Journal of Statistical Software

August 2009, Volume 31, Issue 7. http://www.jstatsoft.org/

Computing and Visualizing Dynamic Time Warping Alignments in R: The dtw Package

Toni Giorgino
University of Pavia

Abstract

Dynamic time warping is a popular technique for comparing time series, providing both a distance measure that is insensitive to local compression and stretches and the warping which optimally deforms one of the two input series onto the other. A variety of algorithms and constraints have been discussed in the literature. The **dtw** package provides an unification of them; it allows R users to compute time series alignments mixing freely a variety of continuity constraints, restriction windows, endpoints, local distance definitions, and so on. The package also provides functions for visualizing alignments and constraints using several classic diagram types.

Keywords: timeseries, alignment, dynamic programming, dynamic time warping.

1. Introduction

Dynamic time warping (DTW) is the name of a class of algorithms for comparing series of values with each other. The rationale behind DTW is, given two time series, to stretch or compress them locally in order to make one resemble the other as much as possible. The distance between the two is computed, after stretching, by summing the distances of individual aligned elements (Figure 1). DTW algorithms have been proposed around 1970 in the context of speech recognition, to account for differences in speaking rates between speakers and utterances. The technique is useful, for example, when one is willing to find a low distance score between the sound signals corresponding to utterances *now* and *nooow* respectively, insensitive to the prolonged duration of the *o* sound.

Various types of DTW algorithms differ for the input feature space, the local distance assumed, presence of local and global constraints on the alignment, and so on. This freedom makes DTW a very flexible alignment approach. As said, it has been popularized in the '70s, when it was mainly applied to isolated word recognition (Velichko and Zagoruyko 1970; Sakoe and

Chiba 1971); since then, it has been employed for clustering and classification in countless domains: electro-cardiogram analysis (Huang and Kinsner 2002; Syeda-Mahmood *et al.* 2007; Tuzcu and Nas 2005), clustering of gene expression profiles (Aach and Church 2001; Hermans and Tsiporkova 2007), biometrics (Faundez-Zanuy 2007; Rath and Manmatha 2003), process monitoring (Gollmer and Posten 1996), just to name a few. The term *time* series may even be misleading, because the warped dimensions can be other than time, e.g., an angle for shape recognition (Kartikeyan and Sarkar 1989; Wei *et al.* 2006; Tak 2007).

This paper describes the **dtw** package for the R statistical software (R Development Core Team 2009), which provides a comprehensive solution for the computation and visualization of DTW alignments, whose theory is outlined in Section 2. The stable package version, currently 1.13-1, is available in source and binary form on the Comprehensive R Archive Network (CRAN), while development versions are hosted at R-Forge (Giorgino and Tormene 2009). The package allows users to select among a wide variety of algorithms and constraints described in the literature, simply passing arguments to a single function, `dtw`, introduced in Section 3. The following sections discuss how the user can customize the classic constraints of the algorithm (local slope, endpoints, windowing). Finally, Section 4 describes how the alignment results can be conveniently plotted in a variety of ways.

2. Definition of the algorithm

In the following exposition, the choice of symbols will follow roughly the one in Chapter 4 of Rabiner and Juang (1993), which is an excellent reference on the subject. We assume that we want to compare two time series: a *test*, or *query*, $X = (x_1, \ldots, x_N)$; and a *reference* $Y = (y_1, \ldots, y_M)$. For clarity, in the following we shall reserve the symbol $i = 1 \ldots N$ for indexing the elements in X and $j = 1 \ldots M$ for those in Y. We also assume that a non-negative, *local dissimilarity* function f is defined between any pair of elements x_i and y_j, with the shortcut:

$$d(i, j) = f(x_i, y_j) \geq 0 \tag{1}$$

Note that d, the cross-distance matrix between vectors X and Y, is the only input to the DTW algorithm: elements x_i and y_j only enter the computation through the arguments of f. Therefore, the following discussion applies, with no loss of generality, to cases when X and Y are single- or multi-variate, continuous, nominal, or mixed, as long as $f(\cdot, \cdot)$ is suitably defined. While the most common choice is to assume the Euclidean distance, different definitions (e.g., those provided by the **proxy** package, Meyer and Buchta 2009) may be useful as well.

At the core of the technique lies the *warping curve* $\phi(k)$, $k = 1 \ldots T$:

$$\begin{aligned} \phi(k) &= \Big(\phi_x(k), \phi_y(k)\Big) \quad \text{with} \\ \phi_x(k) &\in \{1 \ldots N\}, \\ \phi_y(k) &\in \{1 \ldots M\} \end{aligned}$$

The warping *functions* ϕ_x and ϕ_y remap the time indices of X and Y respectively. Given ϕ, we compute the average accumulated distortion between the warped time series X and Y:

$$d_\phi(X, Y) = \sum_{k=1}^{T} d(\phi_x(k), \phi_y(k))\, m_\phi(k)/M_\phi$$

where $m_\phi(k)$ is a per-step weighting coefficient and M_ϕ is the corresponding normalization constant, which ensures that the accumulated distortions are comparable along different paths. To ensure reasonable warps, *constraints* are usually imposed on ϕ. For example, *monotonicity* is imposed to preserve their time ordering and avoid meaningless loops:

$$\phi_x(k+1) \geq \phi_x(k)$$
$$\phi_y(k+1) \geq \phi_y(k)$$

The idea underlying DTW is to find the optimal alignment ϕ such that

$$D(X, Y) = \min_\phi d_\phi(X, Y) \tag{2}$$

In other words, one picks the deformation of the time axes of X and Y which brings the two time series as close as possible to each other. Remarkably, despite of the large search space, Equation 2 can be computed in $O(N \cdot M)$ time using dynamic programming. Interested readers can find the derivation of the solution in several references, e.g., Myers *et al.* (1980, Section II.F).

The output of the DTW algorithm is quite rich: first of all, the value of the function $D(X, Y)$ (minimum global dissimilarity, or "DTW distance") can be assumed as the stretch-insensitive measure of the "inherent difference" between two given time series. This distance has a straightforward application in hierarchical clustering and classification (e.g., with k-NN classifiers). The shape of the warping curve ϕ itself provides information about which point matches which, i.e., the pairwise correspondences of time points can be easily inspected. Once obtained, the warping function can be applied to any time series, allowing one to inspect post-hoc aligned signals or measure time distortions.

3. Computing alignments

After loading the **dtw** package, alignments can be computed invoking the `dtw` function. In its simplest incarnation, the function takes two vector arguments representing the input time series, performs the minimization, and returns an object of class `dtw` encapsulating all the alignment information. The elements of the result can be retrieved with the usual `$` notation (see list in Table 1).

By default, the `dtw` function computes a global alignment, with no windowing, a symmetric local continuity constraint, and the Euclidean local distance. In particular, the symmetric continuity constraint implies that arbitrary time compressions and expansions are allowed, and that all elements must be matched:

$$\begin{aligned} |\phi_x(k+1) - \phi_x(k)| &\leq 1, \\ |\phi_y(k+1) - \phi_y(k)| &\leq 1. \end{aligned} \tag{3}$$

Several alternative forms for local continuity constraints however exist; Section 3.1 discusses them and how they can be selected.

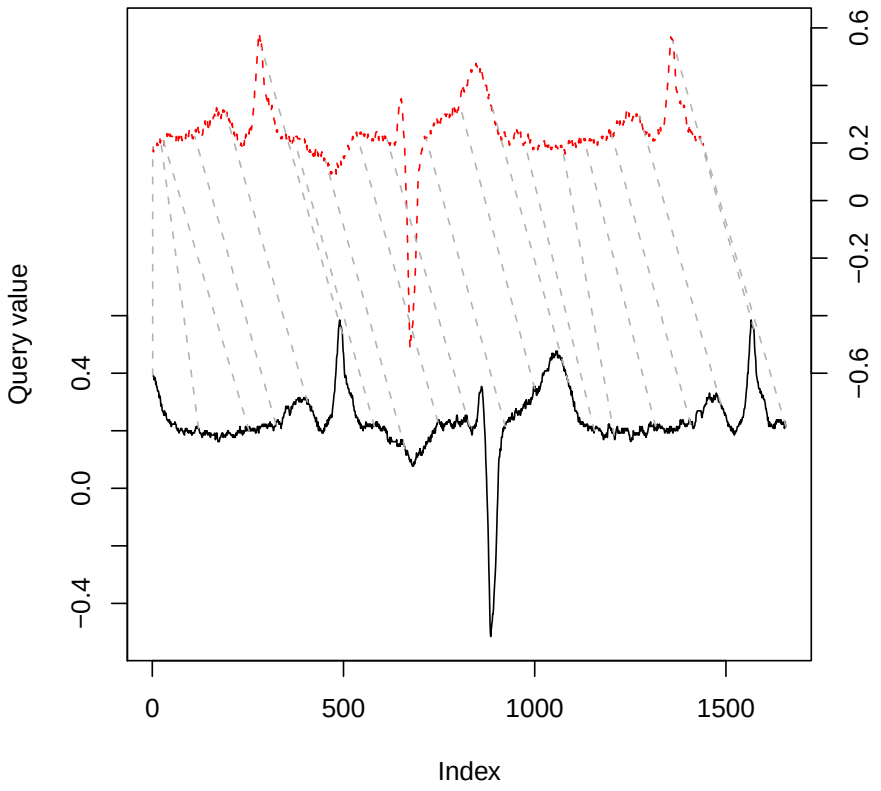

Figure 1: Aligning two time series: a simple example. A query (solid, left axis) and a reference (dashed, right axis) ECG time series, excerpted from `aami3a`.

Computing global alignments means that the time series' heads and tails are constrained to match each other. In other words, the following endpoint constraints are imposed:

$$\phi_x(1) = \phi_y(1) = 1; \tag{4}$$

$$\phi_x(T) = N; \quad \phi_y(T) = M. \tag{5}$$

As we shall see in Section 3.5, one or both of these constraints can be relaxed in order to compute partial time series matches.

As the example below shows, using `dtw` is straightforward.

Example 1 *The* `aami3a` *time series included in the package contains a reference electrocardiogram from the PhysioBank dataset (Goldberger* et al. *2000). We extract two non-overlapping windows from it, and compute their optimal alignment with the default* `dtw` *settings. The two extracted segments are shown in Figure 1.*

```
R> library("dtw")
R> data("aami3a")
```

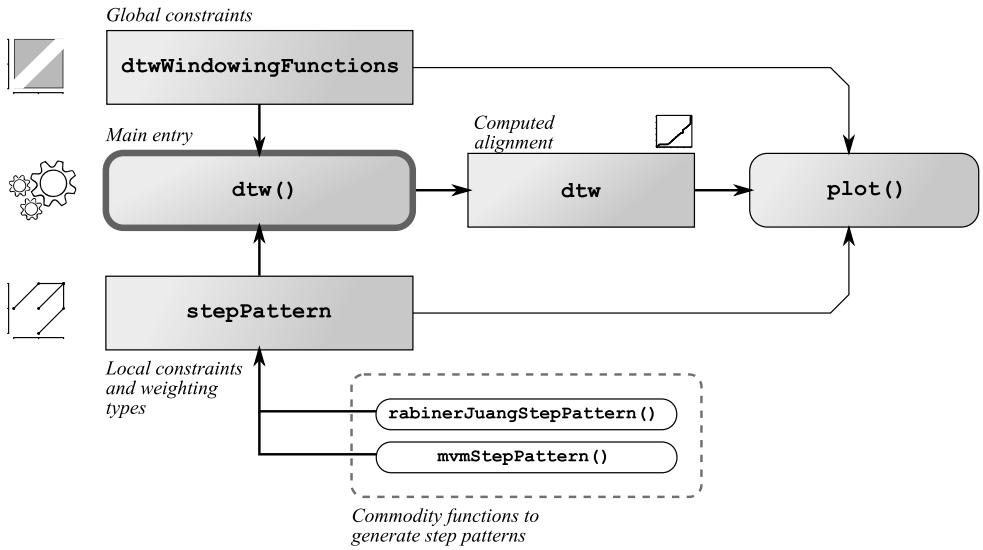

Figure 2: Block diagram of the main functions (rounded boxes) and object classes (rectangles) of the **dtw** package and their relationship. Arrows going in functions are arguments, outgoing are returned values.

```
R> ref <- window(aami3a, start = 0, end = 2)
R> test <- window(aami3a, start = 2.7, end = 5)
R> alignment <- dtw(test, ref)
R> alignment$distance
```

```
[1] 15.824
```

In `alignment$distance` one finds the cumulative (unnormalized) cost for the alignment, i.e., $M_\phi \cdot d_\phi(X, Y)$, while the warping functions ϕ_x and ϕ_y are found in components `$index1` and `$index2`. They are two integer vectors of the same length, listing the matching indices in the query and reference time series, respectively. Although the two vectors may be readily plotted with commands built-in in R, in Section 4 we shall show how this is accomplished even more easily with dedicated plotting commands.

3.1. Step patterns and local slope constraints

The elegant formulation of the DTW alignment problem lends itself to a variety of extensions. For example, one usually wants to limit the number of consecutive elements which are "skipped" in either time series, i.e., are left unmatched (skipping elements is often entirely disallowed by a continuity constraint, indeed). Alignments are in general achieved by *duplicating* elements, i.e., one lets a single time point in X match multiple (consecutive) elements in Y, or vice-versa. How many repeated elements can be matched consecutively, or how many can be skipped, put limits on the local slope of the warping curve. This property can be controlled by a very flexible scheme called *step patterns*. Step patterns list sets of allowed transitions between matched pairs, and the corresponding weights. In other words, step patterns specify the admissible values for $\phi(k+1)$ given $\phi(k), \phi(k-1)$, and so on. It may be useful to remark

Element	Description	Remark
`query`	Query time series, if given	*k,p*
`reference`	Reference time series, if given	*k,p*
`localCostMatrix`	Local distance matrix	
`stepPattern`	Step pattern instance used	
`N`	Length of the query time series	
`M`	Length of the reference time series	
`call`	Function call	
`distance`	Unnormalized minimum cumulative distance	
`normalizedDistance`	Normalized cumulative distance	*n*
`index1`	Warping function $\phi_x(k)$ for the query	*d*
`index2`	Warping function $\phi_y(k)$ for the reference	*d*
`costMatrix`	Computed cumulative cost matrix	*k*
`directionMatrix`	Transition chosen at each alignment point	*k*
`jmin`	End-point, for partial alignments	

Table 1: Elements in a `dtw` result object, as of package version 1.13-1. Remarks: *k* – requires `keep = TRUE`; *n* – requires that the chosen step pattern is normalizable; *d* – unless `distance.only = TRUE`; *p* – unless a local distance matrix is used as input .

that in DTW there is no additive penalty for duplicating or skipping elements, as with other alignment algorithms like Smith-Waterman's or Levenshtein's.

While most authors stay with the simplest recursion types, users of **dtw** have access to almost the full range of step patterns defined in the literature, with no programming burden. Step patterns are represented by objects of class `stepPattern`. Patterns are selected by passing an appropriate instance to the `step.pattern` argument of the `dtw` function call, as in the following example:

Example 2 *Compute the same alignment of the previous example, assuming the well-known* asymmetric *pattern (bottom left of Figure 3).*

```
R> alignment <- dtw(test, ref, step.pattern = asymmetric)
R> alignment$distance

[1] 11.497
```

When printed, `stepPattern` objects display the corresponding DTW recursion in human-readable form; they can also be plotted via the `plot()` overloaded method, and transposed via `t()`. Transposing a pattern means that the role of the query and reference are interchanged.

Example 3 *Display the recursion formula for the properly symmetric continuity constraint (top right in Figure 3), also known as the* symmetric $P = 0$ *from Sakoe and Chiba (1978).*

```
R> symmetric2

Step pattern recursion:
g[i,j] = min(
```

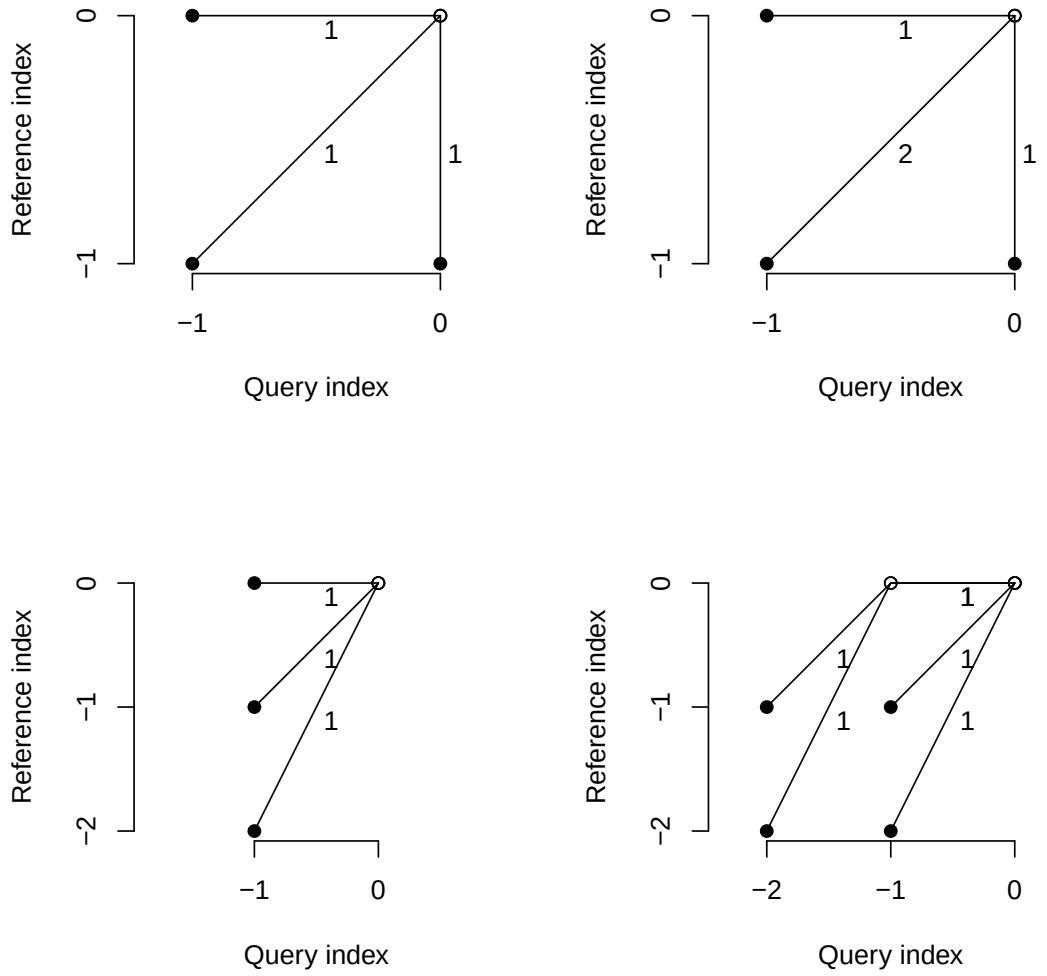

Figure 3: Four well-known step patterns. Left to right, top to bottom: `symmetric1`, `symmetric2`, `asymmetric`, `rabinerJuangStepPattern(4, "c", TRUE)` (i.e., Rabiner-Juang's type IV with slope weighting c). Numbers on transitions indicate the multiplicative weight m_ϕ for the local distance $d(i, j)$, should the corresponding step be followed. See `?stepPattern` for the full list.

```
    g[i  ,j-1] +     d[i  ,j  ] ,
    g[i-1,j-1] + 2 * d[i  ,j  ] ,
    g[i-1,j  ] +     d[i  ,j  ] ,
)

Normalization hint: N+M
```

The so-called *symmetric* recursion printed above allows an unlimited number of elements

of the query to be matched to a single element of the reference, and vice-versa; in other words, there is no limit in the amount of time expansion or compression allowed at any point. Once the package is loaded, an instance representing this recursion is pre-defined under the name `symmetric2`, which is also the default. For this recursion, the average cost per-step is computed by dividing the cumulative distance by $N + M$, where N is the length of the query sequence and M is the length of the reference. Other step patterns require different normalization formulas, as we shall see in Section 3.2. The normalized distance, if defined, is stored in the `$normalizedDistance` component of the result.

Several step patterns have been discussed in the literature. A classic paper by Sakoe and Chiba (1978) classifies them according to two properties: their symmetry (symmetric/asymmetric), and the bounds imposed on the slope expressed through a parameter P. The eight step patterns shown in Sakoe and Chiba (1978, Table I) are pre-defined in **dtw**, with names `symmetricP1`, `asymmetricP05`, and so on.[1] All of them are normalizable.

Rabiner and Juang (1993, Chapter 4) introduced a different classification with three attributes: local continuity constraint type (in Roman numerals, I to VII); slope weighting (Latin letters a to d); and their being "smoothed" or not (boolean). All of Rabiner's step patterns are available through the three-argument function `rabinerJuangStepPattern()`. Slope weighting types c and d are normalizable.

Yet another classification (now obsoleted by the previous one) follows Myers *et al.* (1980); it is similar to Rabiner's, except that only four types (I)-(IV) are defined. The corresponding instances are named like `typeIId`, `typeIIIc` and so on. The latter is a good approximation to the slope-limited pattern proposed by Itakura (1975).

It should be noted that, in general, step patterns impose a lower and/or an upper bound to the *local* slope of the alignment. In other words, they limit the maximum amount of time stretch and compression allowed at any point of the alignment. For example, the `asymmetric` step pattern limits time expansion to a factor of two; it would therefore be impossible to completely align a query with a reference more than twice as long. The Rabiner-Juang type IV, instead, generates the so-called Itakura parallelogram (Itakura 1975).

3.2. Normalization

Computing the average per-step distance along the warping curve is especially important in two cases: (1) when comparing alignments between time series of different lengths, to decide the best match (e.g., for classification); and (2) when performing partial matches. Each step patterns requires a different normalization function. In **dtw**, objects of class `stepPattern` "know" the proper normalization formula they require, called "normalization hint".

Not all step patterns are normalizable. For example, the *quasi-symmetric* recursion (top left in Figure 3, instance name `symmetric1`), like the symmetric one, does not restrict the slope; however, it favors "diagonal" steps over stair-stepping paths. Since similar warping curves have different weights, a path-independent per-step alignment cost is not defined. As a consequence, only the cumulative distance value is available in the `$distance` component of the result.

Table 2 lists the currently supported normalization rules, grouped by weighting type. "R-J" types follow Rabiner's aforementioned slope weight classification (cf. Figure 7 in Myers *et al.*

[1] Sakoe's "asymmetric" property should not be confused with the `asymmetric` step pattern.

Slope weighting (Rabiner-Juang)	Symmetry (Sakoe-Chiba)	Formula	Hint
R-J type (a)	—	—	`NA`
R-J type (b)	—	—	`NA`
R-J type (c)	S-C asymmetric	n	`N`
R-J type (d)	S-C symmetric	$n+m$	`N + M`
Type (c′)	S-C asymmetric	m	`M`
(Others)		—	`NA`

Table 2: Normalization functions for well-known weighting types.

(1980)), while "S-C" refers to the symmetric/asymmetric categories introduced by Sakoe-Chiba (cf. Table I in Sakoe and Chiba (1978)). Type (c′) includes asymmetric patterns similar to R-J type (c) with test and reference interchanged. Step patterns of this category have been used e.g., by Mori *et al.* (2006) (namely, instance `mori2006`), and by Oka (1998). Values n and m are the number of elements actually matched in the query and reference, respectively, i.e.,

$$\begin{aligned} n &= \phi_x(T) - \phi_x(1) + 1 \\ m &= \phi_y(T) - \phi_y(1) + 1. \end{aligned}$$

For global alignments, they are equal to the lengths of the input time series, i.e., $n = N$ and $m = M$.

3.3. A note on indexing conventions and axes

As a general rule, we stick to the convention that the first argument and indices refer to the query time series, and the second to the reference. This implies that when we print alignment-related matrices such as $d(i, j)$, the query index grows row-wise towards the bottom. The reader should not confuse this layout with plot axes, where the query and reference are usually arranged along the abscissa and ordinate respectively (Figure 4).

3.4. Windowing and global constraints

A *global* constraint, or *window*, explicitly forbids warping curves to enter some region of the (i, j) plane. A global constraint translates an a-priori knowledge about the fact that the time distortion is limited. For example, the well-known Sakoe-Chiba band (Sakoe and Chiba 1978) enforces the additional constraint

$$|\phi_x(k) - \phi_y(k)| \leq T_0$$

where T_0 is the maximum allowable absolute time deviation between two matched elements. Intuitively, the constraint creates an allowed band of fixed width about the main diagonal of the alignment plane (Figure 5).

In **dtw**, windowing constraints are enabled through the `window.type` argument of the `dtw` call; it can be either a character string (e.g., `"sakoechiba"`) or a function, specifying the shape of the allowed window. The window size, T_0, is passed through the `window.size` argument.

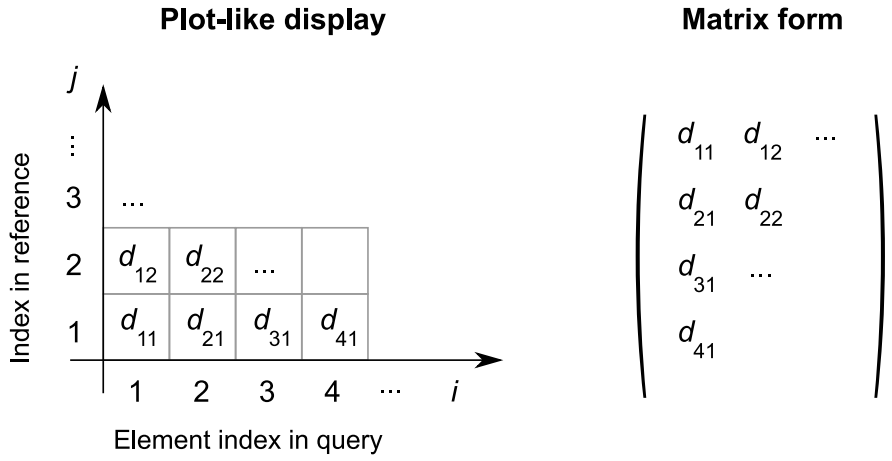

Figure 4: Different conventions for displaying distance matrices: plot-like (left) and matrix (right) arrangements.

Attribute	Section in R-J	Documentation
Local continuity constraints	4.7.2.3	`?stepPattern`
Global path constraints	4.7.2.4	`?dtwWindowingFunctions`
Slope weighting	4.7.2.5	`?stepPattern`

Table 3: Where to find documentation for various DTW parameters. R-J is the book by Rabiner and Juang (1993).

It should be remarked that the Sakoe-Chiba band works well when $N \sim M$, but is inappropriate when the lengths of the two inputs differ significantly. In particular, when $|N - M| > T_0$, the (N, M) endpoint lies outside of the band, thus violating (5), and therefore no solution exists for a global alignment. In general, when enforcing global and/or local constraints one should take care that they are compatible with each other and with the time series' lengths.

The `slantedBandWindow` creates a band centered around the jagged line segment which joins element $(1, 1)$ to element (N, M), and is T_0 elements wide along the first axis. 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.

Arbitrary windows can be specified by supplying a function that takes (at least) the i and j integer arguments, and returns a boolean value specifying whether that match is allowed or not. Arguments unused in `dtw` are passed to the windowing function. More detail on windowing functions, including how to plot them (as in Figure 5), are given in `?dtwWindowingFunctions`.

```
R> dtwWindow.plot(sakoeChibaWindow, window.size = 2, reference = 17,
+       query = 13)
```

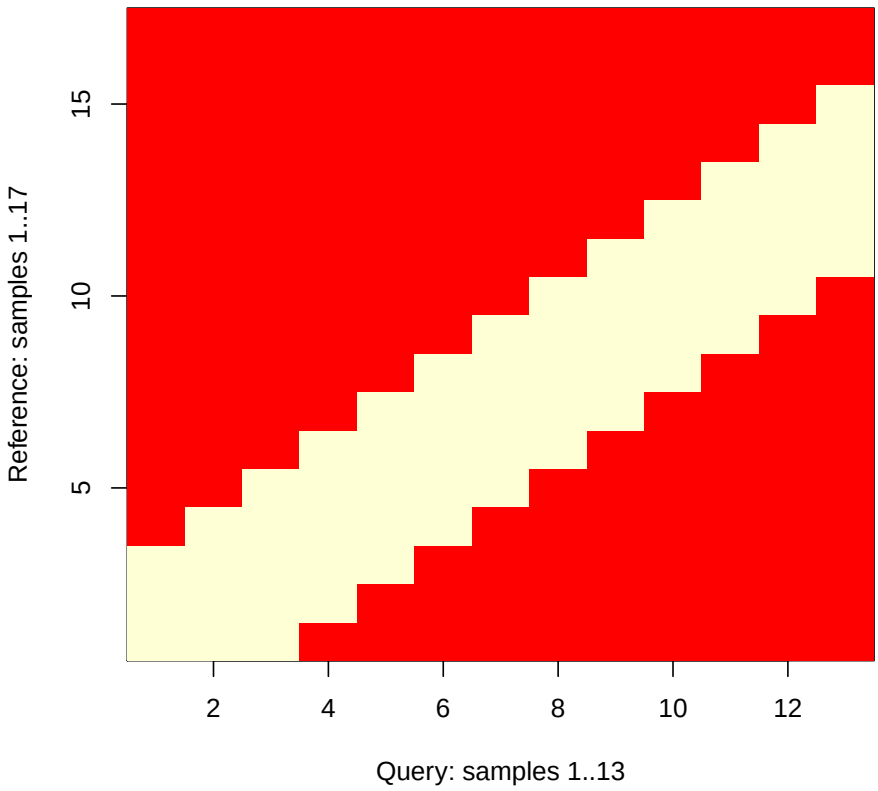

Figure 5: The allowed region for the warping curve under the `"sakoechiba"` global constraint. In this degenerate case, element (N, M) at the upper-right is outside of the band, so the endpoint constraint (5) can't be satisfied: this band is too narrow to be compatible with any global alignment.

3.5. Unconstrained endpoints: Prefix and subsequence matches

Normally, the DTW distance is understood as a *global* alignment, i.e., subject to the conditions (4) and (5). In certain applications, *partial* matches are useful, and one consequently relaxes one or both the constraints. Interested readers can find a review of partial matching algorithms, applications, and the interplay with normalization, in a paper by Tormene *et al.* (2009).

The open-end (OE-DTW) or prefix-matching algorithm is defined by relaxing the end-point constraint (5). The algorithm therefore returns the *prefix* of the reference which best matches the query; it has been used e.g., by Sakoe (1979) and Mori *et al.* (2006). Open-end matching is achieved, in principle, by constructing several incomplete versions $Y^{(p)}$ of the reference Y, each truncated at the index $p = 1, \ldots, M$. One then computes their corresponding DTW distances from X, and picks the best match:

$$Y^{(p)} = (y_1, \ldots, y_p)$$

$$D_{OE}(X,Y) = \min_{1\leq j\leq M} D(X,Y^{(j)}) \tag{6}$$

It is worthwhile noting that in Equation 6 one compares alignments of different lengths with each other. This is only meaningful if the *average* per-step distances are considered, and it is therefore essential to use normalizable step patterns.

In **dtw**, open-end alignment of time series is achieved simply setting the `open.end = TRUE` parameter. The element `$jmin` in the result will hold the size of the prefix matched, i.e., the j that minimizes (6). This is the index of the last element matched in the reference, which in turn equals $\phi_y(T)$. Accordingly, when `oc` is a partial alignment result, `oc$jmin` equals `max(oc$index2)`.

Subsequence match, also called "unconstrained" or open-begin-end (OBE-DTW), is achieved relaxing *both* the start-point constraint in (4) and the end-point constraint of (5). Intuitively, subsequence matching discovers the contiguous part of the template which best matches the *whole* query (Figure 6). This form is used e.g., by Rabiner *et al.* (1978) as algorithm UE2-1; by Sakurai *et al.* (2007); and others. Formally, we define $Y^{(p,q)}$ as the subsequence of Y including elements y_j with $p \leq j \leq q$; the OBE-DTW problem is to minimize the cumulative distance over the initial and final reference indices p and q simultaneously:

$$\begin{aligned} Y^{(p,q)} &= (y_p,\ldots,y_q) \\ D_{OBE}(X,Y) &= \min_{1\leq p\leq q\leq M} D(X,Y^{(p,q)}) \end{aligned} \tag{7}$$

In **dtw**, OBE-DTW alignments are computed setting both the `open.begin` and `open.end` arguments to `TRUE`.[2] Open-begin alignments are supported for step patterns with n-type normalization only.

3.6. Dealing with multivariate time series

As seen in Section 2, the actual time series values only enter the DTW algorithm through their cross-distance matrix, i.e., matrix d in Equation 1. In practice, the choice for the local distance function f influences how "strongly" the alignment will avoid mismatching regions. A variety of dissimilarity functions are available, e.g., the Euclidean, squared Euclidean, Manhattan, Gower coefficient, and many others. Each of them takes two arguments, which may be multi-variate; for example, in speech recognition it is customary to align high-dimensional "frames", whose components are short-time spectral coefficients, or similar quantities. The **proxy** package maintains a database of distance definitions which can be used in cross-distance computations (see `summary(pr_DB)`).

To deal with multivariate time series, one provides `dtw` with two matrices X_{ic} and Y_{jc}, rather than two vectors as for the single-variate case. The time indices $i = 1\ldots N$ and $j = 1\ldots M$ are arranged along rows, while multivariate dimensions $c = 1\ldots C$ are arranged in columns. Multivariate time series objects (R class `mts`) can be used, as well.

By default, `dtw` assumes an Euclidean local distance, i.e.,

[2] The reader may wonder about the last remaining combination: constraining the ends of the time series but not their heads. This is also allowed, but not really necessary: a partial alignment with the query sequence reversed achieves the same effect.

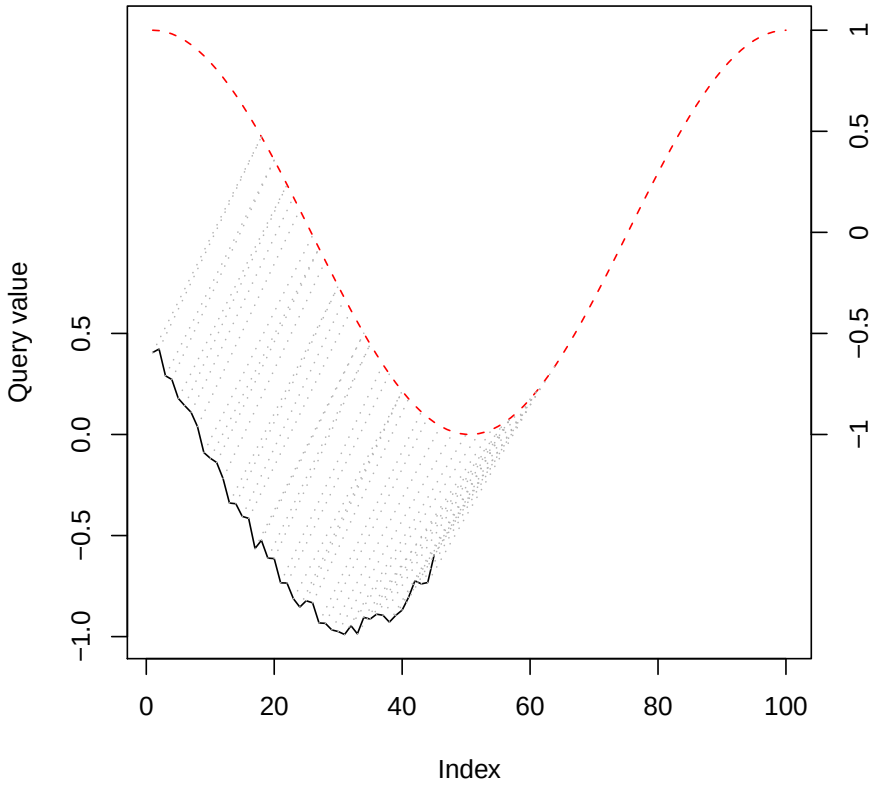

Figure 6: A partial alignment with both endpoints free: the whole query (solid line) is warped into a contiguous subsequence of the template (dashed). Partial alignments are computed setting the `open.begin = TRUE` and/or `open.end = TRUE` arguments. Code available in `example(dtw)`.

$$d(i,j)^2 = \sum_{c=1}^{C} (X_{ic} - Y_{jc})^2$$

Other distance functions can be selected through the `dist.method` character argument, which is forwarded to the `proxy::dist` function.

Alternatively, the user can compute the local distance matrix by herself, and supply that instead of the time series. This is achieved invoking `dtw` with *one* matrix argument rather than two. According to the convention in Section 3.3, matrix rows (first index) are understood as the index in the query sequence, and matrix columns as the index in the reference. The following examples show both the two- and the one-argument call styles.

Example 4 *Align two synthetic bivariate time series (a query with 10 time points, and a reference of length 5), assuming the Manhattan local distance for element pairs:*

```
R> query <- cbind(1:10, 1)
R> ref <- cbind(11:15, 2)
R> dtw(query, ref, dist.method = "Manhattan")$distance
```

```
[1] 83
```

or equivalently, pre-computing `proxy::dist`

```
R> cxdist <- proxy::dist(query, ref, method = "Manhattan")
R> dtw(cxdist)$distance
```

```
[1] 83
```

The optimal alignment of *strings*, i.e., discrete values or *categorial* data, rather than of sequences of real values, is an ubiquitous task in bioinformatics, closely related to DTW. Functions to perform alignments of strings, like Levenshtein's distance or the Needleman-Wunsch algorithm, are implemented, e.g., in package **cba** (cf. function `sdists`), package **TraMineR** (Gabadinho *et al.* 2009), and in **Bioconductor** (Gentleman *et al.* 2004). The DTW algorithm can be used to align strings, as well, by defining a local distance function over all the pairs of symbols that can be formed in the alphabets considered. A cross-distance matrix can then be obtained from this function, and used in the single-argument call of `dtw`, as in the last example.

3.7. Computing several alignments at once

A common situation, arising e.g., in clustering and classification of time series, is to compute several DTW distances between pairs of elements in a database. For instance, we assume to have a database $\{Q_k\}$ of K time series, each of which is a single-variate vector $Q_k = (q_{k1}, \ldots, q_{kN})$. We also assume that all Q_k have the same length N. The database can therefore be encoded as a $K \times N$ matrix, $\{Q_k\} = q_{ki}$. To compute DTW alignments between all possible couples, we iterate two indices h and k over the rows of q, thus obtaining a $K \times K$ self-dissimilarity matrix:

$$\Lambda_{kh} = D(Q_k, Q_h)$$

where D is the dynamic time warping distance of Equation 2. It should be noted that, since the DTW distance is not in general symmetric, Λ will not be, either.

As seen above, the `dtw` function leverages the **proxy** package for building the local distance matrix. There is, however, another important relationship between the two packages. Since dynamic time warping itself *is* a dissimilarity function between vectors (understood as time series), **dtw** registers itself as a distance function in the database of distances `pr_DB`. This feature makes it straightforward to compute many-versus-many alignments. Once the package is loaded, if q_{ki} is given as a matrix `q` (again, time series are arranged in rows), Λ can be straightforwardly computed through `proxy::dist(q, q, method = "DTW")`.[3] Computing $K \times K$ alignments in this way is faster than iterating over the plain `dtw` call, because only

[3]We use the two-argument form because the single-argument form would return a symmetric distance matrix, while DTW is not – in general – symmetric.

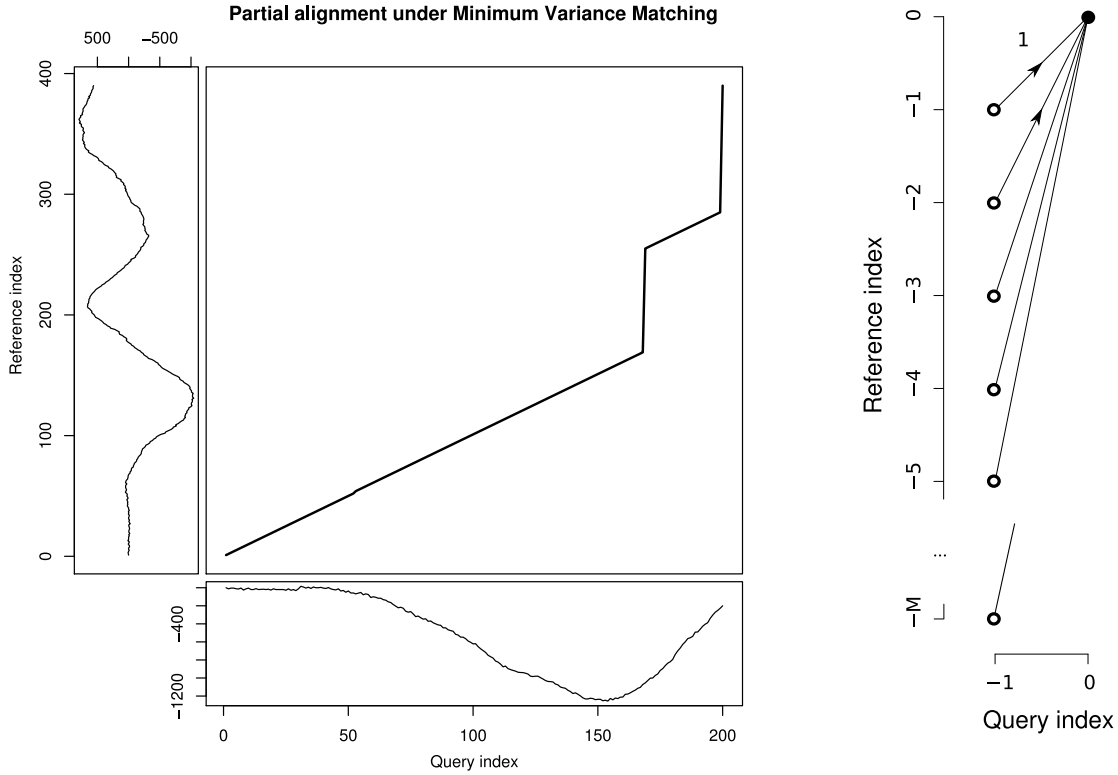

Figure 7: Left: MVM alignment of a truncated query; "jumps" may occur in the middle of the query, e.g., around time 175. Note that since time compression is not allowed, the valley at query index 150 does not match index 125 in the reference. Right: a type (c) DTW step pattern equivalent to the MVM search algorithm.

the numeric value of the distance is computed, while the construction of the actual warping path is bypassed.

Equally useful is the cross-distance form of `dist`, which can be used, for instance, to classify K query time series at once (arranged in a matrix `q`) against a given database of L templates (matrix `p`). Once the $K \times L$ cross-distance matrix is computed by `dist(q, p, method = "DTW")`, one could take column-wise minima to discover the template best matching each of the K query elements. The dissimilarity matrices resulting from calls to `dist` can also be fed into the usual clustering functions.

3.8. Minimal variance matching

Latecki *et al.* (2007) proposed the minimal variance matching (MVM) algorithm to align a given test to a reference, allowing arbitrary portions of the template to be skipped (Figure 7, left-hand side). Interestingly, MVM may be considered a special case of DTW with a large step pattern (Figure 7, right-hand side).

Like the asymmetric step pattern, the MVM algorithm constrains each query element to match exactly one time point in the template, so the degenerate empty-to-empty match is

not allowed. Vice-versa, an arbitrary number of template elements can remain unmatched. Matched reference points must be in strict increasing order, although arbitrary gaps are allowed in the sequence. This translates into the continuity constraints

$$\begin{aligned} \phi_x(k) &= k, \qquad \text{with} \qquad k = 1, \ldots, N \\ \phi_y(k) &< \phi_y(k+1) \end{aligned}$$

Due to the strictly-increasing requirement of the reference index, the minimum local slope of the warping curve is unity, and time compression is therefore ruled out. MVM matching is therefore undefined if the query is longer than the reference, and only one alignment exists if their lengths are equal.

In **dtw**, step pattern instances implementing MVM can be built through the function `mvmStepPattern()`. The `elasticity` integer argument limits the maximum length of a contiguous subsequence which can be skipped in the reference time series. If no limit is desired, elasticity should be made at least as large as the reference length.

3.9. A worked-out exercise

As an additional example, we solve Exercise 4.7 in Rabiner and Juang (1993, page 226). The first question is to find the best path through a 6×6 local distance matrix that is explicitly given in the text. We enter the given matrix as follows:

```
R> lm <- matrix(nrow = 6, ncol = 6, byrow = TRUE, c(
+    1, 1, 2, 2, 3, 3,
+    1, 1, 1, 2, 2, 2,
+    3, 1, 2, 2, 3, 3,
+    3, 1, 2, 1, 1, 2,
+    3, 2, 1, 2, 1, 2,
+    3, 3, 3, 2, 1, 2
+  ))
```

Note that we entered `lm` according to matrix conventions: the first subscript indexes elements in the query time series, while the second indexes the reference (right hand side of Figure 4). Therefore, the query is understood to grow row-wise towards the bottom, while the reference goes column-wise towards the right. Conversely, the exercise text displays the table in plot-like conventions (i_x growing rightwards and i_y upwards).

The exercise requires a specific step pattern which is readily identified as the `asymmetric` recursion (bottom left of Figure 3). To solve the problem and find the best *global* path going through the grid, we invoke `dtw` with one matrix argument. From the result, we extract the cost matrix (see next section) and the normalized distance, 7/6, thus solving the problem:

```
R> alignment <- dtw(lm, step = asymmetric, keep = TRUE)
R> alignment$costMatrix
```

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1   NA   NA   NA   NA   NA
```

```
[2,]    2    2    2   NA   NA   NA
[3,]    5    3    4    4    5   NA
[4,]    8    4    5    4    5    6
[5,]   11    6    5    6    5    6
[6,]   14    9    8    7    6    7
```

```
R> alignment$normalizedDistance
```

```
[1] 1.166667
```

A follow-up question of the same exercise requires the optimal alignment between the whole test and any *prefix* of the reference; in other words, we lift the constraint on the reference end-point. This is achieved with the setting `open.end = TRUE`. As above, we invoke `dtw` in its matrix form; the optimal end-point (that is, $\phi_y(T)$) is found in component `$jmin` of the result.

```
R> alignmentOE <- dtw(lm, step = asymmetric, keep = TRUE, open.end = TRUE)
R> alignmentOE$jmin
```

```
[1] 5
```

```
R> alignmentOE$normalizedDistance
```

```
[1] 1
```

The exercise is thus solved.

3.10. Retrieving cost matrices

If the parameter `keep.internals` is `TRUE`, the local distance matrix and the cumulative cost matrix are preserved after the calculation, and stored in the result elements `$localCostMatrix` and `$costMatrix`, respectively. The matrices can be printed and manipulated as usual.

Figure 8, for example, shows how to display them along with the alignment path superimposed. In the left panel one can hand-check the result of the exercise solved in the previous section: no better warping path exists that passes by $(1, 1)$ through $(6, 6)$ under the constraints of the `asymmetric` pattern, and the cumulative cost is indeed found in the upper-right element of the right panel. An analogous plot style, suitable for larger alignments, is presented in Section 4.3.

```
R> lcm <- alignment$localCostMatrix
R> image(x = 1:nrow(lcm), y = 1:ncol(lcm), lcm)
R> text(row(lcm), col(lcm), label = lcm)
R> lines(alignment$index1, alignment$index2)

R> ccm <- alignment$costMatrix
R> image(x = 1:nrow(ccm), y = 1:ncol(ccm), ccm)
R> text(row(ccm), col(ccm), label = ccm)
R> lines(alignment$index1, alignment$index2)
```

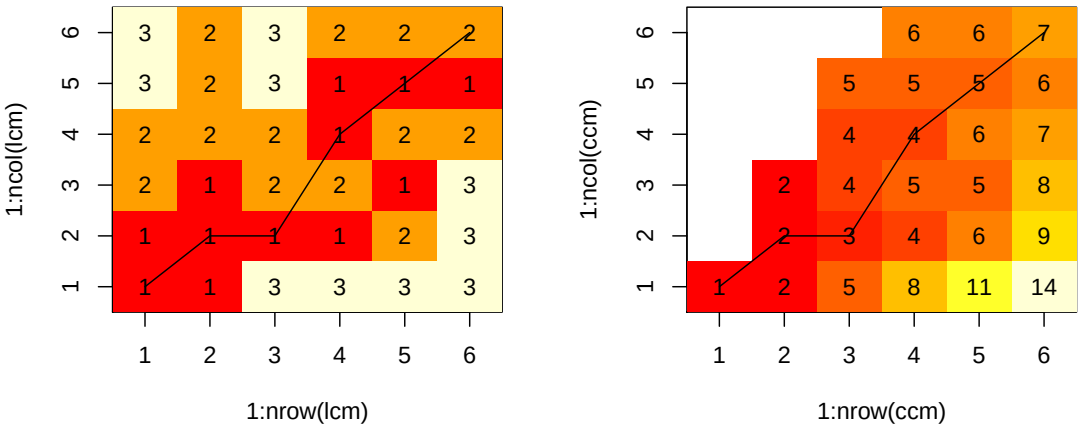

Figure 8: The local distance matrix for the exercise in Section 3.9 (left) and the corresponding cumulative distance matrix (right) under the symmetric step pattern. The optimal alignment path is superimposed.

4. Displaying alignments

Plots are valuable tools to inspect time series pairs together with their alignments. Several plotting styles are available in **dtw** for producing publication-quality figures. They are achieved through the `plot` method overloaded for objects of type `dtw`.

The first two plot styles discussed below, namely two- and three-way, are used to inspect the time series themselves along with their alignment. For this reason, the two actual input time series must be available, and they have to be single-variate. For convenience, time series are retrieved from the alignment object itself, if available (elements `$query` and `$reference`); they can be overridden via the `xts` and `yts` plot arguments.

A *density* plot style is also available to display the relative costs of different warping curves. It builds on the global cost matrix only, and does not therefore require the knowledge of the two original time series.

4.1. Two-way plotting

One intuitive alignment visualization style places both time series in the same plane, and connects the matching point pairs with segments (see e.g., Figure 1). This plot style is selected passing the argument `type = "twoway"` to the plot call. The optional numeric argument `offset` can be used to visually separate query and reference time series; if set, the scales for the reference time series are displayed on the secondary (right-hand) vertical axis. Other options affecting the visual appearance are explained in the `?dtwPlotTwoWay` manual page.

Example 5 *Generate the plot in Figure 1.*

```
R> library("dtw")
```

```
R> data("aami3a")
R> ref <- window(aami3a, start = 0, end = 2)
R> test <- window(aami3a, start = 2.7, end = 5)
R> plot(dtw(test, ref, k = TRUE), type = "two", off = 1, match.lty = 2,
+      match.indices = 20)
```

4.2. Three-way plotting

Another effective layout to display alignments places the query time series horizontally in a small lower panel, the reference time series vertically on the left; a larger inner panel holds the warping curve (Figure 7). In this way, matching points can be recovered by tracing indices on the query time series, moving upwards until the warping curve is met, and then moving leftwards to discover the index of the reference matched. The advantage of this method is that the warping curve is directly exposed, so it becomes easy to visualize the impact of the local and global constraints.

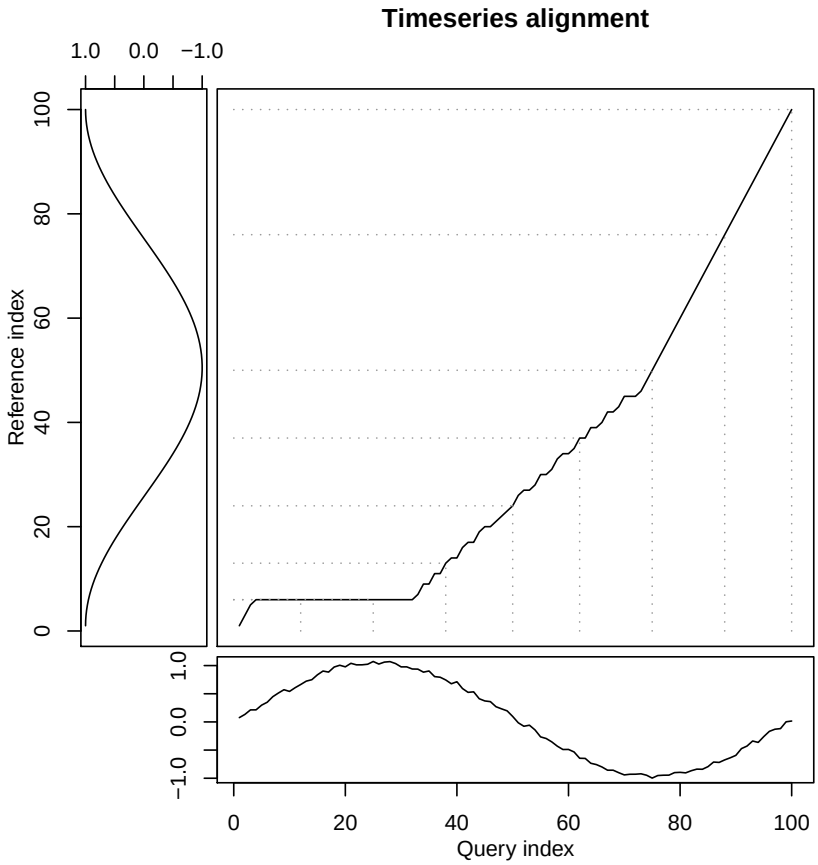

Figure 9: Three-way plot of the `asymmetric` alignment between a noisy sine and a cosine in $[0, 2\pi]$, with visual guide lines drawn every $\pi/4$. Code available in `example(dtwPlotThreeWay)`.

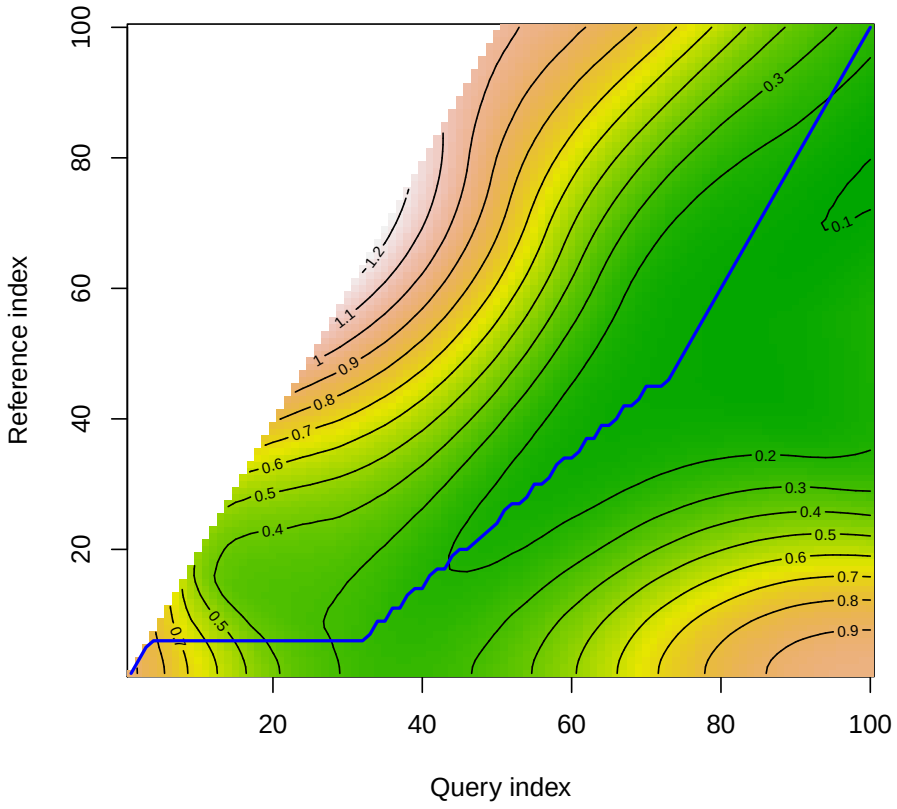

Figure 10: Cost density plot: average per-step cost density of the sine-cosine global alignment. The local constraints and weighting are chosen from the `asymmetric` step pattern. Code in `example(dtwPlotDensity)`.

Three-way plots can be achieved with the `type = "threeway"` argument to the `plot` call. Match lines can also be explicitly visualized, as shown in Figure 9; documentation for all options is available in `?dtwPlotThreeWay`.

4.3. Displaying the cost density

A third plot style displays the "cost density" of alignments, and can therefore be useful to inspect qualitatively how much "slack" is present around the optimal alignment (Figure 10). By default, the *cumulative* distance distribution is displayed as a density distribution with contours superimposed. The result is a terse display of the cumulative distance matrix, analogous to the right panel of Figure 8.

For normalizable step patterns, a plot of the *per-step* density can also be requested with the `normalize = TRUE` argument. Since the density plot is based on the cumulative distance matrix, `keep = TRUE` is required in the `dtw` function call. Density plotting is selected with the `type = "density"` parameter to the plot call, and documented in `?dtwPlotDensity`.

Computational details

The computing kernel of the `dtw` function is written in the C programming language for efficiency. Alignment computations are quite fast, as long as the cross-distance matrices fit in the machine's RAM. A standard quad-core Linux x86-64 PC with 4 GB of RAM and 4 GB of swap computes (using only one core) an unconstrained alignment of 100×100 time points in 7 ms, 6000×6000 points in under 10 s, and 8000×8000 points (close to the virtual memory limit) in 10 minutes. Larger problems may be addressed by approximate strategies, e.g., computing a preliminary alignment between downsampled time series (Salvador and Chan 2004); indexing (Keogh and Ratanamahatana 2005); or breaking one of the sequences into chunks and then iterating subsequence matches.

The results and plots in this paper have been obtained using R 2.9.1 with the packages **dtw** 1.13-1 and **proxy** 0.4-3 (Meyer and Buchta 2009). R itself and all packages used are available from the Comprehensive R Archive Network at http://CRAN.R-project.org/.

Acknowledgments

I am grateful to P. Tormene for testing the package extensively. Thanks to M. J. Harvey and the anonymous reviewers for their valuable suggestions.

References

Aach J, Church GM (2001). "Aligning Gene Expression Time Series with Time Warping Algorithms." *Bioinformatics*, **17**(6), 495–508.

Faundez-Zanuy M (2007). "On-Line Signature Recognition Based on VQ-DTW." *Pattern Recognition*, **40**(3), 981–992.

Gabadinho A, Ritschard G, Studer M, Müller NS (2009). "Mining Sequence Data in R with **TraMineR**: A User's Guide." *Technical report*, Department of Econometrics and Laboratory of Demography, University of Geneva, Geneva. URL http://mephisto.unige.ch/TraMineR/.

Gentleman RC, Carey VJ, Bates DM, Bolstad B, Dettling M, Dudoit S, Ellis B, Gautier L, Ge Y, Gentry J, Hornik K, Hothorn T, Huber W, Iacus S, Irizarry R, Leisch F, Li C, Maechler M, Rossini AJ, Sawitzki G, Smith C, Smyth G, Tierney L, Yang JYH, Zhang J (2004). "**Bioconductor**: Open Software Development for Computational Biology and Bioinformatics." *Genome Biology*, **5**, R80. URL http://genomebiology.com/2004/5/10/R80.

Giorgino T, Tormene P (2009). ***dtw**: Dynamic Time Warping Algorithms*. R package version 1.13-1, URL http://CRAN.R-project.org/package=dtw.

Goldberger AL, Amaral LA, Glass L, Hausdorff JM, Ivanov PC, Mark RG, Mietus JE, Moody GB, Peng CK, Stanley HE (2000). "PhysioBank, PhysioToolkit, and PhysioNet: Components of a new Research Resource for Complex Physiologic Signals." *Circulation*, **101**(23), E215–E220.

Gollmer K, Posten C (1996). "Supervision of Bioprocesses Using a Dynamic Time Warping Algorithm." *Control Engineering Practice*, **4**(9), 1287–1295.

Hermans F, Tsiporkova E (2007). "Merging Microarray Cell Synchronization Experiments Through Curve Alignment." *Bioinformatics*, **23**(2), 64–70. doi:10.1093/bioinformatics/btl320.

Huang B, Kinsner W (2002). "ECG Frame Classification Using Dynamic Time Warping." In W Kinsner, A Sebak, K Ferens (eds.), *Proceedings of the Canadian Conference on Electrical and Computer Engineering – IEEE CCECE 2002*, volume 2, pp. 1105–1110. IEEE Computer Society, Los Alamitos, CA, USA. doi:10.1109/CCECE.2002.1013101.

Itakura F (1975). "Minimum Prediction Residual Principle Applied to Speech Recognition." *IEEE Transactions on Acoustics, Speech, and Signal Processing*, **23**(1), 67–72.

Kartikeyan B, Sarkar A (1989). "Shape Description by Time Series." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **11**(9), 977–984. doi:10.1109/34.35501.

Keogh E, Ratanamahatana CA (2005). "Exact Indexing of Dynamic Time Warping." *Knowledge and Information Systems*, **7**(3), 358–386. doi:10.1007/s10115-004-0154-9.

Latecki LJ, Megalooikonomou V, Wang Q, Yu D (2007). "An Elastic Partial Shape Matching Technique." *Pattern Recognition*, **40**(11), 3069–3080.

Meyer D, Buchta C (2009). ***proxy**: Distance and Similarity Measures*. R package version 0.4-3, URL http://CRAN.R-project.org/package=proxy.

Mori A, Uchida S, Kurazume R, Taniguchi R, Hasegawa T, Sakoe H (2006). "Early Recognition and Prediction of Gestures." In B Werner (ed.), *Proceedings of the 18th International Conference on Pattern Recognition – ICPR 2006*, volume 3, pp. 560–563. IEEE Computer Society, Los Alamitos, CA, USA. doi:10.1109/ICPR.2006.467.

Myers C, Rabiner L, Rosenberg A (1980). "Performance Tradeoffs in Dynamic Time Warping Algorithms for Isolated Word Recognition." *IEEE Transactions on Acoustics, Speech, and Signal Processing*, **28**(6), 623–635.

Oka R (1998). "Spotting Method for Classification of Real World Data." *The Computer Journal*, **41**(8), 559–565. doi:10.1093/comjnl/41.8.559.

Rabiner L, Juang BH (1993). *Fundamentals of Speech Recognition*. Prentice-Hall, Upper Saddle River, NJ, USA.

Rabiner L, Rosenberg A, Levinson S (1978). "Considerations in Dynamic Time Warping Algorithms for Discrete Word Recognition." *IEEE Transactions on Acoustics, Speech, and Signal Processing*, **26**(6), 575–582.

Rath TM, Manmatha R (2003). "Word Image Matching Using Dynamic Time Warping." In R Manmatha (ed.), *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pp. II–521–II–527. IEEE Computer Society, Los Alamitos, CA, USA.

R Development Core Team (2009). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

Sakoe H (1979). "Two-Level DP-matching – A Dynamic Programming-Based Pattern Matching Algorithm for Connected Word Recognition." *IEEE Transactions on Acoustics, Speech, and Signal Processing*, **27**(6), 588–595.

Sakoe H, Chiba S (1971). "A Dynamic Programming Approach to Continuous Speech Recognition." In *Proceedings of the Seventh International Congress on Acoustics*, volume 3, pp. 65–69. Akadémiai Kiadó, Budapest.

Sakoe H, Chiba S (1978). "DynaMic Programming Algorithm Optimization for Spoken Word Recognition." *IEEE Transactions on Acoustics, Speech, and Signal Processing*, **26**(1), 43–49.

Sakurai Y, Faloutsos C, Yamamuro M (2007). "Stream Monitoring Under the Time Warping Distance." In L Liu, A Yazici, R Chirkova, V Oria (eds.), *Proceedings of the IEEE 23rd International Conference on Data Engineering – ICDE 2007*, pp. 1046–1055. IEEE Computer Society, Los Alamitos, CA, USA. doi:10.1109/ICDE.2007.368963.

Salvador S, Chan P (2004). "FastDTW: Toward Accurate Dynamic Time Warping in Linear Time and Space." In KP Unnikrishnan, R Uthurusamy, J Han (eds.), *KDD Workshop on Mining Temporal and Sequential Data*, pp. 70–80. ACM, New York, NY, USA.

Syeda-Mahmood T, Beymer D, Wang F (2007). "Shape-Based Matching of ECG Recordings." In A Dittmar, J Clark, E McAdams, N Lovell (eds.), *Engineering in Medicine and Biology Society, 2007. EMBS 2007. 29th Annual International Conference of the IEEE*, pp. 2012–2018. IEEE Computer Society, Los Alamitos, CA, USA. doi:10.1109/IEMBS.2007.4352714.

Tak YS (2007). "A Leaf Image Retrieval Scheme Based on Partial Dynamic Time Warping and Two-Level Filtering." In D Wei, T Miyazaki, I Paik (eds.), *Proceedings of the 7th IEEE International Conference on Computer and Information Technology – CIT 2007*, pp. 633–638. IEEE Computer Society, Los Alamitos, CA, USA. doi:10.1109/CIT.2007.158.

Tormene P, Giorgino T, Quaglini S, Stefanelli M (2009). "Matching Incomplete Time Series with Dynamic Time Warping: An Algorithm and an Application to Post-Stroke Rehabilitation." *Artificial Intelligence in Medicine*, **45**(1), 11–34. doi:10.1016/j.artmed.2008.11.007.

Tuzcu V, Nas S (2005). "Dynamic Time Warping as a Novel Tool in Pattern Recognition of ECG Changes in Heart Rhythm Disturbances." In M Jamshidi, M Johnson, P Chen (eds.), *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, volume 1, pp. 182–186 Vol. 1. IEEE Computer Society, Los Alamitos, CA, USA. doi:10.1109/ICSMC.2005.1571142.

Velichko VM, Zagoruyko NG (1970). "Automatic Recognition of 200 Words." *International Journal of Man-Machine Studies*, **2**, 223–234.

Wei L, Keogh E, Xi X (2006). "SAXually Explicit Images: Finding Unusual Shapes." In CW Clifton, N Zhong, J Liu, BW Wah, X Wu (eds.), *Sixth International Conference on Data Mining 2006 – ICDM '06*, pp. 711–720. IEEE Computer Society, Los Alamitos, CA, USA. doi:10.1109/ICDM.2006.138.

Affiliation:

Toni Giorgino
Laboratory for Biomedical Informatics
Dipartimento di Informatica e Sistemistica
Università di Pavia

via Ferrata 1
I-27100, Pavia, Italy
E-mail: toni.giorgino@gmail.com
URL: http://www.labmedinfo.org/people/cv/giorgino.htm

Currently with:
Computational Biochemistry and Biophysics Laboratory
Research Group on Biomedical Informatics (GRIB-IMIM)
Parc de Recerca Biomèdica de Barcelona
c/ Dr. Aiguader, 88
E-08003, Barcelona, Spain

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 31, Issue 7 *Submitted:* 2008-09-30
August 2009 *Accepted:* 2009-06-26