Package: mmand (via r-universe)

July 16, 2024

Version 1.6.3

Date 2023-02-07

Title Mathematical Morphology in Any Number of Dimensions

Author Jon Clayden

Maintainer Jon Clayden <code@clayden.org>

Imports methods, Rcpp

LinkingTo Rcpp

Suggests loder, tinytest, covr

Description Provides tools for performing mathematical morphology operations, such as erosion and dilation, on data of arbitrary dimensionality. Can also be used for finding connected components, resampling, filtering, smoothing and other image processing-style operations.

Encoding UTF-8

License GPL-2

URL https://github.com/jonclayden/mmand

BugReports https://github.com/jonclayden/mmand/issues

Roxygen list(old_usage=TRUE)

RoxygenNote 7.2.3

Repository https://jonclayden.r-universe.dev

RemoteUrl https://github.com/jonclayden/mmand

RemoteRef HEAD

RemoteSha 1e73bb40e9ff4c57ba6de7dc89fca81d36fc86f3

Contents

`binarise` *Binarise a numeric array*

Description

This function binarises an array, setting all nonzero elements to unity.

Usage

```
binarise(x)
```

Arguments

`x`	An object that can be coerced to an array, or for which a `morph` method exists.

Value

A morphed array with the same dimensions as the original array.

Author(s)

Jon Clayden <code@clayden.org>

See Also

`morph` for the function underlying this operation, and `erode` for mathematical morphology functions.

binary *Check for a binary array*

Description

This function checks whether a numeric array is binary, with only one unique nonzero value, or not.

Usage

```
binary(x)
```

Arguments

x	An object that can be coerced to a numeric array.

Value

A logical value indicating whether the array is binary or not. Binary in this case means that the array contains only one unique nonzero value, which is stored with the return value in an attribute.

Author(s)

Jon Clayden <code@clayden.org>

components *Find connected components*

Description

The `components` function finds connected components in a numeric array. The kernel determines which neighbours are considered connected (e.g. including or excluding diagonal neighbours), and will usually have width 3 in each dimension.

Usage

```
components(x, kernel, ...)

## Default S3 method:
components(x, kernel, ...)
```

Arguments

x	Any object. For the default method, this must be coercible to an array.
kernel	An object representing the kernel to be used, which must be coercible to an array. It must have odd width in all dimensions, but does not have to be isotropic in size. The kernel's dimensionality may be less than that of the target array, `x`. See `kernels` for kernel-generating functions.
...	Additional arguments to methods.

Value

An array of the same dimension as the original, whose integer-valued elements identify the component to which each element in the array belongs. Zero values in the original array will result in NAs.

Author(s)

Jon Clayden <code@clayden.org>

See Also

kernels for kernel-generating functions.

Examples

```
x <- c(0,0,1,0,0,0,1,1,1,0,0)
k <- c(1,1,1)
components(x,k)
```

---

display *Display a 2D image*

Description

This function displays a 2D greyscale or RGB colour image. It is a wrapper around image, with more sensible defaults for images. It is (S3) generic. A method for 3D arrays is provided, which assumes that the third dimension corresponds to channel (grey/alpha for two channels, red/green/blue for three, red/green/blue/alpha for four).

Usage

```
display(x, ...)

## Default S3 method:
display(x, transpose = TRUE, useRaster = TRUE,
  add = FALSE, col = grey(0:255/255), ...)

## S3 method for class 'matrix'
display(x, ...)

## S3 method for class 'array'
display(x, max = NULL, ...)
```

Arguments

x	An R object. For the default method, it must be coercible to a numeric matrix.
...	Additional arguments to image, or the default method.
transpose	Whether to transpose the matrix before display. This is usually necessary due to the conventions of image.
useRaster	Whether to use raster graphics if possible. This is generally preferred for speed. Passed to image.
add	Whether to add the image to an existing plot. If TRUE, zero values in the image will be converted to NAs for plotting purposes, to make them transparent. This will not affect the original image data.
col	The colour scale to use. The default is 256 grey levels. The array method overrides this appropriately.
max	The maximum colour value for each channel. If NULL, the default, this is taken from the "range" attribute, if there is one, otherwise it is 255 for integer-mode arrays, and 1 otherwise. Passed to rgb.

Details

Relative to the defaults for image (from the graphics package), this function transposes and then inverts the matrix along the y-direction, uses a grey colour scale, fills the entire device with the image, and tries to size the image correctly given the dot pitch of the display. Unfortunately the latter is not always possible, due to downstream limitations.

If x has attributes "range", "background", "asp" or "dpi", these are respected.

Value

This function is called for its side-effect of displaying an image on a new R device.

Author(s)

Jon Clayden <code@clayden.org>

---

distanceTransform *Distance transforms*

Description

The Euclidean distance transform produces an array like its argument, but with element values representing the Euclidean distance to the nearest nonzero element. The input is treated as logically binary, with all nonzero values treated as "on", and all zeroes as "off".

Usage

```
distanceTransform(x, ...)

## Default S3 method:
distanceTransform(x, pixdim = TRUE, signed = FALSE,
  threads = getOption("mmand.threads"), ...)
```

Arguments

x	Any object. For the default method, this must be coercible to an array.
...	Additional arguments to methods.
pixdim	An optional numeric vector or logical value. In the former case it will be taken as giving the physical size of the array elements of x along each dimension, and these will be incorporated into the distance calculation. If TRUE, the default, the "pixdim" attribute of x will be used for this purpose, if it is present. If FALSE, any such attribute will be ignored, and distances will always be counted in array elements, with all dimensions treated equally.
signed	Logical value. If TRUE, the signed distance transform is returned, such that distances from the region boundary are negative within the region and positive outside. Otherwise, distances are zero within the region.
threads	If a positive integer, and the package is compiled with OpenMP support, the number of threads to use during the calculation.

Value

An array of the same dimension as the original, whose elements give the Euclidean distance from that element to the nearest "on" element in the original.

Author(s)

Jon Clayden <code@clayden.org>

References

This implementation is based on the "marching parabolas" algorithm described by Felzenszwalb and Huttenlocher in the paper below.

P.F. Felzenszwalb & D.P. Huttenlocher (2012). Distance transforms of sampled functions. Theory of Computing 8(19):415-428.

Examples

```
x <- c(0,0,1,0,0,0,1,1,1,0,0)
distanceTransform(x)
distanceTransform(x, pixdim=2)
```

erode *Standard mathematical morphology operations*

Description

These functions provide standard mathematical morphology operations, which can be applied to array data with any number of dimensions. Binary and greyscale morphology is supported.

Usage

```
erode(x, kernel)

dilate(x, kernel)

opening(x, kernel)

closing(x, kernel)
```

Arguments

x	An object that can be coerced to an array, or for which a morph method exists.
kernel	An array representing the kernel to be used. See shapeKernel for functions to generate a suitable kernel.

Details

The erode function uses the kernel as an eraser, centring it on each zero-valued pixel, which has the effect of eroding the extent of nonzero areas. Dilation has the opposite effect, extending the nonzero regions in the array. Opening is an erosion followed by a dilation, and closing is a dilation followed by an erosion, using the same kernel in both cases.

If the kernel has only one unique nonzero value, it is described as "flat". For a flat kernel, the erosion is the minimum value of x within the nonzero region of kernel. For a nonflat kernel, this becomes the minimum value of x - kernel. Dilation is the opposite operation, taking the maximum within the kernel.

Value

A morphed array with the same dimensions as the original array.

Author(s)

Jon Clayden <code@clayden.org>

See Also

morph for the function underlying all of these operations, kernels for kernel-generating functions, binarise for binarising an array, and gaussianSmooth for smoothing. The EBImage Bioconductor package also supplies functions to perform these operations, and may be slightly faster, but only works in two dimensions.

Examples

```
x <- c(0,0,1,0,0,0,1,1,1,0,0)
k <- c(1,1,1)
erode(x,k)
dilate(x,k)
```

---

gameOfLife *Conway's Game of Life*

Description

An implementation of Conway's Game of Life, a classical cellular automaton, using the morph function. The gosperGliderGun function provides an interesting starting configuration.

Usage

```
gameOfLife(init, size, density = 0.3, steps = 200, viz = FALSE,
  tick = 0.5)

gosperGliderGun()
```

Arguments

init	The initial state of the automaton, a binary matrix. If missing, the initial state will be randomly generated, with a population density given by density.
size	The dimensions of the board. Defaults to the size of init, but must be given if that parameter is missing. If both are specified and size is larger than the dimensions of init, then the latter will be padded with zeroes.
density	The approximate population density of the starting state. Ignored if init is provided.
steps	The number of generations of the automaton to simulate.
viz	If TRUE, the state of the system at each generation is plotted.
tick	The amount of time, in seconds, to pause before plotting each successive generation. Ignored if viz is FALSE.

Details

Conway's Game of Life is a simple cellular automaton, based on a 2D matrix of "cells". It shows complex behaviour based on four simple rules. These are:

1. Any live cell with fewer than two live neighbours dies, as if caused by under-population.
2. Any live cell with two or three live neighbours lives on to the next generation.
3. Any live cell with more than three live neighbours dies, as if by overcrowding.
4. Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

Live and dead cells are represented by 1s and 0s in the matrix, respectively.

The initial state and the rules above completely determine the behaviour of the system. The Gosper glider gun is an interesting starting configuration that generates so-called "gliders", which propagate across the board.

In principle the size of the board in a cellular automaton is infinite. Of course this is not easy to simulate, but this implementation adds a border of two extra cells around the board on all sides to approximate an infinite board slightly better. These are not visualised, nor returned in the final state.

Value

A binary matrix representing the final state of the system after `steps` generations.

Author(s)

Jon Clayden <code@clayden.org>

See Also

The `morph` function, which powers this simulation.

Examples

```
## Not run: gameOfLife(init=gosperGliderGun(), size=c(40,40), steps=50, viz=TRUE)
```

---

gaussianSmooth *Smooth a numeric array with a Gaussian kernel*

Description

This function smoothes an array using a Gaussian kernel with a specified standard deviation.

Usage

```
gaussianSmooth(x, sigma)
```

Arguments

x	An object that can be coerced to an array, or for which a morph method exists.
sigma	A numeric vector giving the standard deviation of the kernel in each dimension. Can have lower dimensionality than the target array.

Details

This implementation takes advantage of the separability of the Gaussian kernel for speed when working in multiple dimensions. It is therefore equivalent to, but much faster than, directly applying a multidimensional kernel.

Value

A morphed array with the same dimensions as the original array.

Author(s)

Jon Clayden <code@clayden.org>

See Also

morph for the function underlying this operation, gaussianKernel for generating Gaussian kernels (which is also used by this function), and erode for mathematical morphology functions.

---

isKernel *Kernel-generating functions*

Description

These functions can be used to generate kernels for morphological, smoothing or resampling operations. There are two types of kernels: kernel arrays, which are used with morph, and kernel functions, which are used with resample.

Usage

```
isKernel(object)

isKernelArray(object)

isKernelFunction(object)

kernelArray(values)

shapeKernel(width, dim = length(width), type = c("box", "disc", "diamond"),
  binary = TRUE, normalised = FALSE)

gaussianKernel(sigma, dim = length(sigma), size = 6 * sigma,
```

```
  normalised = TRUE)

sobelKernel(dim, axis = 1)

kernelFunction(name = c("box", "triangle", "mitchell-netravali", "lanczos"),
  ...)

boxKernel()

triangleKernel()

mitchellNetravaliKernel(B = 1/3, C = 1/3)

mnKernel(B = 1/3, C = 1/3)

lanczosKernel()
```

Arguments

object	Any object.
values	A numeric vector or array, containing the values of the kernel array.
width	A numeric vector giving the width of the shape in each dimension, in array elements. Does not need to be integer-valued, or equal for all dimensions. Will be recycled to length dim if that parameter is also specified.
dim	An integer value giving the dimensionality of the kernel. Defaults to the length of width or sigma, where available.
type	A string giving the type of shape to produce. In one dimension, these shapes are all equivalent.
binary	If FALSE, the value of the kernel at each point represents the proportion of the array element within the shape. If TRUE, these values are binarised to be 1 if at least half of the element is within the shape, and 0 otherwise.
normalised	If TRUE, the sum of non-missing elements of the kernel will be unity. Note that this is the default for gaussianKernel, but not for shapeKernel.
sigma	A numeric vector giving the standard deviation of the underlying Gaussian distribution in each dimension, in array elements. Does not need to be equal for all dimensions. Will be recycled to length dim if that parameter is also specified.
size	A numeric vector giving the width of the kernel in each dimension, which will be rounded up to the nearest odd integer. Defaults to four times the corresponding sigma value.
axis	The axis along which the gradient operator will be applied.
name	A string giving the name of the kernel function required.
...	Parameters for the kernel function.
B, C	Mitchell-Netravali coefficients, each of which must be between 0 and 1.

Details

There are two forms of kernel used by this package. Kernel arrays, otherwise known in mathematical morphology as structuring elements, are numeric arrays with class `kernelArray`. They are defined on a grid of odd width, and are used by `morph` and related functions. Kernel functions, by contrast, are represented in R as a list containing a name and, optionally, some parameters. The real implementation is in C++. They are defined everywhere within the support of the kernel, and are used by `resample` and friends. The key distinction is in whether the kernel will always be centred exactly on the location of an existing value in the data (for kernel arrays) or not (for kernel functions).

The `kernelArray` and `kernelFunction` functions create objects of the corresponding classes, while `isKernelArray` and `isKernelFunction` test for them. In addition, `isKernel` returns `TRUE` if its argument is of either kernel class.

The remaining functions generate special-case kernels: `shapeKernel` generates arrays with nonzero elements in a box, disc or diamond shape for use with `morphology` functions; `gaussianKernel` generates Gaussian coefficients and is used by `gaussianSmooth`; `sobelKernel` generates the Sobel-Feldman gradient operator, for use by `sobelFilter`; `boxKernel` is used for "nearest neighbour" resampling, and `triangleKernel` for linear, bilinear, etc. The Mitchell-Netravali kernel, a.k.a. BC-spline, is based on a family of piecewise-cubic polynomial functions, with support of four times the pixel separation in each dimension. The default parameters are the ones recommended by Mitchell and Netravali as a good trade-off between various artefacts, but other well-known special cases include B=1, C=0 (the cubic B-spline) and B=0, C=0.5 (the Catmull-Rom spline). `mnKernel` is a shorter alias for `mitchellNetravaliKernel`. Finally, the Lanczos kernel is a five-lobe windowed sinc function.

Value

For `isKernel`, `isKernelArray` and `isKernelFunction`, a logical value. For `kernelArray`, `shapeKernel`, `gaussianKernel` and `sobelKernel`, a kernel array. For `kernelFunction`, `boxKernel`, `triangleKernel`, `mitchellNetravaliKernel` and `mnKernel`, a kernel function.

Author(s)

Jon Clayden <code@clayden.org>

References

The Mitchell-Netravali kernel is described in the following paper.

D.P. Mitchell & A.N. Netravali (1988). Reconstruction filters in computer graphics. Computer Graphics 22(4):221-228.

See Also

`morph` for general application of kernel arrays to data, `morphology` for mathematical morphology functions, `resample` for resampling, and `gaussianSmooth` for smoothing. Also see `sampleKernelFunction` for kernel sampling and plotting.

Examples

```
shapeKernel(c(3,5), type="diamond")
gaussianKernel(c(0.3,0.3))
mnKernel()
```

`meanFilter` *Apply a filter to an array*

Description

These functions apply mean, median or Sobel filters to an array.

Usage

```
meanFilter(x, kernel)

medianFilter(x, kernel)

sobelFilter(x, dim, axis = 0)
```

Arguments

`x`	An object that can be coerced to an array, or for which a `morph` method exists.
`kernel`	A kernel array, indicating the scope of the filter.
`dim`	For `sobelFilter`, the dimensionality of the kernel. If missing, this defaults to the dimensionality of `x`.
`axis`	For `sobelFilter`, the axis along which to apply the operator, or 0 to apply it along all directions and generate a magnitude image. See also `sobelKernel`.

Value

A morphed array with the same dimensions as the original array.

Author(s)

Jon Clayden <code@clayden.org>

See Also

`morph` for the function underlying these operations, and `kernels` for kernel-generating functions.

morph *Morph an array with a kernel*

Description

The morph function applies a kernel to a target array. Optionally, applying the kernel to a particular array element can be made conditional on its value, or the number of nonzero immediate neighbours that it has. The morph function is (S3) generic.

Usage

```
morph(x, kernel, ...)

## Default S3 method:
morph(x, kernel, operator = c("+", "-", "*", "i", "1", "0",
  "=="), merge = c("sum", "min", "max", "mean", "median", "all", "any"),
  value = NULL, valueNot = NULL, nNeighbours = NULL,
  nNeighboursNot = NULL, renormalise = TRUE, ...)
```

Arguments

x	Any object. For the default method, this must be coercible to an array.
kernel	An object representing the kernel to be applied, which must be coercible to an array. It must have odd width in all dimensions, but does not have to be isotropic in size. The kernel's dimensionality may be less than that of the target array, x. See kernels for kernel-generating functions.
...	Additional arguments to methods.
operator	The operator applied elementwise within the kernel, as a function of the original image value and the kernel value. Arithmetic operators are as usual; "i" is the identity operator, where every value within the kernel will be included as-is; "1" and "0" include a 1 or 0 for each element within the kernel's nonzero region; "==" produces a 1 where the image matches the kernel, and 0 elsewhere.
merge	The operator applied to combine the elements into a final value for the centre pixel. All have their usual meanings.
value	An optional vector of values in the target array for which to apply the kernel. Takes priority over valueNot if both are specified.
valueNot	An optional vector of values in the target array for which not to apply the kernel.
nNeighbours	An optional numeric vector giving allowable numbers of nonzero neighbours (including diagonal neighbours) for array elements where the kernel will be applied. Takes priority over nNeighboursNot if both are specified.
nNeighboursNot	An optional numeric vector giving nonallowable numbers of nonzero neighbours (including diagonal neighbours) for array elements where the kernel will be applied.
renormalise	If TRUE, the default, and merge is "sum", the sum will be renormalised relative to the sum over the visited part of the kernel. This avoids low-intensity bands around the edges of a morphed image.

Value

A morphed array with the same dimensions as the original array.

Author(s)

Jon Clayden <code@clayden.org>

See Also

`kernels` for kernel-generating functions, and `morphology` for more specific mathematical morphology functions. `gameOfLife` shows how this function can be used for non-morphological purposes, in that case to power a cellular automaton. See also the `kernel` and `kernapply` functions in the `stats` package, particularly if you want to smooth time series.

---

neighbourhood *Get neighbourhood information for an array*

Description

This function provides information about the structure of a neighbourhood of a given width within a specified array.

Usage

```
neighbourhood(x, width)
```

Arguments

`x`	An object that can be coerced to an array.
`width`	A vector giving the width of the neighbourhood in each dimension, which will be recycled if necessary. Must not be greater than the size of the array. Even values are rounded up to the next odd integer.

Value

A list with the following elements.

`widths`	The width of the neighbourhood along each dimension. Currently all elements of this vector will be the same.
`size`	The number of pixels within the neighbourhood.
`locs`	A matrix giving the coordinates of each neighbourhood pixel relative to the centre pixel, one per row.
`offsets`	Vector offsets of the neighbourhood values within `x`.

Author(s)

Jon Clayden <code@clayden.org>

`resample` *Resample an array*

Description

The `resample` function uses a kernel function to resample a target array. This can be thought of as a generalisation of array indexing which allows fractional indices. It is (S3) generic. The `rescale` function is an alternative interface for the common case where the image is being scaled to a new size.

Usage

```
resample(x, points, kernel, ...)

## Default S3 method:
resample(x, points, kernel, pointType = c("auto",
  "general", "grid"), threads = getOption("mmand.threads"), ...)

rescale(x, factor, kernel, ...)
```

Arguments

`x`	Any object. For the default method, this must be coercible to an array.
`points`	Either a matrix giving the points to sample at, one per row, or a list giving the locations on each axis, which will be made into a grid.
`kernel`	A kernel function object, used to provide coefficients for each resampled value, or the name of one.
`...`	Additional options, such as kernel parameters.
`pointType`	A string giving the type of the point specification being used. Usually can be left as `"auto"`.
`threads`	If a positive integer, and the package is compiled with OpenMP support, the number of threads to use during the calculation.
`factor`	A vector of scale factors, which will be recycled to the dimensionality of `x`.

Value

If a generalised sampling scheme is used (i.e. with `points` a matrix), the result is a vector of sampled values. For a grid scheme (i.e. with `points` a list, including for `rescale`), it is a resampled array.

Author(s)

Jon Clayden <code@clayden.org>

See Also

kernels for kernel-generating functions.

Examples

```
resample(c(0,0,1,0,0), seq(0.75,5.25,0.5), triangleKernel())
```

---

sampleKernelFunction *Sampling and plotting kernels*

Description

These functions can be used to sample and plot kernel profiles.

Usage

```
sampleKernelFunction(kernel, values)

## S3 method for class 'kernelArray'
plot(x, y, axis = 1, lwd = 2, col = "red", ...)

## S3 method for class 'kernelFunction'
plot(x, y, xlim = c(-2, 2), lwd = 2,
  col = "red", ...)
```

Arguments

kernel	A kernel function object.
values	A vector of values to sample the function at. These are in units of pixels, with zero representing the centre of the kernel.
x	A kernel object of the appropriate class.
y	Ignored.
axis	The axis to profile along.
lwd	The line width to use for the kernel profile.
col	The line colour to use for the kernel profile.
...	Additional plot parameters.
xlim	The limits of the range used to profile the kernel.

Value

For sampleKernelFunction a vector of kernel values at the locations requested. The plot methods are called for their side-effects.

Author(s)

Jon Clayden <code@clayden.org>

See Also

kernels for kernel-generating functions.

Examples

```
sampleKernelFunction(mnKernel(), -2:2)
plot(mnKernel())
```

---

skeletonise *Skeletonise a numeric array*

Description

Skeletonisation is the process of thinning a shape to a medial line or surface, and can be achieved using elementary mathematical morphology operations in a number of ways. Three methods are available through this function. They are all iterative and therefore relatively time-consuming.

Usage

```
skeletonise(x, kernel = NULL, method = c("lantuejoul", "beucher",
  "hitormiss"))
```

Arguments

x	An object that can be coerced to an array, or for which a morph method exists.
kernel	An array representing the kernel to be used for the underlying morphology operations. The kernel is fixed for the "hitormiss" method, so this argument will be ignored.
method	A string giving the method to use. See Details.

Details

The default method is Lantuéjoul's formula, a union across repeated erosions, which works in any number of dimensions and may produce reasonable results on greyscale images, but does not in general produce a connected skeleton. Beucher introduced an alternative which may produce a better result (although again the skeleton may not be connected), but this implementation of the latter algorithm only applies to binary arrays. The final method uses the so-called hit-or-miss transform, which searches for exact patterns in the source array. This is guaranteed to produce a connected skeleton, which is often desirable, but uses fixed kernels (so the kernel argument is ignored) and is currently only implemented for 2D binary arrays.

Value

A skeletonised array with the same dimensions as the original array.

Author(s)

Jon Clayden <code@clayden.org>

References

C. Lantuéjoul (1977). Sur le modèle de Johnson-Mehl généralisé. Technical report, Centre de Morphologie Mathématique, Fontainebleau, France.

S. Beucher (1994). Digital skeletons in Euclidean and geodesic spaces. Signal Processing 38(1):127-141. doi:10.1016/01651684(94)900612.

See Also

morphology

Examples

```
x <- c(0,0,1,0,0,0,1,1,1,0,0)
k <- c(1,1,1)
skeletonise(x,k)
```

---

sketch *Show an ASCII art representation of a 2D image or matrix*

Description

This function prints a rough, text-only representation of an image argument to the R terminal, mapping image intensities to a 10-level pseudo-greyscale. The image is first rescaled to fit into the terminal or other specified width, and downsampled in the row direction to correct for nonsquare character shapes.

Usage

```
sketch(x, invert = FALSE, width = getOption("width"), squash = 0.5)
```

Arguments

x	An object that can be coerced to a numeric matrix or array. 3D arrays with third dimension no greater than 4 will be taken as multichannel 2D images, and their channels averaged before display. Plain vectors and 1D arrays will be treated as single-row matrices.
invert	By default the mapping uses heavier type for brighter areas. If this option is TRUE, the sense of the scale will be reversed.
width	The width of sketch to draw, in characters.
squash	The factor by which to scale the row direction of the image. Generally this should be markedly less than one, to preserve the aspect ratio of the image, since most fixed-width font characters are taller than they are wide.

Details

The result is a compact representation of a matrix that can be used for visualising kernel arrays, sparse matrices and other non-images.

Value

This function is called for the side-effect of printing an ASCII representation of its argument.

Note

If the terminal does not used a fixed-width font, the result is unlikely to be useful.

Author(s)

Jon Clayden <code@clayden.org>

See Also

`display`

Examples

```
sketch(shapeKernel(c(9,15), type="diamond"))
sketch(shapeKernel(c(9,15), type="diamond"), squash=1)
```

---

symmetric *Check for a symmetric array*

Description

This function checks whether a numeric array is symmetric, in the sense of transposition. This is tested by comparing the reversed vectorised array to the unreversed equivalent.

Usage

```
symmetric(x)
```

Arguments

x	An object that can be coerced to a numeric array.

Value

A logical value indicating whether the array is symmetric or not.

Author(s)

Jon Clayden <code@clayden.org>

`threshold` *Threshold a numeric array or vector*

Description

This function thresholds an array or vector, setting elements below the threshold value to zero. The threshold can be given literally or calculated using k-means clustering.

Usage

```
threshold(x, level, method = c("literal", "kmeans"), binarise = TRUE)
```

Arguments

`x`	A numeric vector or array.
`level`	The literal threshold level, if required.
`method`	The method to use to calculate the threshold. If `"literal"` (the default) then the value of `level` will be used. If `"kmeans"` then the threshold value will be determined implicitly using k-means clustering.
`binarise`	Whether to set suprathreshold elements to unity (if `TRUE`), or leave them at their original values (if `FALSE`).

Author(s)

Jon Clayden <code@clayden.org>

See Also

`binarise`

Examples

```
x <- c(0.1, 0.05, 0.95, 0.85, 0.15, 0.9)
threshold(x, method="kmeans")
threshold(x, 0.5)
```

Index