Distance Measures for Time Series in R: the TSdist Package

Usue Mori
University of the Basque Country
UPV/EHU.

Alexander Mendiburu
University of the Basque Country
UPV/EHU.

Jose A. Lozano
University of the Basque Country
UPV/EHU.

Abstract

The definition of a distance measure between time series is critical for many time series data mining tasks such as clustering and classification. For this reason, and based on the specific characteristics of time series data, a vast number of distance measures have been published in the past few years. However, many of the most popular distance measures for time series are not included in any R package. With the objective of filling this gap, a complete set of the most popular distance measures for time series are implemented in the **TSdist** `R` package . This package is designed to work with different time series data types supported in R such as `ts`, `xts` or `zoo`, as well as with numeric vectors. Furthermore, the efficient calculation of distance matrices for entire databases is supported in the package.

Keywords: time series, distance measures, R.

1. Introduction

In recent years, the increase in data collecting technologies and sensors has enabled access to a large amount of temporal data, also denominated time series. The main characteristics of this type of data are its high dimensionality, dynamism, auto-correlation and noisy nature, all which complicate the study and pattern extraction to a large extent. In view of this, many researchers have focused on finding specific methods and on adapting the existing data mining algorithms to obtain useful information from these databases. So, tasks such as regression, classification, clustering or segmentation have been extended and modified successfully (Fu 2011).

Many of these tasks require the definition of a distance measure which will indicate the level of similarity or dissimilarity between time series. Because of this, the scientific community has focused on elaborating suitable measures for this specific type of data. The fruit of this work, a vast portfolio of distance measures, has been published.

A few R packages such as **dtw** (Giorgino 2009) or **TSClust** (Montero and Vilar 2014) provide implementations of some of these distance measures. However, to the extent of our knowledge, many of the most popular distances reviewed by Liao (2005); Esling and Agon (2012); Wang, Mueen, Ding, Trajcevski, Scheuermann, and Keogh (2012) are not available in R.

In this paper, the **TSdist** package for the R statistical software (R Core Team 2014) is presented. This package provides the implementation of a set of distance measures designed for univariate numerical time series. These distance measures have been selected based on their popularity and because they are mentioned in recent reviews on the topic (Liao 2005; Esling and Agon 2012; Wang *et al.* 2012). In

this manner, this package largely contributes to achieving a more complete coverage of the published time series distance measures in R.

The rest of the paper is organized as follows. In Section 2, the distance measures included in the package are described and the details concerning their implementation and computation are summarized. Furthermore, some examples of their use are included. In Section 3, the distance computation between objects of type `ts`, `zoo` or `xts` is explained. In Section 4 the calculation of distance matrices of entire time series databases by means of the **TSdist** package is studied. To finish, in the last section, a summary of the distance measures implemented in the different R packages is given, providing a complete view of the possibilities for time series distance computation in this language.

2. Definition of similarity measures

In this section, the distance measures implemented in the **TSdist** package are described and the functions included for their calculation are detailed.

These basic distance function will be designed to work only with numeric vectors. To calculate distances between time series objects of type `zoo`, `xts` or `ts` (see Section 3).

Following the categorization introduced by Esling and Agon (2012), the time series distance measures are usually divided into four categories: shape based, edit based, features based and structure based. This package focuses on the first three categories because they are applicable to all cases.

2.1. Shape based distance measures

This first category of distances is based on directly comparing the raw values and the shape of the series in different manners.

L_p distances

L_p distances are those that derive from the different L_p norms (Yi and Faloutsos 2000). These distances are rigid metrics that can only compare series of the same length. However, due to their simplicity, they have been widely used in many tasks related to time series analysis and mining. Given two time series $X = \{x_0, x_1, ..., x_{N-1}\}$ and $Y = \{y_0, y_1, ..., y_{N-1}\}$, the different variations of the L_p distances and their formulas are provided in Table 1. It must be noted that the Euclidean Distance is a special case of the Minkowski distance, but it is explicitly included because it is a baseline distance measure in the area of time series data mining.

These distances are implemented in the `manhattanDistance()`, `minkowskiDistance()`, `euclideanDistance()` and `infinitenormDistance()` functions. They receive two time series, represented by numeric vectors as the only input, except the Minkowski distance which also needs a specification of `p`. The output of these functions is the distance between the two series. A wrapper function, `lpDistances()`, has also been defined for the L_p distances. This function also takes two numeric vectors as input but, in addition, the `method` argument must be set to the desired L_p distance measure (e.g `method = "euclidean"`). In the case of the Minkowski distance, the argument `p` must also be specified.

Short Time Series distance

The Short Time Series distance (STS) is introduced by Möller-Levet, Klawonn, Cho, and Wolkenhauer (2003) with the idea of proposing a distance adapted to the characteristics of irregularly sampled

Distance	p	Formula
Manhattan	$p = 1$	$\sum_{i=0}^{N-1} \lvert x_i - y_i \rvert$
Minkowski	$1 < p < \inf$	$\sqrt[p]{\sum_{i=0}^{N-1} (x_i - y_i)^{\frac{1}{p}}}$
Euclidean	$p = 2$	$\sqrt{\sum_{i=0}^{N-1} (x_i - y_i)^2}$
Infinite norm	$p = \inf$	$\max_{i=0,\ldots,N-1} \lvert x_i - y_i \rvert$

Table 1: L_p distances

series. It is defined as:

$$d_{STS}(X,Y) = \sqrt{\sum_{k=0}^{N-1} \left(\frac{y_{k+1} - y_k}{t_{k+1} - t_k} - \frac{x_{k+1} - x_k}{t'_{k+1} - t'_k} \right)} \tag{1}$$

where t and t' are the temporal indexes of series X and Y respectively.

This distance can be calculated by invoking the `stsDistance()` function and specifying four numeric vectors that represent the two series `x` and `y` and their temporal indexes `t` and `t'`. `x` and `y` must have the same length and, although their temporal indexes may start in different time contexts, the increments must be equal:

$$t_{k+1} - t_k = t'_{k+1} - t'_k, \;\; \forall k = 0, ..., N-1 \tag{2}$$

Furthermore, if the temporal indexes are not specified, a constant sampling rate will be assumed.

Dissim Distance

The Dissim distance was introduced by (Frentzos, Gratsias, and Theodoridis 2007) and is specifically designed for series collected at different sampling rates. This means that each series will be defined in a finite set of time instants, but these can be different for each series (see Figure 1).

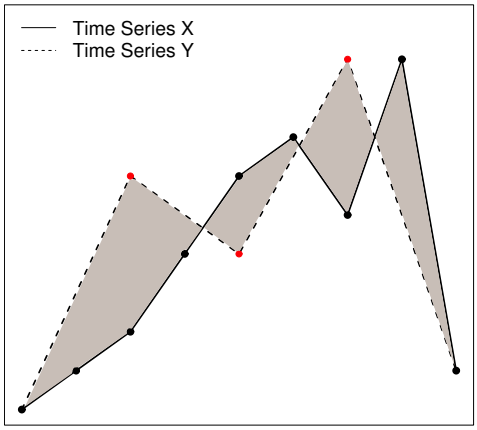


Figure 1: Representation of the Dissim distance calculation.

The Dissim distance requires a continuous representation of the series and so, the series that are being compared are assumed to be linear between sampling points (see Figure 1). Once this is done, the idea is to calculate the definite integral of the Euclidean distance between them:

$$Dissim(X,Y) = \sum_{i=0}^{K-1} \int_{t_i}^{t_{i+1}} D_{X,Y}(t)dt \tag{3}$$

where, $T = \{t_0, ..., t_{K-1}\}$ is a global time index that fuses the time indexes of both series by taking all the points that appear in both sets. $D_{X,Y}(t)$ represents the Euclidean distance between the series at time-stamp t.

This distance measure is implemented in the `dissimDistance()` function that takes the two numeric series (`x` and `y`) and their corresponding temporal indexes (`tx` and `ty`) as input. It must be noted that, although these indexes may differ, they must start and end at the same instants of time.

Finally, in order to avoid the computational expense of calculating the integral, Frentzos *et al.* (2007) present an approximation of Dissim that simplifies Equation 3 by using the trapezoid rule. In this manner, the following simpler formula is obtained:

$$Dissim\_approx(X,Y) = \sum_{i=0}^{N-1} (D_{X,Y}(t_i) + D_{X,Y}(t_{i+1})) \cdot (t_{i+1} - t_i) \tag{4}$$

This approximation may be calculated by invoking the `dissimapproxDistance()` function, which is defined with the same input arguments as in the previous case.

Dynamic Time Warping distance (DTW)

In order to overcome the inconveniences of rigid distances such as Euclidean Distance, many similarity measures have been specifically designed for time series data. Among them the most popular is probably Dynamic Time Warping (DTW) (Berndt and Clifford 1994).

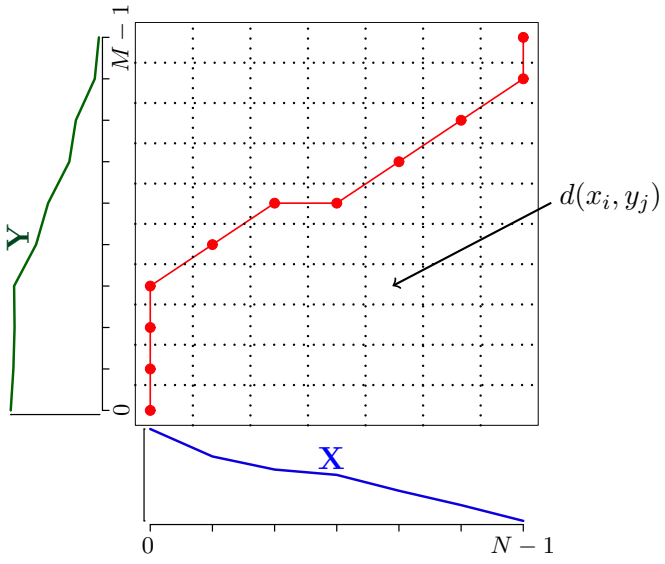


Figure 2: Illustration of the Dynamic Time Warping distance calculation

As shown in Figure 2, the objective of this distance is to find the optimal alignment between two series $X = \{x_0, x_1, ..., x_{N-1}\}$ and $Y = \{y_0, y_1, ..., y_{M-1}\}$, by searching for the minimal path in a

distance matrix (D) that defines a mapping between them. Each entry of the matrix D is defined by the Euclidean distance between a pair of points (x_i, y_j). In R, the calculation of D is efficiently done by using the **proxy** package (Meyer and Buchta 2013).

This optimization problem is subject to three restrictions (Liao 2005). The *boundary condition* forces the path to start in position $D(0,0)$ and to end in $D(N-1, M-1)$. The *continuity condition* restricts the step size, forcing the path to continue through one of the adjacent cells. Finally, the *monotonicity condition* forbids the path to move backwards in the positions of the matrix. Based on this, the problem is reduced to solving the following recurrence:

$$DTW(X,Y) = \begin{cases} 0 & \text{if } M-1 = N-1 = 0 \\ \inf & \text{if } M-1 = 0 \text{ or } N-1 = 0 \\ d(x_0, y_0) + min\{DTW(Rest(X), Rest(Y)), \\ DTW(Rest(X), Y), DTW(X, Rest(Y))\} & \text{otherwise} \end{cases} \quad (5)$$

where d is the Euclidean distance and, being $X = \{x_0, x_1, ..., x_{N-1}\}$ and $Y = \{y_0, y_1, ..., y_{M-1}\}$, $Rest(X)$ and $Rest(Y)$ are defined as $\{x_1, ..., x_{N-1}\}$ and $\{y_1, ..., y_{M-1}\}$.

Based on its definition, this distance is able to deal with transformations such as local warping and shifting and, furthermore, it allows the comparison between series of different length.

By applying dynamic programming, this recurrence can be solved as shown in Algorithm 1.

Algorithm 1 Dynamic Time Warping calculation

Input: $X = \{x_0, ..., x_{N-1}\}, Y = \{y_0, ..., y_{M-1}\}$
Initialize: $costMatrix = matrix(N+1, M+1) * 0$
  for $i = 1, ..., N$ **do**
    $costMatrix(i, 0) = 10000$
  end for
  for $j = 1, ..., M$ **do**
    $costMatrix(0, j) = 10000$
  end for
  for $i = 1, ..., N$ **do**
    for $j = 1, ..., M$ **do**

$$costMatrix(i,j) = d(x_i, y_j) + min \begin{cases} costMatrix(i-1, j) \\ costMatrix(i, j-1) \\ costMatrix(i-1, j-1) \end{cases}$$

    end for
  end for
  return $costmatrix(N, M)$

The calculation of this basic DTW distance is implemented in the `dtwDistance()` function. The core of this function has been implemented in C, due to computational efficiency issues. The output of the function is the DTW distance between the two series.

Additionally, it must be noted that it is quite common to add an extra temporal constraint to DTW by limiting the number of vertical or horizontal steps that the path can take consecutively. The windowing type implemented in this package is the classical Sakoe-Chiba band (Sakoe and Chiba 1978) which places a symmetric band around the main diagonal and forces the path to stay inside this band (see Figure 3).

This adjustment avoids the matching of points that are very far from each other in time and, in addition, it reduces the computation cost (Wang *et al.* 2012).

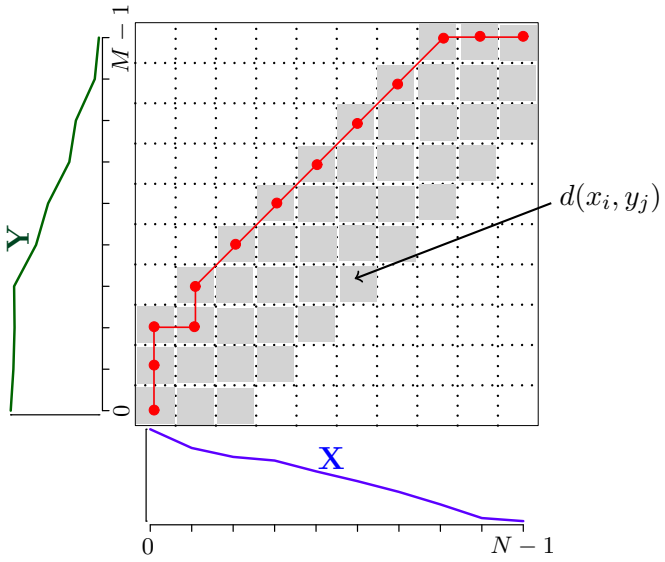


Figure 3: Illustration of the Sakoe-Chiba windowing applied to the Dynamic Time Warping distance calculation.

If we want to restrict the window size in the DTW calculation, an additional argument `sigma` must be added to the `dtwDistance()` function invocation. This argument represents the window size and must be a positive integer value. The window size can not exceed the length of the series and in addition, the function will return an error message if $|length(X) - length(Y)| >$`sigma`, because in this case the boundary condition can not be fulfilled and no possible warping solution will exist.

LB-Keogh for Dynamic Time Warping

Calculating the Dynamic Time Warping between a pair of series is computationally quite expensive and, because of this, many lower bounds have been introduced in the literature for this distance. These lower bounds are essentially approximations of the DTW distance that provide a value which is always lower than the actual DTW measure. They are computationally more efficient than DTW and become especially useful for the task of similar series retrieval, because they allow the pruning of sequences that are largely dissimilar to the query series. One of the most common of such lower bounds is that introduced by Keogh and Ratanamahatana (2004).

Given a query series X and a reference series Y, the first step in the calculation of this lower bound is obtaining an upper and lower envelope series for the query series. Given an allowed range of warping r, the upper and lower envelope series are defined as follows:

$$U_i = max(X_i - r, X_i + r) \tag{6}$$
$$L_i = min(X_i - r, X_i + r) \tag{7}$$

where r can take a fixed value or it can be a function of i, depending on the windowing function used. In this case, the Sakoe-Chiba band (Sakoe and Chiba 1978) will be used to calculate the envelopes, so r will take a constant value indicating the width of the window.

Once the upper and lower envelopes have been calculated, the lower bounding distance between X

and Y is calculated by using the following formula:

$$LB\_Keogh(X,Y) = \begin{cases} (Y_i - U_i)^2 & \text{if } Y_i > U_i \\ (Y_i - L_i)^2 & \text{if } Y_i < L_i \\ 0 & \text{otherwise} \end{cases} \quad (8)$$

This lower bound is implemented in the `lbKeoghDistance()` function that takes a numeric vector that represents the query series `x`, a numeric series that defines the reference series `y` and a window size `sigma` as input arguments and returns the Keogh lower bound for the DTW distance.

2.2. Edit based distances

Edit distance was initially presented to calculate the similarity between two sequences of strings and is based on the idea of counting the minimum number of edit operations (delete, insert and replace) that are necessary to transform one sequence into the other.

The problem of working with real numbers is that it is difficult to find exact matching points in two different sequences and, therefore, the edit distance is not directly applicable. Different adaptations have been proposed in the literature and those included in the **TSdist** package are the most common according to recent reviews Wang *et al.* (2012); Esling and Agon (2012).

By using the delete and insert operations, all these distances are able to work with series of different length.

Edit Distance for Real Sequences (EDR)

In this first edit based distance, in order to adapt it to numerical values, the distance between the points in the time series is reduced to 0 or 1 (Chen, Ozsu, and Oria 2005). If two points x_i and y_j are closer to each other in the absolute sense than a user specified ϵ, they will be considered equal. On the contrary, if they are farther apart, they will be considered distinct and the distance between them will be considered 1. In R, this initial mapping between two series is efficiently done by using the **proxy** package and applying vector operations to the resulting matrix.

As an additional property, EDR permits gaps or unmatched regions in the database but it penalizes them with a value equal to their length. All this summarizes into the following recursion that is converted into an iteration by means of dynamic programming as in the previous case:

$$EDR(X,Y) = \begin{cases} N & \text{if } M-1=0 \\ M & \text{if } N-1=0 \\ min\{EDR(Rest(X),Rest(Y)) + d_{edr}(x_0,y_0), & \\ EDR(Rest(X),Y)+1, EDR(X,Rest(Y))+1\} & \text{otherwise} \end{cases} \quad (9)$$

where d_{edr} represents the distance between two points in the series and takes a value of 0 or 1, as explained above.

The value of this distance measure can be calculated by means of the `edrDistance()` function, which takes two numeric vectors and a threshold parameter `epsilon` as input.

Finally, as with DTW, a Sakoe-Chiba windowing may be added to the EDR distance by simply adding a positive integer `sigma` value to the function call.

Longest Common Subsequence distance (LCSS)

The second edit based distance included in the **TSdist** package, which is in fact a similarity measure, is the **Longest Common Subsequence distance (LCSS)** (Vlachos, Kollios, and Gunopulos 2002).

In this case, the similarity between two time series is quantified in terms of the longest common sub-sequence but, taking into account that gaps or unmatched regions are permitted.
As with EDR, the initial mapping between the series is done with the aid of the **proxy** package and the Euclidean distance. Then, the distance between two points is reduced to 0 or 1 depending on a threshold ϵ. All this is reduced to the following recurrence:

$$LCSS(X,Y) = \begin{cases} 0 & \text{if } M-1=0 \text{ or } N-1=0 \\ LCSS(Rest(X), Rest(Y)) + 1 & \text{if } |x_0 - y_0| \leq \epsilon \\ max\{LCSS(Rest(X), Y), LCSS(X, Rest(Y))\} & \text{otherwise} \end{cases} \tag{10}$$

This recurrence is usually solved by using dynamic programming, similar to the two previous distance measures and is implemented in the `lcssDistance()` function. The input arguments to this function are two numerical vectors `x` and `y`, a threshold parameter `epsilon` and an optional `sigma` value that will be added if a temporal constraint is desired. As with the other distances, the function will return the distance value between the two series (`d`).

Edit Distance with Real Penalty (ERP)

The third adaptation to the edit distance is **Edit Distance with Real Penalty (EDR)** (Chen and Ng 2004) which is a combination of DTW and EDR.
In this case, given two time series $X = \{x_0, x_1, ..., x_{N-1}\}$ and $Y = \{y_0, y_1, ..., y_{M-1}\}$, the initial mapping is done by using the Euclidean distance and the **proxy** package as with DTW. The similarity with EDR relies on the fact that gaps are permitted. However, penalization will be carried out differently, by setting a user specified constant g and adding the euclidean distance (d) of the unmatched points to this constant.

$$ERP(X,Y) = \begin{cases} \sum_{i=0}^{N-1} |y_i - g| & \text{if } M-1=0 \\ \sum_{i=0}^{M-1} |x_i - g| & \text{if } N-1=0 \\ min\{ERP(Rest(X), Rest(Y)) + d(x_0, y_0), & \\ ERP(Rest(X), Y) + d(x_0, g), ERP(X, Rest(Y)) + d(g, y_0)\} & \text{otherwise} \end{cases} \tag{11}$$

The `R` function that calculates this distance is `erpDistance()` and in order to invoke it, two numeric vectors and a user defined `g` parameter must be provided necessarily as input. In addition, a Sakoe-Chiba temporal constraint can be added to the calculation if the `sigma` argument is added to the call.

2.3. Feature based distances

This category of distance measures focuses on extracting a set of features from the time series and calculating the similarity between these features instead of using the raw values of the series.

Distances based on Pearson's correlation

Pearson's correlation between two time series $X = \{x_0, x_1, ..., x_{N-1}\}$ and $Y = \{y_0, y_1, ..., y_{N-1}\}$ is defined as:

$$PC(X,Y) = \frac{\sum_{i=0}^{N-1} (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{(x_i - \bar{x})^2}\sqrt{(y_i - \bar{y})^2}} \tag{12}$$

where $\bar{x}$ and $\bar{y}$ are the mean values of the series.
Based on this value, two distance measures were introduced by Golay, Kollias, Stoll, Meier, Valavanis, and Boesiger (1998):

$$d_{PC1}(X,Y) = \left(\frac{1 - PC}{1 + PC}\right)^{\beta} \tag{13}$$

$$d_{PC2}(X,Y) = 2(1-PC) \tag{14}$$

where β is a positive parameter defined by the user.

These two distance measures are implemented in the `correlationDistance()` function. If the `beta` parameter is specified, the first distance will be calculated and if not, then, the second definition will be applied. Other than that, the function takes two numeric vectors `x` and `y` as input and returns the distance between them (`d`).

Distance based on the cross-correlation

This distance is presented in (Liao 2005) and is based on the cross-correlation between two series. The cross-correlation between two series at lag k is calculated as:

$$CC_k(X,Y) = \frac{\sum_{i=0}^{N-1-k}(x_i-\bar{x})(y_{i+k}-\bar{y})}{\sqrt{(x_i-\bar{x})^2}\sqrt{(y_{i+k}-\bar{y})^2}} \tag{15}$$

where $\bar{x}$ and $\bar{y}$ are the mean values of the series as in the previous case. Based on this, the distance measure is defined as:

$$d_{CC}(X,Y) = \sqrt{\frac{(1-CC_0(X,Y))}{\sum_{k=1}^{max} CC_k(X,Y)}} \tag{16}$$

This distance measure can be calculated by using the `crosscorrelationDistance()` function included in the **TSdist** package and specifying two numeric vectors (`x` and `y`) and a value for `max`. This last argument represents the maximum lag that is considered in the calculation and should not exceed the length of the series. As in the other cases, this function will return the distance between the two series.

Fourier Coefficients based distance

As its name indicates, the similarity calculation in this case is based on comparing the Discrete Fourier Transform coefficients of the series.

Given a numeric series $X = \{x_0, x_1, ..., x_{N-1}\}$, its Discrete Fourier Transform can be easily calculated in R by using the `fft()` function included in the **stats** package. This function simply returns an array containing the Fourier Coefficients of the series. The value of each coefficient measures the contribution of its associated frequency to the series and, based on this, the Inverse Fourier Transform provides the means to represent the sequences as a combination of sinusoidal forms.

It is important to note that the Fourier coefficients are complex numbers that can be expressed as $X_f = a_f + b_f i$. In the case of real sequences such as time series, the Discrete Fourier Transform is symmetric and therefore it is sufficient to study the first $\frac{N}{2}+1$ coefficients. Furthermore, it is commonly considered that, for many time series, most of the information is kept in their first n Fourier Coefficients, where $n < \frac{N}{2}+1$ (Agrawal, Faloutsos, and Swami 1993).

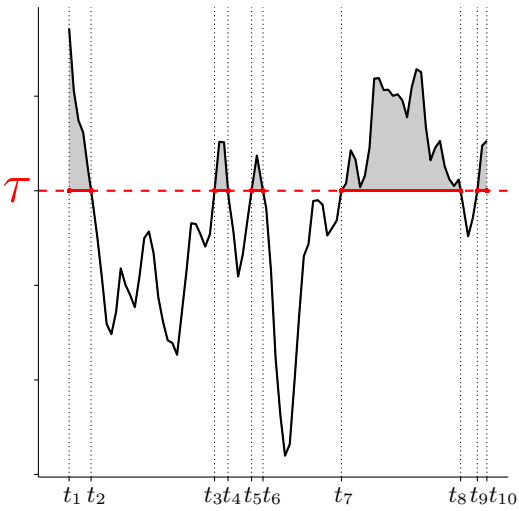
Based on all this information, the distance between two time series X and Y with Fourier Coefficients $\{(a_0,b_0),...,(a_{\frac{N}{2}},b_{\frac{N}{2}})\}$ and $\{(a'_0,b'_0),...,(a'_{\frac{N}{2}},b'_{\frac{N}{2}})\}$ is given by the Euclidean distance between the first n coefficients:

$$F(X,Y) = \sqrt{\sum_{i=0}^{n}((a_i-a'_i)^2+(b_i-b'_i)^2)} \tag{17}$$

This distance is implemented in the `fourierDistance()` function and apart from two numeric series, the number of parameters to be considered (`n`) must be specified.

TQuest distance

TQuest was presented by Aß falg, Kriegel, Kröger, Kunath, Pryakhin, and Renz (2006) and is classified as a feature based distance in (Esling and Agon 2012). In this manner, instead of comparing the raw values of the series, it studies the similarity of a set of features extracted from them. As can be



$$S(X,\tau) = \{(t_1,t_2),(t_3,t_4),(t_5,t_6),(t_7,t_8),(t_9,t_{10})\}$$

Figure 4: Time series representation method used by the TQuest distance.

seen in Figure 4, the idea is to define the set of time intervals in a time series that fulfill the following conditions:

1. All the values that the time series takes during these time intervals must be strictly above a user specified threshold τ.
2. They are the largest possible intervals that satisfy the previous condition.

The distance between two time series X and Y that are represented by the interval sets $S(X,\tau)$ and $S(Y,\tau)$ is defined as follows:

$$TQuest(X,Y) = \frac{1}{|S(X,\tau)|}\sum_{s\in S(X,\tau)}\min_{t\in S(Y,\tau)} d(s,s') + \frac{1}{|S(Y,\tau)|}\sum_{s'\in S(Y,\tau)}\min_{s\in S(X,\tau)} d(s',s) \tag{18}$$

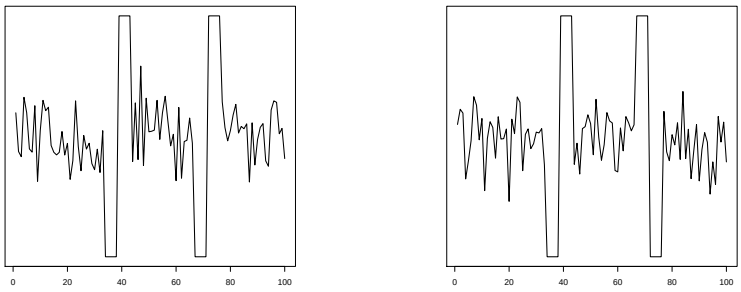
where the distance between two intervals $s = (s_l, s_u)$ and $s' = (s'_l, s'_u)$ is calculated as:

$$d(s,s') = \sqrt{(s_l - s'_l)^2 + (s_u - s'_u)^2} \tag{19}$$

This distance is implemented in the `tquestDistance()` function and relies on the potential of R for working with vectors to find the threshold passing points and to define the threshold passing intervals. The input values of the function are two numeric series, `x` and `y`, their temporal indices `tx` and `ty` and a threshold `tau`. It provides the TQuest distance between the two series.

2.4. Examples of distance calculations between numeric vectors

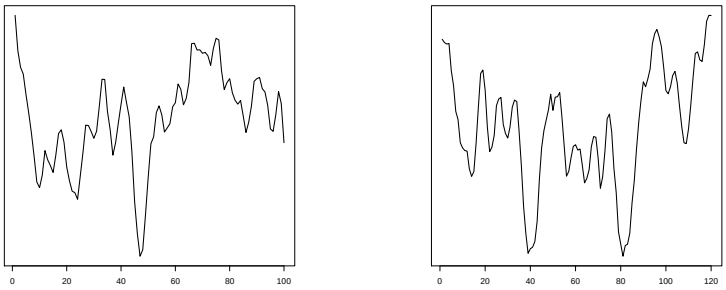
The `example.series1` and `example.series2` objects included in the **TSdist** package are two numeric vectors that represent two different synthetic series which are generated based on the shapes that define the Two Patterns synthetic database of series (Geurts 2002) (See Figure 5).



(a) example.series1 (b) example.series2

Figure 5: The two example series of the same length included in the TSdist package.

Additionaly, `example.series3` and `example.series4` represent two ARMA(3,2) series of coefficients AR=(1, -0.24, 0.1) and MA=(1, 1.2) but with different lengths, 100 and 120, and generated with different random seeds (See Figure 6).



(a) example.series3 (b) example.series4

Figure 6: The two example series of different length included in the TSdist package.

Example 1 *The basic calculation of the distance between two series such as* `example.series1` *and* `example.series2` *is done as follows:*

```
> crossCorrelationDistance(example.series1, example.series2)

[1] 11.80617

> correlationDistance(example.series1, example.series2)

[1] 1.958172
```

Example 2 *A special case are the L_p distances that can also be called by the wrapper function* `lpDistance`*:*

```
> manhattanDistance(example.series1, example.series2)

[1] 185.1962
```

is equivalent to:

```
> lpDistance(example.series1, example.series2, method="manhattan")

[1] 185.1962
```

Example 3 *Many of the distance measures require the definition of a parameter, which must be included in the call to the corresponding function:*

```
> edrDistance(example.series1, example.series2, epsilon=0.1)

[1] 80

> erpDistance(example.series1, example.series2, g=0)

[1] 98.29833
```

Example 4 *Some measures are able to compute the distance between series of different lengths but others are not and will return an error in this case:*

```
>  lcssDistance(example.series3, example.series4, epsilon=0.1)

[1] 16

> fourierDistance(example.series3, example.series4)
```

```
Error : The length of the series must be equal
```

Example 5 *Other errors will be specific to each distance measure and will be determined based on their definition. As an example, the window function defined for DTW can not exceed the size of the time series being compared.*

```
> length(example.series3)

[1] 100

> length(example.series4)

[1] 120

> dtwDistance(example.series3, example.series4, sigma=105)

Error : The window size exceeds the length of the first series
```

Furthermore, the window size must be larger that the difference of the series lengths:

```
> dtwDistance(example.series3, example.series4, sigma=10)

Error : It is not possible to compare those two series
with the defined window size
```

3. The use of time series objects of R in TSdist

Given their relevance and special characteristics, many specific classes or object types have been included in R to define time series. The most common and popular examples are the `ts` objects introduced in the basic **stats** package of R or the more complex `zoo` (Zeileis and Grothendieck 2005) and `xts` (Ryan and Ulrich 2013) classes implemented in two separate packages.

All these objects provide the possibility of saving information about the temporal index and, in addition, provide a complete set of methods to work with them. However, there are slight differences between them. The first is the most basic and is addressed exclusively for regularly sampled time series. The `zoo` objects incorporate the possibility of dealing with irregularly sampled time series. Finally, the **xts** package further extends the **zoo** package to provide a uniform handling of all the time series data types in R.

The **TSdist** package reviewed in this paper supports the use of these three time series objects by means of the function `tsDistances`. This function admits `ts` objects or univariate `zoo` and `xts` objects as well as numeric vectors and works as a wrapper function for all the distances presented in the previous sections. All the input arguments and the results obtained by this function are summarized in Table 2.

Argument	Description	Remarks
x	Univariate time series object of type numeric vector, ts, zoo or xts	
y	Univariate time series object of type numeric vector, ts, zoo or xts	
tx	Sampling index of series x (optional)	Necessary if x is a numeric vector and the sampling is not constant.
ty	Sampling index of series y (optional)	Necessary if y is a numeric vector and the sampling is not constant.
method	Distance measure	"euclidean", "manhattan", "minkowski", "infinitenorm", "pearsoncorrelation", "crosscorrelation", "sts", "dtw", "keogh_lb", "edr", "erp", "lcss", "fourier", "tquest", "dissim" or "dissimapprox"
...	Parameters associated to the selected distance measure	
Result		
d	Distance between x and y.	

Table 2: Summary of the function tsDistances

As can be seen, this function simply acts as a link between the time series object types and the functions that calculate the distance measures.

3.1. Examples of distance calculations between time series objects

The zoo.series1 and zoo.series2 time series included in the package are replicas of the example.series1 and example.series2 objects introduced previously but saved in a zoo format with a specific time index.

Example 6 *A basic call to the* tsDistance *function for two series like these is done in the following manner:*

```
> tsDistances(zoo.series1, zoo.series2,
+ distance="pearsoncorrelation")

[1] 1.958172

> tsDistances(zoo.series1, zoo.series2, distance="dtw", sigma=10)

[1] 93.66541
```

The distance calculation between ts or xts objects is done in the same manner.

4. Computing distance matrices

On some occasions it is necessary to calculate the distance between each pair of series in a given database of series ($X = \{X_1, X_2, ..., X_N\}$). This will result in a distance matrix:

$$D(X) = \begin{pmatrix} d(X_1, X_1) & d(X_1, X_2) & \cdots & d(X_1, X_N) \\ d(X_2, X_1) & d(X_2, X_2) & \cdots & d(X_2, X_N) \\ \vdots & \vdots & \ddots & \vdots \\ d(X_N, X_1) & d(X_N, X_2) & \cdots & d(X_N, X_N) \end{pmatrix}$$

The **proxy** package provides the means to calculate pairwise distances between the rows of a given matrix. In view of this, together with loading the **TSdist** package, the distance measures included in it are loaded into the registry of distances of the **proxy** package. With this, distance matrices obtained from the measures included in the **TSdist** package may be computed directly by simply invoking the `dist` method implemented in **proxy**. The only requirement is that the time series database must be saved in a numeric matrix, where each series is set in a row.

A more direct way of performing this computation is by using the `tsDatabaseDistances` function implemented in the package **TSdist**. The input and output arguments to this function are summarized in Table 3.

Argument	**Description**	**Remarks**
`x`	Object of type `matrix`, `list`, `mts`, `xts` or `zoo` that represents a set of series set in a row-wise format .	
`method`	Distance measure to be computed.	`"euclidean"`, `"manhattan"`, `"minkowski"`, `"infinitenorm"`, `"pearsoncorrelation"`, `"crosscorrelation"`, `"sts"`, `"dtw"`, `"keogh_lb"`, `"edr"`, `"erp"`, `"lcss"`, `"fourier"`, `"tquest"`, `"dissim"` or `"dissimapprox"`
`diag`	Logical value indicating if the diagonal of the distance matrix should be printed.	
`upper`	Logical value indicating if the upper triangle of the distance matrix should be printed.	
...	Any other parameters associated to the distance measures	
Result		
`D`	Distance matrix of series in `x`.	An object of the class `dist`.

Table 3: Summary of the function `tsDatabaseDistances`

As can be seen in the table, this function also provides a solution to calculate the pairwise distance between databases saved on `list`, `mts`, `zoo` or `xts` objects. It works as a wrapper function and accommodates the structure of this type of objects to the requirements of the `dist` function. Note that the series in the database must be of the same length.

The direct distance matrix calculation of databases with series of different sizes or sampling rates are not supported in this package. However, they can be easily obtained by using the `tsDistances` combined with `for` loops or derivations of the function `apply` in R.

4.1. Examples of distance matrix calculations

The `example.database` object included in the package is a matrix that represents a database with 6 ARMA(3,2) series of coefficients AR=(1, -0.24, 0.1) and MA=(1, 1.2) but with different innovations.

The 6 series are set in a matrix in a row-wise format.

Additionally, the `zoo.database` object included in the package is a multivariate `zoo` object that saves the series of `example.database` but with a specific time index.

Example 7 *The* `dist` *function calculates the pairwise distance between all the rows in a matrix, so the calculation of the distance matrix can be done easily for the* `example.database` *object:*

```
> dist(example.database, method="tsDistances",
+ distance="tquest", tau=mean(example.database),
+ diag=TRUE, upper=TRUE)
```

```
          series1   series2   series3   series4   series5   series6
series1 0.0000000 1.9070456 1.4856385 1.4856385 1.3499311 2.0070405
series2 1.9070456 0.0000000 1.1856538 2.0213254 1.9427580 1.3570736
series3 1.4856385 1.1856538 0.0000000 0.5642569 2.1284628 0.8071017
series4 1.4856385 2.0213254 0.5642569 0.0000000 1.4284985 0.9499515
series5 1.3499311 1.9427580 2.1284628 1.4284985 0.0000000 2.7141472
series6 2.0070405 1.3570736 0.8071017 0.9499515 2.7141472 0.0000000
```

This can also be calculated by using the `tsDatabaseDistances` *function:*

```
> tsDatabaseDistances(example.database, method="tquest",
+ tau=mean(example.database), diag=TRUE, upper=TRUE)
```

Example 8 *The* `zoo.database` *object is not a matrix, so the distance matrix calculation can not be done by using the* `dist` *function directly. In this case, the calculation must be done in the following manner:*

```
> tsDatabaseDistances(zoo.database, method="tquest",
+ tau=mean(zoo.database), diag=TRUE, upper=TRUE)
```

```
          series1   series2   series3   series4   series5   series6
series1 0.0000000 1.9070456 1.4856385 1.4856385 1.3499311 2.0070405
series2 1.9070456 0.0000000 1.1856538 2.0213254 1.9427580 1.3570736
series3 1.4856385 1.1856538 0.0000000 0.5642569 2.1284628 0.8071017
series4 1.4856385 2.0213254 0.5642569 0.0000000 1.4284985 0.9499515
series5 1.3499311 1.9427580 2.1284628 1.4284985 0.0000000 2.7141472
series6 2.0070405 1.3570736 0.8071017 0.9499515 2.7141472 0.0000000
```

5. Summary and comparison of time series distance packages in R

As commented previously and to the extent of our knowledge, apart from **TSdist**, two other packages of R provide implementations of distances for time series: **dtw** and **TSclust**.

In Table 4, a summary of the distance measures included in these packages is presented, providing a complete view of the different options available for time series data mining purposes. As can be seen, the three packages focus on different distances and are, therefore, complementary.

Distance Measure	**dtw**	**TSclust**	**TSdist**
Shape-based distances			
L_p distances			✓
Short Time Series Distance (STS)			✓
Complexity Invariant Time Series Distance		✓	
DISSIM			✓
Approximated DISSIM			✓
Dynamic Time Warping (DTW)	✓		✓
Keogh_LB (DTW)			✓
Edit based distances			
Edit Distance for Real Sequences (EDR)			✓
Edit Distance with Real Penalty (ERP)			✓
Longest Common Subsequence (LCSS)			✓
Feaure-based distances			
Autocorrelation and Partial Autocorrelation based		✓	
Pearson correlation based		✓	✓
Cross-correlation based			✓
Discrete Fourier Decomposition based			✓
Wavelet Decomposition based		✓	
Periodogram based		✓	
SAX based		✓	
Spectral Density based		✓	
TQuest			✓
Structure-based distances			
ARIMA model based		✓	
Compression based		✓	
Non Parametric Forecast based		✓	

Table 4: Summary of distance measures for time series implemented in R

The first package, **dtw**, focuses on a unique distance measure, DTW, and provides an in depth approximation to it, providing many different options and variations of this distance measure. On the contrary, **TSclust** provides a wide spectrum of simple distance measures, mostly based on the structure of the series or on different features extracted from them. It must be noted that some of these distances are quite ad-hoc and uncommon and are not mentioned in recent reviews (Liao 2005; Esling and Agon 2012; Wang *et al.* 2012). Finally, our package **TSdist** fills the gaps left by these two packages by implementing many of the most common and popular distance measures proposed in the reviews mentioned. Specifically, in addition to adding new measures to the shape-based and feature-based distance categories, edit based distances for numeric time series have been included, which was a completely overlooked category of distance measures in previous R packages.

References

Agrawal R, Faloutsos C, Swami A (1993). "Efficient similarity search in sequence databases." In *Proceedings of the 4th International Conference of Foundations of Data Organization and Algorithms*, volume 5, pp. 69–84.

Aß falg J, Kriegel Hp, Kröger P, Kunath P, Pryakhin A, Renz M (2006). "Similarity Search on Time Series based on Threshold Queries." In *Proceedings of the 10th international conference on Advances in Database Technology*, pp. 276–294.

Berndt DJ, Clifford J (1994). "Using Dynamic Time Warping to Find Patterns in Time Series." In *AAAI-94 Workshop on Knowledge Discovery in Databases.*, pp. 359–370.

Chen L, Ng R (2004). "On The Marriage of Lp-norms and Edit Distance." In *Proceedings of the Thirtieth International Conference on Very Large Data Bases*, pp. 792–803.

Chen L, Ozsu MT, Oria V (2005). "Robust and Fast Similarity Search for Moving Object Trajectories." In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, pp. 491–502. ISBN 1595930604.

Esling P, Agon C (2012). "Time-series data mining." *ACM Computing Surveys*, **45**(1), 1–34. ISSN 03600300.

Frentzos E, Gratsias K, Theodoridis Y (2007). "Index-based Most Similar Trajectory Search." In *2007 IEEE 23rd International Conference on Data Engineering*, pp. 816–825. Ieee. ISBN 1-4244-0802-4.

Fu TC (2011). "A review on time series data mining." *Engineering Applications of Artificial Intelligence*, **24**(1), 164–181. ISSN 09521976.

Geurts P (2002). *Contributions to decision tree induction: bias/variance tradeoff and time series classification.* Ph.D. thesis, University of Liege, Belgium.

Giorgino T (2009). "Computing and Visualizing Dynamic Time Warping Alignments in R: the dtw Package." *Journal of Statistical Software*, **31**(7).

Golay X, Kollias S, Stoll G, Meier D, Valavanis A, Boesiger P (1998). "A new correlation-based fuzzy logic clustering algorithm for FMRI." *Magnetic Resonance in Medicine*, **40**(2), 249–260. ISSN 07403194.

Keogh E, Ratanamahatana CA (2004). "Exact indexing of dynamic time warping." *Knowledge and Information Systems*, **7**(3), 358–386. ISSN 0219-1377.

Liao TW (2005). "Clustering of time series data, a survey." *Pattern Recognition*, **38**(11), 1857–1874. ISSN 00313203.

Meyer D, Buchta C (2013). "proxy: Distance and Similarity Measures." URL http://cran.r-project.org/package=proxy.

Möller-Levet CS, Klawonn F, Cho Kh, Wolkenhauer O (2003). "Fuzzy Clustering of Short Time-Series and Unevenly Distributed Sampling Points." In *Proceedings of the 5th International Symposium on Intelligent Data Analysis*.

Montero P, Vilar JA (2014). "Time series clustering utilities."

R Core Team (2014). "R: A Language and Environment for Statistical Computing." URL http://www.r-project.org/.

Ryan JA, Ulrich JM (2013). "xts: eXtensible Time Series." URL http://cran.r-project.org/package=xts.

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.

Vlachos M, Kollios G, Gunopulos D (2002). "Discovering similar multidimensional trajectories." In *Proceedings 18th International Conference on Data Engineering*, pp. 673–684. IEEE Comput. Soc. ISBN 0-7695-1531-2.

Wang X, Mueen A, Ding H, Trajcevski G, Scheuermann P, Keogh E (2012). "Experimental comparison of representation methods and distance measures for time series data." *Data Mining and Knowledge Discovery*, **26**(2), 275–309. ISSN 1384-5810.

Yi Bk, Faloutsos C (2000). "Fast Time Sequence Indexing for Arbitrary." In *In Proceedings of the 26th International Conference on Very Large Databases*, pp. 385–394.

Zeileis A, Grothendieck G (2005). "zoo: S3 Infrastructure for Regular and Irregular Time Series." *Journal of Statistical Software*, **14**(6), 1–27.

Affiliation:

Usue Mori, Jose A. Lozano
Intelligent Systems Group (ISG)
Department of Computer Science and Artificial Intelligence
University of the Basque Country UPV/EHU
Manuel de Lardizabal 1
20018 Donostia-San Sebastian, Spain.
E-mail: usue.mori@ehu.es, ja.lozano@ehu.es

Alexander Mendiburu
Intelligent Systems Group (ISG)
Department of Computer Architecture and Technology
University of the Basque Country UPV/EHU
Manuel de Lardizabal 1
20018 Donostia-San Sebastian, Spain.
E-mail: alexander.mendiburu@ehu.es