Appendix C

The \texttt{dts} software package

C.1 Implementation and installation

The methods of distance-based analysis can only be used in practice if there exists a reliable computational code. We have therefore implemented a number of algorithms as a software package for the statistical computing environment \texttt{R} (\texttt{R Development Core Team}, 2008).

The main computational routine \texttt{td} solves discrete transportation or assignment problems in $\mathbb{R}^n$ for a variety of ground distances and Wasserstein orders. Distances can be wrapped for phase distributions. One-dimensional problems are efficiently solved by monotone arrangement. All other problems are either solved by a dedicated minimum-cost flow solver or by a general linear programming solver. The first possibility is offered by the MCF code (Löbel, 1996) which is freely available for academic users\(^\text{1}\). Due to license issues, this code was not incorporated into the \texttt{dts} package, but compiles into the library if it is present. The second possibility is offered by the \texttt{lpSolve} package, which needs to be separately installed\(^\text{2}\).

The package \texttt{dts} is available as a source-code distribution under the terms of the Creative Commons Attribution-Noncommerical-Share Alike 3.0 Unported License\(^\text{3}\). We will discuss its installation for a general UNIX system here. To install \texttt{dts} when MCF is present, let \texttt{MCF\_ROOT} be the directory containing the MCF files. Issuing the command

\begin{verbatim}
R CMD INSTALL dts.tar.gz --configure-vars='MCF\_ROOT=${MCF\_ROOT}$'
\end{verbatim}

configures and installs \texttt{dts} from the source package \texttt{dts.tar.gz}. If MCF is not present, the package can be installed in the usual way, but for computations the package \texttt{lpSolve} is needed, and they will be slower. In this case, installation can be performed by executing

\begin{verbatim}
R CMD INSTALL dts.tar.gz
\end{verbatim}

on the command line. Note that administrative rights might be needed for a global installation.

Further packages that are required to use all features of \texttt{dts} are \texttt{MASS} (for multi-
dimensional scaling), vegan (for MRPP permutation testing), odesolve (for integration of differential equations) and ROCR (for receiver-operator curves).

C.2 Reference

The following is a complete description of all functions available in the dts package, version 1.0-04.

**cmdscale.add**

*Out-of-sample classical multidimensional scaling*

**Description**

Obtain the coordinates of an additional point in classical multidimensional scaling.

**Usage**

```r
cmdscale.add(x, m, k = 2, points = NULL, verbose = FALSE, ntries = 10,
max.iter = 100)
```

**Arguments**

- `x`: Vector of distances between the additional point and all previous points.
- `m`: Distance matrix of all previous points.
- `k`: Dimension of Euclidean space in which to represent the points.
- `points`: Reconstructed coordinates of previous points (optional).
- `verbose`: Logical value to indicate whether details of the computation should be shown.
- `ntries`: Number of times the solution is attempted.
- `max.iter`: Maximal number of iterations in the minimalization problem.

**Details**

The out-of-sample problem consists in approximating the coordinates of an additional point in a representation of n previous points obtained by multidimensional scaling, from its distances with all previous points. In the case of classical multidimensional scaling considered here, the problem can be solved by minimizing a nonlinear error functional (Troset and Priebe, 2008). The R function `optim` is called a number `ntries` of times to perform a simplex search (Nelder and Mead, 1965), and the coordinates that result in the minimal error are returned. For the previous points, coordinates in `points` are used if given; otherwise these are calculated by multidimensional scaling from the distances in `m`. Since the coordinates in multidimensional scaling are unique up to a rotation, this is useful to ensure that the out-of-sample point lies in an already established coordinate system.

---

4 The version number follows the major/minor convention, where the first number indicates significant (major) changes, the second number minor changes (with even numbers indicating stable releases and odd numbers indicating developmental versions), and the last number is used to indicate consecutive bug fixes.
Value

A list containing the following components:

- **points**: A matrix whose rows contain the coordinates of all points (including the out-of-sample point)
- **eig**: A vector of the largest $k$ eigenvalues (see `cmdscale` in package MASS)
- **y**: A vector containing the coordinates of the out-of-sample point
- **points0**: A matrix whose rows contain the coordinates of all previous points (excluding the out-of-sample point)

Note

This function uses the `cmdscale` code from the package MASS to obtain the representation of the original points.

Examples

```r
library(MooreRayleigh) # rsphere
x <- rsphere(10) # uniform sample on the sphere
x0 <- rsphere(1) # one additional point
m.all <- as.matrix(dist(rbind(x,x0))) # all mutual Euclidean distances
attributes(m.all)$dimnames <- NULL
m <- m.all[1:10,1:10]
m0 <- m.all[11,1:10]
library(MASS)
par(mfrow=c(1,2))
mds <- cmdscale(m.all,k=2,eig=TRUE)$points # project to the plane
eqscplot(mds[,1],mds[,2],xlab="x",ylab="y",
tol=0.3,main="MDS")
points(mds[11,1],mds[11,2],pch=4) # mark additional point
mds.add <- cmdscale.add(m0,m,k=2,points=mds[1:10,])$points
eqscplot(mds.add[,1],mds.add[,2],xlab="x",ylab="y",
tol=0.3,main="Out-of-sample MDS")
points(mds.add[11,1],mds.add[11,2],pch=4) # mark additional point
cat(paste("Distance between MDS and out-of-sample MDS =",
round(sqrt(sum(mds.add[11,]-mds[11,])^2),4),"\n"))
```

Description

Linear discriminant analysis for distance matrices.

Usage

```r
ldadist.cv(x, classes, pc = NULL, search = FALSE, verbose = TRUE)
```
Arguments

x  A square matrix of mutual distances between n data items.
classes  A factor specifying the class membership of each data item.
pc  The number of components to use in the reconstruction. Can be empty if search is TRUE.
search  A logical value indicating whether to search for the optimal number of components (see details).
verbose  A logical value indicating whether to show details when search is TRUE.

Details

Linear discriminant analysis is performed on coordinates obtained by classical multidimensional scaling. This assumes the data to be represented by points in an n-dimensional Euclidean space. The class membership is estimated, and the fraction of correct classifications defines the accuracy. To reliably estimate this, leave-one-out crossvalidation is implemented by the out-of-sample method. In detail, for each data item its distance information is removed from x, the coordinates of the remaining points are calculated by classical multidimensional scaling, and the coordinates of the present point are approximated from its distances to the remaining points (see cmdscale.add). The classification of the single point is obtained by predict.lda from the MASS package, with the remaining points as training data for the linear discriminant function (using lda from the MASS package).

The number of components n can be specified (parameter pc). If search is TRUE, the number of components is searched that results in the best accuracy. In this case, the parameter pc is the maximal number of components to use. If pc is not given, the value n-4 is used (the points usually become collinear for larger numbers of components).

Value

A list containing the following components:

predict, cv.predict  A vector containing the predicted class membership.
posterior, cv.posterior  A vector containing the maximum a posteriori classification probabilities for the estimated class membership.
pc, cv.pc  The number of components used in the representation by multidimensional scaling.
tab, cv.tab  A two-by-k summary table of the classification, where k is the number of distinct classes.
accuracy, cv.accuracy  A single number containing the (average) classification accuracy.
correct, cv.correct  A vector containing ones (correct) and zeros (false) for the classification of each item.
acc, cv.acc  A vector containing the (average) classification accuracies for each number of components evaluated. Equal to accuracy (cv.accuracy) if search is FALSE.

If search is TRUE, the values are returned for the number of components with the highest accuracy among all searched.
Note

If the number of components is determined by searching, the resulting accuracies are slightly over-estimated and have to be interpreted with care. For large sample sizes this bias can be avoided by an additional stage of crossvalidation, but this has not been implemented.

Also note that the number of components should at most be equal to the number of samples from the smallest class.

Examples

```r
ndim <- 6
# generate 20 normal variates in "ndim" dimensions
x <- matrix(rnorm(20*ndim),ncol=ndim)
# translate the second half
x[11:20,] <- x[11:20,] + c(0,0,0,rnorm(ndim-3,mean=1))
m <- as.matrix(dist(x))
attributes(m)$dimnames <- NULL
grouping <- as.factor(c(rep(1,10),rep(2,10)))
res <- ldadist.cv(m,grouping,search=TRUE,pc=ndim-1,verbose=TRUE)
```

mfdfa

Multifractal detrended fluctuation analysis

Description

Multifractal detrended fluctuation analysis.

Usage

```r
MF DFA(x, detrend = "poly1", q = c(1, 2), sum.order = 0,
    scale.max = trunc(length(x)/4), scale.min = 16, scale.ratio = 2,
    verbose = FALSE)
```

Arguments

- `x` Time series
- `detrend` Detrending method. Can be either 'bridge' for bridge regression or 'polyN' for polynomial detrending, where N > 0 indicates the order to use.
- `q` A numerical vector that indicates which scaling exponents to extract. Standard detrended fluctuation analysis corresponds to q = 2.
- `sum.order` Number of integrations (positive order) or differentiations (negative order) to perform before the analysis.
- `scale.max` Maximal scale to consider.
- `scale.min` Minimal scale to consider.
- `scale.ratio` Ratio between successive scales.
- `verbose` A logical value that indicates whether to show details of the computation.
Details

Deviations of a time series $X_i$ ($i = 1, \ldots, N$) from its mean $\bar{X}$ are first integrated,

$$Y_i = \sum_{j=1}^{N} (X_i - \bar{X}),$$

leading to an unbounded profile. For a given scale $s > 0$, the profile $Y_i$ is then divided into $N_s = \text{int}(N/s)$ nonoverlapping segments of length $s$. Since the length $N$ of the time series is usually not a multiple of the scale $s$, a short part at the end of the profile may remain. In order not to disregard this part, the procedure is repeated starting from the opposite end, leading to a total of $2N_s$ segments, denoted by $Y_{k;j}$ ($k = 1, \ldots, 2N_s; j = 1, \ldots, s$). For each segment a trend $Z_{k;j}$ (usually linear or quadratic, corresponding to detrend="poly1" or detrend="poly2") is individually estimated by least-squares and subtracted. The fluctuation function for a given order $q \geq 0$ is given by

$$F_q(s) = \left( \frac{1}{2N_s} \sum_k \left( \frac{1}{s} \sum_j (Y_{k;j} - Z_{k;j})^2 \right)^{q/2} \right)^{1/q}.$$

This procedure is repeated for a number of scales $s$ and exponents $q$. The scaling behavior is then assessed by weighted least-squares fitting a line to the scaling function $F_q(s)$ with respect to scale $s$ in a double logarithmic plot, such that $\log F_q(s) \propto \alpha \log s$, with weights proportional to the index of scale (e.g., the third scale is weighted 2/3 relative to the second scale), to compensate for the reduction in data points on which the estimation of the corresponding $F_q(s)$ is based. If the residual error of the fit $R^2$ is large enough, the estimate $\alpha \geq 0$ is the $q$-scaling exponent of $x$ for each value of $q$.

Exponents for $q = 0$ can also be evaluated, but need special treatment, with the scaling function given by

$$F_0(s) = \exp \left( \frac{1}{4N_s} \sum_k \ln \left( \frac{1}{s} \sum_j (Y_{k;j} - Z_{k;j})^2 \right) \right).$$

Note that only nonnegative exponents can be evaluated. However, additional integrations (or finite differences) can be performed before the analysis, indicated by sum.order. Since each integration increases the exponent by one, the estimated exponent $\alpha$ is corrected by subtracting sum.order-1 from it at the end. This allows to also resolve negative exponents.

Value

A list containing the following components:

- h: A vector containing the estimated scaling exponents.
- r.squared: The residual errors of the linear fits of the scaling relationship.
- scale: A vector that contains the scales that were considered.
- rmse: A matrix that contains the residual errors for each exponent at each scale.
- q: A vector that indicates which scaling exponents were extracted.

Note

This implementation is based heavily on the DFA code in the fractal package of W. Constantine and D. Percival (unpublished), correcting an error in the detrending and adding functionality for calculation of multifractal exponents.
Examples

```r
x <- rnorm(2000)
foo <- mfdfa(x, q=seq(0,10), detrend="poly2", verbose=TRUE)
plot(res$r.squared, ylim=c(0,1), type="b", xlab="") # goodness-of-fit
lines(res$h, type="b", pch=2) # scaling exponents
legend("bottomleft", pch=c(1,2), legend=c("goodness-of-fit", "exponents"))
abline(h=1/2) # theoretical value
```

Description

Usage

```r
mle.pl(x, min.tail = 50, cut = 0, verbose = FALSE, nboot = NULL)
plot.pl(x, ...)
print.pl(x)
test.pl(x, min.tail = 50, cut = 0, verbose = FALSE, nboot = 2500)
```

Arguments

- **x**: A numerical vector of positive measurements.
- **min.tail**: Minimum number of measurements to use in the estimation.
- **cut**: Proportion of measurements to throw away (see details).
- **verbose**: A logical value that indicates whether to show details of the estimation procedure.
- **...**: Additional arguments for plotting.
- **nboot**: Number of bootstrap replicates to use.

Details

Maximum likelihood estimation is implemented in function `mle.pl` to fit the Pareto distribution (see `powerlaw`) to the samples in `x`. The lower-cut off point `x_{min}` is determined by the method of Clauset et al. (2009): The `n` samples in `x` are sorted and each of the smallest `n-min.tail` samples is considered as a candidate for the lower cut-off point `x_{min}`. For each such candidate the Pareto distribution is fitted, resulting in an estimate of the power-law exponent `α`, and the Kolmogorov-Smirnov statistic KS quantifies the maximal difference between the fitted distribution and the empirical distribution function (for samples from `x` greater or equal to `x_{min}`). Finally, the value of `x_{min}` that minimizes KS is chosen and the corresponding parameter values are returned.

It is recommended to keep at least about 50–100 samples for the estimation, which can be adjusted by the parameter `min.tail`. To speed up the computation, a fraction `cut` of the smallest samples can be discarded before the estimation.

To quantify estimation uncertainty, a number `nboot` of bootstrap replicates can be optionally specified. Each replicate is generated by resampling with replacement from `x` and estimating the parameters `x_{min}` and `α` by the above procedure.
Function `print.pl` provides a summary of the estimation results and `plot.pl` a diagnostic plot. Function `test.pl` performs the simple test of the power-law hypothesis described in Clauset et al. (2009)[Section 4.1]. First the parameters $x_{\text{min}}$ and $\alpha$ are estimated as above. Then a number $n_{\text{boot}}$ of bootstrap samples are generated where samples are independently drawn either from the fitted power-law model for the tail, or sampled with replacement from the samples of $x$ smaller than $x_{\text{min}}$. The probability to choose a sample from the tail is given by its (relative) length. For each of these replicates the estimation procedure in `mle.pl` is repeated, resulting in a set of $n_{\text{boot}}$ values of the KS statistic (for the best fit of the Pareto distribution). The significance probability for the null hypothesis of power-law behaviour is given by the fraction of these that are larger than the KS statistic for the fit of the original data $x$. If this is large enough (Clauset et al. (2009) recommend a value of 0.10 for a conservative test), the general alternative is rejected and the power-law hypothesis is accepted.

**Value**

Function `mle.pl` returns a list with class “`pl`” containing the following components:

- `x` The original data.
- `xmin.all, n.all` All values of $x_{\text{min}}$ considered in the estimation procedure and the length of the remaining tail for which the maximum-likelihood estimation of the exponent $\alpha$ was performed.
- `alpha.all, D.all` Values of $\alpha$ and the KS statistic corresponding to the cut-off points in `xmin.all`.
- `ntail` The length of the tail for the optimal parameter choice.
- `xmin, alpha, D` Values of the parameters and the KS statistic for the optimal parameter choice.
- `nboot` The number of bootstrap replicates used to quantify estimation error.
- `alpha.boot, xmin.boot` If `nboot` is given, these contain the parameter estimates for each bootstrap replicate.

Function `test.pl` returns a list with class “`htest`” containing the following components:

- `statistic` value of the test statistic for the data in $x$.
- `p.value` the significance probability for the test.
- `alternative` a character string describing the alternative hypothesis (‘not symmetric’).
- `method` a character string describing the type of test.
- `data.name` a character string giving the names of the data.
- `nboot` the number of bootstrap replicates used for the test.
- `xmin.boot, alpha.boot, D.boot` the values of the estimated parameters and the KS statistic for each replicate.
- `xmin, alpha, D` the values of the estimated parameters for the original data in $x$.
- `x` the original data.

**Examples**

```r
x <- ppl(1000,2,10)  # generate synthetic power-law data
pl <- mle.pl(x,cut=0.2,verbose=TRUE,nboot=100)
pl  # summary output
plot(pl)  # diagnostic plot
foo <- test.pl(x,cut=0.2,nboot=100,verbose=TRUE)
foo  # should reject the alternative
```
The power-law distribution

Description

Density, distribution function and random generation for the Pareto distribution, i.e., the power-law distribution with lower cut-off point.

Usage

dpl(x, alpha, xmin)
ppl(q, alpha, xmin)
rpl(n, alpha, xmin)

Arguments

x, q Vector of quantiles.
n Number of observations. If length(n) > 1, the length is taken to be the required number.
alpha Exponent \( \alpha \geq 1 \) of the power-law.
xmin Lower cut-off point \( x_{\text{min}} \geq 0 \).

Details

The density of the power-law distribution is proportional to \( x^{-\alpha} \) with exponent \( \alpha \geq 1 \). Since this is not integrable for \( x \to 0 \), it is customary to restrict the power-law distribution to values of \( x \) greater than a (lower) cut-off point \( x_{\text{min}} > 0 \). This is called the Pareto distribution, and its density is given by

\[
f(x) = (\alpha - 1)x_{\text{min}}^{-\alpha - 1}x^{-\alpha}.
\]

Its two parameters \( \alpha \) and \( x_{\text{min}} \) are usually called the shape and scale parameters.

Value

dpl gives the density, ppl gives the distribution function, and rpl generates random variates by the transformation method.

Examples

x <- seq(0,100,0.1)
# power-law leads to a straight line in a double logarithmic plot
plot(log10(x),log10(1-ppl(x,3,1)),ylab=expression(log10(1-F(x))))
abline(v=log10(1),lty=2)  # cut-off point
Sample entropy and cross-entropy

Description

Calculate sample entropy (SampEn) and cross-sample entropy to estimate the rate of information production in dynamical systems.

Usage

    samp.en(x, y = NULL, r = NULL, r.ratio = NULL, edim = 2, tlag = 1, normalize = TRUE, size = NULL, verbose = FALSE)

Arguments

- **x**: A numerical vector containing the time series for which sample entropy is calculated.
- **y**: Optional vector containing a second time series.
- **r**: The threshold when to consider two vectors as being neighbours (see details). If not given, use r.ratio times the standard deviation of x. If r.ratio is also not given, use a value of 0.2 for it.
- **r.ratio**: If r is not given, use r.ratio times the standard deviation of x as the threshold.
- **edim, tlag**: Embedding dimension and time lag to use.
- **normalize**: A logical value indicating whether to center and normalize the time series to zero mean and unit standard deviation.
- **size**: If present, draw randomly size vectors with replacement from the delay vector embedding for the calculation.
- **verbose**: A logical value that indicates whether to show details of the calculation.

Details

Calculates the sample entropy (SampEn) introduced by Richman and Moorman (2000) from the time series in x. If additional y is given, calculates their cross-sample entropy (Cross-SampEn). Both time series are delay embedded with time lag tlag in edim and edim+1 dimensions (see rdelay). To ensure an equal number of delay vectors in both embeddings, a few vectors at the end of the former are discarded. Then the number of pairs of delay vectors that lie within a distance r of each other is counted for both these delay embeddings, resulting in two counts B (in dimension edim) and A (in dimension edim+1). For computational efficiency the maximum distance is used, and in contrast to the approximate entrop (ApEn) of Pincus (1991) self-matches are not counted. If y is given, the distances are calculated between all pairs of delay vectors where one arises from the embedding of x and the other from the corresponding one for y. Sample entropy is then defined as the negative (natural) logarithm of A/B and is a finite approximation of the Kolmogorov-Sinai entropy (obtained in an appropriate limit of infinite data and vanishing threshold r).
Value

A list containing the following components:

- **s**: Sample entropy.
- **B**: The number of close neighbours in an $\text{edim}$-dimensional embedding.
- **A**: The number of close neighbours in an $\text{edim} + 1$-dimensional embedding.

Examples

```r
x <- seq(0,100,0.2)
y1 <- rnorm(length(x))  # a random sample
samp.en(y1)  # large sample entropy
y2 <- sin(x*2*pi/10)  # a deterministic process
samp.en(y2)  # low sample entropy
samp.en(y1, y2)  # large cross-sample entropy
```

**td**

*Wasserstein distances for finite distributions of points*

**Description**

Calculates Wasserstein distance between two sets of multi-dimensional vectors.

**Usage**

```r
td(x, y, wx = NULL, wy = NULL, dist = "l2", order = 2, 
    cost.scale = NULL, phases = FALSE, verbosity = 0)
```

```r
td.1d(x,y, wx = NULL, wy = NULL, dist = "l2", order = 2, 
    phases = FALSE, verbosity = 0)
```

```r
td.lpSolve(x, y, wx = NULL, wy = NULL, dist = "l2", order = 2, 
    cost.scale = NULL, phases = FALSE, verbosity = 0)
```

**Arguments**

- **x**: Multi-dimensional point data as a matrix with individual points represented by columns. Can also be numerical vector (for one-dimensional problems).
- **y**: Multi-dimensional point data as a matrix with individual points represented by columns. Can also be a numerical vector (for one-dimensional problems).
- **wx**: Optional weights for x. Should sum to 1.
- **wy**: Optional weights for y. Should sum to 1.
- **dist**: Choose one from "l1", "l2", "max", or a numerical value $\geq 1$ for a Minkowski ($L_p$) distance.
- **order**: The order of the Wasserstein distance. Defaults to quadratic Wasserstein distance.
- **cost.scale**: Optional scaling factor for weights. Only needed for multidimensional data.
- **phases**: Logical value that indicates whether to wrap distances (for phase distributions) or not.
- **verbosity**: Verbosity level of output. Higher values result in more diagnostic output.
Details

The Wasserstein distance between $k$-dimensional point sets $x$ and $y$ is the cost associated with an optimal transportation problem. Both $x$ and $y$ are interpreted as discrete probability measures on a Euclidean space $\mathbb{R}^k$, and their Wasserstein distance is the minimal total cost when transforming one measure into the other. Each unit of probability mass transported incurs a cost equal to the distance it is moved.

The distance used is in principle arbitrary; however, at present only the most common distances are implemented: ‘l1’ is the $L_1$ (Manhattan) distance, ‘l2’ is $L_2$ (Euclidean) distance and ‘max’ is supremum distance. Additionally, the ‘order’ can be given. Explicitly, the Wasserstein distance of order $p$ is

$$W_p(x, y) = \inf \left( \int d(x-y)^p \, d\pi(x, y) \right)^{1/p},$$

where the infimum is taken over all probability measures (transportation plans) $\pi(x, y)$ such that $\pi[A, R^k] = x[A]$ and $\pi[R^k, B] = y[B]$ for all subsets $A, B \subseteq R^k$. The quadratic Wasserstein distance (default) has very interesting theoretical properties, in particular, it is possible to interpolate measures. More commonly used, however, is the Wasserstein distance of order 1, also known as the Kantorovich-Rubinstein distance.

In the discrete case considered here, the calculation of the Wasserstein distance is equivalent to solving a so-called (discrete) transportation problem: Let $x$ and $y$ be discrete probability measures, $x = \sum_i a_i \delta_{x_i}$ and $y = \sum_j b_j \delta_{y_j}$, where $\delta_x$ is the Dirac measure at the point $x \in R^k$. These can be interpreted as weighted point sets. The supplies $a_i \in (0, 1]$ and the demands $b_j \in (0, 1]$ need to be normalized, such that $\sum_i a_i = 1$ and $\sum_j b_j = 1$. Each transportation plan can then be represented as a nonnegative matrix $f_{ij}$ that fulfills the source and sink conditions $\sum_j f_{ij} = a_i$ and $\sum_i f_{ij} = b_j$ for all $i$ and $j$. The Wasserstein distance of order $p$ is then

$$W_p(x, y) = \min \left( \sum_{ij} f_{ij} d(x_i - y_j)^p \right)^{1/p}.$$

In the one-dimensional case, the problem is solved by monotone arrangement, i.e., by sorting the points of both samples and iteratively matching the largest value of $x$ to its nearest neighbour from $y$. In the multivariate case, this routine will use the minimum-cost flow solver MCF (Loebel, 1996), if available. Alternatively, the much slower lpSolve package is used. In both these cases internally integer arithmetic is used, so the distances and weights will be scaled to the nearest integer in a suitable range (see discretize.factor).

Value

The Wasserstein distance of the data.

Note

The computational complexity is high, theoretically on the order of $O(n^5)$ where $n = \max\{|x|, |y|\}$, although in practice often an almost quadratic complexity can be observed. Problems with more than 1000 data points will therefore need to be approximated by resampling smaller point sets a number of times (bootstrapping), or binning the points.

Examples

```r
data1 <- c(1,2)
data2 <- c(1,2,3)
td(data1,data2)

data1 <- matrix(c(1,1,0,1,0.5,0.5),nrow=2)
```

value: Wasserstein distance of the data.
stress <- matrix(c(1,1,0,0,0.3,0.3),nrow=2)
tdp(data1,data2) # will be 7/15

data1 <- c(1,2)
data2 <- c(1,2,3)
weights1 <- c(0.9,0.1)
weights2 <- c(0.4,0.5,0.1)
td(data1,data2,weights1,weights2) # will be 0.6

---

Diagnostics of misrepresentation error

Description
Diagnostic measures of misrepresentation error in multidimensional scaling

Usage
stress(x, mds.dim)

Arguments
- x: Distance matrix
- mds.dim: Reconstruction dimension.

Details
The misrepresentation error of metric multidimensional scaling is evaluated. Given a n-by-n distance matrix $D_{ij}$, raw stress is the total residual square error of the Euclidean distances $\Delta_{ij}$ of metric multidimensional scaling in mds.dim dimensions,

$$\sigma = \sum_{ij} (D_{ij} - \Delta_{ij})^2.$$

The average of the residual square error with respect to the $i$-th point,

$$\sigma^{(i)} = \frac{1}{n} \sum_{j=1}^{n} (D_{ij} - \Delta_{ij})^2$$

is called stress-per-point and a useful measure of the local misrepresentation error. To compare these measures between distinct metric spaces, normalized stress is $\sigma_1 = \sigma / \sum_{ij} \Delta_{ij}^2$, and normalized stress-per-point is $\sigma_1^{(i)} = n \sigma_i / \sum_{ij} \Delta_{ij}^2$.

Value
A list containing the following components:
- stress: Raw total stress
- spp: A vector containing stress-per-point for each row of x
- stress1: Normalized total stress
- spp1: A vector containing normalized stress-per-point for each row of x
Examples

```r
library(MASS)
library(MooreRayleigh) # rsphere
x <- rsphere(10)
d <- as.matrix(dist(x))
attributes(d)$dimnames <- NULL
cmd <- cmdscale(d,k=2,eig=TRUE)
plot(cmd$points[,1],cmd$points[,2])
str <- stress(d,2)
symbols(cmd$points[,1],cmd$points[,2],
circles=(1/pi*sqrt(str$spp)),inches=FALSE,add=TRUE)
```

---

**td.interp**

Interpolate two distributions along an optimal transport ray

**Description**

Calculates Wasserstein distance between two sets of multi-dimensional vectors and shifts both distributions a fraction along the optimal transport rays.

**Usage**

```r
td.interp(x, y, frac, dist = "l2", order = 2, cost.scale = NULL, verbosity = 0)
```

**Arguments**

- **x**
  Multi-dimensional point data as a matrix with individual points represented by columns. Can also be numerical vector (for one-dimensional problems).

- **y**
  Multi-dimensional point data as a matrix with individual points represented by columns. Can also be a numerical vector (for one-dimensional problems).

- **frac**
  Fraction of distance to move points on the optimal rays.

- **dist**
  Choose one from “l1”, “l2”, “max”, or a numerical value ≥ 1 for a Minkowski ($L_p$) distance.

- **order**
  The order of the Wasserstein distance. Defaults to quadratic Wasserstein distance.

- **cost.scale**
  Optional scaling factor for weights. Only needed for multidimensional data.

- **verbosity**
  Verbosity level of output. Higher values result in more diagnostic output.

**Details**

The calculation of the optimal transport is the same as for the function td. The optimal transport mapping is used to shift each point of `x` and `y` a fraction `frac/2` into the direction of its partner (along an optimal transport ray) under the optimal matching.
Value

A list containing the following components:

- `td`: The optimal transportation distance between x and y
- `x`: The shifted data points of the original x
- `y`: The shifted data points of the original y

Note

The state of this function is experimental. Currently, the interpolation is only available with the MCF solver and for equal-sized point sets (trivial weights).

Examples

```r
library(MooreRayleigh)
x <- rsphere(10,2) # 10 points on the unit circle
y <- matrix(rep(0,20),nrow=2)
foo <- td.interp(x,y,frac=0.5)
plot(x)
points(y,pch=19)
points(foo$x,col="blue")
points(foo$y,col="blue",pch=19)
```

Description

Delay vector embedding of scalar time series

Usage

```r
ts.delay(x, edim, tlag = 1, ofs = 0)
```

Arguments

- `x`: A numerical vector (scalar time series).
- `edim`: Embedding dimension to use.
- `tlag`: Distance between indices of adjacent components of delay vectors. Defaults to 1.
- `ofs`: Number of data points to skip from the beginning of x. Defaults to 0.

Details

The delay representation of a numerical vector $(x_1, \ldots, x_n)$ with time lag $k$ and offset $l$ in embedding dimension $d$ is the vector-valued series $y = (y_1, y_2, \ldots)$ given by:

\[
y_1 = (x_l, x_{l+k}, \ldots, x_{l+k(d-1)})^t,
\]

\[
y_2 = (x_{l+1}, x_{l+k+1}, \ldots, x_{l+k(d-1)+1})^t,
\]

\cdots
Value

A matrix whose columns contain the delay vectors.

Examples

```r
x <- seq(1, 9)
ts.delay(x, 3)
ts.delay(x, 3, 3)
```