Package ‘akima’

May 30, 2020

Version 0.6-2.1

Date 2016-12-16

Title Interpolation of Irregularly and Regularly Spaced Data

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

Description Several cubic spline interpolation methods of H. Akima for irregular and regular gridded data are available through this package, both for the bivariate case (irregular data: ACM 761, regular data: ACM 760) and univariate case (ACM 433 and ACM 697). Linear interpolation of irregular gridded data is also covered by reusing D. J. Renkas triangulation code which is part of Akimas Fortran code. A bilinear interpolator for regular grids was also added for comparison with the bicubic interpolator on regular grids.

License ACM | file LICENSE

Depends R (>= 2.0.0)

Imports sp

NeedsCompilation yes

Author Hiroshi Akima [aut, cph] (Fortran code (TOMS 760, 761, 697 and 433)),
Albrecht Gebhardt [aut, cre, cph] (R port (interp* functions), bicubic* functions),
Thomas Petzold [ctb, cph] (aspline function),
Martin Maechler [ctb, cph] (interp2xyz function + enhancements),
YYYY Association for Computing Machinery, Inc. [cph] (covers code from TOMS 760, 761, 697 and 433)

License_restricts_use yes

Repository CRAN

Date/Publication 2020-05-30 09:50:55 UTC

R topics documented:

---

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)
```

---

akima760 — *Sample data from Akima's Bicubic Spline Interpolation code (TOMS 760)*

Description

akima760 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, "

", ACM Transactions on Mathematical Software, Vol. 22, No. 3, September 1996, pp. 357-361.

Examples

```
## Not run:
library(rgl)
data(akima)
# data
rgl.spheres(akima760$x,akima760$z , akima760$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)
```

---

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="original", 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 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, and 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.

Value

A list with components x and y, containing n coordinates which interpolate the given data points.

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

approx, 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))
lines(spline(x, y, xmin=min(xnew), xmax=max(xnew), n=200), col="blue")

lines(aspline(x, y, xnew), 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))
lines(spline(x, y, xmin=min(xnew), xmax=max(xnew), n=200), col="blue")

lines(aspline(x, y, xnew), 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))
lines(spline(x, y, n=200), col="blue")

lines(aspline(x, y, n=200), 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

The description in the Fortran code says:

This subroutine performs interpolation of a bivariate function, z(x,y), on a rectangular grid in the x-y plane. It is based on the revised Akima method.

In this subroutine, the interpolating function is a piecewise function composed of a set of bicubic (bivariate third-degree) polynomials, each applicable to a rectangle of the input grid in the x-y plane. Each polynomial is determined locally.

This subroutine has the accuracy of a bicubic polynomial, i.e., it interpolates accurately when all data points lie on a surface of a bicubic polynomial.

The grid lines can be unevenly spaced.

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 functiuon is a R interface to Akima's Rectangular-Grid-Data Fitting algorithm (TOMS 760). The algorithm has the accuracy of a bicubic (bivariate third-degree) polynomial.

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(akima760)
# interpolate at the diagonal of the grid [0,8]x[0,10]
akima.bic <- bicubic(akima760$x,akima760$y,akima760$z,
                     seq(0,8,length=50), seq(0,10,length=50))
plot(sqrt(akima.bic$x^2+akima.bic$y^2), akima.bic$z, type="l")
```

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

Description

The description in the Fortran code says:

This subroutine performs interpolation of a bivariate function, z(x,y), on a rectangular grid in the x-y plane. It is based on the revised Akima method.

In this subroutine, the interpolating function is a piecewise function composed of a set of bicubic (bivariate third-degree) polynomials, each applicable to a rectangle of the input grid in the x-y plane. Each polynomial is determined locally.

This subroutine has the accuracy of a bicubic polynomial, i.e., it interpolates accurately when all data points lie on a surface of a bicubic polynomial.

The grid lines can be unevenly spaced.

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 functiuon is a R interface to Akima's Rectangular-Grid-Data Fitting algorithm (TOMS 760). The algorithm has the accuracy of a bicubic (bivariate third-degree) polynomial.

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(akima760)
# interpolate at a grid [0,8]x[0,10]
akima.bic <- bicubic.grid(akima760$x,akima760$y,akima760$z)
zmin <- min(akima.bic$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)
```

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

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`, `bicubic.grid`

Examples

```
data(akima760)
# interpolate at the diagonal of the grid [0,8]x[0,10]
akima.bil <- bilinear(akima760$x,akima760$y,akima760$z,
                      seq(0,8,length=50), seq(0,10,length=50))
plot(sqrt(akima.bil$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)
```

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

Use `interp` 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/

See Also

`interp`, `bicubic`

Examples

```
data(akima760)
# interpolate at a grid [0,8]x[0,10]
akima.bil <- bilinear.grid(akima760$x,akima760$y,akima760$z)
zmin <- min(akima.bil$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)
```

`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 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 out of [Akima 1996].

Author(s)

A. Gebhardt,

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)
```

---

interp *Gridded Bivariate Interpolation for Irregular Data*

Description

These functions implement bivariate interpolation onto a grid for irregularly spaced input data. Bilinear or bicubic spline interpolation is applied using different versions of algorithms from Akima.

Usage

```
interp(x, y=NULL, z, xo=seq(min(x), max(x), length = nx),
       yo=seq(min(y), max(y), length = ny),
       linear = TRUE, extrap=FALSE, duplicate = "error", dupfun = NULL,
       nx = 40, ny = 40,
       jitter = 10^-12, jitter.iter = 6, jitter.random = FALSE)
```

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` 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). Some heuristics is built in to avoid this case by adding small jitter to x and y when the number of NA values in the result exceeds 10%.

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
: vector of x-coordinates of output grid. The default is 40 points evenly spaced over the range of x. If extrapolation is not being used (extrap=FALSE, the default), xo should have a range that is close to or inside of the range of x for the results to be meaningful.

yo
: vector of y-coordinates of output grid; analogous to xo, see above.

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?

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

jitter
: Jitter of amount of diff(range(XX))*jitter (XX=x or y) will be added to coordinates if collinear points are detected. Afterwards interpolation will be tried once again.

  Note that the jitter is not generated randomly unless jitter.random is set to TRUE. This ensures reproducable result. tri.mesh of package tripack uses the same jitter mechanism. That means you can plot the triangulation on top of the interpolation and see the same triangulation as used for interpolation, see examples below.

jitter.iter
: number of iterations to retry with jitter, amount will be multiplied in each iteration by iter^1.5

jitter.random
: logical, see jitter, defaults to FALSE

Details

If linear is TRUE (default), linear interpolation is used in the triangles bounded by data points. Cubic interpolation is done if linear is set to FALSE. If extrap is FALSE, z-values for points outside the convex hull are returned as NA. No extrapolation can be performed for the linear case.

The interp function handles duplicate (x,y) points in different ways. As default it will stop with an error message. But it can give duplicate points an unique z value according to the parameter duplicate (mean,median or any other user defined function).

The triangulation scheme used by `interp` works well if `x` and `y` have similar scales but will appear stretched if they have very different scales. The spreads of `x` and `y` must be within four orders of magnitude of each other for `interp` to work.

Value

list with 3 components:

`x,y` 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`.

`z` matrix of fitted z-values. The value `z[i,j]` is computed at the x,y point `xo[i],yo[j]`. `z` has dimensions `length(xo)` times `length(yo)`.

If input is a `SpatialPointsDataFrame` a `SpatialPixelssDataFrame` is returned.

Note

`interp` uses Akimas new Fortran code from 1996 for spline interpolation, the triangulation (based on Renkas tripack) is reused for linear interpolation. In this newer version Akima switched from his own triangulation to Renkas tripack (=TOMS 751).

Note that S-Plus uses (used?) the old Fortran code from Akima 1978.

The resulting structure is suitable for input to the functions `contour` and `image`. Check the requirements of these functions when choosing values for `xo` and `yo`.

References

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.

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

See Also

`contour`, `image`, `approx`, `spline`, `aspline`, `outer`, `expand.grid`, `link{franke.data}`.

Examples

```
data(akima)
plot(y ~ x, data = akima, main = "akima example data")
with(akima, text(x, y, formatC(z,dig=2), adj = -0.1))

## linear interpolation
akima.li <- interp(akima$x, akima$y, akima$z)
li.zmin <- min(akima.li$z,na.rm=TRUE)
li.zmax <- max(akima.li$z,na.rm=TRUE)
breaks <- pretty(c(li.zmin,li.zmax),10)
```

```
colors <- heat.colors(length(breaks)-1)
with(akima.li, image  (x,y,z, breaks=breaks, col=colors))
with(akima.li,contour(x,y,z, levels=breaks, add=TRUE))
points (akima, pch = 3)

## increase smoothness (using finer grid):
akima.smooth <-
    with(akima, interp(x, y, z, nx=100, ny=100))
si.zmin <- min(akima.smooth$z,na.rm=TRUE)
si.zmax <- max(akima.smooth$z,na.rm=TRUE)
breaks <- pretty(c(si.zmin,si.zmax),10)
colors <- heat.colors(length(breaks)-1)

image  (akima.smooth, main = "interp(<akima data>, *) on finer grid",
        breaks=breaks, col=colors)
contour(akima.smooth, add = TRUE, levels=breaks, col = "thistle")
points(akima, pch = 3, cex = 2, col = "blue")

## use triangulation package to show underlying triangulation:
## Not run:
if(library(tripack, logical.return=TRUE))
   plot(tri.mesh(akima), add=TRUE, lty="dashed")

## End(Not run)
## use only 15 points (interpolation only within convex hull!)
akima.part <- with(akima, interp(x[1:15], y[1:15], z[1:15]))
p.zmin <- min(akima.part$z,na.rm=TRUE)
p.zmax <- max(akima.part$z,na.rm=TRUE)
breaks <- pretty(c(p.zmin,p.zmax),10)
colors <- heat.colors(length(breaks)-1)

image(akima.part, breaks=breaks, col=colors)
title("interp() on subset of only 15 points")
contour(akima.part, levels=breaks, add=TRUE)
points(akima$x[1:15],akima$y[1:15], col = "blue")

## spline interpolation
akima.spl <- with(akima, interp(x, y, z, nx=100, ny=100, linear=FALSE))

contour(akima.spl, main = "smooth  interp(*, linear = FALSE)")
points(akima)

full.pal <- function(n) hcl(h = seq(340, 20, length = n))
cool.pal <- function(n) hcl(h = seq(120, 0, length = n) + 150)
warm.pal <- function(n) hcl(h = seq(120, 0, length = n) - 30)

filled.contour(akima.spl, color.palette = full.pal,
        plot.axes = { axis(1); axis(2);
                      title("smooth  interp(*, linear = FALSE)");
                      points(akima, pch = 3, col= hcl(c=100, l = 20))})
## no extrapolation!

## Not run:
```

```
    ## interp can handle spatial point dataframes created by the sp package:
    library(sp)
    data(meuse)
    coordinates(meuse) <- ~x+y
    ## argument z has to be named, y has to be omitted!
    z <- interp(meuse,z="zinc",nx=100,ny=150)
    spplot(z,"zinc")
    z <- interp(meuse,z="zinc",nx=100,ny=150,linear=FALSE)
    spplot(z,"zinc")

## End(Not run)

## Not run:
### An example demonstrating the problems that occur for rectangular
### gridded data.
###
require(tripack)
### Create irregularly spaced sample data on even values of x and y
### (the "14" makes it irregular spacing).
x   <- c(seq(2,10,2),14)
nx  <- length(x)
y   <- c(seq(2,10,2),14)
ny  <- length(y)
nxy <- nx*ny
xy  <- expand.grid(x,y)
colnames(xy) <- c("x","y")
### prepare a dataframe for interp
df  <- cbind(xy,z=rnorm(nxy))
### and a matrix for bicubic and bilinear
z   <- matrix(df$z,nx,ny)

old.par <- par(mfrow=c(2,2))
### First: bicubic spline interpolation:
### This is Akimas bicubic spline implementation for regular gridded
### data:
iRbic <- bicubic.grid(x,y,z,nx=250,ny=250)
### Note that this interpolation tends to extreme values in large cells.
### Therefore zmin and zmax are taken from here to generate the same
### color scheme for the next plots.
zmin <- min(iRbic$z, na.rm=TRUE)
zmax <- max(iRbic$z, na.rm=TRUE)
breaks <- pretty(c(zmin,zmax),10)
colors <- heat.colors(length(breaks)-1)
image(iRbic,breaks=breaks,col = colors)
contour(iRbic,col="black",levels=breaks,add=TRUE)
points(xy$x,xy$y)
title(main="bicubic interpolation",
      xlab="bcubic.grid(...)",
      sub="Akimas regular grid version, ACM 760")

### Now Akima splines with accurracy of bicubic polynomial
### for irregular gridded data:
iRspl <- with(df,interp(x,y,z,linear=FALSE,nx=250,ny=250))
```

```
### Note that the triangulation is created by adding small amounts
### of jitter to the coordinates, resulting in an unique triangulation.
### This jitter is not randomly choosen to get reproducable results.
### tri.mesh() from package tripack uses the same code and so produces the
### same triangulation.
image(iRspl,breaks=breaks,col = colors)
contour(iRspl,col="black",levels=breaks,add=TRUE)
plot(tri.mesh(xy$x,xy$y),col="white",add=TRUE)
title(main="bicubic* interpolation",
      xlab="interp(...,linear=FALSE)",
      ylab="*: accuracy of bicubic polynomial"
      sub="Akimas irregular grid version, ACM 761")

### Just for comparison an implementation of bilinear interpolation,
### only applicable to regular gridded data:
iRbil <- bilinear.grid(x,y,z,nx=250,ny=250)
### Note the lack of differentiability at grid cell borders.
image(iRbil,breaks=breaks,col = colors)
contour(iRbil,col="black",levels=breaks,add=TRUE)
points(xy$x,xy$y)
title(main="bilinear interpolation",
      xlab="bilinear.grid(...)",
      sub="only works for regular grid")

### Linear interpolation using the same triangulation as
### Akima bicubic splines for irregular gridded data.
iRlin <- with(df,interp(x,y,z,linear=TRUE,nx=250,ny=250))
### Note how the triangulation influences the interpolation.
### For this rectangular gridded dataset the triangulation
### in each rectangle is arbitraryly choosen from two possible
### solutions, hence the interpolation would change drastically
### when the triangulation changes. For this reason interp()
### is not meant for regular (rectangular) gridded data!
image(iRlin,breaks=breaks,col = colors)
contour(iRlin,col="black",levels=breaks,add=TRUE)
plot(tri.mesh(xy$x,xy$y),col="white",add=TRUE)
title(main="linear interpolation",
      xlab="interp(...,linear=TRUE)",
      sub="same triangulation as Akima irregular grid")

### And now four times Akima 761 with random jitter for
### triangulation correction, note that now interp() and tri.mesh()
### need the same random seed to produce identical triangulations!
for(i in 1:4){
    set.seed(42+i)
    iRspl <- with(df,interp(x,y,z,linear=FALSE,nx=250,ny=250,jitter.random=TRUE))
    image(iRspl,breaks=breaks,col = colors)
    contour(iRspl,col="black",levels=breaks,add=TRUE)
    set.seed(42+i)
    plot(tri.mesh(xy$x,xy$y,jitter.random=TRUE),col="white",add=TRUE)
    title(main="bicubic* interpolation",
          xlab="interp(...,linear=FALSE)",
          ylab="random jitter added",
```

```
          sub="Akimas irregular grid version, ACM 761")
}
par(old.par)


## End(Not run)
### Use all datasets from Franke, 1979:
data(franke)
for(i in 1:5)
    for(j in 1:3){
        FR <- franke.data(i,j,franke)
        IL <- with(FR, interp(x,y,z,linear=FALSE))
        image(IL)
        contour(IL,add=TRUE)
        with(FR,points(x,y))
    }
```

---

interp.old *Gridded Bivariate Interpolation for Irregular Data*

Description

These functions implement bivariate interpolation onto a grid for irregularly spaced input data. These functions are only for backward compatibility, use `interp` instead.

Usage

```
interp.old(x, y, z, xo= seq(min(x), max(x), length = 40),
           yo=seq(min(y), max(y), length = 40), ncp = 0,
           extrap=FALSE, duplicate = "error", dupfun = NULL)
interp.new(x, y, z, xo = seq(min(x), max(x), length = 40),
           yo = seq(min(y), max(y), length = 40), linear = FALSE,
           ncp = NULL, extrap=FALSE, duplicate = "error", dupfun = NULL)
```

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). 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
: vector of x-coordinates of output grid. The default is 40 points evenly spaced over the range of x. If extrapolation is not being used (extrap=FALSE, the default), xo should have a range that is close to or inside of the range of x for the results to be meaningful.

yo
: vector of y-coordinates of output grid; analogous to xo, see above.

linear
: logical – indicating wether linear or spline interpolation should be used. supersedes old ncp parameter

ncp
: deprecated, use parameter linear. Now only used by interp.old().

  old meaning was: number of additional points to be used in computing partial derivatives at each data point. ncp must be either 0 (partial derivatives are not used), or at least 2 but smaller than the number of data points (and smaller than 25).

extrap
: logical flag: should extrapolation be used outside of the convex hull determined by the data points?

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".

Details

see interp

Value

list with 3 components:

x,y
: 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.

z
: matrix of fitted z-values. The value z[i,j] is computed at the x,y point xo[i],yo[j]. z has dimensions length(xo) times length(yo).

If input is a SpatialPointsDataFrame a SpatialPixelssDataFrame is returned.

Note

interp.new is deprecated and interp.old will soon be deprecated.

References

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

`contour`, `image`, `approx`, `spline`, `aspline`, `outer`, `expand.grid`.

`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, linear = FALSE,
                             xo= seq(0,25, length=100),
                             yo= seq(0,20, length= 96)))
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 Bivariate Interpolation for Irregular Data*

Description

These functions implement bivariate interpolation onto a set of points for irregularly spaced input data. These functions are only for backward compatibility, use `interpp` instead.

If `linear` is \codeTRUE, linear interpolation is used in the triangles bounded by data points, otherwise cubic interpolation is done.

If `extrap` is `FALSE`, z-values for points outside the convex hull are returned as `NA`. No extrapolation can be performed for linear interpolation.

The `interpp` function handles duplicate `(x,y)` points in different ways. As default it will stop with an error message. But it can give duplicate points an unique `z` value according to the parameter `duplicate` (`mean`,`median` or any other user defined function).

The triangulation scheme used by `interp` works well if `x` and `y` have similar scales but will appear stretched if they have very different scales. The spreads of `x` and `y` must be within four orders of magnitude of each other for `interpp` to work.

Usage

```
interpp(x, y=NULL, z, xo, yo=NULL, linear=TRUE, extrap=FALSE,
        duplicate = "error", dupfun = NULL,
        jitter = 10^-12, jitter.iter = 6, jitter.random = FALSE)
```

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 `SpatialPointsDataFrame`s 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"`

jitter
: Jitter of amount of `diff(range(XX))*jitter` (XX=x or y) will be added to coordinates if collinear points are detected. Afterwards interpolation will be tried once again.

  Note that the jitter is not generated randomly unless `jitter.random` is set to `TRUE`. This ensures reproducable result. `tri.mesh` of package `tripack` uses the same jitter mechanism. That means you can plot the triangulation on top of the interpolation and see the same triangulation as used for interpolation, see examples below.

jitter.iter
: number of iterations to retry with jitter, amount will be increased in each iteration by `iter^1.5`

jitter.random
: logical, see `jitter`, defaults to `FALSE`

Value

list with 3 components:

x
: vector of x-coordinates of output points, the same as the input argument `xo`.

y
: vector of y-coordinates of output points, the same as the input argument `yo`.

z
: fitted z-values. The value `z[i]` is computed at the x,y point `x[i],y[i]`.

If input is `SpatialPointsDataFrame` than an according `SpatialPointsDataFrame` is returned.

NOTE

Use `interp` if interpolation on a regular grid is wanted.

See `interp` for more details.

References

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.

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

See Also

`contour`, `image`, `approxfun`, `splinefun`, `outer`, `expand.grid`, `interp`, `aspline`.

Examples

```
data(akima)
# linear interpolation at points (1,2), (5,6) and (10,12)
akima.lip<-interpp(akima$x, akima$y, akima$z,c(1,5,10),c(2,6,12))
akima.lip$z
# spline interpolation
akima.sip<-interpp(akima$x, akima$y, akima$z,c(1,5,10),c(2,6,12),
  linear=FALSE)
akima.sip$z
## Not run:
    ## interaction with sp objects:
    library(sp)
    ## take 30 sample points out of meuse grid:
    data(meuse.grid)
    m0 <- meuse.grid[sample(1:3103,30),]
    coordinates(m0) <- ~x+y
    ## interpolate on this 30 points:
    ## note: both "meuse" and "m0" are sp objects
    ## (SpatialPointsDataFrame) !!
    ## arguments z and xo have to named, y has to be omitted!
    ipp <- interpp(meuse,z="zinc",xo=m0)
    spplot(ipp)

## End(Not run)
```

`interpp.old` *Pointwise Bivariate Interpolation for Irregular Data*

Description

If ncp is zero, linear interpolation is used in the triangles bounded by data points. Cubic interpolation is done if partial derivatives are used. If `extrap` is `FALSE`, z-values for points outside the convex hull are returned as `NA`. No extrapolation can be performed if ncp is zero.

The interpp function handles duplicate (x,y) points in different ways. As default it will stop with an error message. But it can give duplicate points an unique z value according to the parameter duplicate (mean,median or any other user defined function).

The triangulation scheme used by interp works well if x and y have similar scales but will appear stretched if they have very different scales. The spreads of x and y must be within four orders of magnitude of each other for interpp to work.

Usage

```
interpp.old(x, y, z, xo, yo, ncp = 0, extrap = FALSE,
                      duplicate = "error", dupfun = NULL)
interpp.new(x, y, z, xo, yo, extrap = FALSE,
                      duplicate = "error", dupfun = NULL)
```

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
ncp	deprecated, use parameter linear. Now only used by interpp.old(). meaning was: number of additional points to be used in computing partial derivatives at each data point. ncp must be either 0 (partial derivatives are not used, = linear interpolation), or at least 2 but smaller than the number of data points (and smaller than 25).
extrap	logical flag: should extrapolation be used outside of the convex hull determined by the data points?
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"

Value

list with 3 components:

- `x` vector of x-coordinates of output points, the same as the input argument `xo`.
- `y` vector of y-coordinates of output points, the same as the input argument `yo`.
- `z` fitted z-values. The value `z[i]` is computed at the x,y point `x[i],y[i]`.

If input is `SpatialPointsDataFrame` than an according `SpatialPointsDataFrame` is returned.

NOTE

Use `interp` if interpolation on a regular grid is wanted.

The two versions `interpp.old` and `interpp.new` are now deprecated, use `interpp` instead, see details there.

Earlier versions (pre 0.5-1) of `interpp` used the parameter `ncp` to choose between linear and cubic interpolation, this is now done by setting the logical parameter `linear`. Use of `ncp` is still possible, but is deprecated.

References

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

`contour`, `image`, `approxfun`, `splinefun`, `outer`, `expand.grid`, `interp`, `aspline`.

Index