## Asymmetric step makes it "natural" to warp
## the reference, because every query index has
## exactly one image (q->t is a function)
##

alignment<-dtw(query,reference,step=asymmetric)
wt<-warp(alignment,index.reference=TRUE);

plot(query,type="l",col="blue",
      main="Warping reference, asymmetric step");
points(reference[wt]);

```

warpArea

Compute Warping Path Area

Description

Compute the area between the warping function and the diagonal (no-warping) path, in unit steps.

Usage

```
warpArea(d)
```

Arguments

d an object of class dtw

Details

Above- and below- diagonal unit areas all count *plus* one (they do not cancel with each other). The "diagonal" goes from one corner to the other of the possibly rectangular cost matrix, therefore having a slope of M/N, not 1, as in [slantedBandWindow](#).

The computation is approximate: points having multiple correspondences are averaged, and points without a match are interpolated. Therefore, the area can be fractionary.

Value

The area, not normalized by path length or else.

Note

There could be alternative definitions to the area, including considering the envelope of the path.

Author(s)

Toni Giorgino

Examples

```
ds<-dtw(1:4,1:8);  
  
plot(ds);lines(seq(1,8,len=4),col="red");  
  
warpArea(ds)  
  
## Result: 6  
## index 2 is 2 while diag is 3.3 (+1.3)  
##      3    3          5.7 (+2.7)  
##      4   4:8 (avg to 6)    8 (+2 )  
##                                     -----  
##                                     6
```

Index

- *Topic **cluster**
 - triangleFixing, 29
- *Topic **datasets**
 - aami, 4
- *Topic **hplot**
 - dtwPlot, 14
 - dtwPlotThreeWay, 17
 - dtwPlotTwoWay, 19
- *Topic **package**
 - dtw-package, 2
- *Topic **ts**
 - countPaths, 6
 - dtw, 7
 - dtw-package, 2
 - dtwDist, 12
 - dtwPlot, 14
 - dtwWindowingFunctions, 21
 - mvm, 23
 - stepPattern, 24
 - warp, 30
 - warpArea, 32
- aami, 4
- aami3a (aami), 4
- aami3b (aami), 4
- asymmetric (stepPattern), 24
- asymmetricP0 (stepPattern), 24
- asymmetricP05, 9
- asymmetricP05 (stepPattern), 24
- asymmetricP1 (stepPattern), 24
- asymmetricP2 (stepPattern), 24
- countPaths, 6
- dist, 3, 7, 8, 10, 13
- distance, 4, 10, 13
- dtw, 3, 4, 7, 12, 15, 18, 19, 21, 23, 24, 31
- dtw-package, 2
- dtwDist, 10, 12
- dtwPlot, 14, 20
- dtwPlotAlignment (dtwPlot), 14
- dtwPlotDensity (dtwPlot), 14
- dtwPlotThreeWay, 15, 16, 17
- dtwPlotTwoWay, 15, 16, 19
- dtwWindow.plot (dtwWindowingFunctions), 21
- dtwWindowingFunctions, 3, 4, 8, 10, 21
- is.dtw (dtw), 7
- is.stepPattern (stepPattern), 24
- itakuraWindow (dtwWindowingFunctions), 21
- matplot, 19, 20
- mori2006 (stepPattern), 24
- mvm, 23
- mvmStepPattern, 27, 28
- mvmStepPattern (mvm), 23
- noWindow (dtwWindowingFunctions), 21
- outer, 4, 10
- plot.dtw, 10
- plot.dtw (dtwPlot), 14
- plot.stepPattern (stepPattern), 24
- print, 25
- print.dtw (dtw), 7
- print.stepPattern (stepPattern), 24
- rabinerJuangStepPattern (stepPattern), 24
- rigid (stepPattern), 24
- sakoeChibaWindow (dtwWindowingFunctions), 21
- slantedBandWindow, 32
- slantedBandWindow (dtwWindowingFunctions), 21
- stepPattern, 3, 4, 7, 8, 10, 15, 24, 24
- symmetric1 (stepPattern), 24

symmetric2 (stepPattern), 24
symmetricP0 (stepPattern), 24
symmetricP05 (stepPattern), 24
symmetricP1 (stepPattern), 24
symmetricP2 (stepPattern), 24

t.stepPattern (stepPattern), 24
tri.ineq, 30
tri.ineq.show (triangleFixing), 29
triangleFixing, 29
typeIa (stepPattern), 24
typeIas (stepPattern), 24
typeIb (stepPattern), 24
typeIbs (stepPattern), 24
typeIc (stepPattern), 24
typeIcs (stepPattern), 24
typeId (stepPattern), 24
typeIds (stepPattern), 24
typeIIa (stepPattern), 24
typeIIb (stepPattern), 24
typeIIc (stepPattern), 24
typeIIId (stepPattern), 24
typeIIIc, 22
typeIIIc (stepPattern), 24
typeIVc (stepPattern), 24

vegdist, 13

warp, 30
warpArea, 32