Package ‘akima’

February 19, 2015

Version 0.5-11
Date 2013-01-19
Title Interpolation of irregularly spaced data
Author Fortran code by H. Akima
R port by Albrecht Gebhardt <albrecht.gebhardt@uni-klu.ac.at>
aspline function by Thomas Petzoldt <thomas.petzoldt@tu-dresden.de>
interp2xyz, enhancements and corrections by Martin Maechler <maechler@stat.math.ethz.ch>
Maintainer Albrecht Gebhardt <albrecht.gebhardt@uni-klu.ac.at>
Description Linear or cubic spline interpolation for irregular gridded data
License ACM | file LICENSE
Depends R (>= 2.0.0)
NeedsCompilation yes
License_restricts_use yes
Repository CRAN
Date/Publication 2013-09-16 14:11:47

R topics documented:

akima ................................................................. 2
akima760 ............................................................ 3
aspline .............................................................. 4
bicubic ............................................................... 6
bicubic.grid .......................................................... 8
interp ................................................................. 9
interp2xyz .......................................................... 13
interpp ............................................................. 14

Index 16
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


Examples

```r
## 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, extrapol = TRUE)
```
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, 


Examples

## Not run:
library(rgl)
data(akima)
# data
rgl.spheres(akima760$x, akima760$y, akima760$z, color="red")
rgl.bbox()
# bivariate linear interpolation
# interp:
akima.li <- interp(akima$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(akimationli$x, akimationli$y, akimationli$z, color="green", alpha=c(0.5))
# interpp:
akima.p <- interpp(akima$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$x, akima.p$y, akima.p$z, size=4, color="yellow")
# bivariate spline interpolation
# data
rgl.spheres(akima$x, akima$z, akima$y, 0.5, color="red")
rgl.box()
# 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))
# interp 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**

```r
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

See Also
approx, spline

Examples
```r
## 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)
```
bicubic

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 function 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.

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")
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


See Also

`interp, bicubic.grid`

Examples

```r
data(akima76P)
# interpolate at the diagonal of the grid [0,8]x[0,10]
akima.bic <- bicubic(akima76Px, akima76Py, akima76Pz,
                      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")

## Should be DIRECTLY executable !! ----
## or do help(data=index) for the standard data sets.

## The function is currently defined as
function (x, y, z, x0, y0)
{
  nx <- length(x)
  ny <- length(y)
  if (dim(z)[1] != nx)
    stop("dim(z)[1] and length of x differs!")
  if (dim(z)[2] != ny)
    stop("dim(z)[2] and length of y differs!")
  n0 <- length(x0)
  if (length(y0) != n0)
    stop("length of y0 and x0 differs!")
  ret <- .Fortran("rgbicp", md = as.integer(1), nxd = as.integer(nx),
                  nyd = as.integer(ny), xd = as.double(x),
                  yd = as.double(y),
                  zd = as.double(z), nip = as.integer(n0),
                  xi = as.double(x0),
                  yi = as.double(y0), zi = double(n0), ier = integer(1),
                  ...)
  return(ret)
}
```
bicubic.grid

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.grid(x,y,z,xlim,ylim,dx,dy)

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.
dx output grid spacing in x direction.
dy output grid spacing in y direction.

Details

This function is an 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**


**See Also**

`interp, bicubic`

**Examples**

```r
data(akima76P)
# interpolate at a grid [0.8]x[0.10]
akima.bic <- bicubic.grid(akima76P$x, akima76P$y, akima76P$z,
c(0.8), c(0.10), 0.1, 0.1)
image(akima.bic)
contour(akima.bic, add=TRUE)
```

---

** 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**

```r
typ(x, y, z, xo=seq(min(x), max(x), length = 40),
yo=seq(min(y), max(y), length = 40),
linear = TRUE, extrap=FALSE, duplicate = "error", dupfun = NULL, ncp = NULL)
typ.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)
typ.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. Missing values are not accepted.
- **y** – vector of y-coordinates of data points. Missing values are not accepted.
- **z** – vector of z-coordinates of data points. Missing values are not accepted.

x, y, and z must be the same length 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 whether linear or spline interpolation should be used. supercedes old ncp parameter
- **ncp** – deprecated, use parameter linear. Now only used by interp.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

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.
*interp*

**Value**

list with 3 components:

- **x, y** vectors of x- and y- coordinates of output grid, the same as the input argument \( x_0 \) or \( y_0 \), 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 \( x_0[i], y_0[j] \). z has dimensions \( \text{length}(x_0) \times \text{length}(y_0) \).

**Note**

*interp* is a wrapper for the two versions *interp.old* (it uses (almost) the same Fortran code from Akima 1978 as the S-Plus version) and *interp.new* (it is based on new Fortran code from Akima 1996). For linear interpolation the old version is chosen, but spline interpolation is done by the new version.

Earlier versions (pre 0.5-1) of *interp* 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.

The resulting structure is suitable for input to the functions *contour* and *image*. Check the requirements of these functions when choosing values for \( x_0 \) and \( y_0 \).

**References**


**See Also**

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

**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)
image (akima.li, add=TRUE)
contour(akima.li, add=TRUE)
points (akima, pch = 3)
```

```
# increase smoothness (using finer grid):
akima.smooth <-
with(akima, interp(x, y, z, xo=seq(0,25, length=100),
                  yo=seq(0,25, length=100)))
image (akima.smooth, main = "interp(<akima data>, *) on finer grid")
```
contour(akima.smooth, add = TRUE, 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]))
image(akima.part)
title("interp() on subset of only 15 points")
contour(akima.part, add=TRUE)
points(akima$x[1:15], akima$y[1:15], col = "blue")

## spline interpolation, two variants (AMS 526 "Old", AMS 761 "New")
## -----------------------------------------------
## "Old": use 5 points to calculate derivatives -> many NAs
akima.so <- interp.old(akima$x, akima$, akima$,
  xo=seq(0,25, length=100), yo=seq(0,20, length=100), ncp=5)
table(is.na(akima.so$z)) ## 3990 NA's; 40 %
akima.so <- with(akima,
  interp.old(x,y,z, xo=seq(0,25, length=100), yo=seq(0,20, len=100), ncp = 4))
sum(is.na(akima.so$z)) ## still 3429
image (akima.so, main = "interp.old(*, ncp = 4) [almost useless]")
contour(akima.so, add = TRUE)

## "New:"
akima.spl <- with(akima, interp.new(x,y,z, xo=seq(0,25, length=100),
  yo=seq(0,20, length=100)))
## equivalent call via setting linear=FALSE in interp():
akima.spl <- with(akima, interp(x,y,z, xo=seq(0,25, length=100),
  yo=seq(0,20, length=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!

## example with duplicate points :
data(airquality)
air <- subset(airquality,
  !is.na(Temp) & !is.na(Ozone) & !is.na(Solar.R))
# gives an error (duplicate ..):
try( air.ip <- interp(air$Temp, air$Solar.R, air$Ozone, linear=FALSE) )
# use mean of duplicate points:
air.ip <- with(air, interp(Temp, Solar.R, log(Ozone), duplicate = "mean",
linear = FALSE))
image(air.ip, main = "Airquality: Ozone vs. Temp and Solar.R")
with(air, points(Temp, Solar.R))

interp2xyz FROM interp() Result, Produce 3-column Matrix

Description
From an \texttt{interp()} result, produce a 3-column matrix or \texttt{data.frame}\ cbind(x, y, z).

Usage
\texttt{interp2xyz(\textit{al}, \textit{data.frame} = \texttt{FALSE})}

Arguments
\begin{itemize}
\item \texttt{al} \hspace{1cm} \texttt{a list} as produced from \texttt{interp()}.
\item \texttt{data.frame} \hspace{1cm} \texttt{logical} indicating if result should be \texttt{data.frame} or matrix (default).
\end{itemize}

Value
a matrix (or \texttt{data.frame}) with three columns, called "x", "y", "z".

Author(s)
Martin Maechler, Jan.18, 2013

See Also
\texttt{expand.grid()} is the “essential ingredient” of \texttt{interp2xyz()}.
\texttt{interp}.

Examples
\begin{verbatim}
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)) )
\end{verbatim}
**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**

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

**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`-coordinates of data points. Missing values are not accepted.
- `xo` vector of `x`-coordinates of points at which to evaluate the interpolating function.
- `yo` vector of `y`-coordinates of points at which to evaluate the interpolating function.
- `linear` logical – indicating whether linear or spline interpolation should be used. superseded old `ncp` parameter
- `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 $x_i$, $y_i$.

**NOTE**

Use interp if interpolation on a regular grid is wanted.

The two versions interp.old and interp.new refer to Akimas Fortran code from 1978 and 1996 resp. The call wrapper interp chooses interp.old for linear and interp.new for cubic spline interpolation.

Earlier versions (pre 0.5-1) of interp 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**


**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<-interp(akima$x, akima$y, akima$z, c(1,5,10), c(2,6,12))

akima.lip$z

# spline interpolation
akima.sip<-interp(akima$x, akima$y, akima$z, c(1,5,10), c(2,6,12),
 linear=FALSE)

akima.sip$z
Index

*Topic arith
  aspline, 4

*Topic datasets
  akima, 2
  akima760, 3

*Topic dplot
  aspline, 4
  bicubic, 6
  bicubic.grid, 8
  interp, 9
  interpp, 14

*Topic manip
  interp2xyz, 13

akima, 2
akima760, 3
approx, 5, 11
approxfun, 15
aspline, 4, 11, 15
bicubic, 6, 9
bicubic.grid, 7, 8
contour, 9, 11, 15
data.frame, 13
expand.grid, 11, 13, 15
image, 9, 11, 15
interp, 7, 9, 9, 13, 15
interp2xyz, 13
interpp, 14
list, 13
outer, 11, 15
spline, 5, 11
splinefun, 15
xy.coords, 4