Package 'interp'

November 27, 2023

Type Package

Title Interpolation Methods

Version 1.1-5

Date 2023-11-27

Maintainer Albrecht Gebhardt <albrecht.gebhardt@aau.at>

Description Bivariate data interpolation on regular and irregular grids, either linear or using splines are the main part of this package. It is intended to provide FOSS replacement functions for the ACM licensed akima::interp and tripack::tri.mesh functions. Linear interpolation is implemented in interp::interp(..., method=``linear''), this corresponds to the call akima::interp(..., linear=TRUE) which is the default setting and covers most of akima::interp use cases in depending packages. A re-implementation of Akimas irregular grid spline interpolation (akima::interp(..., linear=FALSE)) is now also available via interp::interp(..., method=``akima''). Estimators for partial derivatives are now also available in interp::locpoly(), these are a prerequisite for the spline interpolation. The basic part is a GPLed triangulation algorithm (sweep hull algorithm by David Sinclair) providing the starting point for the irregular grid interpolator. As side effect this algorithm is also used to provide replacements for almost all functions of the tripack package which also suffers from the same ACM license restrictions. All functions are designed to be backward compatible with their akima / tripack counterparts.

License GPL (>= 2)

Imports Rcpp (>= 0.12.9), deldir

Suggests sp, Deriv, Ryacas, ggplot2, gridExtra, lattice, stringi, stringr, scatterplot3d, MASS

Enhances RcppEigen

LinkingTo Rcpp, RcppEigen

Depends R (>= 3.5.0)

NeedsCompilation yes

Author Albrecht Gebhardt [aut, cre, cph],
Roger Bivand [aut],
David Sinclair [aut, cph] (author of the shull library)

Repository CRAN

Date/Publication 2023-11-27 19:20:07 UTC

R topics documented:

`interp-package` *Interpolation of data*

Description

Interpolation of z values given regular or irregular gridded data sets containing coordinates (x_i, y_i) and function values z_i is (will be) available through this package. As this interpolation is (for the irregular gridded data case) based on trianglation of the data locations also triangulation functions are implemented. Moreover the (not yet finished) spline interpolation needs estimators for partial derivates, these are also made available to the end user for direct use.

Details

The interpolation use can be divided by the used method into piecewise linear (finished in 1_0.27) and spline (not yet finished) interpolation and by input and output settings into gridded and pointwise setups.

Note

This package is a FOSS replacement for the ACM licensed packages `akima` and `tripack`. The function calls are backward compatible.

Author(s)

Albrecht Gebhardt <albrecht.gebhardt@aau.at>, Roger Bivand <roger.bivand@nhh.no>

Maintainer: Albrecht Gebhardt <albrecht.gebhardt@aau.at>

See Also

`interp`, `tri.mesh`, `voronoi.mosaic`, `locpoly`

akima *Waveform Distortion Data for Bivariate Interpolation*

Description

akima is a list with components x, y and z which represents a smooth surface of z values at selected points irregularly distributed in the x-y plane.

The data was taken from a study of waveform distortion in electronic circuits, described in: Hiroshi Akima, "A Method of Bivariate Interpolation and Smooth Surface Fitting Based on Local Procedures", CACM, Vol. 17, No. 1, January 1974, pp. 18-20.

References

Hiroshi Akima, "A Method of Bivariate Interpolation and Smooth Surface Fitting for Irregularly Distributed Data Points", ACM Transactions on Mathematical Software, Vol. 4, No. 2, June 1978, pp. 148-159. Copyright 1978, Association for Computing Machinery, Inc., reprinted by permission.

Examples

```
## Not run:
library(rgl)
data(akima)
# data
rgl.spheres(akima$x,akima$z , akima$y,0.5,color="red")
rgl.bbox()
# bivariate linear interpolation
# interp:
akima.li <- interp(akima$x, akima$y, akima$z,
                   xo=seq(min(akima$x), max(akima$x), length = 100),
                   yo=seq(min(akima$y), max(akima$y), length = 100))
# interp surface:
rgl.surface(akima.li$x,akima.li$y,akima.li$z,color="green",alpha=c(0.5))
# interpp:
akima.p <- interpp(akima$x, akima$y, akima$z,
                    runif(200,min(akima$x),max(akima$x)),
                    runif(200,min(akima$y),max(akima$y)))
# interpp points:
rgl.points(akima.p$x,akima.p$z , akima.p$y,size=4,color="yellow")

# bivariate spline interpolation
# data
rgl.spheres(akima$x,akima$z , akima$y,0.5,color="red")
rgl.bbox()
# bivariate cubic spline interpolation
# interp:
akima.si <- interp(akima$x, akima$y, akima$z,
                   xo=seq(min(akima$x), max(akima$x), length = 100),
                   yo=seq(min(akima$y), max(akima$y), length = 100),
                   linear = FALSE, extrap = TRUE)
```

```
# interp surface:
rgl.surface(akima.si$x,akima.si$y,akima.si$z,color="green",alpha=c(0.5))
# interpp:
akima.sp <- interpp(akima$x, akima$y, akima$z,
                    runif(200,min(akima$x),max(akima$x)),
                    runif(200,min(akima$y),max(akima$y)),
                   linear = FALSE, extrap = TRUE)
# interpp points:
rgl.points(akima.sp$x,akima.sp$z , akima.sp$y,size=4,color="yellow")


## End(Not run)
```

---

akima474 *Sample data from Akima's Bicubic Spline Interpolation code (TOMS 474)*

Description

akima474 is a list with vector components x, y and a matrix z which represents a smooth surface of z values at the points of a regular grid spanned by the vectors x and y.

References

Hiroshi Akima, Bivariate Interpolation and Smooth Surface Fitting Based on Local Procedures [E2], Communications of ACM, Vol. 17, No. 1, January 1974, pp. 26-30

Examples

```
## Not run:
library(rgl)
data(akima474)
# data
rgl.spheres(akima474$x,akima474$z , akima474$y,0.5,color="red")
rgl.bbox()
# bivariate linear interpolation
# interp:
akima474.li <- interp(akima474$x, akima474$y, akima474$z,
                   xo=seq(min(akima474$x), max(akima474$x), length = 100),
                   yo=seq(min(akima474$y), max(akima474$y), length = 100))
# interp surface:
rgl.surface(akima474.li$x,akima474.li$y,akima474.li$z,color="green",alpha=c(0.5))
# interpp:
akima474.p <- interpp(akima474$x, akima474$y, akima474$z,
                    runif(200,min(akima474$x),max(akima474$x)),
                    runif(200,min(akima474$y),max(akima474$y)))
# interpp points:
rgl.points(akima474.p$x,akima474.p$z , akima474.p$y,size=4,color="yellow")

# bivariate spline interpolation
```

```
# data
rgl.spheres(akima474$x,akima474$z , akima474$y,0.5,color="red")
rgl.bbox()
# bivariate cubic spline interpolation
# interp:
akima474.si <- interp(akima474$x, akima474$y, akima474$z,
                  xo=seq(min(akima474$x), max(akima474$x), length = 100),
                  yo=seq(min(akima474$y), max(akima474$y), length = 100),
                  linear = FALSE, extrap = TRUE)
# interp surface:
rgl.surface(akima474.si$x,akima474.si$y,akima474.si$z,color="green",alpha=c(0.5))
# interpp:
akima474.sp <- interpp(akima474$x, akima474$y, akima474$z,
                   runif(200,min(akima474$x),max(akima474$x)),
                   runif(200,min(akima474$y),max(akima474$y)),
                  linear = FALSE, extrap = TRUE)
# interpp points:
rgl.points(akima474.sp$x,akima474.sp$z , akima474.sp$y,size=4,color="yellow")


## End(Not run)
```

`arcs` *Extract a list of arcs from a triangulation object.*

Description

This function extracts a list of arcs from a triangulation object created by `tri.mesh`.

Usage

```
arcs(tri.obj)
```

Arguments

`tri.obj`	object of class `triSht`

Details

This function acesses the `arcs` component of a triangulation object returned by `tri.mesh` and extracts the arcs contained in this triangulation. This is e.g. used for plotting.

Value

A matrix with two columns `"from"` and `"to"` containing the indices of points connected by the arc with the corresponding row index.

Author(s)

Albrecht Gebhardt <albrecht.gebhardt@aau.at>, Roger Bivand <roger.bivand@nhh.no>

See Also

triSht, triangles, area

Examples

```
data(franke)
tr <- tri.mesh(franke$ds3)
arcs(tr)
```

---

area *Extract a list of triangle areas from a triangulation object.*

Description

This function returns a list containing the areas of each triangle of a triangulation object created by tri.mesh.

Usage

```
area(tri.obj)
```

Arguments

tri.obj	object of class triSht

Details

This function acesses the cclist component of a triangulation object returned by tri.mesh and extracts the areas of the triangles contained in this triangulation.

Value

A vector containing the area values.

Author(s)

Albrecht Gebhardt <albrecht.gebhardt@aau.at>, Roger Bivand <roger.bivand@nhh.no>

See Also

triSht, triangles, arcs

Examples

```
data(franke)
tr <- tri.mesh(franke$ds3)
area(tr)
```

aspline *Univariate Akima interpolation*

Description

The function returns a list of points which smoothly interpolate given data points, similar to a curve drawn by hand.

Usage

```
aspline(x, y = NULL, xout, n = 50, ties = mean, method = "improved",
degree = 3)
aSpline(x, y, xout, method = "improved", degree = 3)
```

Arguments

x, y	vectors giving the coordinates of the points to be interpolated. Alternatively a single plotting structure can be specified: see xy.coords.
xout	an optional set of values specifying where interpolation is to take place.
n	If xout is not specified, interpolation takes place at n equally spaced points spanning the interval [min(x), max(x)].
ties	Handling of tied x values. Either a function with a single vector argument returning a single number result or the string "ordered".
method	either "original" method after Akima (1970) or "improved" method (default) after Akima (1991)
degree	if improved algorithm is selected: degree of the polynomials for the interpolating function

Details

The original algorithm is based on a piecewise function composed of a set of polynomials, each of degree three, at most, and applicable to successive interval of the given points. In this method, the slope of the curve is determined at each given point locally by fitting a third degree polynomial to four consecutive points. Each polynomial representing a portion of the curve between a pair of given points is determined by the coordinates of and the slopes at the points. The data set is prolonged below and above minimum and maximum x values to enable estimation of derivatives at the boundary. The improved algorithm uses polynomials of degree two and one at the boundary. Additionally four overlapping sequences of points are used for the estimation via a residual based weighting scheme.

Value

x	x coordinates of the interpolated data as given by 'xout' or 'n'.
y	interpolated y values.

Note

'aspline' is a wrapper call for the underlying Rcpp function 'aSpline' which could also be called directly with 'x' and 'y' arguments if 'xout' is given and no 'ties' argument is needed.

This is a reimplementation of Akimas algorithms (original and improved version). It is only based on the original articles. It does not involve or resemble the Fortran code associated with those articles. For this reason results may differ slightly because different expressions can result in different numerical errors.

This code is under GPL in contrast to original Fortran code as provided in package 'akima'.

The function arguments are identical to the call in package 'akima', only the 'method' argument has its default now set to 'improved'.

Author(s)

Albrecht Gebhardt <albrecht.gebhardt@aau.at>, Thomas Petzold <thomas.petzoldt@tu-dresden.de>

References

Akima, H. (1970) A new method of interpolation and smooth curve fitting based on local procedures, J. ACM **17**(4), 589-602

Akima, H. (1991) A Method of Univariate Interpolation that Has the Accuracy of a Third-degree Polynomial. ACM Transactions on Mathematical Software, **17**(3), 341-366.

See Also

spline

Examples

```
## regular spaced data
x <- 1:10
y <- c(rnorm(5), c(1,1,1,1,3))

xnew <- seq(-1, 11, 0.1)
plot(x, y, ylim=c(-3, 3), xlim=range(xnew))
## stats::spline() for comparison
lines(spline(x, y, xmin=min(xnew), xmax=max(xnew), n=200), col="blue")

lines(aspline(x, y, xnew, method="original"), col="red")
lines(aspline(x, y, xnew, method="improved"), col="black", lty="dotted")
lines(aspline(x, y, xnew, method="improved", degree=10), col="green", lty="dashed")

## irregular spaced data
x <- sort(runif(10, max=10))
y <- c(rnorm(5), c(1,1,1,1,3))

xnew <- seq(-1, 11, 0.1)
plot(x, y, ylim=c(-3, 3), xlim=range(xnew))
## stats::spline() for comparison
lines(spline(x, y, xmin=min(xnew), xmax=max(xnew), n=200), col="blue")
```

```
lines(aspline(x, y, xnew, method="original"), col="red")
lines(aspline(x, y, xnew, method="improved"), col="black", lty="dotted")
lines(aspline(x, y, xnew, method="improved", degree=10), col="green", lty="dashed")

## an example of Akima, 1991
x <- c(-3, -2, -1, 0,  1,  2, 2.5, 3)
y <- c( 0,  0,  0, 0, -1, -1, 0,   2)

plot(x, y, ylim=c(-3, 3))
## stats::spline() for comparison
lines(spline(x, y, n=200), col="blue")

lines(aspline(x, y, n=200, method="original"), col="red")
lines(aspline(x, y, n=200, method="improved"), col="black", lty="dotted")
lines(aspline(x, y, n=200, method="improved", degree=10), col="green", lty="dashed")
```

`bicubic` *Bivariate Interpolation for Data on a Rectangular grid*

Description

This is a placeholder function for backward compatibility with packaga akima.

In its current state it simply calls the reimplemented Akima algorithm for irregular grids applied to the regular gridded data given.

Later a reimplementation of the original algorithm for regular grids may follow.

Usage

```
bicubic(x, y, z, x0, y0)
```

Arguments

`x`	a vector containing the `x` coordinates of the rectangular data grid.
`y`	a vector containing the `y` coordinates of the rectangular data grid.
`z`	a matrix containing the `z[i,j]` data values for the grid points (`x[i]`,`y[j]`).
`x0`	vector of `x` coordinates used to interpolate at.
`y0`	vector of `y` coordinates used to interpolate at.

Details

This function is a call wrapper for backward compatibility with package akima.

Currently it applies Akimas irregular grid splines to regular grids, later a FOSS reimplementation of his regular grid splines may replace this wrapper.

Value

This function produces a list of interpolated points:

x	vector of x coordinates.
y	vector of y coordinates.
z	vector of interpolated data z.

If you need an output grid, see bicubic.grid.

Note

Use interp for the general case of irregular gridded data!

References

Akima, H. (1996) Rectangular-Grid-Data Surface Fitting that Has the Accuracy of a Bicubic Polynomial, J. ACM **22**(3), 357-361

See Also

interp, bicubic.grid

Examples

```
data(akima474)
# interpolate at the diagonal of the grid [0,8]x[0,10]
akima.bic <- bicubic(akima474$x,akima474$y,akima474$z,
                     seq(0,8,length=50), seq(0,10,length=50))
plot(sqrt(akima.bic$x^2+akima.bic$y^2), akima.bic$z, type="l")
```

---

bicubic.grid *Bicubic Interpolation for Data on a Rectangular grid*

Description

This is a placeholder function for backward compatibility with packaga akima.

In its current state it simply calls the reimplemented Akima algorithm for irregular grids applied to the regular gridded data given.

Later a reimplementation of the original algorithm for regular grids may follow.

Usage

```
bicubic.grid(x,y,z,xlim=c(min(x),max(x)),ylim=c(min(y),max(y)),
             nx=40,ny=40,dx=NULL,dy=NULL)
```

Arguments

x	a vector containing the x coordinates of the rectangular data grid.
y	a vector containing the y coordinates of the rectangular data grid.
z	a matrix containing the z[i,j] data values for the grid points (x[i],y[j]).
xlim	vector of length 2 giving lower and upper limit for range x coordinates used for output grid.
ylim	vector of length 2 giving lower and upper limit for range of y coordinates used for output grid.
nx	output grid dimension in x direction.
ny	output grid dimension in y direction.
dx	output grid spacing in x direction, not used by default, overrides nx if specified.
dy	output grid spacing in y direction, not used by default, overrides ny if specified..

Details

This function is a call wrapper for backward compatibility with package akima.

Currently it applies Akimas irregular grid splines to regular grids, later a FOSS reimplementation of his regular grid splines may replace this wrapper.

Value

This function produces a grid of interpolated points, feasible to be used directly with `image` and `contour`:

x	vector of x coordinates of the output grid.
y	vector of y coordinates of the output grid.
z	matrix of interpolated data for the output grid.

Note

Use `interp` for the general case of irregular gridded data!

References

Akima, H. (1996) Rectangular-Grid-Data Surface Fitting that Has the Accuracy of a Bicubic Polynomial, J. ACM **22**(3), 357-361

See Also

`interp`, `bicubic`

Examples

```
data(akima474)
# interpolate at a grid [0,8]x[0,10]
akima.bic <- bicubic.grid(akima474$x,akima474$y,akima474$z)
zmin <- min(akima.bic$z, na.rm=TRUE)
zmax <- max(akima.bic$z, na.rm=TRUE)
breaks <- pretty(c(zmin,zmax),10)
colors <- heat.colors(length(breaks)-1)
image(akima.bic, breaks=breaks, col=colors)
contour(akima.bic, levels=breaks,  add=TRUE)
```

`bilinear` *Bilinear Interpolation for Data on a Rectangular grid*

Description

This is an implementation of a bilinear interpolating function.

For a point (x0,y0) contained in a rectangle (x1,y1),(x2,y1), (x2,y2),(x1,y2) and x1<x2, y1<y2, the first step is to get z() at locations (x0,y1) and (x0,y2) as convex linear combinations z(x0,y*)=a*z(x1,y*)+(1-a)*z(x2,y*) where a=(x2-x1)/(x0-x1) for y*=y1,y2. In a second step z(x0,y0) is calculated as convex linear combination between z(x0,y1) and z(x0,y2) as z(x0,y1)=b*z(x0,y1)+(1-b)*z(x0,y2) where b=(y2-y1)/(y0-y1).

Finally, z(x0,y0) is a convex linear combination of the z values at the corners of the containing rectangle with weights according to the distance from (x0,y0) to these corners.

The grid lines can be unevenly spaced.

Usage

```
bilinear(x, y, z, x0, y0)
BiLinear(x, y, z, x0, y0)
```

Arguments

`x`	a vector containing the `x` coordinates of the rectangular data grid.
`y`	a vector containing the `y` coordinates of the rectangular data grid.
`z`	a matrix containing the `z[i,j]` data values for the grid points (`x[i]`,`y[j]`).
`x0`	vector of `x` coordinates used to interpolate at.
`y0`	vector of `y` coordinates used to interpolate at.

Value

This function produces a list of interpolated points:

`x`	vector of `x` coordinates.
`y`	vector of `y` coordinates.
`z`	vector of interpolated data `z`.

If you need an output grid, see `bilinear.grid`.

Note

This Fortran function was part of the akima package but not related to any of Akimas algorithms and under GPL. So it could be transfered into the interp package without changes.

`BiLinear` is a C++ reimplementation, maybe it will replace the Fortran implementation later, so its name may change in futrure versions.

Note

Use `interpp` for the general case of irregular gridded data!

References

Pascal Getreuer, Linear Methods for Image Interpolation, Image Processing On Line, 2011, http://www.ipol.im/pub/art/2011/g

See Also

`interp`, `bilinear.grid`

Examples

```
data(akima474)
# interpolate at the diagonal of the grid [0,8]x[0,10]
akima.bil <- bilinear(akima474$x,akima474$y,akima474$z,
                      seq(0,8,length=50), seq(0,10,length=50))
plot(sqrt(akima.bil$x^2+akima.bil$y^2), akima.bil$z, type="l")
```

---

bilinear.grid *Bilinear Interpolation for Data on a Rectangular grid*

Description

This is an implementation of a bilinear interpolating function.

For a point (x0,y0) contained in a rectangle (x1,y1),(x2,y1), (x2,y2),(x1,y2) and x1<x2, y1<y2, the first step is to get z() at locations (x0,y1) and (x0,y2) as convex linear combinations z(x0,y*)=a*z(x1,y*)+(1-a)*z(x2,y*) where a=(x2-x1)/(x0-x1) for y*=y1,y2. In a second step z(x0,y0) is calculated as convex linear combination between z(x0,y1) and z(x0,y2) as z(x0,y1)=b*z(x0,y1)+(1-b)*z(x0,y2) where b=(y2-y1)/(y0-y1).

Finally, z(x0,y0) is a convex linear combination of the z values at the corners of the containing rectangle with weights according to the distance from (x0,y0) to these corners.

The grid lines can be unevenly spaced.

Usage

```
bilinear.grid(x,y,z,xlim=c(min(x),max(x)),ylim=c(min(y),max(y)),
                          nx=40,ny=40,dx=NULL,dy=NULL)
BiLinear.grid(x,y,z,xlim=c(min(x),max(x)),ylim=c(min(y),max(y)),
                          nx=40,ny=40,dx=NULL,dy=NULL)
```

Arguments

x	a vector containing the x coordinates of the rectangular data grid.
y	a vector containing the y coordinates of the rectangular data grid.
z	a matrix containing the z[i,j] data values for the grid points (x[i],y[j]).
xlim	vector of length 2 giving lower and upper limit for range x coordinates used for output grid.
ylim	vector of length 2 giving lower and upper limit for range of y coordinates used for output grid.
nx	output grid dimension in x direction.
ny	output grid dimension in y direction.
dx	output grid spacing in x direction, not used by default, overrides nx if specified.
dy	output grid spacing in y direction, not used by default, overrides ny if specified..

Value

This function produces a grid of interpolated points, feasible to be used directly with image and contour:

x	vector of x coordinates of the output grid.
y	vector of y coordinates of the output grid.
z	matrix of interpolated data for the output grid.

Note

This Fortran function was part of the akima package but not related to any of Akimas algorithms and under GPL. So it could be transfered into the interp package without changes.

BiLinear.grid is a C++ reimplementation, maybe this will replace the Fortran implementation later. So its name may change in future versions, dont rely on it currently.

References

Pascal Getreuer, Linear Methods for Image Interpolation, Image Processing On Line, 2011, http://www.ipol.im/pub/art/2011/g

See Also

interp

Examples

```
data(akima474)
# interpolate at a grid [0,8]x[0,10]
akima.bil <- bilinear.grid(akima474$x,akima474$y,akima474$z)
zmin <- min(akima.bil$z, na.rm=TRUE)
zmax <- max(akima.bil$z, na.rm=TRUE)
breaks <- pretty(c(zmin,zmax),10)
colors <- heat.colors(length(breaks)-1)
image(akima.bil, breaks=breaks, col=colors)
contour(akima.bil, levels=breaks, add=TRUE)
```

`cells` *extract info about voronoi cells*

Description

This function returns some info about the cells of a voronoi mosaic, including the coordinates of the vertices and the cell area.

Usage

```
cells(voronoi.obj)
```

Arguments

`voronoi.obj`	object of class `voronoi`

Details

The function calculates the neighbourhood relations between the underlying triangulation and translates it into the neighbourhood relations between the voronoi cells.

Value

retruns a list of lists, one entry for each voronoi cell which contains

`cell`	cell index
`center`	cell 'center'
`neighbours`	neighbour cell indices
`nodes`	2 times `nnb` matrix with vertice coordinates
`area`	cell area

Note

outer cells have `area=NA`, currently also `nodes=NA` which is not really useful – to be done later

Author(s)

A. Gebhardt

See Also

`voronoi.mosaic`, `voronoi.area`

Examples

```
data(tritest)
tritest.vm <- voronoi.mosaic(tritest$x,tritest$y)
tritest.cells <- cells(tritest.vm)
# higlight cell 12:
plot(tritest.vm)
polygon(t(tritest.cells[[12]]$nodes),col="green")
# put cell area into cell center:
text(tritest.cells[[12]]$center[1],
     tritest.cells[[12]]$center[2],
     tritest.cells[[12]]$area)
```

---

circles *plot circles*

Description

This function plots circles at given locations with given radii.

Usage

```
circles(x, y, r, ...)
```

Arguments

x	vector of x coordinates
y	vector of y coordinates
r	vactor of radii
...	additional graphic parameters will be passed through

Note

This function needs a previous plot where it adds the circles.

Author(s)

A. Gebhardt

See Also

lines, points

Examples

```
x<-rnorm(10)
y<-rnorm(10)
r<-runif(10,0,0.5)
plot(x,y, xlim=c(-3,3), ylim=c(-3,3), pch="+")
circles(x,y,r)
```

circtest *circtest / sample data*

Description

Sample data for the link{circumcircle} function.

circtest2 are points sampled from a circle with some jitter added, i.e. they represent the most complicated case for the link{circumcircle} function.

circum *Determine the circumcircle (and some other characteristics) of a triangle*

Description

This function returns the circumcircle of a triangle and some additonal values used to determine them.

Usage

```
circum(x, y)
```

Arguments

x	Vector of three elements, giving the x coordinatres of the triangle nodes.
y	Vector of three elements, giving the y coordinatres of the triangle nodes.

Details

This is an interface to the Fortran function CIRCUM found in TRIPACK.

Value

x	'x' coordinate of center
y	'y' coordinate of center
radius	circumcircle radius
signed.area	signed area of riangle (positive iff nodes are numbered counter clock wise)
aspect.ratio	ratio "radius of inscribed circle"/"radius of circumcircle", varies between 0 and 0.5 0 means collinear points, 0.5 equilateral trangle.

Note

This function is mainly intended to be used by circumcircle.

Author(s)

A. Gebhardt

References

https://math.fandom.com/wiki/Circumscribed_circle#Coordinates_of_circumcenter, visited march 2022.

See Also

circumcircle

Examples

```
circum(c(0,1,0),c(0,0,1))

tr <- list()
tr$t1 <-list(x=c(0,1,0),y=c(0,0,1))
tr$t2 <-list(x=c(0.5,0.9,0.7),y=c(0.2,0.9,1))
tr$t3 <-list(x=c(0.05,0,0.3),y=c(0.2,0.7,0.1))
plot(0,0,type="n",xlim=c(-0.5,1.5),ylim=c(-0.5,1.5))
for(i in 1:3){
    x <- tr[[i]]$x
    y <- tr[[i]]$y
    points(x,y,pch=c("1","2","3"),xlim=c(-0.5,1.5),ylim=c(-0.5,1.5))
    cc =circum(x,y)
    lines(c(x,x[1]),c(y,y[1]))
    points(cc$x,cc$y)
    if(cc$signed.area<0)
      circles(cc$x,cc$y,cc$radius,col="blue",lty="dotted")
    else
      circles(cc$x,cc$y,cc$radius,col="red",lty="dotted")
}
```

---

circumcircle *Determine the circumcircle of a set of points*

Description

This function returns the (smallest) circumcircle of a set of n points

Usage

```
circumcircle(x, y = NULL, num.touch=2, plot = FALSE, debug = FALSE)
```

Arguments

x	vector containing x coordinates of the data. If y is missing x should contain two elements $x and $y.
y	vector containing y coordinates of the data.
num.touch	How often should the resulting circle touch the convex hull of the given points? default: 2 possible values: 2 or 3 Note: The circumcircle of a triangle is usually defined to touch at 3 points, this function searches by default the minimum circle, which may be only touching at 2 points. Set parameter num.touch accordingly if you dont want the default behaviour!
plot	Logical, produce a simple plot of the result. default: FALSE
debug	Logical, more plots, only needed for debugging. default: FALSE

Details

This is a (naive implemented) algorithm which determines the smallest circumcircle of n points:

First step: Take the convex hull.

Second step: Determine two points on the convex hull with maximum distance for the diameter of the set.

Third step: Check if the circumcircle of these two points already contains all other points (of the convex hull and hence all other points).

If not or if 3 or more touching points are desired (num.touch=3), search a point with minimum enclosing circumcircle among the remaining points of the convex hull.

If such a point cannot be found (e.g. for data(circtest2)), search the remaining triangle combinations of points from the convex hull until an enclosing circle with minimum radius is found.

The last search uses an upper and lower bound for the desired miniumum radius:

Any enclosing rectangle and its circumcircle gives an upper bound (the axis-parallel rectangle is used).

Half the diameter of the set from step 1 is a lower bound.

Value

x	'x' coordinate of circumcircle center
y	'y' coordinate of circumcircle center
radius	radius of circumcircle

Author(s)

Albrecht Gebhardt

See Also

convex.hull

Examples

```
data(circtest)
# smallest circle:
circumcircle(circtest,num.touch=2,plot=TRUE)

# smallest circle with maximum touching points (3):
circumcircle(circtest,num.touch=3,plot=TRUE)

# some stress test for this function,
data(circtest2)
# circtest2 was generated by:
# 100 random points almost one a circle:
# alpha <- runif(100,0,2*pi)
# x <- cos(alpha)
# y <- sin(alpha)
# circtest2<-list(x=cos(alpha)+runif(100,0,0.1),
#                 y=sin(alpha)+runif(100,0,0.1))
#
circumcircle(circtest2,plot=TRUE)
```

---

convex.hull *Return the convex hull of a triangulation object*

Description

Given a triangulation `tri.obj` of n points in the plane, this subroutine returns two vectors containing the coordinates of the nodes on the boundary of the convex hull.

`ConvexHull` is an experimental C++ implementation of Grahams Scan without previous triangulation, should be much faster.

Usage

```
convex.hull(tri.obj, plot.it=FALSE, add=FALSE,...)
       ConvexHull(x,y)
```

Arguments

`tri.obj`	object of class `triSht`
`plot.it`	logical, if `TRUE` the convex hull of `tri.obj` will be plotted.
`add`	logical. if `TRUE` (and `plot.it=TRUE`), add to a current plot.
`...`	additional plot arguments
`x`	only for `ConvexHull()`: `x` coordinates for C++ call to `ConvexHull`
`y`	only for `ConvexHull()`: see `x`

Value

x	x coordinates of boundary nodes.
y	y coordinates of boundary nodes.

Author(s)

Albrecht Gebhardt <albrecht.gebhardt@aau.at>, Roger Bivand <roger.bivand@nhh.no>

See Also

triSht, print.triSht, plot.triSht, summary.triSht, triangles.

Examples

```
## random points:
rand.tr<-tri.mesh(runif(10),runif(10))
plot(rand.tr)
rand.ch<-convex.hull(rand.tr, plot.it=TRUE, add=TRUE, col="red")
## use a part of the quakes data set:
data(quakes)
quakes.part<-quakes[(quakes[,1]<=-17 & quakes[,1]>=-19.0 &
                     quakes[,2]<=182.0 & quakes[,2]>=180.0),]
quakes.tri<-tri.mesh(quakes.part$lon, quakes.part$lat, duplicate="remove")
plot(quakes.tri)
convex.hull(quakes.tri, plot.it=TRUE, add=TRUE, col="red")
```

---

franke.data *Test datasets from Franke for interpolation of scattered data*

Description

franke.data generates the test datasets from Franke, 1979, see references.

Usage

```
franke.data(fn = 1, ds = 1, data)
franke.fn(x, y, fn = 1)
```

Arguments

fn	function number, from 1 to 5.
x	'x' value
y	'y' value
ds	data set number, from 1 to 3. Dataset 1 consists of 100 points, dataset 2 of 33 points and dataset 3 of 25 points scattered in the square $[0,1] \times [0,1]$. (and partially slightly outside).
data	A list of dataframes with 'x' and 'y' to choose from, dataset franke should be used here.

Details

These datasets are mentioned in Akima, (1996) as a testbed for the irregular scattered data interpolator.

Franke used the five functions:

$$0.75e^{-\frac{(9x-2)^2+(9y-2)^2}{4}} + 0.75e^{-\frac{(9x+1)^2}{49}-\frac{9y+1}{10}} + 0.5e^{-\frac{(9x-7)^2+(9y-3)^2}{4}} - 0.2e^{-((9x-4)^2-(9y-7)^2)}$$

$$\frac{\tanh(9y-9x)+1}{9}$$

$$\frac{1.25+\cos(5.4y)}{6(1+(3x-1)^2)}$$

$$e^{-\frac{81((x-0.5)^2+\frac{(y-0.5)^2}{16})}{3}}$$

$$e^{-\frac{81((x-0.5)^2+\frac{(y-0.5)^2}{4})}{3}}$$

$$\frac{\sqrt{64-81((x-0.5)^2+(y-0.5)^2)}}{9} - 0.5$$

and evaluated them on different more or less dense grids over $[0,1] \times [0,1]$.

Value

A data frame with components

x	'x' coordinate
y	'y' coordinate
z	'z' value

Note

The datasets have to be generated via `franke.data` before use, the dataset `franke` only contains a list of 3 dataframes of 'x' and 'y' coordinates for the above mentioned irregular grids. Do not forget to load the `franke` dataset first.

The 'x' and 'y' values have been taken from Akima (1996).

Author(s)

Albrecht Gebhardt <albrecht.gebhardt@aau.at>, Roger Bivand <roger.bivand@nhh.no>

References

FRANKE, R., (1979). A critical comparison of some methods for interpolation of scattered data. Tech. Rep. NPS-53-79-003, Dept. of Mathematics, Naval Postgraduate School, Monterey, Calif.

Akima, H. (1996). Algorithm 761: scattered-data surface fitting that has the accuracy of a cubic polynomial. ACM Transactions on Mathematical Software **22**, 362–371.

See Also

interp

Examples

```
## generate Frankes data set for function 2 and dataset 3:
data(franke)
F23 <- franke.data(2,3,franke)
str(F23)
```

`identify.triSht` *Identify points in a triangulation plot*

Description

Identify points in a plot of "x" with its coordinates. The plot of "x" must be generated with `plot.tri`.

Usage

```
## S3 method for class 'triSht'
identify(x,...)
```

Arguments

x	object of class triSht
...	additional paramters for identify

Value

an integer vector containing the indexes of the identified points.

Author(s)

Albrecht Gebhardt <albrecht.gebhardt@aau.at>, Roger Bivand <roger.bivand@nhh.no>

See Also

triSht, print.triSht, plot.triSht, summary.triSht

Examples

```
## Not run:
data(franke)
tr <- tri.mesh(franke$ds3$x, franke$ds3$y)
plot(tr)
identify(tr)

## End(Not run)
```

---

interp *Interpolation function*

Description

This function currently implements piecewise linear interpolation (=barycentric interpolation).

Usage

```
interp(x, y = NULL, z, xo = seq(min(x), max(x), length = nx),
       yo = seq(min(y), max(y), length = ny),
       linear = (method == "linear"), extrap = FALSE,
       duplicate = "error", dupfun = NULL,
       nx = 40, ny = 40, input="points", output = "grid",
       method = "linear", deltri = "shull", h=0,
       kernel="gaussian", solver="QR", degree=3,
       baryweight=TRUE, autodegree=FALSE, adtol=0.1,
       smoothpde=FALSE, akimaweight=TRUE, nweight=25)
```

Arguments

x	vector of x-coordinates of data points or a SpatialPointsDataFrame object. Missing values are not accepted.
y	vector of y-coordinates of data points. Missing values are not accepted. If left as NULL indicates that x should be a SpatialPointsDataFrame and z names the variable of interest in this dataframe.
z	vector of z-values at data points or a character variable naming the variable of interest in the SpatialPointsDataFrame x. Missing values are not accepted. x, y, and z must be the same length (execpt if x is a SpatialPointsDataFrame) and may contain no fewer than four points. The points of x and y should not be collinear, i.e, they should not fall on the same line (two vectors x and y such that y = ax + b for some a, b will not produce menaningful results). interp is meant for cases in which you have x, y values scattered over a plane and a z value for each. If, instead, you are trying to evaluate a mathematical function, or get a graphical interpretation of relationships that can be described by a polynomial, try outer.

xo
: If `output="grid"` (default): sequence of x locations for rectangular output grid, defaults to `nx` points between `min(x)` and `max(x)`.
If `output="points"`: vector of x locations for output points.

yo
: If `output="grid"` (default): sequence of y locations for rectangular output grid, defaults to `ny` points between `min(y)` and `max(y)`.
If `output="points"`: vector of y locations for output points. In this case it has to be same length as `xo`.

input
: text, possible values are `"grid"` (not yet implemented) and `"points"` (default).
This is used to distinguish between regular and irregular gridded data.

output
: text, possible values are `"grid"` (=default) and `"points"`.
If `"grid"` is choosen then `xo` and `yo` are interpreted as vectors spanning a rectangular grid of points $(xo[i], yo[j])$, $i = 1, ..., nx$, $j = 1, ..., ny$. This default behaviour matches how `akima::interp` works.
In the case of `"points"` `xo` and `yo` have to be of same length and are taken as possibly irregular spaced output points $(xo[i], yo[i])$, $i = 1, ..., no$ with `no=length(xo)`. `nx` and `ny` are ignored in this case. This case is meant as replacement for the pointwise interpolation done by `akima::interpp`. If the input `x` is a `SpatialPointsDataFrame` and `output="points"` then `xo` has to be a `SpatialPointsDataFrame`, `yo` will be ignored.

linear
: logical, only for backward compatibility with `akima::interp`, indicates if piecewise linear interpolation or Akima splines should be used.
Please use the new `method` argument instead!

method
: text, possible methods are (currently only, more is under developement) `"linear"` (piecewise linear interpolation within the triangles of the Delaunay triangulation, also referred to as barycentric interpolation based on barycentric coordinates) and `"akima"` (a reimplementation for Akimas spline algorithms for irregular gridded data with the accuracy of a bicubic polynomial).
This replaces the old `linear` argument of `akima::interp`.

extrap
: logical, indicates if extrapolation outside the convex hull is intended, will not work for piecewise linear interpolation!

duplicate
: character string indicating how to handle duplicate data points. Possible values are
`"error"` produces an error message,
`"strip"` remove duplicate z values,
`"mean"`,`"median"`,`"user"` calculate mean , median or user defined function (`dupfun`) of duplicate z values.

dupfun
: a function, applied to duplicate points if `duplicate= "user"`.

nx
: dimension of output grid in x direction

ny
: dimension of output grid in y direction

deltri
: triangulation method used, this argument will later be moved into a control set together with others related to the spline interpolation! Possible values are `"shull"` (default, sweep hull algorithm) and `"deldir"` (uses package`deldir`).

h
: bandwidth for partial derivatives estimation, compare `locpoly` for details

kernel	kernel for partial derivatives estimation, compare locpoly for details
solver	solver used in partial derivatives estimation, compare locpoly for details
degree	degree of local polynomial used for partial derivatives estimation, compare locpoly for details
baryweight	calculate three partial derivatives estimatiors and weight them
autodegree	try to reduce degree automatically
adtol	used for autodegree
smoothpde	Use a an averaged version of partial derivatives estimates, by default simple average of nweight estimates. Currently disabled by default (FALSE), underlying code still a bit experimental.
akimaweight	apply Akima weighting scheme on partial derivatives estimations instead of simply averaging
nweight	size of search neighbourhood for weighting scheme, default: 25

Value

a list with 3 components:

x,y	If output="grid": vectors of x- and y-coordinates of output grid, the same as the input argument xo, or yo, if present. Otherwise, their default, a vector 40 points evenly spaced over the range of the input x and y. If output="points": vectors of x- and y-coordinates of output points as given by xo and yo.
z	If output="grid": matrix of fitted z-values. The value z[i,j] is computed at the point $(xo[i], yo[j])$. z has dimensions length(xo) times length(yo). If output="points": a vector with the calculated z values for the output points as given by xo and yo. If the input was a SpatialPointsDataFrame a SpatialPixelsDataFrame is returned for output="grid" and a SpatialPointsDataFrame for output="points".

Note

Please note that this function tries to be a replacement for the interp() function from the akima package. So it should be call compatible for most applications. It also offers additional tuning parameters, usually the default settings will fit. Please be aware that these additional parameters may change in the future as they are still under development.

Author(s)

Albrecht Gebhardt <albrecht.gebhardt@aau.at>, Roger Bivand <roger.bivand@nhh.no>

References

Moebius, A. F. (1827) Der barymetrische Calcul. Verlag v. Johann Ambrosius Barth, Leipzig, https://books.google.at/books?id=eFPluv_UqFEC&hl=de&pg=PR1#v=onepage&q&f=false

Franke, R., (1979). A critical comparison of some methods for interpolation of scattered data. Tech. Rep. NPS-53-79-003, Dept. of Mathematics, Naval Postgraduate School, Monterey, Calif.

Akima, H. (1978). A Method of Bivariate Interpolation and Smooth Surface Fitting for Irregularly Distributed Data Points. ACM Transactions on Mathematical Software **4**, 148-164.

Akima, H. (1996). Algorithm 761: scattered-data surface fitting that has the accuracy of a cubic polynomial. ACM Transactions on Mathematical Software **22**, 362–371.

See Also

interpp

Examples

```
### Use all datasets from Franke, 1979:
data(franke)
## x-y irregular grid points:
oldseed <- set.seed(42)
ni <- 64
xi <- runif(ni,0,1)
yi <- runif(ni,0,1)
xyi <- cbind(xi,yi)
## linear interpolation
fi <- franke.fn(xi,yi,1)
IL <- interp(xi,yi,fi,nx=80,ny=80,method="linear")
## prepare breaks and colors that match for image and contour:
breaks <- pretty(seq(min(IL$z,na.rm=TRUE),max(IL$z,na.rm=TRUE),length=11))
db <- breaks[2]-breaks[1]
nb <- length(breaks)
breaks <- c(breaks[1]-db,breaks,breaks[nb]+db)
colors <- terrain.colors(length(breaks)-1)
image(IL,breaks=breaks,col=colors,main="Franke function 1",
      sub=paste("linear interpolation, ", ni,"points"))
contour(IL,add=TRUE,levels=breaks)
points(xi,yi)
## spline interpolation
fi <- franke.fn(xi,yi,1)
IS <- interp(xi,yi,fi,method="akima",
             kernel="gaussian",solver="QR")
## prepare breaks and colors that match for image and contour:
breaks <- pretty(seq(min(IS$z,na.rm=TRUE),max(IS$z,na.rm=TRUE),length=11))
db <- breaks[2]-breaks[1]
nb <- length(breaks)
breaks <- c(breaks[1]-db,breaks,breaks[nb]+db)
colors <- terrain.colors(length(breaks)-1)
image(IS,breaks=breaks,col=colors,main="Franke function 1",
      sub=paste("spline interpolation, ", ni,"points"))
contour(IS,add=TRUE,levels=breaks)
        points(xi,yi)
## regular grid:
nx <- 8; ny <- 8
xg<-seq(0,1,length=nx)
yg<-seq(0,1,length=ny)
```

```
xx <- t(matrix(rep(xg,ny),nx,ny))
yy <- matrix(rep(yg,nx),ny,nx)
xyg<-expand.grid(xg,yg)
## linear interpolation
fg <- outer(xg,yg,function(x,y)franke.fn(x,y,1))
IL <- interp(xg,yg,fg,input="grid",method="linear")
## prepare breaks and colors that match for image and contour:
breaks <- pretty(seq(min(IL$z,na.rm=TRUE),max(IL$z,na.rm=TRUE),length=11))
db <- breaks[2]-breaks[1]
nb <- length(breaks)
breaks <- c(breaks[1]-db,breaks,breaks[nb]+db)
colors <- terrain.colors(length(breaks)-1)
image(IL,breaks=breaks,col=colors,main="Franke function 1",
      sub=paste("linear interpolation, ", nx,"x",ny,"points"))
contour(IL,add=TRUE,levels=breaks)
points(xx,yy)
## spline interpolation
fg <- outer(xg,yg,function(x,y)franke.fn(x,y,1))
IS <- interp(xg,yg,fg,input="grid",method="akima",
             kernel="gaussian",solver="QR")
## prepare breaks and colors that match for image and contour:
breaks <- pretty(seq(min(IS$z,na.rm=TRUE),max(IS$z,na.rm=TRUE),length=11))
db <- breaks[2]-breaks[1]
nb <- length(breaks)
breaks <- c(breaks[1]-db,breaks,breaks[nb]+db)
colors <- terrain.colors(length(breaks)-1)
image(IS,breaks=breaks,col=colors,main="Franke function 1",
      sub=paste("spline interpolation, ", nx,"x",ny,"points"))
contour(IS,add=TRUE,levels=breaks)
        points(xx,yy)
set.seed(oldseed)
```

`interp2xyz` *From interp() Result, Produce 3-column Matrix*

Description

From an `interp()` result, produce a 3-column matrix or `data.frame` `cbind(x, y, z)`.

Usage

```
interp2xyz(al, data.frame = FALSE)
```

Arguments

`al`	a `list` as produced from `interp()`.
`data.frame`	logical indicating if result should be data.frame or matrix (default).

Value

a matrix (or data.frame) with three columns, called "x", "y", "z".

Author(s)

Martin Maechler, Jan.18, 2013

See Also

expand.grid() is the "essential ingredient" of interp2xyz().

interp.

Examples

```
data(akima)
ak.spl <- with(akima, interp(x, y, z, method = "akima"))
str(ak.spl)# list (x[i], y[j], z = <matrix>[i,j])

## Now transform to simple  (x,y,z)  matrix / data.frame :
str(am <- interp2xyz(ak.spl))
str(ad <- interp2xyz(ak.spl, data.frame=TRUE))
## and they are the same:
stopifnot( am == ad | (is.na(am) & is.na(ad)) )
```

---

interpp *Pointwise interpolate irregular gridded data*

Description

This function implements bivariate interpolation onto a set of points for irregularly spaced input data.

This function is meant for backward compatibility to package akima, please use interp with its output argument set to "points" now. Especially newer options to the underlying algorithm are only available there.

Usage

```
interpp(x, y = NULL, z, xo, yo = NULL, linear = TRUE,
  extrap = FALSE, duplicate = "error", dupfun = NULL,
  deltri = "shull")
```

Arguments

x	vector of x-coordinates of data points or a SpatialPointsDataFrame object. Missing values are not accepted.

y	vector of y-coordinates of data points. Missing values are not accepted. If left as NULL indicates that x should be a SpatialPointsDataFrame and z names the variable of interest in this dataframe.
z	vector of z-coordinates of data points or a character variable naming the variable of interest in the SpatialPointsDataFrame x. Missing values are not accepted. x, y, and z must be the same length (execpt if x is a SpatialPointsDataFrame) and may contain no fewer than four points. The points of x and y cannot be collinear, i.e, they cannot fall on the same line (two vectors x and y such that y = ax + b for some a, b will not be accepted).
xo	vector of x-coordinates of points at which to evaluate the interpolating function. If x is a SpatialPointsDataFrame this has also to be a SpatialPointsDataFrame.
yo	vector of y-coordinates of points at which to evaluate the interpolating function. If operating on SpatialPointsDataFrames this is left as NULL
linear	logical – indicating wether linear or spline interpolation should be used.
extrap	logical flag: should extrapolation be used outside of the convex hull determined by the data points? Not possible for linear interpolation.
duplicate	indicates how to handle duplicate data points. Possible values are "error" - produces an error message, "strip" - remove duplicate z values, "mean","median","user" - calculate mean , median or user defined function of duplicate z values.
dupfun	this function is applied to duplicate points if duplicate="user"
deltri	triangulation method used, this argument will later be moved into a control set together with others related to the spline interpolation!

Value

a list with 3 components:

x,y	If output="grid": vectors of x- and y-coordinates of output grid, the same as the input argument xo, or yo, if present. Otherwise, their default, a vector 40 points evenly spaced over the range of the input x and y. If output="points": vectors of x- and y-coordinates of output points as given by xo and yo.
z	If output="grid": matrix of fitted z-values. The value z[i,j] is computed at the point $(xo[i], yo[j])$. z has dimensions length(xo) times length(yo). If output="points": a vector with the calculated z values for the output points as given by xo and yo. If the input was a SpatialPointsDataFrame a SpatialPixelssDataFrame is returned for output="grid" and a SpatialPointsDataFrame for output="points".

Note

This is only a call wrapper meant for backward compatibility, see interp for more details!

Author(s)

Albrecht Gebhardt <albrecht.gebhardt@aau.at>, Roger Bivand <roger.bivand@nhh.no>

References

Moebius, A. F. (1827) Der barymetrische Calcul. Verlag v. Johann Ambrosius Barth, Leipzig, https://books.google.at/books?id=eFPluv_UqFEC&hl=de&pg=PR1#v=onepage&q&f=false

Franke, R., (1979). A critical comparison of some methods for interpolation of scattered data. Tech. Rep. NPS-53-79-003, Dept. of Mathematics, Naval Postgraduate School, Monterey, Calif.

See Also

interp

Examples

```
### Use all datasets from Franke, 1979:
### calculate z at shifted original locations.
data(franke)
for(i in 1:5)
    for(j in 1:3){
        FR <- franke.data(i,j,franke)
        IL <- with(FR, interpp(x,y,z,x+0.1,y+0.1,linear=TRUE))
        str(IL)
    }
```

`locpoly` *Local polynomial fit.*

Description

This function performs a local polynomial fit of up to order 3 to bivariate data. It returns estimated values of the regression function as well as estimated partial derivatives up to order 3. This access to the partial derivatives was the main intent for writing this code as there already many other local polynomial regression implementations in R.

Usage

```
locpoly(x, y, z, xo = seq(min(x), max(x), length = nx), yo = seq(min(y),
 max(y), length = ny), nx = 40, ny = 40, input = "points", output = "grid",
 h = 0, kernel = "gaussian", solver = "QR", degree = 3, pd = "")
```

Arguments

x	vector of x-coordinates of data points. Missing values are not accepted.
y	vector of y-coordinates of data points. Missing values are not accepted.
z	vector of z-values at data points. Missing values are not accepted. x, y, and z must be the same length

xo — If output="grid" (default): sequence of x locations for rectangular output grid, defaults to nx points between min(x) and max(x).
If output="points": vector of x locations for output points.

yo — If output="grid" (default): sequence of y locations for rectangular output grid, defaults to ny points between min(y) and max(y).
If output="points": vector of y locations for output points. In this case it has to be same length as xo.

input — text, possible values are "grid" (not yet implemented) and "points" (default).
This is used to distinguish between regular and irregular gridded data.

output — text, possible values are "grid" (=default) and "points".
If "grid" is choosen then xo and yo are interpreted as vectors spanning a rectangular grid of points $(xo[i], yo[j])$, $i = 1, ..., nx$, $j = 1, ..., ny$. This default behaviour matches how akima::interp works.
In the case of "points" xo and yo have to be of same lenght and are taken as possibly irregular spaced output points $(xo[i], yo[i])$, $i = 1, ..., no$ with no=length(xo). nx and ny are ignored in this case.

nx — dimension of output grid in x direction

ny — dimension of output grid in y direction

h — bandwidth parameter, between 0 and 1. If a scalar is given it is interpreted as ratio applied to the dataset size to determine a local search neighbourhood, if set to 0 a minimum useful search neighbourhood is choosen (e.g. 10 points for a cubic trend function to determine all 10 parameters).
If a vector of length 2 is given both components are interpreted as ratio of the x- and y-range and taken as global bandwidth.

kernel — Text value, implemented kernels are uniform, triangle, epanechnikov, biweight, tricube, triweight, cosine and gaussian (default).

solver — Text value, determines used solver in fastLM algorithm used by this code
Possible values are LLt, QR (default), SVD, Eigen and CPivQR (compare fastLm).

degree — Integer value, degree of polynomial trend, maximum allowed value is 3.

pd — Text value, determines which partial derivative should be returned, possible values are "" (default, the polynomial itself), "x", "y", "xx", "xy", "yy", "xxx", "xxy", "xyy", "yyy" or "all".

Value

If pd="all":

x — x coordinates

y — y coordinates

z — estimates of z

zx — estimates of dz/dx

zy — estimates of dz/dy

zxx — estimates of d^2z/dx^2

zxy	estimates of $d^2z/dxdy$
zyy	estimates of d^2z/dy^2
zxxx	estimates of d^3z/dx^3
zxxy	estimates of d^3z/dx^2dy
zxyy	estimates of $d^3z/dxdy^2$
zyyy	estimates of d^3z/dy^3

If pd!="all" only the elements x, y and the desired derivative will be returned, e.g. zxy for pd="xy".

Note

Function locpoly of package KernSmooth performs a similar task for univariate data.

Author(s)

Albrecht Gebhardt <albrecht.gebhardt@aau.at>, Roger Bivand <roger.bivand@nhh.no>

References

Douglas Bates, Dirk Eddelbuettel (2013). Fast and Elegant Numerical Linear Algebra Using the RcppEigen Package. Journal of Statistical Software, 52(5), 1-24. URL http://www.jstatsoft.org/v52/i05/.

See Also

locpoly, fastLm

Examples

```
## choose a kernel
knl <- "gaussian"

## choose global and local bandwidth
bwg <- 0.25 # *100% means: percentage of x- y-range used
bwl <- 0.1  # *100% means: percentage of data set (nearest neighbours) used

## a bivariate polynomial of degree 5:
f <- function(x,y) 0.1+ 0.2*x-0.3*y+0.1*x*y+0.3*x^2*y-0.5*y^2*x+y^3*x^2+0.1*y^5

## degree of model
dg=3

## part 1:
## regular gridded data:
ng<- 11 # x/y size of a square data grid

## build and fill the grid with the theoretical values:

xg<-seq(0,1,length=ng)
```

```
yg<-seq(0,1,length=ng)

# xg and yg as matrix matching fg
nx <- length(xg)
ny <- length(yg)
xx <- t(matrix(rep(xg,ny),nx,ny))
yy <- matrix(rep(yg,nx),ny,nx)

fg   <- outer(xg,yg,f)

## local polynomial estimate
## global bw:
ttg <- system.time(pdg <- locpoly(xg,yg,fg,
  input="grid", pd="all", h=c(bwg,bwg), solver="QR", degree=dg, kernel=knl))
## time used:
ttg

## local bw:
ttl <- system.time(pdl <- locpoly(xg,yg,fg,
  input="grid", pd="all", h=bwl, solver="QR", degree=dg, kernel=knl))
## time used:
ttl

image(pdl$x,pdl$y,pdl$z,main="f and its estimated first partial derivatives",
      sub="colors: f, dotted: df/dx, dashed: df/dy")
contour(pdl$x,pdl$y,pdl$zx,add=TRUE,lty="dotted")
contour(pdl$x,pdl$y,pdl$zy,add=TRUE,lty="dashed")
points(xx,yy,pch=".")


## part 2:
## irregular data,
## results will not be as good as with the regular 21*21=231 points.

nd<- 121 # size of data set

## random irregular data
oldseed <- set.seed(42)
x<-runif(ng)
y<-runif(ng)
set.seed(oldseed)

z <- f(x,y)

## global bw:
ttg <- system.time(pdg <- interp::locpoly(x,y,z, xg,yg, pd="all",
  h=c(bwg,bwg), solver="QR", degree=dg,kernel=knl))

ttg

## local bw:
ttl <- system.time(pdl <- interp::locpoly(x,y,z, xg,yg, pd="all",
  h=bwl, solver="QR", degree=dg,kernel=knl))
```

```
ttl

image(pdl$x,pdl$y,pdl$z,main="f and its estimated first partial derivatives",
      sub="colors: f, dotted: df/dx, dashed: df/dy")
contour(pdl$x,pdl$y,pdl$zx,add=TRUE,lty="dotted")
contour(pdl$x,pdl$y,pdl$zy,add=TRUE,lty="dashed")
points(x,y,pch=".")
```

`nearest.neighbours` *Nearest neighbour structure for a data set*

Description

This function can be used to generate nearest neighbour information for a set of 2D data points.

Usage

```
nearest.neighbours(x, y)
```

Arguments

`x`	vector containing x ccordinates of points.
`y`	vector containing x ccordinates of points.

Details

The C++ implementation of this function is used inside the `locpoly` and `interp` functions.

Value

A list with two components

`index`	A matrix with one row per data point. Each row contains the indices of the nearest neigbours to the point associated with this row, currently the point itself is also listed in the first row, so this matrix is of dimension n times n (will change to n times $n-1$ later).
`dist`	A matrix containing the distances according to the neigbours listed in component `index`.

Author(s)

Albrecht Gebhardt <albrecht.gebhardt@aau.at>, Roger Bivand <roger.bivand@nhh.no>

See Also

`convex.hull`

Examples

```
data(franke)
## use only a small subset
fd <- franke$ds1[1:5,]
nearest.neighbours(fd$x,fd$y)
```

---

neighbours *List of neighbours from a triangulation or voronoi object*

Description

Extract a list of neighbours from a triangulation or voronoi object

Usage

```
neighbours(obj)
```

Arguments

obj	object of class "triSht" or "voronoi.mosaic"

Value

nested list of neighbours per point

Author(s)

A. Gebhardt

See Also

triSht, print.triSht, plot.triSht, summary.triSht, triangles

Examples

```
data(tritest)
tritest.tr<-tri.mesh(tritest$x,tritest$y)
tritest.nb<-neighbours(tritest.tr)
```

on *Determines if a point is on or left of the vector described by two other points.*

Description

A simple test function to determine the position of one (or more) points relative to a vector spanned by two points.

Usage

```
on(x1, y1, x2, y2, x0, y0, eps = 1e-16)
left(x1, y1, x2, y2, x0, y0, eps = 1e-16)
```

Arguments

x1	x coordinate of first point determinig the vector.
y1	y coordinate of first point determinig the vector.
x2	x coordinate of second point determinig the vector.
y2	y coordinate of second point determinig the vector.
x0	vector of x coordinates to locate relative to the vector $(x_2 - x_1, y_2 - y_1)$.
y0	vector of x coordinates to locate relative to the vector $(x_2 - x_1, y_2 - y_1)$.
eps	tolerance for checking if x_0, y_0 is on or left of $(x_2 - x_1, y_2 - y_1)$, defaults to 10^{-16}.

Value

logical vector with the results of the test.

Author(s)

Albrecht Gebhardt <albrecht.gebhardt@aau.at>, Roger Bivand <roger.bivand@nhh.no>

See Also

in.convex.hull, on.convex.hull.

Examples

```
y <- x <- c(0,1)
## should be TRUE
on(x[1],y[1],x[2],y[2],0.5,0.5)
## note the default setting of eps leading to
on(x[1],y[1],x[2],y[2],0.5,0.50000000000000001)
## also be TRUE

## should be TRUE
```

```
left(x[1],y[1],x[2],y[2],0.5,0.6)
## note the default setting of eps leading to
left(x[1],y[1],x[2],y[2],0.5,0.50000000000000001)
## already resulting to FALSE
```

`on.convex.hull` *Determines if points are on or in the convex hull of a triangulation object*

Description

Given a triangulation object `tri.obj` of n points in the plane, this subroutine returns a logical vector indicating if the points (x_i, y_i) lay on or in the convex hull of `tri.obj`.

Usage

```
on.convex.hull(tri.obj, x, y, eps=1E-16)
in.convex.hull(tri.obj, x, y, eps=1E-16, strict=TRUE)
```

Arguments

`tri.obj`	object of class `triSht`
`x`	vector of x-coordinates of points to locate
`y`	vector of y-coordinates of points to locate
`eps`	accuracy for checking the condition
`strict`	logical, default `TRUE`. It indicates if the convex hull is treated as an open (`strict=TRUE`) or closed (`strict=FALSE`) set. (applies only to `in.convex.hull`)

Value

Logical vector.

Author(s)

Albrecht Gebhardt <albrecht.gebhardt@aau.at>, Roger Bivand <roger.bivand@nhh.no>

See Also

`triSht`, `print.triSht`, `plot.triSht`, `summary.triSht`, `triangles`, `convex.hull`.

Examples

```
# use a part of the quakes data set:
data(quakes)
quakes.part<-quakes[(quakes[,1]<=-10.78 & quakes[,1]>=-19.4 &
                     quakes[,2]<=182.29 & quakes[,2]>=165.77),]
q.tri<-tri.mesh(quakes.part$lon, quakes.part$lat, duplicate="remove")
on.convex.hull(q.tri,quakes.part$lon[1:20],quakes.part$lat[1:20])
# Check with part of data set:
# Note that points on the hull (see above) get marked FALSE below:
in.convex.hull(q.tri,quakes.part$lon[1:20],quakes.part$lat[1:20])
# If points both on the hull and in the interior of the hull are meant
# disable strict mode:
in.convex.hull(q.tri,quakes.part$lon[1:20],quakes.part$lat[1:20],strict=FALSE)
# something completely outside:
in.convex.hull(q.tri,c(170,180),c(-20,-10))
```

`outer.convhull` *Version of outer which operates only in a convex hull*

Description

This version of `outer` evaluates `FUN` only on that part of the grid cx times cy that is enclosed within the convex hull of the points (px, py).

This can be useful for spatial estimation if no extrapolation is wanted.

Usage

```
outer.convhull(cx,cy,px,py,FUN,duplicate="remove",...)
```

Arguments

`cx`	x cordinates of grid
`cy`	y cordinates of grid
`px`	vector of x coordinates of points
`py`	vector of y coordinates of points
`FUN`	function to be evaluated over the grid
`duplicate`	indicates what to do with duplicate (px_i, py_i) points, default `"remove"`.
`...`	additional arguments for `FUN`

Value

Matrix with values of `FUN` (`NA`s if outside the convex hull).

Author(s)

Albrecht Gebhardt <albrecht.gebhardt@aau.at>, Roger Bivand <roger.bivand@nhh.no>

See Also

in.convex.hull

Examples

```
x<-runif(20)
y<-runif(20)
z<-runif(20)
z.lm<-lm(z~x+y)
f.pred<-function(x,y)
  {predict(z.lm,data.frame(x=as.vector(x),y=as.vector(y)))}
xg<-seq(0,1,0.05)
yg<-seq(0,1,0.05)
image(xg,yg,outer.convhull(xg,yg,x,y,f.pred))
points(x,y)
```

---

plot.triSht *Plot a triangulation object*

Description

plots the triangulation object "x"

Usage

```
## S3 method for class 'triSht'
plot(x, add = FALSE, xlim = range(x$x),
  ylim = range(x$y), do.points = TRUE, do.labels = FALSE, isometric = TRUE,
  do.circumcircles = FALSE, segment.lty = "dashed", circle.lty =
  "dotted", ...)
```

Arguments

x	object of class "triSht"
add	logical, if TRUE, add to a current plot.
do.points	logical, indicates if points should be plotted. (default TRUE)
do.labels	logical, indicates if points should be labelled. (default FALSE)
xlim,ylim	x/y ranges for plot
isometric	generate an isometric plot (default TRUE)
do.circumcircles	logical, indicates if circumcircles should be plotted (default FALSE)
segment.lty	line type for triangulation segments
circle.lty	line type for circumcircles
...	additional plot parameters

Value

None

Author(s)

Albrecht Gebhardt <albrecht.gebhardt@aau.at>, Roger Bivand <roger.bivand@nhh.no>

See Also

triSht, print.triSht, summary.triSht

Examples

```
## random points
plot(tri.mesh(rpois(100,lambda=20),rpois(100,lambda=20),duplicate="remove"))
## use a part of the quakes data set:
data(quakes)
quakes.part<-quakes[(quakes[,1]<=-10.78 & quakes[,1]>=-19.4 &
                     quakes[,2]<=182.29 & quakes[,2]>=165.77),]
quakes.tri<-tri.mesh(quakes.part$lon, quakes.part$lat, duplicate="remove")
plot(quakes.tri)
## use the whole quakes data set
## (will not work with standard memory settings, hence commented out)
## plot(tri.mesh(quakes$lon, quakes$lat, duplicate="remove"), do.points=F)
```

---

plot.voronoi *Plot a voronoi object*

Description

Plots the mosaic "x". Dashed lines are used for outer tiles of the mosaic.

Usage

```
## S3 method for class 'voronoi'
plot(x,add=FALSE,
                            xlim=c(min(x$tri$x)-
                              0.1*diff(range(x$tri$x)),
                              max(x$tri$x)+
                              0.1*diff(range(x$tri$x))),
                            ylim=c(min(x$tri$y)-
                              0.1*diff(range(x$tri$y)),
                              max(x$tri$y)+
                              0.1*diff(range(x$tri$y))),
                            all=FALSE,
                            do.points=TRUE,
                            main="Voronoi mosaic",
                            sub=deparse(substitute(x)),
```

```
                    isometric=TRUE,
                    ...)
```

Arguments

x	object of class "voronoi"
add	logical, if TRUE, add to a current plot.
xlim	x plot ranges, by default modified to hide dummy points outside of the plot
ylim	y plot ranges, by default modified to hide dummy points outside of the plot
all	show all (including dummy points in the plot
do.points	logical, indicates if points should be plotted.
main	plot title
sub	plot subtitle
isometric	generate an isometric plot (default TRUE)
...	additional plot parameters

Value

None

Author(s)

Albrecht Gebhardt <albrecht.gebhardt@aau.at>, Roger Bivand <roger.bivand@nhh.no>

See Also

voronoi, print.voronoi, summary.voronoi, plot.voronoi.polygons

Examples

```
data(franke)
tr <- tri.mesh(franke$ds3)
vr <- voronoi.mosaic(tr)
plot(tr)
plot(vr,add=TRUE)
```

---

plot.voronoi.polygons *plots an voronoi.polygons object*

Description

plots an voronoi.polygons object

Usage

```
## S3 method for class 'voronoi.polygons'
plot(x, which, color=TRUE, isometric=TRUE, ...)
```

Arguments

x	object of class voronoi.polygons
which	index vector selecting which polygons to plot
color	logical, determines if plot should be colored, default: TRUE
isometric	generate an isometric plot (default TRUE)
...	additional plot arguments

Author(s)

A. Gebhardt

See Also

voronoi.polygons

Examples

```
data(franke)
fd3 <- franke$ds3
fd3.vm <- voronoi.mosaic(fd3$x,fd3$y)
fd3.vp <- voronoi.polygons(fd3.vm)
plot(fd3.vp)
plot(fd3.vp,which=c(3,4,6,10))
```

---

print.summary.triSht *Print a summary of a triangulation object*

Description

Prints some information about tri.obj

Usage

```
## S3 method for class 'summary.triSht'
print(x, ...)
```

Arguments

x	object of class "summary.triSht", generated by summary.triSht.
...	additional paramters for print

Value

None

Note

This function is meant as replacement for the function of same name in package `tripack`.

The only difference is that no constraints are possible with `triSht` objects of package `interp`.

Author(s)

Albrecht Gebhardt <albrecht.gebhardt@aau.at>, Roger Bivand <roger.bivand@nhh.no>

See Also

`triSht`,`tri.mesh`, `print.triSht`, `plot.triSht`, `summary.triSht`.

`print.summary.voronoi` *Print a summary of a voronoi object*

Description

Prints some information about object `x`

Usage

```
## S3 method for class 'summary.voronoi'
print(x, ...)
```

Arguments

`x`	object of class `"summary.voronoi"`, generated by `summary.voronoi`.
`...`	additional paramters for `print`

Value

None

Note

This function is meant as replacement for the function of same name in package `tripack` and should be fully backward compatible.

Author(s)

Albrecht Gebhardt <albrecht.gebhardt@aau.at>, Roger Bivand <roger.bivand@nhh.no>

See Also

`voronoi`,`voronoi.mosaic`, `print.voronoi`, `plot.voronoi`, `summary.voronoi`.

print.triSht *Print a triangulation object*

Description

prints a adjacency list of "x"

Usage

```
## S3 method for class 'triSht'
print(x,...)
```

Arguments

x	object of class "triSht"
...	additional paramters for print

Value

None

Author(s)

Albrecht Gebhardt <albrecht.gebhardt@aau.at>, Roger Bivand <roger.bivand@nhh.no>

See Also

triSht, plot.triSht, summary.triSht

print.voronoi *Print a voronoi object*

Description

prints a summary of "x"

Usage

```
## S3 method for class 'voronoi'
print(x,...)
```

Arguments

x	object of class "voronoi"
...	additional paramters for print

Value

None

Author(s)

Albrecht Gebhardt <albrecht.gebhardt@aau.at>, Roger Bivand <roger.bivand@nhh.no>

See Also

voronoi, plot.voronoi, summary.voronoi

---

summary.triSht *Return a summary of a triangulation object*

Description

Returns some information (number of nodes, triangles, arcs) about object.

Usage

```
## S3 method for class 'triSht'
summary(object,...)
```

Arguments

object	object of class "triSht"
...	additional paramters for summary

Value

An object of class "summary.triSht", to be printed by print.summary.triSht.

It contains the number of nodes (n), of arcs (na), of boundary nodes (nb) and triangles (nt).

Note

This function is meant as replacement for the function of same name in package tripack.

The only difference is that no constraints are possible with triSht objects of package interp.

Author(s)

Albrecht Gebhardt <albrecht.gebhardt@aau.at>, Roger Bivand <roger.bivand@nhh.no>

See Also

triSht, print.triSht, plot.triSht, print.summary.triSht.

summary.voronoi *Return a summary of a voronoi object*

Description

Returns some information about `object`

Usage

```
## S3 method for class 'voronoi'
summary(object,...)
```

Arguments

`object`	object of class `"voronoi"`
`...`	additional parameters for `summary`

Value

Object of class `"summary.voronoi"`.

It contains the number of nodes (`nn`) and dummy nodes (`nd`).

Note

This function is meant as replacement for the function of same name in package `tripack` and should be fully backward compatible.

Author(s)

Albrecht Gebhardt <albrecht.gebhardt@aau.at>, Roger Bivand <roger.bivand@nhh.no>

See Also

`voronoi`,`voronoi.mosaic`, `print.voronoi`, `plot.voronoi`, `print.summary.voronoi`.

tri.find *Locate a point in a triangulation*

Description

This subroutine locates a point $P = (x, y)$ relative to a triangulation created by `tri.mesh`. If P is contained in a triangle, the three vertex indexes are returned. Otherwise, the indexes of the rightmost and leftmost visible boundary nodes are returned.

Usage

```
tri.find(tri.obj,x,y)
```

Arguments

tri.obj	an triangulation object of class triSht
x	x-coordinate of the point
y	y-coordinate of the point

Value

A list with elements i1,i2,i3 containing nodal indexes, in counterclockwise order, of the vertices of a triangle containing $P = (x, y)$. tr contains the triangle index and bc contains the barycentric coordinates of P w.r.t. the found triangle.

If P is not contained in the convex hull of the nodes this indices are 0 (bc is meaningless then).

Author(s)

Albrecht Gebhardt <albrecht.gebhardt@aau.at>, Roger Bivand <roger.bivand@nhh.no>

See Also

triSht, print.triSht, plot.triSht, summary.triSht, triangles, convex.hull

Examples

```
data(franke)
tr<-tri.mesh(franke$ds3$x,franke$ds3$y)
plot(tr)
pnt<-list(x=0.3,y=0.4)
triangle.with.pnt<-tri.find(tr,pnt$x,pnt$y)
attach(triangle.with.pnt)
lines(franke$ds3$x[c(i1,i2,i3,i1)],franke$ds3$y[c(i1,i2,i3,i1)],col="red")
points(pnt$x,pnt$y)
```

---

tri.mesh *Delaunay triangulation*

Description

This function generates a Delaunay triangulation of arbitrarily distributed points in the plane. The resulting object can be printed or plotted, some additional functions can extract details from it like the list of triangles, arcs or the convex hull.

Usage

```
tri.mesh(x, y = NULL, duplicate = "error", jitter = FALSE)
```

Arguments

x	vector containing x coordinates of the data. If y is missing x should be a list or dataframe with two components x and y.
y	vector containing y coordinates of the data. Can be omitted if x is a list with two components x and y.
duplicate	flag indicating how to handle duplicate elements. Possible values are: • "error" – default, • "strip" – remove all duplicate points, • "remove" – leave one point of the duplicate points.
jitter	logical, adds some jitter to both coordinates as this can help in situations with too much colinearity. Default is FALSE. Some error conditions within C++ code can also lead to enabling this internally (a warning will be displayed).

Details

This function creates a Delaunay triangulation of a set of arbitrarily distributed points in the plane referred to as nodes.

The Delaunay triangulation is defined as a set of triangles with the following five properties:

1. The triangle vertices are nodes.
2. No triangle contains a node other than its vertices.
3. The interiors of the triangles are pairwise disjoint.
4. The union of triangles is the convex hull of the set of nodes (the smallest convex set which contains the nodes).
5. The interior of the circumcircle of each triangle contains no node.

The first four properties define a triangulation, and the last property results in a triangulation which is as close as possible to equiangular in a certain sense and which is uniquely defined unless four or more nodes lie on a common circle. This property makes the triangulation well-suited for solving closest point problems and for triangle-based interpolation.

This triangulation is based on the s-hull algorithm by David Sinclair. It consist of two steps:

1. Create an initial non-overlapping triangulation from the radially sorted nodes (w.r.t to an arbitrary first node). Starting from a first triangle built from the first node and its nearest neigbours this is done by adding triangles from the next node (in the sense of distance to the first node) to the hull of the actual triangulation visible from this node (sweep hull step).
2. Apply triange flipping to each pair of triangles sharing a border until condition 5 holds (Cline-Renka test).

This algorithm has complexicity $O(n * log(n))$.

Value

an object of class "triSht", see triSht.

Note

This function is meant as a replacement for function `tri.mesh` from package `tripack`. Please note that the underlying algorithm changed from Renka's method to Sinclair's sweep hull method. Delaunay triangulations are unique if no four or more points exist which share the same circumcircle. Otherwise several solutions are available and different algorithms will give different results. This especially holds for regular grids, where in the case of rectangular gridded points each grid cell can be triangulated in two different ways.

The arguments are backward compatible, but the returned object is not compatible with package `tripack` (it provides a `tri` object type)! But you can apply methods with same names to the object returned in package `interp` which is of type `triSht`, so you can reuse your old code but you cannot reuse your old saved workspace.

Author(s)

Albrecht Gebhardt <albrecht.gebhardt@aau.at>, Roger Bivand <roger.bivand@nhh.no>

References

B. Delaunay, Sur la sphere vide. A la memoire de Georges Voronoi, Bulletin de l'Academie des Sciences de l'URSS. Classe des sciences mathematiques et na, 1934, no. 6, p. 793–800

D. A. Sinclair, S-Hull: A Fast Radial Sweep-Hull Routine for Delaunay Triangulation. https://arxiv.org/pdf/1604.01428.pdf, 2016.

See Also

`triSht`, `print.triSht`, `plot.triSht`, `summary.triSht`, `triangles`, `convex.hull`, `arcs`.

Examples

```
## use Frankes datasets:
data(franke)
tr1 <- tri.mesh(franke$ds3$x, franke$ds3$y)
tr1
tr2 <- tri.mesh(franke$ds2)
summary(tr2)
```

---

triangles *Extract a list of triangles from a triangulation object*

Description

This function extracts a list of triangles from an triangulation object created by `tri.mesh`.

Usage

```
triangles(tri.obj)
```

Arguments

tri.obj	object of class triSht

Details

The vertices in the returned matrix (let's denote it with retval) are ordered counterclockwise. The columns tr$x$ and arc$x$, $x = 1, 2, 3$ index the triangle and arc, respectively, which are opposite (not shared by) node node$x$, with tri$x = 0$ if arc$x$ indexes a boundary arc. Vertex indexes range from 1 to n, the number of nodes, triangle indexes from 0 to nt, and arc indexes from 1 to $na = nt + n - 1$.

Value

A matrix with columns node1, node2, node3, representing the vertex nodal indexes, tr1, tr2, tr3, representing neighboring triangle indexes and arc1, arc2, arc3 reresenting arc indexes.

Each row represents one triangle.

Author(s)

Albrecht Gebhardt <albrecht.gebhardt@aau.at>, Roger Bivand <roger.bivand@nhh.no>

See Also

triSht, print.triSht, plot.triSht, summary.triSht, triangles

Examples

```
# use the smallest Franke data set
data(franke)
fr3.tr<-tri.mesh(franke$ds3$x, franke$ds3$y)
triangles(fr3.tr)
```

---

triSht *A triangulation object*

Description

R object that represents the triangulation of a set of 2D points, generated by tri.mesh.

Arguments

n	Number of nodes
x	x coordinates of the triangulation nodes
y	y coordinates of the triangulation nodes
nt	number of triangles

`trlist` Matrix of indices which defines the triangulation, each row corresponds to a triangle.

Columns `i1`, `i2`, `i3` of the row i contain the node indices defining the ith triangle.

Columns `j1`, `j2`, `j3` of the row i contain the indices of neighbour triangles (or 0 if no neighbour available along the convex hull).

Columns `k1`, `k2`, `k3` of the row i contain the indices of the arcs of the ith triangle as returned by the `arcs` function.

`cclist` Matrix describing the circumcircles and triangles.

Columns `x` and `y` contain coordinates of the circumcircle centers, `r` is the circumcircle radius.

`area` is the triangle area and `ratio` is the ratio of the radius of the inscribed circle to the circumcircle radius. It takes it maximum value 0.5 for an equilateral triangle.

The radius of the inscribed circle can be get via $r_i = \frac{r}{ratio}$.

`nchull` number of points on the convex hull

`chull` A vector containing the indices of nodes forming the convec hull (in counterclockwise ordering).

`narcs` number of arcs forming the triangulation

`arcs` A matrix with node indices describing the arcs, contains two columns `from` and `to`.

`call` call, which generated this object

Note

This object is not backward compatible with `tri` objects generated from package `tripack` but the functions and methods are! So you have to regenerate these objects and then you can continue to use the same calls as before.

The only difference is that no constraints to the triangulation are possible in package `interp`.

Function `triSht2tri` provides an option to convert this object into the older form from package `tripack`, but it will not generate exact copies as if the object would have been created with `tripack::tri.mesh`! The old data structure consists of three lists describing adjacency lists of triangulation nodes in counterclockwise order, the translation function only genrates such a valid (but not unique) description.

Author(s)

Albrecht Gebhardt <albrecht.gebhardt@aau.at>, Roger Bivand <roger.bivand@nhh.no>

See Also

`tri.mesh`, `print.triSht`,`triSht2tri`, `plot.triSht`, `summary.triSht`

triSht2tri *Converter to tripack objects*

Description

This function converts `triSht` objects (from this package) to `tri` objects (from tripack package).

Usage

```
triSht2tri(t.triSht)
```

Arguments

`t.triSht`	a class `triSht` object as returned by `tri.mesh`

Value

A class `tri` object, see tripack package.

Note

The converted objects are not fully compatible with `tripack` functions. Basic stuff (printing, plotting) works, `tripack::triangles` e.g. does not work. Voronoi functions from package `tripack` are working correctly with translated objects.

Author(s)

A. Gebhardt

See Also

`tri.mesh`, `triSht`

tritest *tritest / sample data*

Description

A very simply set set of points to test the tripack functions, taken from the FORTRAN original. `tritest2` is a slight modification by adding `runif(,-0.1,0.1)` random numbers to the coordinates.

References

R. J. Renka (1996). Algorithm 751: TRIPACK: a constrained two-dimensional Delaunay triangulation package. ACM Transactions on Mathematical Software. **22**, 1-8.

voronoi *Voronoi object*

Description

A voronoi object is created with voronoi.mosaic

Arguments

x,y	x and y coordinates of nodes of the voronoi mosaic. Each node is a circumcircle center of some triangle from the Delaunay triangulation.
node	logical vector, indicating real nodes of the voronoi mosaic. These nodes are the centers of circumcircles of triangles with positive area of the delaunay triangulation. If node[i]=FALSE, (c[i],x[i]) belongs to a triangle with area 0.
n1,n2,n3	indices of neighbour nodes. Negative indices indicate dummy points as neighbours.
tri	triangulation object, see triSht.
area	area of triangle i.
ratio	aspect ratio (inscribed radius/circumradius) of triangle i.
radius	circumradius of triangle i.
dummy.x,dummy.y	x and y coordinates of dummy points. They are used for plotting of unbounded tiles.

Note

This version of voronoi object is generated from the tri.mesh function from package interp. That's the only difference to voronoi objects generated with package tripack.

Author(s)

Albrecht Gebhardt <albrecht.gebhardt@aau.at>, Roger Bivand <roger.bivand@nhh.no>

See Also

voronoi.mosaic,plot.voronoi

`voronoi.area` *Calculate area of Voronoi polygons*

Description

Computes the area of each Voronoi polygon. For some sites at the edge of the region, the Voronoi polygon is not bounded, and so the area of those sites cannot be calculated, and hence will be `NA`.

Usage

```
voronoi.area(voronoi.obj)
```

Arguments

`voronoi.obj`	object of class `"voronoi"`

Value

A vector of polygon areas.

Author(s)

S. J. Eglen

See Also

`voronoi.mosaic`,`voronoi.polygons`,

Examples

```
data(franke)
fd3 <- franke$ds3
fd3.vm <- voronoi.mosaic(fd3$x,fd3$y)
fd3.vm.areas <- voronoi.area(fd3.vm)
plot(fd3.vm)
text(fd3$x, fd3$y, round(fd3.vm.areas,5))
```

`voronoi.findrejectsites` *Find the Voronoi sites at the border of the region (to be rejected).*

Description

Find the sites in the Voronoi tesselation that lie at the edge of the region. A site is at the edge if any of the vertices of its Voronoi polygon lie outside the rectangle with corners (xmin,ymin) and (xmax,ymax).

Usage

```
voronoi.findrejectsites(voronoi.obj, xmin, xmax, ymin, ymax)
```

Arguments

`voronoi.obj`	object of class `"voronoi"`
`xmin`	minimum x-coordinate of sites in the region
`xmax`	maximum x-coordinate of sites in the region
`ymin`	minimum y-coordinate of sites in the region
`ymax`	maximum y-coordinate of sites in the region

Value

A logical vector of the same length as the number of sites. If the site is a reject, the corresponding element of the vector is set to TRUE.

Author(s)

S. J. Eglen

See Also

`voronoi.polygons`

`voronoi.mosaic` *Voronoi mosaic*

Description

This function creates a Voronoi mosaic out of a given set of arbitraryly located points in the plane. Each cell of a voronoi mosaic is associated with a data point and contains all points (x, y) closest to this data point.

Usage

```
voronoi.mosaic(x, y = NULL, duplicate = "error")
```

Arguments

`x`	vector containing x coordinates of the data. If `y` is missing `x` should be a list or dataframe with two components `x` and `y`. `x` can also be an object of class `triSht` generated by `tri.mesh`. In this case the internal triangulation step can be skipped.
`y`	vector containing y coordinates of the data. Can be omitted if `x` is a list with two components `x` and `y`.
`duplicate`	flag indicating how to handle duplicate elements. Possible values are: • `"error"` – default, • `"strip"` – remove all duplicate points, • `"remove"` – leave one point of the duplicate points.

Details

The function creates first a Delaunay triangulation (if not already given), extracts the circumcircle centers of these triangles, and then connects these points according to the neighbourhood relations between the triangles.

Value

An object of class `voronoi`.

Note

This function is meant as a replacement for function `voronoi.mosaic` from package `tripack`. Please note that the underlying triangulation uses a different algorithm, see `tri.mesh`. Contrary to `tri.mesh` this should not affect the result for non unique triangulations e.g. on regular grids as the voronoi mosaic in this case will still be unique.

The arguments are backward compatible, even the returned object should be compatible with functions from package `tripack`.

Author(s)

Albrecht Gebhardt <albrecht.gebhardt@aau.at>, Roger Bivand <roger.bivand@nhh.no>

References

G. Voronoi, Nouvelles applications des parametres continus a la theorie des formes quadratiques. Deuxieme memoire. Recherches sur les parallelloedres primitifs, Journal fuer die reine und angewandte Mathematik, 1908, vol 134, p. 198-287

See Also

voronoi,voronoi.mosaic, print.voronoi, plot.voronoi

Examples

```
data(franke)
fd <- franke$ds3
vr <- voronoi.mosaic(fd$x, fd$y)
summary(vr)
```

---

voronoi.polygons *extract polygons from a voronoi mosaic*

Description

This functions extracts polygons from a voronoi.mosaic object.

Usage

```
voronoi.polygons(voronoi.obj)
```

Arguments

voronoi.obj	object of class voronoi.mosaic

Value

Returns an object of class voronoi.polygons with unamed list elements for each polygon. These list elements are matrices with columns x and y. Unbounded polygons along the border are represented by NULL instead of a matrix.

Author(s)

Denis White

See Also

plot.voronoi.polygons,voronoi.mosaic

Examples

```
data(franke)
fd3 <- franke$ds3
fd3.vm <- voronoi.mosaic(fd3$x,fd3$y)
fd3.vp <- voronoi.polygons(fd3.vm)
fd3.vp
```

Index