Package: detpack (via r-universe)

August 23, 2024

Type Package

Title Density Estimation and Random Number Generation with Distribution Element Trees

Version 1.1.3

Author Daniel Meyer

Maintainer Daniel Meyer <meyerda@ethz.ch>

Description Density estimation for possibly large data sets and conditional/unconditional random number generation or bootstrapping with distribution element trees. The function 'det.construct' translates a dataset into a distribution element tree. To evaluate the probability density based on a previously computed tree at arbitrary query points, the function 'det.query' is available. The functions 'det1' and 'det2' provide density estimation and plotting for one- and two-dimensional datasets. Conditional/unconditional smooth bootstrapping from an available distribution element tree can be performed by 'det.rnd'. For more details on distribution element trees, see: Meyer, D.W. (2016) <arXiv:1610.00345> or Meyer, D.W., Statistics and Computing (2017) <doi:10.1007/s11222-017-9751-9> and Meyer, D.W. (2017) <arXiv:1711.04632> or Meyer, D.W., Journal of Computational and Graphical Statistics (2018) <doi:10.1080/10618600.2018.1482768>.

Imports parallel, graphics, grDevices, stats

License GPL-2

RoxygenNote 6.0.1

NeedsCompilation no

Repository CRAN

Date/Publication 2019-07-24 11:00:03 UTC

Contents

`allequal` *Are All Columns in a Matrix Equal?*

Description

Check if all column vectors in a matrix are equal.

Usage

```
allequal(x)
```

Arguments

`x`	matrix with column vectors.

Value

TRUE if matrix `x` has zero or one column, or if all column vectors in matrix `x` are equal. FALSE if `x` contains at least two different columns.

chi2indeptest *Pairwise Mutual Independence Test*

Description

Pearson's Chi-square test of pairwise mutual independence.

Usage

```
chi2indeptest(x, alpha)
```

Arguments

x	matrix with n rows comprised of normalized data between 0 and 1.
alpha	significance level.

Value

Object with test outcomes h[i,j] = h[j,i] = TRUE/FALSE for 1 <= i,j <= n meaning rejection/acceptance of independence null hypothesis involving rows i and j of matrix x, and p-value or confidence level of acceptance.

chi2testlinear *Goodness-of-Fit Test for Linear Distributions*

Description

Composite Pearson's Chi-square test for goodness-of-fit of linear distributions.

Usage

```
chi2testlinear(x, alpha)
```

Arguments

x	vector with normalized data between 0 and 1.
alpha	significance level.

Value

Object with test outcome h = TRUE/FALSE meaning rejection/acceptance of linear null hypothesis, and p-value or confidence level of acceptance.

chi2testtable *(Contingency) Tables for Chi-square Tests*

Description

(Contingency) tables for Pearson's Chi-square goodness-of-fit and independence tests.

Usage

```
chi2testtable(x, alpha, cf = FALSE)
```

Arguments

x	vector with normalized data between 0 and 1.
alpha	significance level.
cf	flag, if TRUE, bin counts for a multi-variate contingency table are applied (Bagnato et al., 2012). Otherwise rules for univariate tables are used (Cochran, 1952; Mann and Wald, 1942).

Value

Object with observed counts co of data x inside bins defined by edges be.

References

Cochran, W.G., The Chi Square Test of Goodness of Fit. The Annals of Mathematical Statistics, 1952. 23(3): p. 315-345.

Mann, H.B. and A. Wald, On the Choice of the Number of Class Intervals in the Application of the Chi Square Test. The Annals of Mathematical Statistics, 1942. 13(3): p. 306-317.

Bagnato, L., A. Punzo, and O. Nicolis, The autodependogram: a graphical device to investigate serial dependences. Journal of Time Series Analysis, 2012. 33(2): p. 233-254.

chi2testuniform *Goodness-of-Fit Test for Uniform Distribution*

Description

Pearson's Chi-square test for goodness-of-fit of a uniform distribution.

Usage

```
chi2testuniform(x, alpha)
```

Arguments

x	vector with normalized data between 0 and 1.
alpha	significance level.

Value

Object with test outcome h = TRUE/FALSE meaning rejection/acceptance of uniform null hypothesis, and p-value or confidence level of acceptance.

---

contourRect *Draw Contours in a Rectangle*

Description

The function contourRect draws the z contour levels of a rectangular domain in x-y-space with z-values given at the corners of the rectangle.

Usage

```
contourRect(xy, z, n = 20, zlb = 0, zub = 1,
  color = grDevices::colorRamp(c("white", "black")))
```

Arguments

xy	matrix with two rows and four columns containing x- and y-coordinates of the four corner points of the rectangle. The corner points are ordered in clockwise or counter-clockwise direction.
z	vector with four z-values at the four corner points.
n	abs(n) gives the number of local or global contour levels. If n > 0, n local contours are drawn within [min(z),max(z)]. If n < 0, n global contours result in [zlb,zub], but only the contours falling inside [min(z),max(z)] are drawn.
zlb, zub	determines the global range of z-values used to determine the contour colors. All values in z have to be contained in [zlb,zub].
color	function to assign plot colors that is generated, e.g., by colorRamp. color returns a color based on an argument in [0,1].

de.split *Split a Distribution Element*

Description

Splits a parent distribution element characterized by x, sze, and id along dimension dimens based on mode into two child elements.

Usage

```
de.split(dimens, x, sze, id, mode)
```

Arguments

dimens	split dimension.
x	data in element given as matrix with d rows or components and n columns or samples.
sze	vector representing the size of the element.
id	element index or identifier.
mode	for splitting: > 0 or < 0 for equal-size or -score split, respectively.

Value

Object containing the properties of the resulting two child distribution elements and the split position within the parent element.

det.construct *Distribution Element Tree (DET) Construction*

Description

The function det.construct generates a distribution element tree DET from available data. The DET can be used firstly in connection with det.query for density estimation. Secondly, with det.rnd, DETs can be used for smooth bootstrapping or more specifically conditional or unconditional random number generation.

Usage

```
det.construct(dta, mode = 2, lb = NA, ub = NA, alphag = 0.001,
  alphad = 0.001, progress = TRUE, dtalim = Inf, cores = 1)
```

Arguments

dta	matrix with d rows representing components or dimensions and n columns corresponding to data points or samples.
mode	order of distribution elements applied, default is mode = 2. Use +/-1 for constant or +/-2 for linear elements. mode > 0 and mode < 0 lead to equal-size and -score splits, respectively, in the element-refinement process.
lb, ub	vectors of length d with lower and upper sample-space bounds. If not provided or set to NA or 0, the bounds are determined from the data dta. If bounds are provided or given as 0, the data is not pre-whitened before the DET is computed.
alphag, alphad	significance levels for goodness-of-fit and independence tests, respectively, in element refinement or splitting process. Default is alphag = alphad = 1.0e-3. alphad is irrelevant for univariate data dta with d = 1.
progress	optional logical, if set to TRUE, a progress report about the DET construction process is provided.
dtalim	for large datasets, det.construct can be accelerated (with negligible impact on the resulting DET if dtalim is sufficiently large) by using only up to dtalim samples for element splitting tests. Setting dtalim < n impacts mainly the splitting at the tree root, with elements being large and thus containing many samples. Default is dtalim = Inf, which corresponds to using all available samples (no acceleration). When using dtalim < n, the samples have to be randomly arranged in dta: use for example dta[,sample(1:ncol(dta), ncol(dta), replace = FALSE)] to randomly rearrange the data.
cores	> 1 allows for parallel tree construction or branch splitting using the indicated number of cores. With cores = Inf, half of the available cores (see detectCores) are allocated. cores = 1 corresponds to serial tree construction (default).

Value

A DET object, which reflects the tree and pre-white transform, is returned.

References

Meyer, D.W. (2016) http://arxiv.org/abs/1610.00345 or Meyer, D.W., Statistics and Computing (2017) https://doi.org/10.1007/s11222-017-9751-9 and Meyer, D.W. (2017) http://arxiv.org/abs/1711.04632

Examples

```
## Gaussian mixture data
require(stats)
det <- det.construct(t(c(rnorm(1e5),rnorm(1e4)/100+2))) # default linear det (mode = 2)
x <- t(seq(-4,6,0.01)); p <- det.query(det, x); plot(x, p, type = "l")

## piecewise uniform data with peaks
x <- matrix(c(rep(0,1e3),rep(1,1e3), 2*runif(1e4),
              rep(0,5e2),rep(1,25e2),2*runif(9e3)), nrow = 2, byrow = TRUE)
det <- det.construct(x, mode = 1, lb = 0, ub = 0) # constant elements, no pre-whitening
```

det.cut *Identify Tree Leafs Intersected by Condition(s)*

Description

Identify distribution element tree (DET) leafs that are cut by conditions. The latter are defined in terms of positions xc along probability-space components with indices dc.

Usage

```
det.cut(det, xc, dc)
```

Arguments

det	distribution element tree object resulting from det.construct.
xc	vector with conditioning values of probability-space components listed in dc.
dc	integer vector with indices of conditioning components corresponding to xc.

Value

A vector containing the leaf indices that are cut by conditions xc of components dc is returned. If no leafs are found, the return vector has length 0.

Examples

```
# DET based on Gaussian data
require(stats); require(graphics)
n <- 8e4; x <- rnorm(n)
x <- matrix(c(x, x+rnorm(n,0,0.2)), ncol = 2)
det <- det.construct(t(x), lb = 0, ub = 0) # no pre-whitening
plot(x, type = "p", pch = ".", asp = 1)
# leaf elements that are cut by x1 = 2
leafs <- det.cut(det, xc = 2, dc = 1) # condition x1 = 2
# draw probability space (black) with cut leaf elements (red)
rect(det$lb[1], det$lb[2], det$ub[1], det$ub[2], border = "black")
for (k in 1:length(leafs)) {
   p <- det.de(det, leafs[k])$lb; w <- det.de(det, leafs[k])$size
   rect(p[1],p[2],p[1]+w[1],p[2]+w[2], border = "red")
}
# leafs cut by two conditions x1 = -3, x2 = -2 (blue)
leafs <- det.cut(det, xc = c(-2,-3), dc = c(2,1))
p <- det.de(det, leafs[1])$lb; w <- det.de(det, leafs[1])$size
rect(p[1],p[2],p[1]+w[1],p[2]+w[2], border = "blue")
```

det.de *Extract Distribution Element Characteristics*

Description

The function det.de extracts the distribution element with index ind from a distribution element tree (DET) generated by the function det.construct.

Usage

```
det.de(det, ind)
```

Arguments

det	distribution element tree object resulting from det.construct.
ind	index of element to extract from det.

Value

A list with the element characteristics is returned: p probability density, theta element parameters, lb lower bound, size of element, div divisions or splits along dimensions leading to final element.

det.leafs *Extract Leaf Elements from Distribution Element Tree*

Description

The function det.leafs extracts the distribution elements at the branch ends of a DET generated by the function det.construct.

Usage

```
det.leafs(det)
```

Arguments

det	distribution element tree object resulting from det.construct.

Value

A list of vectors containing the leaf element data is returned: p probability density, theta element parameters, lb lower bound, size of element, div divisions or splits along dimensions leading to final element.

Examples

```
require(stats); require(graphics)
# generate DET based on bi-variate Gaussian data
n <- 1e4; x <- rnorm(n)
x <- matrix(c(x, x+rnorm(n,0,0.2)), nrow = 2, byrow = TRUE)
det <- det.construct(x)
# plot data and element pattern
leafs <- det.leafs(det)
plot(t(x), type = "p", pch = ".", asp = 1)
for (k in 1:length(leafs$p)) {
   p <- leafs$lb[,k] # element corner point
   w <- leafs$size[,k] # element size
   elem <- rbind(c(p[1],p[1]+w[1],p[1]+w[1],p[1],p[1]),
                 c(p[2],p[2],p[2]+w[2],p[2]+w[2],p[2])) # element rectangle
   elem <- t(det$A) %*% elem + det$mu %*% t(rep(1,5)) # pre-white transform
   lines(elem[1,],elem[2,]) # draw element
}
```

`det.query` *Density Estimation Based on Distribution Element Trees*

Description

The function `det.query` evaluates probability densities at the query points `x` based on a distribution element tree (DET). The latter is calculable with `det.construct` based on available data.

Usage

```
det.query(det, x, cores = 1)
```

Arguments

`det`	distribution element tree object resulting from `det.construct` based on data with `d` components or dimensions.
`x`	matrix containing `n` query points (columns) with `d` components or dimensions (rows).
`cores`	for large query-point sets, `cores` > 1 allows for parallel tree query using the indicated number of cores. `cores` = `Inf` allocates half of the available cores (see `detectCores`). The default is `cores` = 1 corresponding to serial tree query.

Value

A vector containing the probability density at the query points `x` is returned.

Examples

```
## 1d example
require(stats); require(graphics)
# DET generation based on Gaussian/uniform data
det <- det.construct(t(c(rnorm(1e5,2,3),runif(1e5)-3)))
# density evaluation based on DET at equidistant query points
x <- t(seq(-10,14,0.01)); p <- det.query(det, x)
# compare DET estimate (black) against Gaussian/uniform reference (red)
plot(x, p, type = "l", col = "black")
lines(x, (dnorm(x,2,3)+dunif(x+3))/2, col = "red")

## 2d example
require(stats); require(graphics)
# mean and covariance of Gaussian, data generation
mu <- c(3,5); C <- matrix(c(4.0,-2.28,-2.28,1.44), nrow = 2)
A <- eigen(C); B <- diag(A$values); A <- A$vectors
x <- matrix(rnorm(2e4), nrow = 2)
x <- t(A %*% (sqrt(B) %*% x) + mu %*% t(rep(1,ncol(x))))
# bounds and resolution of x1-x2 query grid
lb <- c(-5,0); ub <- c(11,10); np <- c(320,200)
x1 <- lb[1] + (ub[1]-lb[1])*((1:np[1])-0.5)/np[1]
x2 <- lb[2] + (ub[2]-lb[2])*((1:np[2])-0.5)/np[2]
xp <- rbind(rep(x1,np[2]), rep(x2,each = np[1])) # grid points
# plotting
split.screen(c(2, 2)); screen(1)
plot(x, type = "p", pch = ".", asp = 1, main = "data")
# DET estimator
det <- det.construct(t(x))
yd <- matrix(det.query(det, xp), nrow = np[1])
screen(2)
image(list(x = x1, y = x2, z = yd), asp = 1,
      col = grDevices::gray((100:0)/100), main = "det")
# Gaussian density for comparison
yr <- yr <- exp(-1/2 * colSums(
   (t(solve(C)) %*% (xp - mu%*%t(rep(1,ncol(xp))))) *
                    (xp - mu%*%t(rep(1,ncol(xp))))
                              ) / sqrt((2*pi)^2*det(C))
yr <- matrix(yr, nrow = np[1])
screen(3)
image(list(x = x1, y = x2, z = yr), asp = 1,
      col = grDevices::gray((100:0)/100), main = "reference")
```

`det.rnd` *Bootstrapping from Distribution Element Trees*

Description

Smooth bootstrapping or generation of (un)conditional random vectors based on an existing distribution element tree (DET).

Usage

```
det.rnd(n, det, xc = vector("numeric", length = 0), dc = vector("numeric",
  length = 0), cores = Inf)
```

Arguments

n	number of samples to generate.
det	distribution element tree object resulting from det.construct.
xc	vector with conditioning values of probability-space components listed in dc. If empty (default), unconditional samples are generated.
dc	integer vector with indices of conditioning components corresponding to xc. If empty (default), unconditional samples are generated.
cores	for large n, cores > 1 allows for parallel bootstrapping using the indicated number of cores. The default is cores = Inf, which allocates half of the available cores (see detectCores). cores = 1 corresponds to serial bootstrapping.

Value

A matrix containing n random vectors (columns) with d components or dimensions (rows) is returned. d is equal to the dimensionality of the underlying det object.

Examples

```
## 2d example
require(stats); require(graphics)
# data from uniform distribution on a wedge
x <- matrix(runif(2e4), ncol = 2); x <- x[x[,2]<x[,1],]
x2c <- 0.75 # conditioning component
# data and conditioning line
split.screen(c(2, 1)); screen(1)
plot(x, type = "p", pch = ".", asp = 1)
lines(c(0,1), x2c*c(1,1), col = "red")
# DET construction and bootstrapping
det <- det.construct(t(x), mode = 1, lb = 0, ub = 0) # const. de's, no pre-white
y <- det.rnd(1e3, det, xc = x2c, dc = 2, cores = 2) # conditional bootstrap'g
# compare generated data (black) with exact cond. distribution (red)
screen(2); det1(y[1,], mode = 1)
lines(c(0,x2c,x2c,1,1),c(0,0,1/(1-x2c),1/(1-x2c),0), col = "red")

## example 2d unconditional
require(stats); require(graphics)
x <- matrix(runif(2e4), ncol = 2); x <- x[x[,2]<x[,1],] # uniform wedge
det <- det.construct(t(x), mode = 1, lb = 0, ub = 0) # no pre-white
y <- t(det.rnd(nrow(x), det, cores = 2)) # smooth bootstrapping
split.screen(c(2, 1))
screen(1); plot(x, type = "p", pch = ".", asp = 1, main = "original")
screen(2); plot(y, type = "p", pch = ".", asp = 1, main = "bootstrapped")

## example 3d
require(stats); require(graphics)
```

```
# mean and covariance of Gaussian, data generation
mu <- c(1,3,2); C <- matrix(c(25,7.5,1.75,7.5,7,1.35,1.75,1.35,0.43), nrow = 3)
A <- eigen(C); B <- diag(A$values); A <- A$vectors
x <- matrix(rnorm(3e4), nrow = 3)
x <- A %*% (sqrt(B) %*% x) + mu %*% t(rep(1,ncol(x)))
lbl <- "x1 | x2 = 7 & x3 = 2.5"
pairs(t(x), labels = c("x1","x2","x3"), pch = ".", main = lbl)
# bootstrapping conditional on x2 and x3
det <- det.construct(x, lb = 0, ub = 0)
xc <- c(2.5,7); d <- c(3,2) # conditional on x2 = 7 & x3 = 2.5
y <- det.rnd(1e4, det, xc, d, cores = 1)
det1(y[1,], mode = 1, main = lbl)
# compare with exact conditional density
Cm1 <- solve(C); var1 <- det(C)/det(C[2:3,2:3]) # conditional variance
mu1 <- mu[1] + var1*((mu[2]-xc[d==2])*Cm1[1,2]+(mu[3]-xc[d==3])*Cm1[1,3]) # cond. mean
x1 <- mu1 + seq(-50,50)/50 * 5*sqrt(var1) # x1-axis grid points
lines(x1, dnorm(x1,mu1,sqrt(var1)), col = "red")
```

`det1` *Density Estimation for Univariate Data Based on Distribution Element Trees*

Description

One-dimensional piecewise linear or constant probability density estimator based on distribution element trees (DETs).

Usage

```
det1(dta, mode = 2, bounds = c(0, 0), alpha = 0.001, main = NULL,
  dtalim = Inf, cores = 1)
```

Arguments

`dta`	vector with data
`mode`	order of distribution elements applied, default is `mode = 2`. Use `+/-1` for constant or `+/-2` for linear elements. `mode > 0` and `mode < 0` lead to equal-size and -score splits, respectively, in the element-refinement process.
`bounds`	`c(lb,ub)`, where `lb` and `ub` are lower and upper bounds of the probability space. If both bounds are set to 0 (default), the bounds are determined based on the data `dta`.
`alpha`	significance level for goodness-of-fit testing in element refinement or splitting process. Default is `alpha = 1.0e-3`.
`main`	an overall plot title, see `title`.
`dtalim`	allows to limit the number of samples used in tests guiding the element splitting process. Default is `dtalim = Inf`, which corresponds to using all available samples, see `det.construct`.
`cores`	number of cores for parallel tree construction. Default is `1` for serial construction, see `det.construct`.

Examples

```
require(stats)
det1(rbeta(5e5, shape1 = 1.05, shape2 = 0.8), mode = -1,
     bounds = c(-0.1,1.1), main = "beta, const. elements, equal-scores splits")
x <- seq(-0.1,1.1,0.005); lines(x, dbeta(x,shape1 = 1.05,shape2 = 0.8), col = "red")
det1(rbeta(5e5, shape1 = 1.05, shape2 = 0.8), mode = -2,
     bounds = c(-0.1,1.1), main = "beta, linear elements, equal-scores splits")
x <- seq(-0.1,1.1,0.005); lines(x, dbeta(x,shape1 = 1.05,shape2 = 0.8), col = "red")
det1(rnorm(5e5), mode = 2, cores = 1, main = "Gaussian, linear elements, equal-size splits")
x <- seq(-5,5,0.05); lines(x, dnorm(x), col = "red")
det1(runif(5e5), mode = 1, bounds = c(-0.1,1.1),
     main = "uniform, const. elements, equal-size splits")
x <- seq(-0.1,1.1,0.005); lines(x, dunif(x), col = "red")
```

---

det2 *Density Estimation for Bivariate Data Based on Distribution Element Trees*

Description

Two-dimensional piecewise linear or constant probability density estimator based on distribution element trees (DETs).

Usage

```
det2(dta, mode = 2, bounds = list(NA, NA), alphag = 0.001,
  alphad = 0.001, main = NULL, nc = 20, dtalim = Inf, cores = 1,
  color = grDevices::colorRamp(c("white", "black")))
```

Arguments

dta	matrix with two rows containing data (samples in columns).
mode	order of distribution elements applied, default is mode = 2. Use +/-1 for constant or +/-2 for linear elements. mode > 0 and mode < 0 lead to equal-size and -score splits, respectively, in the element-refinement process.
bounds	list(lb,ub), where lb and ub are vectors representing the lower and upper bounds of the probability space. If both bounds are set to NA (default) or 0, the bounds are determined based on the data dta. Additionally, if the bounds are set to 0, pre-whitening is not applied to the data.
alphag, alphad	significance levels for goodness-of-fit and independence tests, respectively, in element refinement or splitting process. Default is alphag = alphad = 1.0e-3.
main	an overall plot title, see title. If main = NULL (default), the density range is provided as a title.
nc	number of contour levels (default is 20).
dtalim	allows to limit the number of samples used in tests guiding the element splitting process. Default is dtalim = Inf, which corresponds to using all available samples, see det.construct.

cores	number of cores for parallel tree construction. Default is cores = 1 for serial processing, see cores in det.construct.
color	function to assign plot colors that is generated, e.g., by colorRamp. color returns a color based on an argument in [0,1].

Examples

```
## uniform
require(stats)
det2(rbind(runif(5e3),1+2*runif(5e3)), mode = 1, bounds = list(c(-0.1,0),c(1.1,4)))
det2(rbind(1:100,101:200+runif(100)), mode = 2) # data on a line

## Gaussian
require(stats); require(graphics); require(grDevices)
n <- 5e3; x <- rnorm(n)
x <- matrix(c(x, x+rnorm(n,0,0.5)), ncol = 2)
split.screen(c(2,2))
color = colorRamp(c("#FFFFFF","#E6E680","#E6BF1A",
                    "#E68000","#FF4026","#993380",
                    "#4D26BF","#262680","#000000"))
screen(3); plot(x, type = "p", pch = ".", main = "data")
screen(1); det2(t(x), mode = 1, main = "constant det estimator", color = color)
screen(2); det2(t(x), main = "linear det estimator", color = color)
screen(4)
det2(t(x), mode = 1, bounds = list(0,0), main = "const. det, no pre-white", color = color)
```

---

detpack *Distribution Element Trees for Density Estimation and Bootstrapping*

Description

Distribution element trees (DETs) enable the estimation of probability densities based on (possibly large) datasets. Moreover, DETs can be used for random number generation or smooth bootstrapping both in unconditional and conditional modes. In the latter mode, information about certain probability-space components is taken into account when sampling the remaining probability-space components.

Details

The function det.construct translates a dataset into a DET. To evaluate the probability density based on a precomputed DET at arbitrary query points, det.query is used. The functions det1 and det2 provide density estimation and plotting for one- and two-dimensional datasets. (Un)conditional smooth bootstrapping from an available DET, can be performed by det.rnd. To inspect the structure of a DET, the functions det.de and det.leafs are useful. While det.de enables the extraction of an individual distribution element from the tree, det.leafs extracts all leaf elements at branch ends.

Author(s)

Daniel Meyer, <meyerda@ethz.ch>

References

Distribution element tree basics and density estimation, see Meyer, D.W. (2016) http://arxiv.org/abs/1610.00345 or Meyer, D.W., Statistics and Computing (2017) https://doi.org/10.1007/s11222-017-9751-9.

DETs for smooth bootstrapping, see Meyer, D.W. (2017) http://arxiv.org/abs/1711.04632 or Meyer, D.W., Journal of Computational and Graphical Statistics (2018) https://doi.org/10.1080/10618600.2018.1482768.

`dimstosplit` *Determine Split Dimension(s)*

Description

Determine the split dimensions of an existing distribution element in the DET refinement-process based on statistical tests.

Usage

```
dimstosplit(x, sze, mode, alphag, alphad)
```

Arguments

`x`	data in element given as matrix with `d` rows or components and `n` columns or samples.
`sze`	vector representing the element size.
`mode`	element order and split mode as detailed in `det.construct`.
`alphag, alphad`	significance levels for goodness-of-fit and independence tests, respectively, in element refinement or splitting process. `alphad` is irrelevant for univariate data `x` with `d = 1`.

Value

An object comprised of the split dimension(s) or `NA` for no split, and the resulting child element parameters is returned.

Index