

Package ‘dtw’

January 15, 2012

Type Package

Title Dynamic Time Warping algorithms

Version 1.14-3

Date 2009-08-15

Depends methods, proxy

Imports proxy

Enhances proxy

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

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

Description Comprehensive implementation of Dynamic Time Warping algorithms in R. DTW finds the optimal (least cumulative distance) mapping between two time series. All common DTW variants are covered, including local and global constraints, arbitrary timeseries lengths, distance definitions, MVM, etc. Package computes cumulative distance, warping functions, plots, normalizations, etc.

License GPL (>= 2)

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

Repository CRAN

Date/Publication 2010-02-04 15:32:53

R topics documented:

dtw-package	2
aami	3
dtw	5
dtwDist	10
dtwPlot	12
dtwPlotThreeWay	14

dtwPlotTwoWay	16
dtwWindowingFunctions	18
mvm	21
stepPattern	22
warp	26
warpArea	28

Index	30
--------------	-----------

dtw-package	<i>Dynamic Time Warp algorithms in R</i>
-------------	--

Description

Dynamic Time Warp: find the optimal alignment between two time series.

Details

Package: dtw
 Type: Package
 Version: 1.14
 Date: 2009-8-15
 License: GPL-2

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

DTW finds the optimal (least cumulative distance) mapping between a given query into a given reference time series.

Most variants of the algorithm are supported: symmetric, asymmetric and custom step patterns, with weighting (see [stepPattern](#)). Supports windowing: none, "Itakura" parallelogram, Sakoe-Chiba band, custom (see [dtwWindowingFunctions](#)). Handles query and reference of arbitrary lengths. Multivariate matching and arbitrary definition for a distance function are supported via user-supplied local distance matrix. The Minimum Variance Matching algorithm is also supported, as a special case of DTW.

Package provides minimum cumulative distance, warping function, plots, etc. A fast, compiled version of the algorithm is normally used. Should it not be available, a slower pure-R equivalent is automatically used as a fall-back.

Please see documentation for function [dtw](#), which is the main entry point to the package.

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>.

To get the latest stable version from CRAN, use `install.packages("dtw")`. To get the development version (possibly unstable), use `install.packages("dtw", repos="http://r-forge.r-project.org")`.

Author(s)

Toni Giorgino, Copyright (c) 2007-2009

Maintainer: 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/>

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)
    )
```

dtw	<i>Dynamic Time Warp</i>
-----	--------------------------

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

x	query vector <i>or</i> local cost matrix
y	reference vector, unused if x given as cost matrix
dist.method	pointwise (local) distance function to use. See dist in package proxy
step.pattern	a <code>stepPattern</code> object describing the local warping steps allowed with their cost (see stepPattern)
window.type	windowing function. Character: "none", "itakura", "sakoechiba", "slantedband", or a function (see details).
open.begin, open.end	perform open-ended alignments
keep.internals	preserve the cumulative cost matrix, inputs, and other internal structures
distance.only	only compute distance (no backtrack, faster)
d	an arbitrary R object
...	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 DTW are supported via the `step.pattern` argument, which defaults to `symmetric2`. Most common step patterns are pre-defined: see `stepPattern` for details.

Open-ended alignment, i.e. semi-unconstrained alignment, can be selected via the `open.end` argument. 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`. Open-begin alignments usually only make sense when `open.end` is enabled, as well (subsequence alignment); otherwise, it is just as easy to reverse the query sequence. Subsequence alignments are similar e.g. to UE2-1 algorithm by Rabiner (1978) and others. Please find a review in Tormene et al. (2009).

Windowing (enforcing a global constraint) is supported by passing a string or function `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.

A native (fast, compiled) version of the function is normally available. If it is not, an interpreted equivalent will be used as a fall-back, with a warning.

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

Value

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

`distance` the minimum global distance computed, *not* normalized.

normalizedDistance	distance computed, <i>normalized</i> for path length, if normalization is known for chosen step pattern.
N,M	query and reference length
call	the function call that created the object
index1	matched elements: indices in x
index2	corresponding mapped indices in y
stepPattern	the stepPattern object used for the computation
jmin	last element of reference matched, if open.end=TRUE
directionMatrix	if keep.internals=TRUE, the directions of steps that would be taken at each alignment pair (integers indexing step patterns)
costMatrix	if keep.internals=TRUE, the cumulative cost matrix
query, reference	if keep.internals=TRUE 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.

The `partial` argument has been deprecated and renamed `open.end` as of version 1.12.

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

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

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.

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.
##

alignmentOBE <-
  dtw(query[44:88],reference,
      keep=TRUE,step=asymmetric,
      open.end=TRUE,open.begin=TRUE);
plot(alignmentOBE,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)

```

dtwPlot

Plotting of dynamic time warp results

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

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

dtwPlotTwoWay

Plotting of dynamic time warp results: pointwise comparison

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	<i>Minimum Variance Matching algorithm</i>
-----	--

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

[2] 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/>

See Also

Other objects in [stepPattern](#).

Examples

```
## The hand-checkable example given in 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

Local constraints and step patterns for DTW

Description

DTW variants are implemented through step pattern objects. A `stepPattern` object lists the transitions allowed by the `dtw` function in the search for the minimum-distance path. The user can use one of the objects described in this page for the `stepPattern` argument of the `dtw` call.

Usage

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

## Step patterns classified according to Rabiner-Juang [3]
rabinerJuangStepPattern(type,slope.weighting="d",smoothed=FALSE)

## Slope-constrained step patterns from Sakoe-Chiba [1]
symmetricP0; asymmetricP0
```

```

symmetricP05; asymmetricP05
symmetricP1;  asymmetricP1
symmetricP2;  asymmetricP2

## Step patterns classified according to Rabiner-Myers [4]
typeIa;  typeIb;  typeIc;  typeId;
typeIas; typeIbs; typeIcs; typeIds; # smoothed
typeIIa; typeIIb; typeIIc; typeIID;
typeIIc; typeIVc;

## Miscellaneous
mori2006;

## 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

<code>x</code>	a step pattern object
<code>type</code>	path specification, integer 1..7 (see [3], table 4.5)
<code>slope.weighting</code>	slope weighting rule: character "a" to "d" (see [3], sec. 4.7.2.5)
<code>smoothed</code>	logical, whether to use smoothing (see [3], fig. 4.44)
<code>v</code>	a vector defining the stepPattern structure
<code>norm</code>	normalization hint (character)
<code>...</code>	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, or production rules [7].

`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 [1]; Rabiner-Juang [3]; and Rabiner-Myers [4]. 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 [7] for full details.

1. Well-known step patterns

These common transition types are used in quite a lot of implementations.

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

`symmetric2` is 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).

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

2. The Rabiner-Juang set

A comprehensive table of step patterns is proposed by Rabiner-Juang [3], tab. 4.5. All of them can be recovered by the `rabinerJuangStepPattern(type, slope.weighting, smoothed)` function.

Seven families, labelled with Roman numerals I-VII, 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 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

`symmetricPx` is the family of Sakoe's symmetric steps, slope constraint x ; `asymmetricPx` are Sakoe's asymmetric, slope constraint x . These slope-constrained patterns are discussed in Sakoe-Chiba [1], and implemented as shown in page 47, table I. Values available for $P(x)$ are accordingly: 0 (no constraint), 1, 0.5 (one half) and 2. See reference for details.

4. The Rabiner-Myers set

The `typeXXx` step patterns follow the older Rabiner-Myers' classification given in [4-5]. Note that they are a subset of the Rabiner-Juang set [3], which should be preferred to avoid confusion. XX is a roman numeral specifying the shape of the transitions; x is a letter in the range a-d according the type of weighting used per step, as above; `typeIIx` patterns also have a version ending in `s` meaning the path smoothing is used (which does not permit skipping points). The `typeId`, `typeIID` and `typeIIIs` are unbiased and symmetric.

5. Other

Mori's [6] asymmetric step-constrained pattern is called `mori2006`. It is normalized in the reference length.

Note

The `stepPattern` constructor is currently not well documented. For a commented example please see source code for `symmetricP1`.

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

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

[4] Myers, C. S. *A Comparative Study Of Several Dynamic Time Warping Algorithms For Speech Recognition*, MS and BS thesis, MIT Jun 20 1980, dspace.mit.edu/bitstream/1721.1/27909/1/07888629.pdf

[5] 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

[6] 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

[7] 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/>

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 [4] 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))

```

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 **data**
 - aami, 3
- *Topic **hplot**
 - dtwPlot, 12
 - dtwPlotThreeWay, 14
 - dtwPlotTwoWay, 16
- *Topic **optimize**
 - dtw, 5
- *Topic **package**
 - dtw-package, 2
- *Topic **ts**
 - dtw, 5
 - dtw-package, 2
 - dtwDist, 10
 - dtwPlot, 12
 - dtwWindowingFunctions, 18
 - mvm, 21
 - stepPattern, 22
 - warp, 26
 - warpArea, 28
- aami, 3
- aami3a (aami), 3
- aami3b (aami), 3
- asymmetric (stepPattern), 22
- asymmetricP0 (stepPattern), 22
- asymmetricP05 (stepPattern), 22
- asymmetricP1 (stepPattern), 22
- asymmetricP2 (stepPattern), 22
- dist, 5, 6, 8, 10, 11
- distance, 3, 8, 11
- dtw, 2, 3, 5, 10, 12, 13, 15, 17–19, 21, 22, 27
- dtw-package, 2
- dtwDist, 8, 10
- dtwPlot, 12, 17
- dtwPlotAlignment (dtwPlot), 12
- dtwPlotDensity (dtwPlot), 12
- dtwPlotThreeWay, 13, 14
- dtwPlotTwoWay, 13, 16
- dtwWindow.plot (dtwWindowingFunctions), 18
- dtwWindowingFunctions, 2, 3, 6, 8, 18
- is.dtw (dtw), 5
- is.stepPattern (stepPattern), 22
- itakuraWindow (dtwWindowingFunctions), 18
- matplot, 17
- mori2006 (stepPattern), 22
- mvm, 21
- mvmStepPattern (mvm), 21
- noWindow (dtwWindowingFunctions), 18
- outer, 3, 8
- plot.dtw, 8
- plot.dtw (dtwPlot), 12
- plot.stepPattern (stepPattern), 22
- print, 23
- print.dtw (dtw), 5
- print.stepPattern (stepPattern), 22
- rabinerJuangStepPattern (stepPattern), 22
- sakoeChibaWindow (dtwWindowingFunctions), 18
- slantedBandWindow, 28
- slantedBandWindow (dtwWindowingFunctions), 18
- stepPattern, 2, 3, 5, 6, 8, 13, 21, 22
- symmetric1 (stepPattern), 22
- symmetric2 (stepPattern), 22
- symmetricP0 (stepPattern), 22
- symmetricP05 (stepPattern), 22
- symmetricP1 (stepPattern), 22
- symmetricP2 (stepPattern), 22

t.stepPattern (stepPattern), [22](#)
typeIa (stepPattern), [22](#)
typeIas (stepPattern), [22](#)
typeIb (stepPattern), [22](#)
typeIbs (stepPattern), [22](#)
typeIc (stepPattern), [22](#)
typeIcs (stepPattern), [22](#)
typeId (stepPattern), [22](#)
typeIds (stepPattern), [22](#)
typeIIa (stepPattern), [22](#)
typeIIb (stepPattern), [22](#)
typeIIc (stepPattern), [22](#)
typeIIId (stepPattern), [22](#)
typeIIIc, [20](#)
typeIIIc (stepPattern), [22](#)
typeIVc (stepPattern), [22](#)

vegdist, [11](#)

warp, [26](#)
warpArea, [28](#)