Package: fields (via r-universe)

October 14, 2026

Version 18.0

Date 2026-10-12

Title Tools for Spatial Data

Maintainer Douglas Nychka <douglasnychka@gmail.com>

Description For curve, surface and function fitting with an emphasis on splines, spatial data, geostatistics, and spatial statistics. The major methods include cubic, and thin plate splines, Kriging, and compactly supported covariance functions for large data sets. The splines and Kriging methods are supported by functions that can determine the smoothing parameter (nugget and sill variance) and other covariance function parameters by cross validation and also by restricted maximum likelihood. For Kriging there is an easy to use function that also estimates the correlation scale (range parameter). A major feature is that any covariance function implemented in R and following a simple format can be used for spatial prediction. There are also many useful functions for plotting and working with spatial data as images. This package also contains an implementation of sparse matrix methods for large spatial data sets and currently requires the sparse matrix (spam) package. Use help(fields) to get started and for an overview. All graphics functions focus on using base R graphics. The fields source code is deliberately commented and provides useful explanations of numerical details as a companion to the manual pages. The commented source code can be viewed by expanding the source code version and looking in the R subdirectory. The reference for fields can be generated by the citation function in R and has DOI <doi:10.5065/D6W957CT>. Development of this package was supported in part by the National Science Foundation Grant 1417857, the National Center for Atmospheric Research, and Colorado School of Mines. See the Fields URL for a vignette on using this package and some background on spatial statistics.

License GPL (>= 2)

URL https://github.com/dnychka/fieldsRPackage

Depends R (>= 4.0.0), methods, spam, viridisLite, RColorBrewer

Imports maps

Suggests mapproj

NeedsCompilation yes

Author Douglas Nychka [aut, cre], Reinhard Furrer [aut], John Paige [aut], Stephan Sain [aut], Florian Gerber [aut], Matthew Iverson [aut], Rider Johnson [aut]

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

Date/Publication 2026-10-13 20:10:02 UTC

RemoteUrl https://github.com/cran/fields

RemoteRef HEAD

RemoteSha 8f067097b72335d49deb8782127bf917a1229102

Contents

add.image . . . 4
arrow.plot . . . 5
as.image . . . 6
as.surface . . . 8
AudiA4 . . . 10
BD . . . 10
bplot . . . 11
bplot.xy . . . 13
Chicago ozone test data . . . 14
circulantEmbedding . . . 15
CO2 . . . 18
Colorado Monthly Meteorological Data . . . 20
colorbar.plot . . . 21
compactToMat . . . 23
Covariance functions . . . 25
CovarianceUpper . . . 34
cover.design . . . 35
drape.plot . . . 41
envelopePlot . . . 44
Exponential, Matern, Radial Basis . . . 45
fields . . . 47
fields testing scripts . . . 51
fields-stuff . . . 52
fields.grid . . . 54
fields.hints . . . 55
flame . . . 58
glacier . . . 59
grid list . . . 61

image.cov . . . 65
image.plot . . . 69
image.smooth . . . 78
image2lz . . . 81
imagePlot . . . 83
interp.surface . . . 95
Krig . . . 97
Krig.Amatrix . . . 105
Krig.null.function . . . 107
Krig.replicates . . . 108
KrigFindLambda . . . 109
lennon . . . 111
LENSExample . . . 112
minitri . . . 113
mKrig . . . 113
mKrigMLE . . . 122
naturalSplineBasis . . . 128
NorthAmericanRainfall . . . 130
offGridWeights . . . 132
ozone2 . . . 136
panelPlot . . . 137
plot.Krig . . . 145
plot.surface . . . 146
poly.image . . . 147
predict.Krig . . . 150
predictSE . . . 153
predictSurface . . . 155
print.Krig . . . 160
pushpin . . . 161
qsreg . . . 162
QTps . . . 164
quantilePlot . . . 169
quilt.plot . . . 171
rat.diet . . . 175
RCMexample . . . 176
rdist . . . 177
rdist.earth . . . 179
registeringCode . . . 181
ribbon.plot . . . 182
RMprecip . . . 183
set.panel . . . 186
sim.spatialProcess . . . 187
smooth.2d . . . 194
spam2lz . . . 196
spatialProcess . . . 198
splint . . . 209
sreg . . . 211
stats . . . 215

stats.bin . . . 216
summary.Krig . . . 218
summary.ncdf . . . 219
supportsArg . . . 219
surface.Krig . . . 220
The Engines: . . . 222
tim.colors . . . 226
Tps . . . 229
transformx . . . 238
US . . . 239
US.dat . . . 240
vgram . . . 240
vgram.matrix . . . 243
Wendland . . . 244
world . . . 247
WorldBankCO2 . . . 248
xline . . . 249
yline . . . 250

Index **251**

`add.image` *Adds an image to an existing plot.*

Description

Adds an image to an existing plot. Simple arguments control the location and size.

Usage

```
add.image(xpos, ypos, z, adj.x = 0.5, adj.y = 0.5,
image.width = 0.15, image.height = NULL, col = tim.colors(256), ...)
```

Arguments

`xpos`	X position of image in user coordinates
`ypos`	Y position of image in user coordinates
`z`	Matrix of intensities comprising the image.
`adj.x`	Location of image relative to x coordinate. Most common values are .5 (centered), 0 (left side of image at x) and 1 (right side of image at x). These are the same conventions that are used for `adj` in positioning text.
`adj.y`	Location of image relative to y coordinate. Same rules as `adj.x`
`image.width`	Width of image as a fraction of the plotting region in horizontal direction.
`image.height`	Height of image as a fraction of the plotting region in vertical direction. If NULL, height is scaled to make image pixels square.
`col`	Color table for image. Default is tim.colors.
`...`	Any other plotting arguments that are passed to the image function

See Also

image.plot, colorbar.plot, image, tim.colors

Examples

```
plot( 1:10, 1:10, type="n")
data( lennon)

add.image( 5,4,lennon, col=grey( (0:256)/256))
# reference lines
xline( 5, col=2)
yline( 4,col=2)

#
# add lennon right in the corner beyond the plotting region
#

par(new=TRUE, plt=c(0,1,0,1), mar=c(0,0,0,0), usr=c(0,1,0,1))
add.image( 0,0, lennon, adj.x=0, adj.y=0)
```

---

arrow.plot *Adds arrows to a plot*

Description

Adds arrows at specified points where the arrow lengths are scaled to fit on the plot in a reasonable manner. A classic use of this function is to depict a vector field. At each point (x,y) we have a vector with components (u,v). Like the arrows function, this adds arrows to an existing plot.

Usage

```
arrow.plot(a1, a2, u = NA, v = NA, arrow.ex = 0.05,
     xpd = TRUE, true.angle = FALSE, arrowfun=arrows,...)
```

Arguments

a1	The x locations of the tails of the arrows or a 2 column matrix giving the x and y coordinates of the arrow tails.
a2	The y locations of the tails of the arrows or a 2 column matrix giving the u and v coordinates of the arrows.
u	The u components of the direction vectors if they are not specified in the a1 argument
v	The v components of the direction vectors if they are not specified in the a2 argument
arrow.ex	Controls the length of the arrows. The length is in terms of the fraction of the shorter axis in the plot. So, with a default of .05, 20 arrows of maximum length can line up end to end along the shorter axis.

xpd	If TRUE, does not clip arrows to fit inside the plot region, default is not to clip.
true.angle	If TRUE, preserves the true angle of the (u,v) pair on the plot. E.g., if (u,v)=(1,1), then the arrow will be drawn at 45 degrees.
arrowfun	The actual arrow function to use. The default is standard R arrows. However, Tamas K Papp suggests p.arrows from sfsmisc, which makes prettier arrows.
...	Graphics arguments passed to the arrows function that can change the color or arrow sizes. See help on this for details.

Details

This function is useful because (u,v) may be in very different scales from the locations (x,y). So, some careful scaling is needed to plot the arrows. The only tricky thing about this function is whether you want the true angles on the plot. For overlaying a vector field on top of contours that are the streamlines, true.angle should be FALSE. In this case you want u and v to be scaled in the same way as the x and y variables. If the scaling is not the same, then the arrows will not look like tangent vectors to the streamlines. An application where the absolute angles are meaningful might be the hands of a clock showing different time zones on a world map. Here true.angle=T is appropriate; the clock hands should preserve the right angles.

See Also

arrows

Examples

```
#
# 20 random directions at 20 random points

x<- runif( 20)
y<- runif( 20)
u<- rnorm( 20)
v<- rnorm( 20)
plot( x,y)
arrow.plot( x,y,u,v) # a default that is unattractive

plot( x,y, type="n")
arrow.plot( x,y,u,v, arrow.ex=.2, length=.1, col='green', lwd=2)
# thicker lines in green, smaller heads and longer tails. Note length, col and lwd are
# options that the arrows function itself knows about.
```

---

as.image *Creates image from irregular x,y,z*

Description

Discretizes a set of 2-d locations to a grid and produces an image object with the z values in the right cells. For cells with more than one Z value, the average is used.

Usage

```
as.image(Z, ind=NULL, grid=NULL, x=NULL,weights=rep(1, length(Z)),
 na.rm=FALSE, nx=64, ny=64, boundary.grid=FALSE,  nrow=NULL, ncol=NULL,
 FUN = NULL)
```

Arguments

Z	Values of image.
ind	A matrix giving the row and column subscripts for each image value in Z. (Not needed if x is specified.)
grid	A list with components x and y of equally spaced values describing the centers of the grid points. The default is to use nrow and ncol and the ranges of the data locations (x) to construct a grid.
x	Locations of image values. Not needed if ind is specified.
nrow	Same as nx; this is deprecated.
ncol	Same as ny; this is deprecated.
weights	If two or more values fall into the same pixel, a weighted average is used to represent the pixel value. Default is equal weights.
na.rm	If TRUE, NAs are removed from the Z vector.
nx	Number of grid points in X coordinate.
ny	Number of grid points in Y coordinate.
boundary.grid	If FALSE, grid points are assumed to be the grid midpoints. If TRUE they are the grid box boundaries.
FUN	The function to apply to common values in a grid box. The default is a mean (or weighted mean). If FUN is specified, the weights are not used.

Details

The discretization is straightforward once the grid is determined. If two or more Z values have locations in the same cell, the weighted average value is taken as the value. The weights component that is returned can be used to account for means that have different numbers (or precisions) of observations contributing to the grid point averages. The default weights are taken to be one for each observation. See the source code to modify this to get more information about coincident locations. (See the call to fast.1way.)

Value

A list in image format with a few more components. Components x and y are the grid values, z is a nrow X ncol matrix with the Z values. NAs are placed at cell locations where Z data has not been supplied. Component ind is a 2 column matrix with subscripts for the locations of the values in the image matrix. Component weights is an image matrix with the sum of the individual weights for each cell. If no weights are specified, the default for each observation is one and so the weights will be the number of observations in each bin.

See Also

image.smooth, image.plot, Krig.discretize, Krig.replicates

Examples

```
# convert precip data to 50X50 image
data(RMprecip)
look<- as.image( RMprecip$y, x= RMprecip$x, nx=50, ny=50)
image.plot( look)

# reduced grid extent compared to the domain
gridList<- list( x = seq(-105,-101,length.out=10),
                 y = seq(  38, 42,length.out=10) )
look2<- as.image( RMprecip$y, x= RMprecip$x,grid=gridList)
image.plot( look2)

# number of obs in each cell -- in this case equal to the
# aggregated weights because each obs had equal weight in the call

image.plot( look$x ,look$y, look$weights, col=terrain.colors(50))
# hot spot is around Denver
```

---

as.surface *Creates a "surface" object from grid values.*

Description

Reformats the vector from evaluating a function on a grid of points into a list for use with the surface plotting function. The list has the usual components x,y and z and is suitable for use with persp, contour, image and image.plot.

Usage

```
as.surface(obj, z, location=NULL, order.variables="xy")
```

Arguments

obj	A description of the grid used to evaluate the function. This can either be in the form of a grid.list (see help file for grid.list) or the matrix of grid of points produced by make.surface.grid. In the latter case, obj is a matrix with the grid.list as an attribute.
z	The value of the function evaluated at the gridded points.
location	A logical or two column matrix of indices indicating the location of the z values within the image matrix.
order.variables	Either "xy" or "yx" specifies how the x and y variables used to evaluate the function are matched with the x and y grids in the surface object.

Details

This function was written simply to go back and forth between a matrix of gridded values and the stacked vector obtained by stacking columns. The main application is evaluating a function at each grid point and then reforming the results for plotting. (See example below.)

If zimage is a matrix of values, then the input vector is c(zimage). To go from the stacked vector to the matrix, one needs the nrow and ncol, which explains why grid information must also be specified.

Note that the z input argument must be in the order of stacking the columns of the image. This is also the order of the grid points generated by make.surface.grid.

To convert irregular 2-d data to a surface object where there are missing cells, see the function as.image.

Value

A list of class surface. This object is a modest generalization of the list input format (x,y,z) for the S functions contour, image or persp.

x	The grid values in the X-axis
y	The grid values in the Y-axis
z	A matrix of dimensions nrow= length of x and ncol= length of y with entries being the grid point value reformatted from z.

See Also

grid.list, make.surface.grid, surface, contour, image.plot, as.image

Examples

```
# Make a perspective of the surface Z= X**2 -Y**2
# Do this by evaluating quadratic function on a 25 X 25 grid

grid.l<-list( abcissa= seq( -2,2,,15), ordinate= seq( -2,2,,20))
xg<-make.surface.grid( grid.l)
# xg is a 300X2 matrix that has all pairs of X and Y grid values
z<- xg[,1]**2 - xg[,2]**2
# now fold z in the matrix format needed for persp
out.p<-as.surface( xg, z)
persp( out.p)
# also try  plot( out.p) to see the default plot for a surface object
```

AudiA4 *Used car prices for AudiA4 from cars.com*

Description

These data are a small regression data set that was collected in the process of trying to find a "deal" on a used Audi A4. (Note that a good deal could be taken as a car where the regression residual is very negative.)

Usage

```
data(AudiA4)
```

Format

This is a 1565X4 matrix with the rows being individual cars listed for sale on cars.com and the columns being the covariables:

"year" "price" "mileage" "distance"

- year: Model year
- price: Price in dollars
- mileage: Mileage.
- distance: Distance of car from Boulder, Colorado

Examples

```
data(AudiA4)
fields.style()
plot( AudiA4$mileage/1000,AudiA4$price/1000, pch=16,
xlab="Mileage (1000s)",
ylab="Price (1000s)", col="grey" )
fit<- Tps(AudiA4$mileage/1000,AudiA4$price/1000 )
mileageGrid<-seq(0, 1700 )
priceHat<- predict( fit, mileageGrid)
lines(mileageGrid,priceHat, col="orange3"  )
```

BD *Effect of buffer compositions on DNA strand displacement amplification.*

Description

A 4-d regression data set with replication. This is a useful test data set for function fitting methods. BD as a data frame has 89 rows and 5 columns. There are 89 runs with four buffer components (KCl, MgCl2, KPO4, dNTP) systematically varied in a space-filling design. The response is the DNA amplification rate.

Usage

```
data(BD)
```

Format

This data frame contains the following columns:

KCl Buffer component.

MgCl2 Buffer component.

KPO4 Buffer component.

dNTP Buffer component, deoxyribonucleotides.

lnya Exponential amplification rate on a log scale, i.e. the actual amplification rate.

Source

Thanks to Perry Haaland and Michael O'Connell for these data. at Becton Dickinson Research Center, Research Triangle Park, NC

See Also

Tps, spatialProcess

Examples

```
# fitting a DNA strand
# displacement amplification  surface to various buffer compositions
data(BD)
fit<- Tps(BD[,1:4],BD$lnya,scale.type="range")
surface(fit)  # plots fitted surface and contours
```

`bplot` *boxplot*

Description

Plots boxplots of several groups of data and allows for placement at different horizontal or vertical positions or colors. It is also flexible in the input object, accepting either a list or matrix.

Usage

```
bplot(x, by, pos = NULL, at = pos, add = FALSE, axes = !add,
                  boxwex = 0.8, xlim = NULL, ...)
```

Arguments

x	Vector, matrix, list or data frame. A vector may be divided according to the by argument. Matrices and data frames are separated by columns and lists by components.
by	If x is a vector, an optional vector (either character or numerical) specifying the categories to divide x into separate data sets. Boxplots are then made for each group.
pos	The boxplots will be plotted vertically (horizontally) and pos gives the x (y) locations for their centers. If omitted, the boxes are equally spaced at integer values. This is the same as `at` in the `boxplot` function
at	Same as `pos`; this is the name for this argument in the standard `boxplot` function.
add	If TRUE, do not create a new plot; just add the boxplots to a current plot. Note that the pos argument may be useful in this case and should be in the user coordinates of the parent plot.
axes	Add axes?
boxwex	A boxplot argument to control the width of the boxplot. It behaves a little differently than as an argument passed directly to `boxplot`. To make this a general function it is useful to scale this according to the size of the positions. Within bplot this happens as `boxwex<- boxwex* min(diff( sort( at)))` and then the scaled version of `boxwex` is now passed to `boxplot`.
xlim	Same as the usual argument used in plotting. The plotting limits for the x axis.
...	Other arguments to be passed to the boxplot function; some handy favorites are: `names` Labels for each boxplot. `horizontal` If TRUE, draw boxplots horizontally. The default is FALSE, producing vertical box plots. `lwd` Width(s) of lines in box plots. `col` Color(s) of bplots. See `colors()` for some choices.

Details

This function was created as a complement to the usual S/R function for boxplots. The current function makes it possible to put the boxplots at unequal x or y positions in a rational way using the `at` or `pos` arguments. This is useful for visually grouping a large set of boxplots into several groups. Also, placement of the boxplots with respect to the axis can add information to the plot. Another aspect is the emphasis on data structures for groups of data. One useful feature is the by option to break up the x vector into distinct groups.

Use `axis(3)` (`axis(4)`) to add an axis along the top (right side) or omit the category names and draw on the bottom `axis(1)` (left side `axis(2)`).

The older `bplot` function drew the boxplots from scratch, and if one needs to do this refer to the old functions: `describe.bplot`, `draw.bplot.obj`, `bplot.xy`, `bplot.obj`

Finally, to bin data into groups based on a continuous variable and to make bplots of each group see `bplot.xy`.

See Also

bplot.xy

Examples

```
#
set.seed(123)
temp<- matrix( rnorm(12*8), ncol=12)
pos<- c(1:6,9, 12:16)*100
bplot(temp)
#
par(las=2)
bplot( temp, pos=pos, names=paste( "Data",1:12, sep=""))
# add an axis along top for reference
axis(3)

#
# Xmas boxplots in pleasing red and green
bplot( temp, pos=pos,  col=c("red4", "green4"))
# add an axis on top
axis( 3)
```

`bplot.xy` *Boxplots for conditional distribution*

Description

Draws boxplots for y by binning on x. This gives a coarse, but quick, representation of the conditional distribution of [Y|X] in terms of boxplots.

Usage

```
bplot.xy(x, y, N = 10, breaks = NULL, plot = TRUE, add = FALSE,
                  axes = TRUE, ...)
```

Arguments

`x`	Vector to use for bin membership
`y`	Vector to use for constructing boxplot statistics.
`N`	Number of bins on x. Default is 10.
`breaks`	Break points defining bin boundaries. These can be unequally spaced. If NULL, they are found by `pretty(x, N, eps.correct = 1)`.
`plot`	If FALSE, just returns a list with the statistics used for plotting the box plots, bin centers, etc. – More stuff than you can imagine!
`add`	Just add this to an existing plot.
`axes`	The usual plotting argument – if TRUE, axes are plotted.
`...`	Any other optional arguments passed to the standard `boxplot` function.

See Also

bplot, draw.bplot

Examples

```
# condition on swim times to see how run times vary
data(minitri)
bplot.xy( minitri$swim, minitri$run, N=5)

# bivariate normal corr= .8
set.seed( 123)
x<-rnorm( 2000)
y<- .8*x +  sqrt( 1- .8**2)*rnorm( 200)
#
bplot.xy(x,y)
#
bplot.xy( x,y, breaks=seq( -3, 3,,25) ,
                xlim =c(-4,4), ylim =c(-4,4), col="grey80", lwd=2)
points( x,y,col=3, cex=.5)
```

Chicago ozone test data

Data set of ozone measurements at 20 Chicago monitoring stations.

Description

This data set used to be named ozone but was changed to avoid conflict with other packages. The ChicagoO3 data is a list with components x and y. The x component is longitude and latitude position of each of the 20 Chicago monitoring stations; y is the average daily ozone values over the time period 6/3/87-8/30/87. These data are used extensively for the test scripts and simple examples. The scientific value is probably epsilon! See ozone2 for a data set that is more meaningful.

Usage

```
data(ChicagoO3)
```

Format

This data set is a list containing the following components:

lon.lat Longitude-latitude positions of monitoring stations.

x An approximate Cartesian set of coordinates for the locations where the units are in miles. The origin is in the center of the locations.

y Average daily ozone values over 1987 summer.

Source

AIRS, the EPA air quality database.

See Also

Tps, Krig

Examples

```
data(ChicagoO3)
fit<- Tps(ChicagoO3$x, ChicagoO3$y)
# fitting a surface to ozone measurements.
surface( fit, type="I")
```

`circulantEmbedding` *Efficiently simulates stationary 1D and 2D Gaussian random fields*

Description

Simulates a stationary Gaussian random field on a regular grid with unit marginal variance. Makes use of the efficient algorithm based on the FFT known as circulant embedding.

Usage

```
sim.rf(obj)
circulantEmbedding(obj)

circulantEmbeddingSetup(
grid, M = NULL, mKrigObject = NULL, cov.function =
                 "stationary.cov", cov.args = NULL, delta = NULL,
                  giveWarnings = TRUE,...)
```

Arguments

`obj`	A list (aka covariance object) that includes information about the covariance function and the grid for evaluation. Usually this is created by a setup call to Exp.image.cov, stationary.image.cov, matern.image.cov or other related covariance functions for `sim.rf` (see details below) or to `circulantEmbeddingSetup` for `circulantEmbedding`
`giveWarnings`	If TRUE, will warn that the FFT of the circulant embedding has negative values, i.e., the proposed circulant covariance matrix is not positive definite.
`grid`	A list describing the regular grid. `length(grid)` is the dimension of the field (1D, 2D, etc.) and each component gives the regular locations in that dimension.
`M`	A vector of dimensions to embed the field. Simulation will be exact if each `M[i]` is larger than `2*length(grid)`. The default is to choose a power of 2 larger than this minimum bound.
`mKrigObject`	Object from fitting surface using the mKrig or spatialProcess functions.
`cov.function`	A text string with the name of the stationary covariance function to use. Default is `stationary.cov`, a general function that takes advantage of some efficiency in finding distances.

cov.args	A list of arguments to include with the covariance function, e.g., aRange and smoothness for the Matern.
delta	If NULL, the spatial domain is artificially doubled in size in all dimensions to account for the periodic wrapping of the FFT. If passed, this is the amount to extend the domain and can be less than double if a compact covariance function is used.
...	For convenience, any other arguments for the covariance function. Although it is clearer to use either the mKrig/spatialProcess fit or the cov.args list.

Details

The functions circulantEmbedding and circulantEmbeddingSetup are more recent fields functions, easier to read, and recommended over sim.rf. sim.rf is limited to 2D fields while circulantEmbedding can handle any number of dimensions and has some shortcuts to be efficient for the 2D case.

circulantEmbeddingSetup is also the same setup function used to create covObject for fast multiplication of a stationary covariance function with values on a regular, 2D grid (e.g., see stationaryImageCov).

One complication in this code is to take any grid size and enlarge it so that the grid sizes are powers of 2 and 3 (highly composite). This makes the FFT more efficient. (In fact, without any adjustment, grid sizes that are prime numbers will make the FFT quite slow.)

The simulated field has the marginal variance that is determined by the covariance function for zero distance. Within the fields package the exponential and Matern functions set this equal to one (e.g., Matern(0) ==1) so that one simulates a random field with a marginal variance of one. For stationary.cov, the marginal variance is whatever Covariance(0) evaluates to, and we recommend that alternative covariance functions also be normalized so that this is one.

Of course, if one requires a Gaussian field with different marginal variances, one can simply scale the result of this function by the process standard deviations. See the third example below.

Both sim.rf and circulantEmbedding take an object that includes some preliminary calculations and so is more efficient for simulating more than one field from the same covariance.

Adjusting for negative weights

The algorithm using an FFT, known as circulant embedding, may not always work if the correlation range is large. Specifically, the weight function obtained from the FFT of the covariance field will have some negative values. Currently, a lengthy error message is printed if this is the case. A simple fix is to increase the size of the domain so that the correlation scale becomes smaller relative to the extent of the domain. Increasing the size can be computationally expensive, however, and so this method has some limitations. But when it works, it is an exact simulation of the random field.

For a stationary model the covariance object (or list) for circulantEmbedding should have, minimally, the following components. That is, names(obj) should give "m", "grid", "M", "wght"

where m is the number of grid points in each dimension, grid is a list with components giving the grid points in each coordinate. M is the size of the larger grid that is used for "embedding" and simulation. Usually M = 2*m and results in an exact simulation of the stationary Gaussian field. The default, if M is not passed, is to find the smallest power of 2 greater than 2*m. wght is an array from the FFT of the covariance function with dimensions M. Keep in mind that for the final results, only the array that is within the indices 1: m[i] for each dimension i is retained. This can give a much larger intermediate array, however, in the computation. E.g., if m[1] = 100 and m[2]=200, by default M[1] = 256 and M[2] = 512. A 256 X 512 array is simulated to get the 100 by 200 result.

The easiest way to create the object for simulation is to use `circulantEmbeddingSetup`.

For the older function `sim.rf` one uses the image-based covariance functions with `setup=TRUE` to create the list for simulation. See the example below for this usage.

The classic reference for this algorithm is Wood, A.T.A. and Chan, G. (1994). Simulation of Stationary Gaussian Processes in [0,1]^d. Journal of Computational and Graphical Statistics, 3, 409-432. Michael Stein and Tilman Gneiting have also made some additional contributions to the algorithms and theory.

Value

sim.rf: A matrix with the random field values.

circulantEmbedding: An array according to the grid values specified in the setup.

circulantEmbeddingSetup: A list with components

`"m" "grid" "dx" "M" "wght" "call"`

With the information needed to simulate the field.

See Also

stationary.cov, stationary.image.cov, simLocal.spatialProcess

Examples

```
#Simulate a Gaussian random field with an exponential covariance function,
#range parameter = 2.0 and the domain is  [0,5]X [0,5] evaluating the
#field at a 100X100 grid.
  grid<- list( x= seq( 0,5,,100), y= seq(0,5,,100))
  obj<- circulantEmbeddingSetup( grid, Covariance="Exponential", aRange=.5)
  set.seed( 223)
  look<-  circulantEmbedding( obj)
# Now simulate another ...
  look2<- circulantEmbedding( obj)
# take a look at first two
 set.panel(2,1)
 image.plot( grid[[1]], grid[[2]], look)
 title("simulated gaussian fields")
 image.plot( grid[[1]], grid[[2]], look2)
 title("another realization ...")

# Suppose one requires an exponential, range = 2
# but marginal variance = 10 ( sigma in fields notation)
look3<- sqrt( 10)*circulantEmbedding( obj)

## Not run:
# an interesting 3D field

grid<- list(  1:40,  1:40, 1:16  )

obj<- circulantEmbeddingSetup( grid,
                          cov.args=list( Covariance="Matern", aRange=2, smoothness=1.0)
                          )
```

```
# NOTE: choice of aRange is close to giving a negative weight array
set.seed( 122)
look<- circulantEmbedding( obj )
# look at slices in the 3rd dimension
set.panel( 4,4)
zr<- range( look)
par( mar=c(1,1,0,0))
for(  k in 1:16){
image( grid[[1]], grid[[2]], look[,,k], zlim= zr, col=tim.colors(256),
       axes=FALSE, xlab="", ylab="")
}



## End(Not run)


# same as first example using the older sim.rf

grid<- list( x= seq( 0,10,length.out=100) , y= seq( 0,10,length.out=100) )
obj<-Exp.image.cov( grid=grid, aRange=.75, setup=TRUE)
set.seed( 223)
look<- sim.rf( obj)
# Now simulate another ...
look2<- sim.rf( obj)
```

CO2 *Simulated global CO2 observations*

Description

This is an example of a moderately large spatial data set and consists of simulated CO2 concentrations that are irregularly sampled from a lon/lat grid. Also included is the complete CO2 field (CO2.true) used to generate the synthetic observations.

Usage

```
data(CO2)
```

Format

The format of CO2 is a list with two components:

- lon.lat: 26633x2 matrix of the longitude/latitude locations. These are a subset of a larger lon/lat grid (see example below).
- y: 26633 CO2 concentrations in parts per million.

The format of CO2.true is a list in "image" format with components:

- x longitude grid values.
- y latitude grid values.
- z an image matrix with CO2 concentration in parts per million
- mask a logical image that indicates which grid locations were selected for the synthetic data set CO2.

Details

This data was generously provided by Dorit Hammerling and Randy Kawa as a test example for the spatial analysis of remotely sensed (i.e. satellite) and irregular observations. The synthetic data is based on a true CO2 field simulated from a geophysical, numerical model.

Examples

```
## Not run:

data(CO2)
#
# A quick look at the observations with world map
quilt.plot( CO2$lon.lat, CO2$y)
world( add=TRUE)

# Note high concentrations in Borneo (biomass burning), Amazonia and
# ... Michigan (???).

# spatial smoothing using the wendland compactly supported covariance
# see help( fastTps) for details
# First smooth using locations and Euclidean distances
# note taper is in units of degrees
out<-fastTps( CO2$lon.lat, CO2$y, aRange=4, lambda=2.0)
#summary of fit note about 7300 degrees of freedom
# associated with fitted surface
 print( out)
# image plot on a grid  (this takes a while)
surface( out, type="I", nx=300, ny=150)
# smooth with respect to great circle distance
out2<-fastTps( CO2$lon.lat, CO2$y, lon.lat=TRUE,lambda=1.5, aRange=4*68)
print(out2)
#surface( out2, type="I", nx=300, ny=150)

# these data are actually subsampled from a grid.
# create the image object that holds the data
#

temp<- matrix( NA, ncol=ncol(CO2.true$z), nrow=nrow(CO2.true$z))
temp[ CO2.true$mask] <- CO2$y

# look at gridded object.
 image.plot(CO2.true$x,CO2.true$y, temp)

# to predict _exactly_ on this grid for the second fit;
```

```
# (this takes a while)
look<- predictSurface( out2, list( x=CO2.true$x, y=CO2.true$y) )
image.plot(look)


## End(Not run)
```

Colorado Monthly Meteorological Data

Monthly surface meteorology for Colorado 1895-1997

Description

Source: This is a group of R data sets for monthly min/max temperatures and precipitation over the period 1895-1997. It is a subset extracted from the more extensive US data record. Temperature is in degrees C and precipitation is total monthly accumulation in millimeters. Note that minimum (maximum) monthly temperature is the mean of the daily minimum (maximum) temperatures.

Data domain:

A rectangular lon/lat region [-109.5,-101]x [36.5,41.5] larger than the boundary of Colorado comprises approximately 400 stations. Although there are additional stations reported in this domain, stations that only report precipitation or only report temperatures have been excluded. In addition, stations that have mismatches between locations and elevations from the two metadata files have also been excluded. The net result is 367 stations that have colocated temperatures and precipitation.

Format

This group of data sets is organized with the following objects:

CO.info A data frame with columns: station id, elev, lon, lat, station name

CO.elev elevation in meters

CO.elevGrid An image object being elevation in meters on a 4 km grid covering Colorado.

CO.id alphanumeric station id codes

CO.loc locations in lon/lat

CO.Grid Just the grid.list used in the CO.elevGrid.

CO.ppt CO.tmax CO.tmin Monthly means as three dimensional arrays (Year, Month, Station). Temperature is in degrees C and precipitation in total monthly accumulation in millimeters.

CO.ppt.MAM CO.tmax.MAM CO.tmin.MAM Spring seasonal means (March, April, May) as two dimensional arrays (Year, Station).

CO.MAM.ppt.climate CO.MAM.tmax.climate CO.MAM.tmin.climate Spring seasonal means (March, April, May) by station for the period 1960-1989. If fewer than 15 years are present over this period, an NA is recorded. No detrending or other adjustments have been made for these mean estimates.

Examples

```
data(COmonthlyMet)

#Spatial plot of 1997 Spring average daily maximum temps
 quilt.plot( CO.loc,CO.tmax.MAM[103,]  )
 US( add=TRUE)
 title( "Recorded MAM max temperatures (1997)")

# min and max temperatures against elevation

matplot( CO.elev, cbind( CO.tmax.MAM[103,], CO.tmin.MAM[103,]),
  pch="o", type="p",
  col=c("red", "blue"), xlab="Elevation (m)", ylab="Temperature (C)")
title("Recorded MAM max (red) and min (blue) temperatures 1997")

#Fitting a spatial model:
obj<- Tps(CO.loc,CO.tmax.MAM.climate,  XMat= CO.elev )
## Not run:
out<- spatialProcess(CO.loc,CO.tmax.MAM.climate,
          smoothness=1.0, XMat= CO.elev)
surface( out)

## End(Not run)
```

---

colorbar.plot *Adds color scale strips to an existing plot.*

Description

Adds one or more color scales in a horizontal orientation, or vertical orientation, to an existing plot.

Usage

```
colorbar.plot(x, y, strip, strip.width = 0.1, strip.length = 4 * strip.width,
zrange = NULL, adj.x = 0.5, adj.y = 0.5, col = tim.colors(256),
horizontal = TRUE, ...)
```

Arguments

x	x position of strip in user coordinates
y	y position of strip in user coordinates
strip	Either a vector or matrix giving the values of the color strip(s). If a matrix, then strips are assumed to be the columns.
strip.width	Width of strip as a fraction of the plotting region.

strip.length	Length of strip as a fraction of the plotting region. Default is a pleasing 8 times width.
zrange	If a vector, these are the common limits used for assigning the color scale. Default is to use the range of values in strip. If a two column matrix, rows are used as the limits for each strip.
adj.x	Location of strip relative to x coordinate. Most common values are .5 (centered), 0 (right end at x) and 1 (left end at x). These are the same conventions that are used for adj in positioning text.
adj.y	Location of strip relative to y coordinate. Same rules as adj.x
col	Color table used for strip. Default is our favorite tim.colors being a scale from a dark blue to dark red.
horizontal	If TRUE, draws strips horizontally. If FALSE, strips are drawn vertically.
...	Optional graphical arguments that are passed to the image function.

Details

This function draws the strips as a sequence of image plots added to the existing plot. The main work is in creating a grid (x,y) for the image that makes sense when superimposed on the plot. Note that although the columns of strip are considered as separate strips, these can be oriented either horizontally or vertically based on the value of horizontal. The rows of zrange are essentially the zlim argument passed to the image function when each strip is drawn.

Don't forget to use locator to interactively determine positions. text can be used to label points neatly in conjunction with setting adj.x and adj.y. Although this function is inefficient for placing images at arbitrary locations on a plot, the code can be easily adapted to do this.

This function was created to depict univariate posterior distributions on a map. The values are quantiles of the distribution and the strips, when added under a common color scale, give an overall impression of location and scale for several distributions.

Author(s)

Doug Nychka

See Also

image.plot, arrow.plot, add.image

Examples

```
# set up a plot but don't plot points  and no "box"
plot( 1:10, (1:10)*10, type="n", bty="n")
# of course this could be anything

y<- cbind( 1:15, (1:15)+25)

colorbar.plot( 2.5, 30, y)
points( 2.5,30, pch="+", cex=2, adj=.5)
# note that strip is still in 1:8 aspect even though plot has very
# different ranges for x and y.
```

```
# adding legend using image.plot
zr<- range( c( y))
image.plot( legend.only=TRUE, zlim= zr)
# see help(image.plot) to create more room in margin etc.

zr<- rbind( c(1,20), c(1,100)) # separate ranges for columns of y.
colorbar.plot( 5, 70, y, adj.x=0, zrange= zr)
# some reference lines to show placement
xline( 5, lty=2) # strip starts at x=5
yline(70, lty=2)  # strip is centered around y=7 (because adj.y=.5 by default)

# many strips on common scale.

y<- matrix( 1:200, ncol=10)
colorbar.plot( 2, 75, y, horizontal=FALSE, col=rainbow(256))

# Xmas strip
y<- cbind( rep( c(1,2),10))
y[15] <- NA # NA's should work
colorbar.plot( 6, 45, y, adj.y=1,col=c("red", "green"))
text(6,48,"Christmas strip", cex=2)

# lennon thumbnail
# there are better ways to this ... see add.image for example.
data( lennon)
colorbar.plot( 7.5,22, lennon,
           strip.width=.25, strip.length=.25, col=grey(seq( 0,1,,256)))
```

---

compactToMat *Convert Matrix from Compact Vector to Standard Form*

Description

compactToMat transforms a matrix from compact, vector form to a standard matrix. Only symmetric matrices can be stored in this form, since a compact matrix is stored as a vector with elements representing the upper triangle of the matrix. This function assumes the vector does not contain diagonal elements of the matrix.

An example of a matrix stored in compact form is any matrix generated from the rdist function with compact=TRUE.

Usage

```
compactToMat(compactMat, diagVal=0, lower.tri=FALSE, upper.tri=TRUE)
```

Arguments

compactMat	A symmetric matrix stored as a vector containing elements for the lower-triangular portion of the true matrix (and none of the diagonal elements), as returned by rdist with compact=TRUE.
diagVal	A number to put in the diagonal entries of the output matrix.
lower.tri	Whether or not to fill in the upper triangle of the output matrix
upper.tri	Whether or not to fill in the lower triangle of the output matrix

Value

The standard form matrix represented by the input compact matrix

Author(s)

John Paige

See Also

rdist, dist

Examples

```
################
#Calculate distance matrix from compact form:
################

#make a distance matrix
distOut = rdist(1:5, compact=TRUE)
print(distOut)

#note that distOut is in compact form:
print(c(distOut))

#convert to standard matrix form:
distMat = compactToMat(distOut)

################
#fast computation of covariance matrix:
################

#generate 5 random points on [0,1]x[0,1] square
x = matrix(runif(10), nrow=5)

#get compact distance matrix
distOut = rdist(x, compact=TRUE)

#evaluate Exponential covariance with range=1.  Note that
#Covariance function is only evaluated over upper triangle
#so time is saved.
diagVal = Exponential(0, range=1)
```

```
compactCovMat = Exponential(distOut, range=1)
upperCovMat = compactToMat(compactCovMat, diagVal)
lowerCovMat = compactToMat(compactCovMat, diagVal, lower.tri=TRUE, upper.tri=FALSE)
fullCovMat = compactToMat(compactCovMat, diagVal, lower.tri=TRUE, upper.tri=TRUE)
compactCovMat
lowerCovMat
upperCovMat
fullCovMat
```

`Covariance functions` *Exponential family, radial basis functions, cubic spline, compactly supported Wendland family, stationary covariances and non-stationary covariances.*

Description

Given two sets of locations, these functions compute the cross covariance matrix for some covariance families. In addition, these functions can take advantage of sparseness, implement more efficient multiplication of the cross covariance by a vector or matrix, and also return a marginal variance to be consistent with calls by the Krig function.

`stationary.cov` and `Exp.cov` have additional arguments for precomputed distance matrices and for calculating only the upper triangle and diagonal of the output covariance matrix to save time. Also, they support using the `rdist` function with `compact=TRUE` or input distance matrices in compact form, where only the upper triangle of the distance matrix is used to save time.

Note: These functions have been renamed from the previous fields functions using 'Exp' in place of 'exp' to avoid conflict with the generic exponential function (`exp(...)`) in R.

Usage

```
Exp.cov(x1, x2 = NULL, aRange = 1, p = 1, distMat = NA, C =
                 NA, marginal = FALSE, onlyUpper = FALSE, theta = NULL,
                 ...)

Exp.simple.cov(x1, x2, aRange =1, C=NA,marginal=FALSE, theta=NULL)

Rad.cov(x1, x2, p = 1, m=NA, with.log = TRUE, with.constant = TRUE,
               C=NA,marginal=FALSE, derivative=0)

cubic.cov(x1, x2, aRange = 1, C=NA, marginal=FALSE, theta=NULL)

Rad.simple.cov(x1, x2, p=1, with.log = TRUE, with.constant = TRUE,
               C = NA, marginal=FALSE)

stationary.cov(x1, x2=NULL, Covariance = "Exponential", Distance = "rdist",
  Dist.args = NULL, aRange = 1, V = NULL, C = NA, marginal = FALSE,
  derivative = 0, distMat = NA, onlyUpper = FALSE, theta=NULL, ...)
```

```
stationary.taper.cov(x1, x2, Covariance="Exponential",
           Taper="Wendland",
           Dist.args=NULL, Taper.args=NULL,
           aRange=1.0,V=NULL, C=NA, marginal=FALSE,
           spam.format=TRUE,verbose=FALSE, theta=NULL,...)

Tps.cov(x1, x2 = NULL, cardinalX, mDegree=2,
                  C = NA, aRange=NA,
                  marginal = FALSE
         )

wendland.cov(x1, x2, aRange = 1, V=NULL, k = 2, C = NA,
             marginal =FALSE,Dist.args = list(method = "euclidean"),
             spam.format = TRUE, derivative = 0, verbose=FALSE, theta=NULL)

Paciorek.cov(x1,
                           x2 = NULL,
                           Distance = "rdist",
                           Dist.args = NULL,
                           aRangeObj = 1,
                           rhoObj = NULL,
                           C = NA,
                           marginal = FALSE,
                           smoothness = .5)
```

Arguments

x1	Matrix of first set of locations where each row gives the coordinates of a particular point.
x2	Matrix of second set of locations where each row gives the coordinates of a particular point. If this is missing, x1 is used.
aRange	Range (or scale) parameter. This should be a scalar (use the V argument for other scaling options). Any distance calculated for a covariance function is divided by aRange before the covariance function is evaluated. For Tps.cov this argument is ignored.
aRangeObj	A fit object that defines the Range (or scale) parameter for arbitrary locations using the generic predict function.
theta	Old version of the aRange parameter. If passed, it will be copied to aRange.
cardinalX	Locations added to the thin plate radial function to make it positive definite (See Details below).
C	A vector or matrix with the same number of rows as x2. If specified, the covariance matrix will be multiplied by this vector/matrix.
Covariance	Character string that is the name of the covariance shape function for the distance between locations. Choices in fields are Exponential, Matern
derivative	If nonzero, evaluates the partials of the covariance function at locations x1. This must be used with the "C" option and is mainly called from within a predict function. The partial derivative is taken with respect to x1.

`Distance` Character string that is the name of the distance function to use. Choices in fields are `rdist`, `rdist.earth`

`distMat` If the distance matrix between `x1` and `x2` has already been computed, it can be passed via this argument so it won't need to be recomputed.

`Dist.args` A list of optional arguments to pass to the Distance function.

`k` The order of the Wendland covariance function. See help on Wendland.

`marginal` If TRUE, returns just the diagonal elements of the covariance matrix using the `x1` locations. In this case this is just 1.0. The marginal argument is trivial for this function, but is a required argument and capability for all covariance functions used with Krig.

`mDegree` For the radial basis function p = 2m-d, with d being the dimension of the locations, is the exponent applied to the distance between locations. (m is a common way of parametrizing this exponent.) Equivalently, in Tps.cov, the order of the spline. (See Details section below).

`m` Same as mDegree.

`onlyUpper` For internal use only, not meant to be set by the user. Automatically set to `TRUE` by `mKrigMLEJoint` or `mKrigMLEGrid` if `lambda.profile` is set to `TRUE`, but set to `FALSE` for the final parameter fit so output is compatible with rest of `fields`.

If `TRUE`, only the upper triangle and diagonal of the covariance matrix is computed to save time (although if a non-compact distance matrix is used, the onlyUpper argument is set to FALSE). If `FALSE`, the entire covariance matrix is computed. In general, it should only be set to `TRUE` for `mKrigMLEJoint` and `mKrigMLEGrid`, and the default is set to `FALSE` so it is compatible with all of `fields`.

`p` Exponent in the exponential covariance family. p=1 gives an exponential and p=2 gives a Gaussian. Default is the exponential form. For the radial basis function, this is the exponent applied to the distance between locations.

`rhoObj` A fit object that defines a component of the marginal variance (rho) parameter for arbitrary locations using the generic `predict` function. Note that in fields, the complete marginal variance is `sigma2*rho` where `sigma2` can be estimated in `spatialProcess`.

`smoothness` For the Matern family, the smoothness of the process (aka "nu" in formulas).

`spam.format` If TRUE, returns the matrix in the sparse matrix format implemented in the spam package. If FALSE, just returns a full matrix.

`Taper` Character string that is the name of the taper function to use. Choices in fields are listed in help(taper).

`Taper.args` A list of optional arguments to pass to the Taper function. `aRange` should always be the name for the range (or scale) parameter.

`V` A matrix that describes the inverse linear transformation of the coordinates before distances are found. In R code, this transformation is: `x1 %*% t(solve(V))`. Default is NULL, that is, the transformation is just dividing distance by the scalar value `aRange`. See Details below. If one has a vector of "aRange" values that are the scaling for each coordinate, then just express this as `V = diag(aRange)` in the call to this function.

verbose	If TRUE, prints out some useful information for debugging.
with.constant	If TRUE, includes the complicated constant for radial basis functions. See the function radbas.constant for more details. The default is TRUE, include the constant. Without the usual constant, the lambda used here will differ by a constant from spline estimators (e.g., cubic smoothing splines) that use the constant. Also, a negative value for the constant may be necessary to make the radial basis positive definite as opposed to negative definite.
with.log	If TRUE, include a log term for even dimensions. This is needed to be a thin plate spline of integer order.
...	Any other arguments that will be passed to the covariance function, e.g., smoothness for the Matern.

Details

For purposes of illustration, the function Exp.cov.simple is provided in fields as a simple example and implements the R code discussed below. List this function out as a way to see the standard set of arguments that fields uses to define a covariance function. It can also serve as a template for creating new covariance functions for the Krig and mKrig functions. Also see the higher level function stationary.cov to mix and match different covariance shapes and distance functions.

A common scaling for stationary covariances: If x1 and x2 are matrices where nrow(x1)=m and nrow(x2)=n then this function will return a mXn matrix where the (i,j) element is the covariance between the locations x1[i,] and x2[j,]. The exponential covariance function is computed as exp(-(D.ij)) where D.ij is a distance between x1[i,] and x2[j,] but having first been scaled by aRange. Specifically, if aRange is a matrix to represent a linear transformation of the coordinates, then let u= x1%*% t(solve(aRange)) and v= x2%*% t(solve(aRange)). Form the mXn distance matrix with elements:

D[i,j] = sqrt(sum((u[i,] - v[j,])**2)).

and the cross covariance matrix is found by exp(-D). The tapered form (ignoring scaling parameters) is a matrix with i,j entry exp(-D[i,j])*T(D[i,j]). With T being a positive definite tapering function that is also assumed to be zero beyond 1.

Note that if aRange is a scalar, then this defines an isotropic covariance function and the functional form is essentially exp(-D/aRange).

Implementation: The function rdist is a useful FIELDS function that finds the cross Euclidean distance matrix (D defined above) for two sets of locations. Thus, in compact R code, we have

exp(-rdist(u, v))

Note that this function must also support two other kinds of calls:

If marginal is TRUE, then just the diagonal elements are returned (in R code diag(exp(-rdist(u,u)))).

If C is passed, then the returned value is exp(-rdist(u, v)) %*% C.

Some details on particular covariance functions:

Stationary covariance stationary.cov**:** Here the computation is to apply the function Covariance to the distances found by the Distance function. For example,

Exp.cov(x1,x2, aRange=MyTheta)

and

`stationary.cov( x1,x2, aRange=MyTheta, Distance= "rdist",Covariance="Exponential")`

are the same. This is also the same as

`stationary.cov( x1,x2, aRange=MyTheta, Distance= "rdist",Covariance="Matern",smoothness=.5).`

Radial basis functions (`Rad.cov`)**:** The functional form is Constant* rdist(u, v)**p for odd dimensions and Constant* rdist(u,v)**p * log(rdist(u,v)) for even dimensions. For an m-th order thin plate spline in d dimensions p= 2*m-d and must be positive. The constant, depending on m and d, is coded in the fields function `radbas.constant`. This form is only a generalized covariance function – it is only positive definite when restricted to a linear subspace. See `Rad.simple.cov` for a coding of the radial basis functions in R code.

Tps.cov This covariance can be used in a standard "Kriging" computation to give a thin-plate spline (TPS). This is useful because one can use the high level function `spatialProcess` and supporting functions for the returned object, including conditional simulation. The standard computation for a TPS uses the radial basis functions as given in `Rad.cov` and uses a QR decomposition based on a polynomial matrix to reduce the dimension of the radial basis function and yield a positive definite matrix. This reduced matrix is then used in the regular computations to find the spatial estimate. The function `Krig` and specifically `Tps` implements this algorithm. The interested reader should look at `Krig.engine.default` and specifically at the `TMatrix` polynomial matrix and resulting reduced positive definite matrix `tempM`. The difficulty with this approach is that it is not amenable to taking advantage of sparsity in the covariance matrix.

An alternative that is suggested by Grace Wahba in *Spline models for observational data* is to augment the radial basis functions with a low rank set of functions based on a low order polynomial evaluated at a set of points. The set of locations for these modifications is called *cardinal point*s due to the property mentioned below. This is implemented in `Tps.cov` leading to a full rank (non-stationary!) covariance function. If the fixed part of the spatial model also includes this same order polynomial, then the final result gives a TPS and is invariant to the choice of cardinal points. To streamline using this covariance when it is specified in the `spatialProcess` function, the cardinal points will be chosen automatically based on the observation locations and the spline order, `m`, using a space-filling design. A simple example with a fixed smoothing parameter, `lambda <- .1`, may help:

```
data( "ozone2")
s<- ozone2$lon.lat
y<- ozone2$y[16,]

fitTps1<- spatialProcess( s,y, cov.function= "Tps.cov", lambda=.1)
```

and compare the results to the standard algorithm.

```
fitTps2<- Tps( s,y, scale.type ="unscaled", lambda=.1)

stats( abs(fitTps2$fitted.values - fitTps1$fitted.values))
```

Here, the default choice for the order is 2, which in two dimensions implies a linear polynomial. The arguments filled in by default are shown below:

```
 fitTps1$args
$cardinalX
```

```
        [,1]   [,2]
[1,] -85.289 40.981
[2,] -90.160 38.330
[3,] -91.229 43.812
attr(,"scaled:scale")
[1] 1 1
attr(,"scaled:center")
[1] 0 0

$aRange
[1] NA

fitTps1$mKrig.args
[[1]]
NULL

$m
[1] 2

$collapseFixedEffect
[1] TRUE

$find.trA
[1] TRUE
```

cardinalX are the cardinal points chosen using a space-filling criterion. These are attached to the covariance arguments list so they are used in the subsequent computations for this fit (such as predict, predictSE, sim.spatialProcess).

To change the order of the spline, say to m=3:

```
fitTps1<- spatialProcess( s,y, cov.function= "Tps.cov",
               cov.args=list( mDegree=3), lambda=.1)
```

In general, if d is the dimension of the locations and m the order of the spline, one must have 2*m-d >0. The polynomial will have choose(m+d-1,d) terms and so this many cardinal points need to be specified. As mentioned above, these are generated using a space-filling design if spatialProcess is used and the points are not given.

Stationary tapered covariance stationary.taper.cov **:** The resulting cross covariance is the direct or Schur product of the tapering function and the covariance. In R code, given location matrices x1 and x2 and using Euclidean distance:

```
Covariance(rdist( x1, x2)/aRange)*Taper( rdist( x1,x2)/Taper.args$aRange)
```

By convention, the Taper function is assumed to be identically zero outside the interval [0,1]. Some efficiency is introduced within the function to search for pairs of locations that are nonzero with respect to the Taper. This is done by the SPAM function nearest.dist. This search may find more nonzero pairs than dimensioned internally, and SPAM will try to increase the space. One can also reset the SPAM options to avoid these warnings. For spam.format TRUE, the multiplication with the C argument is done with the spam sparse multiplication routines through the "overloading" of the %*% operator.

Nonstationary covariance function, Paciorek.cov This implements the nonstationary model developed by Chris Paciorek and Mark Schervish that allows for a varying range parameter over space and also a varying marginal variance. This is still experimental and spatialProcess has not been generalized to fit the parameter surfaces. It can, however, be used to evaluate the model at fixed parameter surfaces. See the last example below.

This covariance works by specifying an object aRangeObj such that the generic call `predict(aRangeObj, loc)` will evaluate the aRange function at the locations `loc`. This object can be as simple as a fit to local estimated aRange parameters using a fields function such as Tps or spatialProcess. For more specific applications, one can create a special predict function. For example, suppose log aRange follows a linear model in the spatial coordinates and these coefficients are the vector `Afit`. Define a class and a predict function.

```
aRangeObj<- list(coef=Afit)
class(aRangeObj)<- "myclass"

predict.myclass<- function( aRangeObj, x){
aRange<- exp(cbind( 1,x) %*% aRangeObj$coef)
return( aRange)
}
```

Now use `spatialProcess` with this object and make sure `predict.myclass` is also loaded.

A similar strategy will also work for a varying marginal variance by creating `sigmaObj` and, if needed, a companion predict method.

About the FORTRAN: The actual functions `Exp.cov` and `Rad.cov` call FORTRAN to make the evaluation more efficient. This is especially important when the C argument is supplied. So, unfortunately, the actual production code in Exp.cov is not as crisp as the R code sketched above. See `Rad.simple.cov` for an R coding of the radial basis functions.

Value

If the argument C is NULL, the cross covariance matrix is returned. In general, if nrow(x1)=m and nrow(x2)=n, then the returned matrix will be mXn. Moreover, if x1 is equal to x2, then this is the covariance matrix for this set of locations.

If C is a vector of length n, then the returned value is the multiplication of the cross covariance matrix with this vector.

See Also

Krig, rdist, rdist.earth, gauss.cov, Exp.image.cov, Exponential, Matern, Wendland.cov, mKrig

Examples

```
# exponential covariance matrix ( marginal variance =1) for the ozone
#locations
data(ChicagoO3)
out<- Exp.cov( ChicagoO3$x, aRange=100)

# out is a 20X20 matrix
```

```
out2<- Exp.cov( ChicagoO3$x[6:20,],ChicagoO3$x[1:2,], aRange=100)
# out2 is 15X2 matrix

# Kriging fit where the nugget variance is found by GCV
# Matern covariance shape with range of 100.
#

fit<- Krig( ChicagoO3$x, ChicagoO3$y, Covariance="Matern", aRange=100,smoothness=2)

data( ozone2)
x<- ozone2$lon.lat
y<- ozone2$y[16,]
# Omit the NAs
good<- !is.na( y)
x<- x[good,]
y<- y[good]


# example of calling the taper version directly
# Note that default covariance is exponential and default taper is
# Wendland (k=2).

stationary.taper.cov( x[1:3,],x[1:10,] , aRange=1.5, Taper.args= list(k=2,aRange=2.0,
                      dimension=2) )-> temp
# temp is now a tapered 3X10 cross covariance matrix in sparse format.

 is.spam( temp)  # evaluates to TRUE

# should be identical to
# the direct matrix product

 temp2<- Exp.cov( x[1:3,],x[1:10,], aRange=1.5) * Wendland(rdist(x[1:3,],x[1:10,]),
                     aRange= 2.0, k=2, dimension=2)
 test.for.zero(  as.matrix(temp), temp2)

# Testing that the "V" option works as advertised ...
x1<- x[1:20,]
x2<- x[1:10,]

V<- matrix( c(2,1,0,4), 2,2)
Vi<- solve( V)

u1<- t(Vi%*% t(x1))
u2<- t(Vi%*% t(x2))

look<- exp(-1*rdist(u1,u2))
look2<- stationary.cov( x1,x2, V= V)
test.for.zero( look, look2)


# Here is an example of how the cross covariance multiply works
# and lots of options on the arguments
```

```
 Ctest<- rnorm(10)

 temp<- stationary.cov( x,x[1:10,], C= Ctest,
        Covariance= "Wendland",
            k=2, dimension=2, aRange=1.5 )

# do multiply explicitly

 temp2<- stationary.cov( x,x[1:10,],
        Covariance= "Wendland",
            k=2, dimension=2, aRange=1.5 )%*% Ctest

 test.for.zero( temp, temp2)


# use the tapered stationary version
# cov.args is part of the argument list passed to stationary.taper.cov
# within Krig.
# This example needs the spam package.
#

## Not run:

Krig(x,y, cov.function = "stationary.taper.cov", aRange=1.5,
      cov.args= list(Taper.args= list(k=2, dimension=2,aRange=2.0) )
          ) -> out2
# NOTE: Wendland is the default taper here.

## End(Not run)

# BTW  this is very similar to
## Not run:
 Krig(x,y, aRange= 1.5)-> out

## End(Not run)

#################################################
#### nonstationary covariance Paciorek.cov
#################################################
## Not run:
M<- 20
gridList<- list(x=seq( 0,1,length.out=M),
                y=seq( 0,1,length.out=M))
sGrid<- make.surface.grid(gridList )
# An aRange surface
aRangeObj<- list(coef= c( 1,4,0))
class(aRangeObj)<- "myclass"

predict.myclass<- function( aRangeObj, x){
aRange<- exp(cbind( 1,x) %*% aRangeObj$coef)
return( aRange)
```

```
}

covMatrix<- Paciorek.cov( sGrid, sGrid, aRangeObj=aRangeObj)
# examine correlation surface between selected locations and the  full grid.
set.panel( 2,2)
{
imagePlot( as.surface(sGrid, covMatrix[,10]))
imagePlot( as.surface(sGrid, covMatrix[,205]))
imagePlot( as.surface(sGrid, covMatrix[,305]))
imagePlot( as.surface(sGrid, covMatrix[,390]))

}

# simulation of the field
set.seed(222)
n<- nrow( sGrid)
f<- t(chol(covMatrix)) %*% rnorm(M^2)
set.panel()
imagePlot( as.surface(sGrid,f))
y<- f + .05*rnorm(n)
fitP<- spatialProcess( sGrid, y, cov.function="Paciorek.cov",
           cov.args= list(aRangeObj = aRangeObj ) )
# check estimated tau and sigma
fitP$summary

# fitted surface
surface( fitP)


## End(Not run)
```

`CovarianceUpper` *Evaluate covariance over upper triangle of distance matrix*

Description

Evaluates the covariance over the upper triangle of a distance matrix rather than over the entire matrix to reduce computation time. Note that the `chol` function only requires the upper triangle of the covariance matrix to perform the Cholesky decomposition.

Usage

```
ExponentialUpper(distMat, range = 1, alpha = 1/range, theta = NULL)
```

Arguments

`distMat`	The distance matrix to evaluate the covariance over.
`range`	Range parameter; the default is one. Note that the scale can also be specified through the "aRange" scaling argument (used in fields covariance functions).

alpha	1/range
theta	Also the range parameter.

Value

The covariance matrix, where only the upper triangle is calculated.

Author(s)

John Paige

See Also

Exponential

Examples

```
set.seed(123)

#a  distance matrix
coords = matrix(runif(10), ncol=2)
distMat = rdist(coords)

#compute covariance matrix, but only over the upper triangle
upperCov = ExponentialUpper(distMat, range=.1)

print(distMat)
print(upperCov)
```

---

cover.design *Computes Space-Filling "Coverage" designs using Swapping Algorithm*

Description

Finds the set of points on a discrete grid (Candidate Set) which minimize a geometric space-filling criterion. The strength of this method is that the candidate set can satisfy whatever constraints are important for the problem.

Usage

```
cover.design(R, nd, nruns = 1, nn = TRUE, num.nn = 100, fixed = NULL,
    scale.type = "unscaled", R.center, R.scale, P = -20, Q = 20,
    start = NULL, DIST = NULL, return.grid = TRUE, return.transform =
TRUE, max.loop=20, verbose=FALSE)
```

Arguments

R	A matrix of candidate points to be considered in the design. Each row is a separate point.
nd	Number of points to add to the design. If points exist and are to remain in the design (see "fixed" option), nd is the number of points to add. If no points are fixed, nd is the design size.
nruns	The number of random starts to be optimized. Uses random starts unless "start" is specified. If nruns is greater than 1, the final results are the minimum.
nn	Logical value specifying whether or not to consider only nearest neighbors in the swapping algorithm. When nn=FALSE, then the swapping algorithm will consider all points in the candidate space. When nn=TRUE, then the swapping algorithm will consider only the num.nn closest points for possible swapping. The default is to use nearest neighbors only (nn=TRUE).
num.nn	Number of nearest neighbors to search over. The default number is 100. If nn=F, then this argument will be ignored.
fixed	A matrix or vector specifying points to be forced into the experimental design. If fixed is a matrix, it gives coordinates of the fixed points in the design. In this case, fixed must be a subset of R. If fixed is a vector, then fixed gives the row numbers from the candidate matrix R that identify the fixed points. The number of points to be generated, nd, is in addition to the number of points specified by fixed.
scale.type	A character string that tells how to scale the candidate matrix, R, before calculating distances. The default is "unscaled"; no transformation is done. Another option is "range", in which case variables are scaled to a [0,1] range before applying any distance functions. Use "unscaled" when all of the columns of R are commensurate; for example, when R gives x and y in spatial coordinates. When the columns of R are not in the same units, then it is generally thought that an appropriate choice of scaling will provide a better design. This would be the case, for example, for a typical process optimization. Other choices for scale.type are "unit.sd", which scales all columns of R to have 0 mean and unit standard deviation, and "user", which allows a user-specified scaling (see R.center and R.scale arguments).
R.center	A vector giving the centering values if scale.type=user.
R.scale	A vector giving the scale values if scale.type=user.
P	The "p" exponent of the coverage criterion (see below). It affects how the distance from a point x to a set of design points D is calculated. P=1 gives average distance. P=-1 gives harmonic mean distance. P=-Inf would give minimum distance (not available as a value). As P gets large and negative, points will tend to be more spread out.
Q	The "q" exponent of the coverage criterion (see below). It affects how distances from all points not in the design to points in the design are averaged. When Q=1, simple averaging of the distances is employed. Q=Inf (not available as a value) in combination with P=-Inf would give a classical minimax design.
start	A matrix or vector giving the initial design from which to start optimization. If start is a matrix, it gives the coordinates of the design points. In this case, start

must be a subset of the candidate set, R matrix. If start is a vector, then start gives the row numbers of the initial design based on the rows of the candidate matrix. The default is to use a random starting design.

`DIST` This argument is only for cover.design.S. A distance metric in the form of an S function. Default is Euclidean distance (FIELDS rdist function). See details and example below for the correct form.

`return.grid` Logical value that tells whether or not to return the candidate matrix as an attribute of the computed design. The default is return.grid=T. If FALSE, this just reduces the returned object size. The candidate matrix is used by plot.spatial.design if it is available.

`return.transform`
Logical value that tells whether or not to return the transformation attributes of the candidate set. The default is return.transform=T.

`max.loop` Maximum number of outer loops in algorithm. This is the maximum number of passes through the design testing for swaps.

`verbose` If TRUE, prints out debugging information.

Details

OTHER DISTANCE FUNCTIONS: You can supply an R/S-function to be used as the distance metric. The expected calling sequence for this distance function is function(X1,X2){....} where X1 and X2 are matrices with coordinates as the rows. The returned value of this function should be the pairwise distance matrix. If nrow(X1)=m and nrow(X2)=n, then the function should return an m by n matrix of all distances between these two sets of points. See the example for Manhattan distance below.

The candidate set and DIST function can be flexible, and the last example below using sample correlation matrices is an example.

COVERAGE CRITERION: For nd design points in the set D and nc candidate points ci in the set C, the coverage criterion is defined as:

M(D,C) = [sum(ci in C) [sum(di in D) (dist(di,ci)**P]**(Q/P)]**(1/Q)

Where P, less than 0, and Q, greater than 0, are parameters. The algorithm used in "cover.design" to find the set of nd points in C that minimize this criterion is an iterative swapping algorithm which will be described briefly. The resulting design is referred to as a "coverage design" from among the class of space-filling designs. If fixed points are specified, they are simply fixed in the design set and are not allowed to be swapped out.

ALGORITHM: An initial set of nd points is chosen randomly if no starting configuration is provided. The nc x nd distance matrix between the points in C and the points in D is computed, and raised to the power P. The "row sums" of this matrix are computed. Denote these as rs.i and the vector of row sums as rs. Using rs, M(D,C) is computed as:

[sum i (rs.i)**(Q/P)]**(1/Q)

Note that if point d.i is "swapped" for point c.j, one must only recompute 1 column of the original distance matrix, and 1 row. The row elements not in the ith column will be the same for all j and so only need computing when the first swapping occurs for each d.i. Denote the sum of these off-i elements as "newrow(i)". The index is i here, since this is the same for all rows (j=1,...nc). Thus, for each swap, the row sums vector is updated as

rs(new) = rs(old) - column(i,old) + column(i,new)

And the jth element of rs(new) is replaced by:

rs(new)[j] = column(i,new)[k] + newrow(i)

Finally, M(D,C) is computed for this swap of the ith design point for the jth candidate point using [2]. The point in C that, when swapped, produces the minimum value of M(D,C) replaces d.i. This is done for all nd points in the design, and is iterated until M(D,C) does not change. When the nearest neighbor option is selected, then the points considered for swapping are limited to the num.nn nearest neighbors of the current design point.

STABILITY

The algorithm described above is guaranteed to converge. However, upon convergence, the solution is sensitive to the initial configuration of points. Thus, it is recommended that multiple optimizations be done (i.e., set nruns greater than 1). Also, the quality of the solution depends on the density of the points on the region. At the same time, for large regions, optimization can be computationally prohibitive unless the nearest neighbor option is employed.

Value

Returns a design object of class `spatialDesign`. Subscripting this object has the same effect as subscripting the first component (the design). The returned list has the following components:

`design`	The best design in the form of a matrix.
`best.id`	Row numbers of the final design from the original candidate matrix, R.
`fixed`	Row numbers of the fixed points from the original candidate matrix, R.
`opt.crit`	Value of the optimality criterion for the final design.
`start.design`	Row numbers of the starting design from the original candidate matrix, R.
`start.crit`	Value of the optimality criterion for the starting design.
`history`	The swapping history and corresponding values of the optimality criterion for the best design.
`other.designs`	The designs other than the best design generated when nruns is greater than 1.
`other.crit`	The optimality criteria for the other designs when nruns is greater than 1.
`DIST`	The distance function used in calculating the design criterion.
`nn`	Logical value for nearest-neighbor search or not.
`num.nn`	The number of nearest neighbors set.
`grid`	The matrix R is returned if the argument return.grid=T.
`transform`	The type of transformation used in scaling the data and the values of the centering and scaling constants if the argument return.transform=T.
`call`	The calling sequence.
`P`	The parameter value for calculating criterion.
`Q`	The parameter value for calculating criterion.
`nhist`	The number of swaps performed.
`nloop`	The number of outer loops required to reach convergence if nloop is less than max.loop.
`minimax.crit`	The minimax design criterion using DIST.
`max.loop`	The maximum number of outer loops.

References

Johnson, M.E., Moore, L.M., and Ylvisaker, D. (1990). Minimax and maximin distance designs. Journal of Statistical Planning and Inference 26, 131-148. SAS/QC Software. Volume 2: Usage and Reference. Version 6. First Edition (1995). "Proc Optex". SAS Institute Inc. SAS Campus Drive,

See Also

rdist, rdist.earth

Examples

```
##
##
# first generate candidate set
set.seed(123) # setting seed so that you get the same thing I do!
test.df <- matrix( runif( 600), ncol=3)

test1.des<-cover.design(R=test.df,nd=10)

summary( test1.des)
plot( test1.des)

#

## Not run:
candidates<- make.surface.grid( list( seq( 0,5,,20), seq(0,5,,20)))
out<- cover.design( candidates, 15)

# find 10 more points keeping this original design fixed

out3<-cover.design( candidates, 10,fixed=out$best.id)
# see what happened

plot( candidates[,1:2], pch=".")
points( out$design, pch="x")
points( out3$design, pch="o")

# here is a strange graph illustrating the swapping history for the
# the first design. Arrows show the swap done
# at each pass through the design.

h<- out$history
cd<- candidates
plot( cd[,1:2], pch=".")
points( out$design, pch="O", col=2)
points( out$start.design, pch="x", col=5)

arrows(
cd[h[,2],1],
cd[h[,2],2],
cd[h[,3],1],
```

```
cd[h[,3],2],length=.1)

#text( cd[h[,2],1],
#cd[h[,2],2], h[,1], cex=1.0 )


#
# try this out using "Manhattan distance"
#  ( distance following a grid of city streets)

dist.man<- function(x1,x2) {
            d<- ncol( x1)
            temp<- abs(outer( x1[,1], x2[,1],'-'))
            for ( k in 2:d){
               temp<- temp+abs(outer( x1[,k], x2[,k],'-'))
            }
            temp }

# use the design from the Euclidean distance as the starting
#configuration.

cover.design( candidates, 15, DIST=dist.man, start= out3$best.id)-> out2
# this takes a while ...
plot( out2$design)
points( out3$design, col=2)

# find a design on the sphere
#

candidates<- make.surface.grid( list( x=seq( -180,180,,20), y= seq( -85,
85,,20)))

out4<-cover.design( candidates, 15, DIST=rdist.earth)
# this takes a while

plot( candidates, pch="+", cex=2)
points(out4$design, pch="o", cex=2, col="blue")

# covering based on correlation for 153 ozone stations
#
data( ozone2)

cor.mat<-cor( ozone2$y, use="pairwise")

cor.dist<- function( x1,x2)
{matrix( 1-cor.mat[ x1,x2], ncol=length(x2))}

#
# find 25 points out of the 153
# here the "locations" are just the index but the distance is
# determined by the correlation function.
#
out5<-cover.design(cbind(1:153),25, DIST= cor.dist, scale.type="unscaled")
```

```
plot( ozone2$lon.lat, pch=".")
points(  ozone2$lon.lat[out5$best.id,],pch="O", col=4)
#
# this seems a bit strange probably due some funny correlation values
#

# reset panel
set.panel(1,1)

## End(Not run)
```

---

drape.plot *Perspective plot draped with colors in the facets.*

Description

Function to produce the usual wireframe perspective plot with the facets being filled with different colors. By default, the colors are assigned from a color bar based on the z values. `drape.color` can be used to create a color matrix different from the z matrix used for the wireframe.

Usage

```
drape.plot(x, y, z, z2=NULL, col = tim.colors(64), zlim = range(z, na.rm=TRUE),
 zlim2 = NULL, add.legend = TRUE, horizontal = TRUE, theta = 30, phi = 20,
   breaks=NA, ...)

drape.color(z, col = tim.colors(64), zlim = NULL,breaks,
   transparent.color = "white", midpoint=TRUE, eps=1e-8)
```

Arguments

x	grid values for x coordinate (or, if x is a list, the components x, y and z are used).
y	grid values for y coordinate
z	A matrix of z heights
z2	A matrix of z values to use for coloring facets. If NULL, then z is used for this purpose
col	A color table for the z values that will be used for draping
zlim	the z limits for z; these are used to set up the scale of the persp plot. This defaults to `range(z, na.rm=TRUE)` as in `persp`
zlim2	the z limits for z2; these are used to set up the color scale. This defaults to the range of z2.
add.legend	If TRUE, a color strip is added as a legend.
horizontal	If TRUE, the color strip is put at the bottom of the plot, if FALSE it is placed vertically on the right side.

theta	x-y rotation angle for perspective.
phi	z-angle for perspective.
transparent.color	Color to use when given an NA in z
midpoint	If TRUE, the color scale is formed for midpoints of z obtained by averaging 4 corners.
breaks	Numerical divisions for the color scale. The default (NA) is N+1 equally spaced points in the range zlim where N is the number of colors in col. This argument has the same effect as used in the image and image.plot functions.
eps	Amount to inflate the range (1+/- eps) to include points on break endpoints.
...	Other arguments that will be passed to the persp function. The most common is zlim, the z limits for the 3-d plot and also the limits to set up the color scale. The default for zlim is the range of z.

Details

The legend strip may obscure part of the plot. If so, add this as another step using image.plot.

When using drape.color, just drop the results into the col argument of persp. Given this function, there are no surprises in how the higher level drape.plot works: it calls drape.color followed by persp and finally the legend strip is added with image.plot.

The color scales essentially default to the ranges of the z values. However, by specifying zlim and/or zlim2, one has more control of how the perspective plot is scaled and the limits of the color scale used to fill the facets. The color assignments are done by dividing up the zlim2 interval into equally spaced bins and adding a very small inflation to these limits. The mean z2 values, comprising an (M-1)X(N-1) matrix, for each facet are discretized to the bins. The bin numbers then become the indices used for the color scale. If zlim2 is not specified, the range of the z2 matrix is used to generate the ranges of the color bar. Note that this may be different from the range of the mean facets. If z2 is not passed, then z is used in its place and in this case the zlim2 or zlim argument can be used to define the color scale.

This kind of plot is also supported through the wireframe function in the lattice package. The advantage of the fields version is that it uses the standard R graphics functions – and is written in R code.

The drape plot is also drawn by the fields surface function with type="P".

Value

drape.plot If an assignment is made, the projection matrix from persp is returned. This information can be used to add additional 3-d features to the plot. See the persp help file for an example of how to add additional points and lines using the trans3d function and also the example below.

drape.color If dim(z) = M,N this function returns a list with components:

color.index	An (M-1)X(N-1) matrix (midpoint= TRUE) or MXN matrix (midpoint=FALSE) where each element is a text string specifying the color.
breaks	The breaks used to assign the numerical values in z to color categories.

Author(s)

D. Nychka

See Also

image.plot, quilt.plot, persp, plot.surface, surface, lattice, trans3d

Examples

```
# an obvious choice:
# Dr. R's favorite New Zealand Volcano!
data( volcano)
M<- nrow( volcano)
N<- ncol( volcano)
x<- seq( 0,1,,M)
y<- seq( 0,1,,N)

pm<- drape.plot( x,y,volcano, col=terrain.colors(128))

# use different range for color scale and persp plot
# setting of border omits the mesh lines

 drape.plot( x,y,volcano, col=topo.colors(128),zlim=c(0,300),
                     zlim2=c( 120,200), border=NA)

# note transparent color for facets outside the zlim2 range

#The projection has been saved in pm
# add a point marking the summit
zsummit <- max( volcano)
ix<- row( volcano)[volcano==zsummit]
iy <- col( volcano)[volcano==zsummit]
uv <- trans3d( x[ix], y[iy],zsummit,pm)
points( uv, col="magenta", pch="+", cex=2)

# overlay volcano wireframe with gradient in x direction.

dz<- (
     volcano[1:(M-1), 1:(N-1)] - volcano[2:(M), 1:(N-1)] +
     volcano[1:(M-1), 2:(N)] - volcano[2:(M), 2:(N)]
         )/2

# convert dz to a color scale:
  zlim<- range( c( dz), na.rm=TRUE)
  zcol<-drape.color( dz, zlim =zlim, col = viridis(64) )$color.index

# with these colors

  persp( volcano, col=zcol, theta=30, phi=20,
  border=NA,expand=.3 )

# add legend using image.plot function
```

```
image.plot( zlim=zlim, legend.only =TRUE, horizontal =TRUE,
          col= viridis(64))
```

---

envelopePlot *Add a shaded region between two functions to an existing plot*

Description

This function shades the region vertically between two functions, specified as pairs of x and y vectors, and draws the functions in a darker shade. More formally, it shades all points (x,y) such that f1(x) < y < f2(x) or f2(x) < y < f1(x). When both functions have the same group of x values, the x values only need to be set once, but y2 needs to be passed in by name. If the two functions intersect, the vertical space between the functions will be shaded on both sides, as implied in the definition above.

Usage

```
envelopePlot(x1, y1, x2 = x1, y2,
                          col ="thistle1" , lineCol = "thistle3",lwd = 3, ...)
```

Arguments

x1	The x coordinates for the first function (or possibly both functions).
y1	The y coordinates for the first function.
x2	The x coordinates for the second function.
y2	The y coordinates for the second function.
col	The color to make the filling between the functions.
lineCol	The color to make the lines representing the functions.
lwd	Line width of the envelope.
...	Additional arguments to the base R function polygon that is used to draw the shaded region – except for the upper and lower boundary lines.

Author(s)

Matt Iverson

Examples

```
x <- seq(0, 2*pi,, 100)
y1 <- cos(x)
y2 <- sin(x)
plot(x, y1, type="l")
envelopePlot(x, y1, y2=y2)

x1 <- c(0, 0.5, 1)
```

```
y1 <- c(0, 2, 1)
x2 <- c(0, 1)
y2 <- c(-1, 0)
plot(x1, y1, type="l", ylim=c(-1, 2))
envelopePlot(x1, y1, x2, y2)
```

`Exponential, Matern, Radial Basis`
Covariance functions

Description

Functional form of covariance function assuming the argument is a distance between locations. As they are defined here, they are in fact correlation functions. To set the marginal variance (sill) parameter, use the `sigma` argument in `mKrig` or `Krig`. To set the nugget variance, use the `tau2` argument in `mKrig` or `Krig`.

Usage

```
Exponential(d, aRange = 1, phi = 1, theta = NULL, range = NULL)
Matern(d, aRange = 1, range = NULL, alpha = NULL, smoothness
                  = 0.5, nu = smoothness, phi = 1)
Matern.cor.to.range(d, nu, cor.target=.5, guess=NULL,...)
RadialBasis(d,M,dimension, derivative = 0)
```

Arguments

`aRange`	The usual range parameter for a covariance function. We use this name to be distinct from the "range" function and the generic parameter name "theta".
`d`	Vector of distances or, for `Matern.cor.to.range`, just a single distance.
`range`	Range parameter. It is preferred that the scale can also be specified through the "aRange" scaling argument used in fields covariance functions.
`alpha`	1/range
`theta`	Same as alpha
`phi`	This parameter option is added to be compatible with older versions of fields and refers to the marginal variance of the process, e.g., `phi* exp( -d/aRange)` is the exponential covariance for points separated by distance d and range aRange. Throughout fields, this parameter is equivalent to sigma, and it is recommended that sigma be used. If one is simulating random fields, see the help on `sim.rf` for more details.
`smoothness`	Smoothness parameter in Matern. Controls the number of derivatives in the process. Default is 1/2 corresponding to an exponential covariance.
`nu`	Same as smoothness
`M`	Interpreted as a spline, M is the order of the derivatives in the penalty.

dimension	Dimension of function
cor.target	Correlation used to match the range parameter. Default is .5.
guess	An optional starting guess for solution. This should not be needed.
derivative	If greater than zero, finds the first derivative of this function.
...	Additional arguments to pass to the bisection search function.

Details

Exponential:

exp(-d/aRange)

Matern:

con*(d**nu) * besselK(d , nu)

Matern covariance function transcribed from Stein's book page 31 nu==smoothness, alpha == 1/range

GeoR parameters map to kappa==smoothness and phi == range check for negative distances

con is a constant that normalizes the expression to be 1.0 when d=0.

Matern.cor.to.range: This function is useful to find Matern covariance parameters that are comparable for different smoothness parameters. Given a distance d, smoothness nu, target correlation cor.target and range aRange, this function determines numerically the value of aRange so that

Matern(d, range=aRange, nu=nu) == cor.target

See the example for how this might be used.

Radial basis functions:

```
C.m,d  r**(2m-d)        d- odd

C.m,d  r**(2m-d)ln(r)    d-even
```

where C.m.d is a constant based on spline theory and r is the radial distance between points. See radbas.constant for the computation of the constant.

Value

For the covariance functions: a vector or matrix of covariances. (Inherits from d).

For Matern.cor.to.range: the value of the range parameter.

Author(s)

Doug Nychka

References

Stein, M.L. (1999) Statistical Interpolation of Spatial Data: Some Theory for Kriging. Springer, New York.

See Also

stationary.cov, stationary.image.cov, Wendland, stationary.taper.cov, rad.cov

Examples

```
# a Matern correlation function
 d<- seq( 0,10,,200)
 y<- Matern( d, range=1.5, smoothness=1.0)
 plot( d,y, type="l")

# Several Materns of different smoothness with a similar correlation
# range

# find ranges for nu = .5, 1.0 and 2.0
# where the correlation drops to .1 at a distance of 10 units.

 r1<- Matern.cor.to.range( 10, nu=.5, cor.target=.1)
 r2<- Matern.cor.to.range( 10, nu=1.0, cor.target=.1)
 r3<- Matern.cor.to.range( 10, nu=2.0, cor.target=.1)

# note that these equivalent ranges
# with respect to this correlation length are quite different
# due the different smoothness parameters.

 d<- seq( 0, 15,,200)
 y<- cbind(  Matern( d, range=r1, nu=.5),
             Matern( d, range=r2, nu=1.0),
             Matern( d, range=r3, nu=2.0))

 matplot( d, y, type="l", lty=1, lwd=2)
 xline( 10)
 yline( .1)
```

`fields` *fields - tools for spatial data*

Description

`fields` is a collection of functions for curve and function fitting with an emphasis on spatial data and spatial statistics. It was developed from 1997 ar NCAR, Wendy Meiring choose the name. It is designed to provide easy-to-use but sophisticated tools for analyzing spatial data, particularly that encountered in the environmental sciences. For impatient users, jump to the examples below to see how easy this is to use. Please send bugs and questions to Doug Nychka, nychka@mines.edu. Positive comments are also welcome!

The major methods implemented include cubic and thin plate splines, universal Kriging and Kriging for large data sets. A more modern framework for Kriging is spatial process estimation with covariance parameters determined by maximum likelihood and the uncertainty derived from assumptions of a Gaussian process. Throughout, we try to include reasonable defaults in functions

that reflect our experience with analyzing spatial data. For example, the Matern covariance function is the default choice for the main spatial method.

A key feature of this package is that any covariance function implemented in R code can be used for spatial prediction through the spatial functions. Another important feature is that fields will take advantage of compactly supported covariance functions in a seamless way through the spam package. See `library( help=fields)` for a listing of all the fields contents. We also recommend the thoughtful vignette created by Ashton Wiens, Mitch Krock, and Emma Lilly (`fieldsVignette.pdf`) in the fields GitHub repository.

`fields` strives to have readable and tutorial code. Take a look at the source code for `mKrig` to see how things work "under the hood", e.g., how a linear algebra computation is overloaded to handle sparse matrices and how an output object is built up sequentially throughout a computation.

The `fields` source code is liberally commented. Unfortunately, on loading this package, R will strip comments from the source for efficiency. You can go to CRAN fields page to download the latest "tarball" (aka Package Source) and unzip to get code with comments. We also keep the most recent version of this package at the fields GitHub repository, and for commented source, go to the subdirectory `fields/R`. This may be a more recent version, however, than what is posted to CRAN.

Details

Major methods

- `spatialProcess` An easy-to-use method that fits a spatial process model (e.g., Kriging) but also estimates the key spatial parameters: nugget variance, sill variance and range parameter by maximum likelihood. The default covariance model is a Matern covariance. This function and related functions called by this are the core methods in fields and have much flexibility.

  `spatialProcess` allows one to supply a covariance function that is written in native R code. See (`stationary.cov`) that includes several families of covariances, including the Matern and several distance metrics including great circle distance. `sim.spatialProcess` and `simLocal.spatialProcess` provide "one liners" for conditional simulation of the fitted surface.
- `Tps` Thin plate spline regression including GCV and REML estimates for the smoothing parameter. For moderate-size data sets, as a first look, we use `Tps` all the time. See also `fastTps` for an approximate method to handle very large numbers of spatial locations. Also see the help file for `spatialProcess` to see how to fit a thin plate spline using the more extensive set of spatial stats functions.
- `sreg`, `splint` Fast 1-D cubic smoothing splines and interpolating splines, a workhorse algorithm for EDA and more complicated methods.
- `mKrig` (micro Krig) Efficient Universal Kriging and Gaussian process function that can take advantage of sparse covariance functions and is the core algorithm called by optimization functions and for spatial prediction.
- `QTps` An easy-to-use extension of thin plate splines for quantile and robust surface fitting.
- `mKrigMLEGrid` and `mKrigMLEJoint` for maximum likelihood estimates of covariance parameters. These functions also handle replicate fields, assumed to be independent realizations, at the same locations and can also take any covariance function written in R following the `fields` format.

Other noteworthy functions

- vgram and vgram.matrix find variograms for spatial data (and with temporal replications).
- cover.design Generates space-filling designs where the distance function is expressed in R code.
- There are many convenient functions for working with image data and rationally (well, maybe reasonably) creating and placing a color scale on plots. This suite of tools is for the users who want to extend the "base R graphics" and retain control over details. See the last amusing example in help(imagePlot) for an example. as.image, imagePlot, bubblePlot, drape.plot, quilt.plot, panelPlotCommonAxes add.image, crop.image, half.image, average.image, designer.colors, color.scale, in.poly. See also grid.list for how fields works with grids and US and world for adding a map quickly.

Generic functions that support the methods

plot - diagnostic plots of fit
summary- statistical summary of fit
print- shorter version of summary
surface- graphical display of fitted surface
predict- evaluation of fit at arbitrary points
predictSE- prediction standard errors at arbitrary points.
sim.rf- Simulate a random field on a 2-d grid.

Getting Started

Try some of the examples from help files for spatialProcess or Tps.

Some fields datasets

- CO2 Global satellite CO2 concentrations (simulated field)
- COmonthlyMet Monthly mean temperatures and precip for Colorado
- glacier An elevation dataset of a glacier also used by the applied math community to test interpolation methods.
- lennon Image of John Lennon
- NorthAmericanRainfall 50+ year average and trend for summer rainfall at 1700+ stations.
- ozone2 Daily max 8-hour ozone concentrations for the US Midwest for summer 1987.
- PRISMelevation Digital elevations for the continental US at approximately 4km resolution
- rat.diet Small paired study on rat food intake over time.
- RCMexample Regional climate model output
- RMelevation Digital elevations for the Rocky Mountain Empire
- WorldBankCO2 Demographic and carbon emission data for 75 countries and for 1999.

DISCLAIMER:

The authors cannot guarantee the correctness of any function or program in this package.

Examples

```
# some air quality data, daily surface ozone measurements for
# the Midwest:

data(ozone2)
s<-ozone2$lon.lat
y<- ozone2$y[16,] # June 18, 1987

# quick plot of spatial data with map
bubblePlot( s,y)
US( add=TRUE) # add US map

# fitting a thin plate spline surface (always a good place to
# start). Here the  default smoothing (aka lambda) found by cross-validation
  fit0<- Tps(s,y)
# fits a GCV thin plate smoothing spline surface to ozone measurements.
# Hey, it does not get any easier than this!

  summary(fit0) #diagnostic summary of the fit
  set.panel(2,2)
  plot(fit0) # four diagnostic plots of fit and residuals.

# quick plot of predicted surface
  set.panel()
  surface(fit0) # contour/image plot of the fitted surface
# see also predictSurface for more control over the evaluation grid
#
  US( add=TRUE, col="magenta", lwd=2) # US map overlaid
  title("Daily max 8 hour ozone in PPB,  June 18th, 1987")

####
  fit2<- spatialProcess( s,y)
# a "Kriging" model. The covariance defaults to a Matern
# with smoothness 1.0.
# the nugget, sill and range parameters are found by maximum likelihood
# summary, plot, and surface also work for \code{fit2} !

  surface(fit2) # contour/image plot of the fitted surface
  US( add=TRUE, col="magenta", lwd=2) # US map overlaid
  title("Daily max 8 hour ozone in PPB,  June 18th, 1987")
## Not run:
# And 20 approximate conditional draws of the spatial field on a grid
# with uncertainty in the 120PPB contour
   look<- simLocal.spatialProcess(fit2, M=20)
for( k in 1:20){
contour( look$x, look$y, look$z[,,k], add=TRUE, level=c(120),
  col="white",  drawlabels=FALSE)
}


## End(Not run)
```

fields testing scripts
Testing fields functions

Description

Some of the basic methods in fields can be tested by directly implementing the linear algebra using matrix expressions and other functions can be cross-checked within fields. These comparisons are done in the R source code test files in the tests subdirectory of fields. The function `test.for.zero` is useful for comparing the tests in a meaningful and documented way.

Usage

```
test.for.zero(
xtest, xtrue, tol = 1e-08, relative = TRUE, RMSE =
                 FALSE, tag = NULL)
```

Arguments

`xtest`	Vector of target values
`xtrue`	Vector of reference values
`tol`	Tolerance to judge whether the test passes.
`relative`	If TRUE, a relative error comparison is used. (See details below.)
`RMSE`	If TRUE, computes the root mean squared error between `xtest` and `xtrue`
`tag`	A text string to be printed out with the test results as a reference

Details

IMPORTANT: If the R object `test.for.zero.flag` exists with any value (e.g., `test.for.zero.flag <- 1`), then when the test fails, this function will also generate an error in addition to printing a message. This option is added to ensure that any test scripts will generate an error when any individual test fails.

An example:

```
> test.for.zero( 1:10, 1:10 + 1e-10, tag="First test")
Testing:  First test
PASSED test at tolerance  1e-08

> test.for.zero( 1:10, 1:10 + 1e-10, tag="First test", tol=1e-12)
Testing:  First test
FAILED test value =  1.818182e-10  at tolerance  1e-12

> test.for.zero.flag <- 1
Testing:  First test
FAILED test value =  1.818182e-10  at tolerance  1e-12
Error in test.for.zero(1:10, 1:10 + 1e-10, tag = "First test", tol = 1e-12) :
```

The scripts in the `tests` subdirectory are

Krig.test.R: Tests basic parts of the Krig and Tps functions including replicated and weighted observations.

Krig.se.test.R: Tests computations of standard errors for the Kriging estimate.

Krig.se.grid.test.R Tests approximate standard errors for the Krig function found by Monte Carlo conditional simulation.

Krig.test.W.R Tests predictions and A matrix when an off-diagonal observation weight matrix is used.

Krig.se.W.R Tests standard errors when an off-diagonal observation weight matrix is used.

spam.test.R Tests sparse matrix formats and linear algebra.

Wend.test.R Tests form for Wendland covariance family and its use of sparse matrix formats.

diag.multiply.test.R Tests special (efficient) version of matrix multiply for diagonal matrices.

evlpoly.test.R Tests evaluation of univariate and multivariate polynomial evaluation.

mKrig.test.R Tests the micro Krig function with and without sparse matrix methods.

To run the tests, just attach the fields library and source the testing file. In the fields source code, these are in a subdirectory "tests". Compare the output to the "XXX.Rout.save" text file.

`test.for.zero` is used to print out the result for each individual comparison. Failed tests are potentially bad and are reported with a string beginning

"FAILED test value = ... "

If the object test.for.zero.flag exists, then an error is also generated when the test fails.

FORM OF COMPARISON: The actual test done is the sum of absolute differences:

test value = `sum( abs(c(xtest) - c( xtrue) ) ) /denom`

Where `denom` is either `mean( abs(c(xtrue)))` for relative error or 1.0 otherwise.

Note the use of "c" here to stack any structure in xtest and xtrue into a vector.

`fields-stuff` *Fields supporting functions*

Description

Some supporting functions that are internal to fields top-level methods. Variants of these might be found in the R base, but these have been written for cleaner code or efficiency.

Usage

```
fields.duplicated.matrix(mat, digits = 8)

fields.mkpoly(x, m = 2, tag = "term")

fields.derivative.poly(x, m,dcoef)

fields.evlpoly( x, coef)

fields.evlpoly2( x, coef, ptab)
```

Arguments

mat	Arbitrary matrix for examining rows
digits	Number of significant digits to use for comparing elements to determine duplicate values.
x	Arbitrary matrix where rows are components of a multidimensional vector
m	The null space degree – results in a polynomial of degree (m-1)
dcoef	Coefficients of a multidimensional polynomial
coef	Polynomial coefficients.
ptab	Table of powers of different polynomial terms.
tag	mkpoly fills in as column names the higher order terms of the polynomial terms. The tag is the text string prefix for these column names and the powers of the individual variables are appended. Default is just "terms".

Details

fields.duplicated finds duplicate rows in a matrix. The digits argument is the number of digits that are considered in the comparison. The returned value is an array of integers from 1:M where M is the number of unique rows, and duplicate rows are referenced in the same order that they appear as the rows of mat.

fields.mkpoly computes the complete matrix of all monomial terms up to degree (m-1). Each row of x contains the components of a vector. (The fields function mkpoly returns the number of these terms.) In 2 dimensions with m=3, there are 6 polynomial terms up to quadratic (3-1 = 2) order, and these will be returned as the matrix:

cbind(1 , x[,1], x[,2], x[,1]**2, x[,1]*x[,2], x[,2]**2)

This function is used for the fixed effects polynomial or spatial drift used in spatial estimating functions Krig, Tps and mKrig. The matrix ptab is a table of the powers in each term for each variable and is included as an attribute to the matrix returned by this function. See the attr function for extracting an attribute from an object.

ptab for the example above is

```
    [,1] [,2]
[1,]    0    0
[2,]    1    0
[3,]    0    1
[4,]    2    0
[5,]    1    1
[6,]    0    2
```

This information is used in finding derivatives of the polynomial and is also used to create column names for the terms that are of higher order than linear.

fields.derivative.poly finds the partial derivative matrix of a multidimensional polynomial of degree (m-1) at different vector values and with coefficients dcoef. This function has been organized to be a clean utility for predicting the derivative of the estimated function from Krig or mKrig. Within the fields context, the polynomial itself would be evaluated as fields.mkpoly(

x,m)%*%dcoef. If x has d columns (also the dimension of the polynomial) and n rows, the partial derivatives of this polynomial at the locations x can be organized in a nXd matrix. This is the object returned by this function.

evlpoly and evlpoly2 are FORTRAN based functions for evaluating univariate polynomials and multivariate polynomials. The table of powers (ptab) needed for evlpoly2 is the same format as that returned by the fields.mkpoly function.

Author(s)

Doug Nychka

`fields.grid` *Using mKrig for predicting on a grid.*

Description

This is an extended example for using the sparse/fast interpolation methods in mKrig to evaluate a Kriging estimate on a large grid.

Details

mKrig is a flexible function for surface fitting using a spatial process model. It can also exploit sparse matrix methods for large data sets by using a compactly supported covariance. The example below shows how to evaluate a solution on a big grid. (Thanks to Jan Klennin for this example.)

Examples

```
data(RMprecip)
x<- RMprecip$x
y<- RMprecip$y

Tps( x,y)-> obj

# make up an 80X80 grid that has ranges of observations
# use same coordinate names as the x matrix

glist<- fields.x.to.grid(x, nx=80, ny=80) # this is a cute way to get a default grid that covers x

# convert grid list to actual x and y values ( try plot( Bigx, pch="."))
    make.surface.grid(glist)-> Bigx

# include actual x locations along with grid.
    Bigx<- rbind( x, Bigx)

# evaluate the surface on this set of points (exactly)

    predict(obj, x= Bigx)-> Bigy

# set the range for the compact covariance function
```

```
# this will involve  less than 20 nearest neighbors that have
# nonzero covariance
#

     V<- diag(c( 2.5*(glist$lon[2]-glist$lon[1]),
                 2.5*(glist$lat[2]-glist$lat[1])))
## Not run:
# this is an interpolation of the values using a compact
# but thin plate spline like covariance.
    mKrig( Bigx,Bigy, cov.function="wendland.cov",k=4, V=V,
                 lambda=0)->out2
# the big evaluation this takes about 45 seconds on a Mac G4 latop
    predictSurface( out2, nx=400, ny=400)-> look

## End(Not run)

# the nice surface
## Not run:
    surface( look)
    US( add=TRUE, col="white")

## End(Not run)
```

`fields.hints` *fields - graphics hints*

Description

Here are some technical hints for assembling multiple plots with common legends or axes and setting the graphics parameters to make more readable figures. Also, we provide an index to the default colors in R graphics and setting their definitions in LaTeX. All these hints use the standard graphics environment.

Usage

```
fields.style()
fields.color.picker()
```

Details

`fields.style` is a simple function to enlarge the characters in a plot and set the colors. List this out to modify the choices.

```
##Two examples of concentrating a panel of plots together
## to conserve the white space.
## see also the example in image.plot using split.screen.
## The basic trick is to use the oma option to reserve some space around the
```

```
## plots.  Then unset the outer margins to use that room.

library( fields)

# some hokey image data
x<- 1:20
y<- 1:15
z<-  outer( x,y,"+")
zr<- range( c(z))

# add common legend to 3X2 panel

par( oma=c(4,0,0,0))
set.panel( 3,2)
par( mar=c(1,1,0,0))

# squish plots together with just 1 space between
for ( k in 1:6){
image( x,y,z, axes=FALSE, xlab="", ylab="", zlim=zr)
}

par( oma=c(0,0,0,0))
image.plot( zlim=zr, legend.only=TRUE, horizontal=TRUE, legend.mar=5)

# you may have to play around with legend.mar and the oma settings to
# get enough space.



##
### also add some axes on the sides in a lattice style
## note oma adds some more room at bottom.

par( oma=c(8,6,1,1))
set.panel( 3,2)
par( mar=c(1,1,0,0))
##
for ( k in 1:6){
 image( x,y,z, axes=FALSE, xlab="", ylab="", zlim=zr)
 box() # box around figure

# maybe draw an x axis
  if( k  %in% c(5,6) ){
  axis( 1, cex.axis=1.5)
  mtext( line=4, side=1, "Xstuff")}

# maybe draw a y axis
  if( k  %in% c(1,3,5) ){
```

```
  axis( 2, cex.axis=1.5)
  mtext( line=4, side=2, "Ystuff")}
}

# same trick of adding a legend strip.
par( oma=c(0,0,0,0))
image.plot( zlim=zr, legend.only=TRUE, horizontal=TRUE, legend.mar=5)

# reset panel
set.panel()


####
# show colors
## the factory colors:

clab<- colors()
n<- length( clab)
N<- ceiling( sqrt(n) )
M<- N
temp<- rep( NA,M*N)
temp[1:n] <- 1:n
z<- matrix(temp, M,N)

image(seq(.5,M+.5,,M+1), seq(.5,N+.5,,N+1)
       , z,  col=clab, axes=FALSE, xlab="", ylab="")


# see the  function  fields.color.picker() to locate colors



# dumping out colors by name for a latex document
# this creates text strings that are the LaTeX color definitions
# using the definecolor function.

# grab all of the R default colors
clab<- colors()

for( nn in clab){
  temp<- signif(col2rgb(nn)/256, 3)
   cat(
    "\definecolor{",
                nn, "}",
    "{rgb}{", temp[1],
          ",", temp[2],
          ",", temp[3],
           "}", fill=TRUE , sep="")
```

```
 }

# this loop prints out definitions such as
# \definecolor{yellowgreen}{rgb}{0.602,0.801,0.195}
# having loaded the color package in LaTeX
# defining this color
# use the construction  {\color{yellowgreen}  THIS IS A COLOR}
# to use this color in a talk or document.

# this loop prints out all the colors in LaTeX language
# as their names and can be converted to a pdf for handy reference.

sink( "showcolors.tex")

clab<- colors()
for( nn in clab){
  temp<- signif(col2rgb(nn)/256, 3)
   cat(
    "\definecolor{",
                nn, "}",
    "{rgb}{", temp[1],
          ",", temp[2],
          ",", temp[3],
           "}", fill=TRUE , sep="")
   cat( paste("{ \color{",nn,"} ", nn," $\bullet$ \\ }", sep=""),
                    fill=TRUE)
}
sink()
```

---

flame *Response surface experiment ionizing a reagent*

Description

The characteristics of an ionizing flame are varied with the intent of maximizing the intensity of emitted light for lithium in solution. Areas outside of the measurements are where the mixture may explode! Note that the optimum is close to the boundary. Source of data is from a master's level lab experiment in analytical chemistry from Chuck Boss's course at NCSU.

Usage

```
data(flame)
```

Format

This is a list with the following components:

"x" "y"

- x: 2 column matrix with the different fuel and oxygen flow rates for the burner.
- y: the response: the intensity of light at a particular wavelength indicative of lithium ions.

---

glacier *Franke's Glacier Elevation Data*

Description

A moderate size (about 8400 locations) spatial dataset that is well-known in the applied mathematics approximation literature for testing interpolation methods.

Usage

```
data(glacier)
```

Format

The format of glacier is a list with two components:

loc: 8338x2 matrix of the locations (meters??).

y: A vector of elevations (meters ??).

Details

This data set appears in papers that develop interpolation methods for scattered data and serves as an interesting bridge to the examples in applied math that develop radial basis function surface fitting. The data was originally used by R. Franke.

Unfortunately, at this time, we cannot find any background on where these data were collected or indeed even the location of this glacier. However, it is an interesting data set in that it appears that the elevations are reported along lines of equal elevation, i.e., contours, perhaps from a digitization of a topographic map or survey. It is important to estimate the surface in a way that the artifacts from discretization are not present. In the example below, the compactly supported kernel interpolation still has some artifacts.

The glacier data set is available at this website https://oleg-davydov.de/scat_data.html

The examples below are useful for comparing different approximations to a Gaussian spatial process estimate for the elevation surface. Of course, in using a stationary covariance (e.g., the Matern or Wendland), these are also radial basis smoothing or interpolation of the data.

Examples

```
data(glacier)
# EDA for raw obs:

bubblePlot( glacier$loc, glacier$y, highlight=FALSE, size=.5)

# identifying contour levels. Note this is reported at regular levels
# (Every 25m ???)

table( glacier$y)

# find sigma and rho by maximum likelihood
# for a fixed range
#  the default is the Wendland covariance with k=2
# See help(Wendland)

# this takes < 2  minutes
# macbook pro  Apple M4 chip 16GB memory

#options(spam.nearestdistnnz=c(5e7,1e3))
#system.time(
# obj0<- fastTps(glacier$loc, glacier$y,
#                       theta=2,
#                      profileLambda=TRUE)
#)


# set.panel(2,2)
# plot( obj0)
# set.panel()

# just evaluate at MLE
# reset default matrix size that the spam package will use.

## Not run:

options(spam.nearestdistnnz=c(5e7,1e3))
system.time( obj1<-
               fastTps(glacier$loc, glacier$y,
                       theta=2,
                       lambda= 7.58e-5
                        )
)

system.time(
look1<- predictSurface( obj1, nx=150, ny=150)
)

imagePlot( look1)


system.time(
```

```
out<- simLocal.spatialProcess(obj1, M=3, nx=150, ny=150)
)
set.panel( 2,2)
imagePlot( look1)
zlim<- range( out$z, na.rm=TRUE)
for( k in 1:3){
imagePlot(out$x, out$y, out$z[,,k], zlim=zlim)
}

# near interpolation surface using Matern smoothness .5
 system.time(
 obj2<- spatialProcess(glacier$loc, glacier$y,
                           aRange = 1.5,
                           lambda = 1e-5,
                           smoothness = .5)
 )

system.time(
out<- simLocal.spatialProcess(obj2, M=3, nx=150, ny=150,
fast=TRUE)
)

set.panel( 2,2)
imagePlot( look1)
zlim<- range( out$z, na.rm=TRUE)
for( k in 1:3){
imagePlot(out$x, out$y, out$z[,,k], zlim=zlim)
}

system.time(
look2<- predictSurface.mKrig( obj2, nx=150, ny=150,
                fast=TRUE, NNSize=5)
)

system.time(
look2B<- predictSurface( obj2, nx=150, ny=150,
                fast=FALSE)
)

err<- c((look2$z - look2B$z)/look2B$z)
stats( log10( abs(err) ) )

# some error plots ( percent relative error)
imagePlot(look2$x, look2$y, 100*(look2$z - look2B$z)/look2B$z  )

imagePlot(look2$x, look2$y, 100*(look1$z - look2B$z)/look2B$z  )


## End(Not run)
```

`grid list` *Some simple functions for working with gridded data and the grid format (grid.list) used in fields.*

Description

The object grid.list refers to a list that contains information for evaluating a function on a 2-dimensional grid of points. If a function has more than two independent variables, then one also needs to specify the constant levels for the variables that are not being varied. This format is used in several places in fields for functions that evaluate function estimates and plot surfaces. These functions provide some default conversions among information and the grid.list. The function `discretize.image` is a useful tool for "registering" irregular 2-d points to a grid.

Usage

```
makeMultiIndex(M)
parse.grid.list( grid.list, order.variables="xy")
fields.x.to.grid(x,nx=80, ny=80, xy=c(1,2))
fields.convert.grid( midpoint.grid )
discretize.image(x, m = 64, n = 64, grid = NULL,
                 expand = c(1 + 1e-08, 1 + 1e-08),
                 boundary.grid = FALSE, na.rm = TRUE)
make.surface.grid( grid.list)
unrollXMatGrid( grid.list, XMatGrid)
```

Arguments

`M`	A vector of integers.
`grid.list`	No surprises here – a grid list! These can be unequally spaced.
`order.variables`	If "xy", the x variable will be subsequently plotted as the horizontal variable. If "yx", the x variable will be on the vertical axis.
`x`	A matrix of independent variables such as the locations of observations given to Krig.
`nx`	Number of grid points for x variable.
`ny`	Number of grid points for y variable.
`m`	Number of grid points for x variable.
`n`	Number of grid points for y variable.
`na.rm`	Remove missing values if TRUE
`xy`	The column positions that locate the x and y variables for the grid.
`grid`	A grid list!
`expand`	A scalar or two column vector that will expand the grid beyond the range of the observations.
`midpoint.grid`	Grid midpoints to convert to grid boundaries.

`boundary.grid` If TRUE, interpret grid points as boundaries of grid boxes. If FALSE, interpret as the midpoints of the boxes.

`XMatGrid` An array or list form of covariates to use for prediction. This must match the `grid.list` argument, i.e., XMatGrid and grid.list describe the same grid. If XMatGrid is an array, then the first two indices are the x and y locations in the grid. The third index, if present, indexes the covariates. E.g., for evaluation on a 10X15 grid and with 2 covariates, `dim( XMatGrid) == c(10,15, 2)`. If XMatGrid is a list, then the components x and y should match those of grid.list and the z component follows the shape described above for the non-list case.

Details

`makeMultiIndex` creates an expanded set of indices to reference a regular grid. M are L integers with product prodM. This will create a prodM by L matrix that is all combinations of (1:M[i]) for i =1,2, ...L This is organized in the standard array ordering where the first column varies the fastest. For M = c(3,2,4), the result will be a 24X3 matrix with the entries:

```
                    1,1,1
                    2,1,1
                    3,1,1
                    1,2,1
                    2,2,1
                    3,2,1
                 etc ...
...
 and  ending with
                    2,2,4
                    3,2,4
```

All about grid lists:

The form of a grid.list is

`list( var.name1= what1 , var.name2=what2 , ... var.nameN=what3)`

Here var.names are the names of the independent variables. The what options describe what should be done with this variable when generating the grid. These should either be an increasing sequence of points or a single value. Obviously there should only be two variables with sequences to define a grid for a surface.

Most of the time, the gridding sequences are equally spaced and are easily generated using the `seq` function. Also, throughout fields, the grid points are typically the midpoints of the grid rather than the grid box boundaries. However, these functions can handle unequally spaced grids, and the logical boundary.grid can indicate a grid being the box boundaries.

The variables in the list components are assumed to be in the same order as they appear in the data matrix.

A useful function that expands the grid from the grid.list description into a full set of locations is `make.surface.grid`, which is just a wrapper around the R base function `expand.grid`. A typical operation is to go from a grid.list to the set of grid locations, evaluate a function at these locations, and then reformat this as an image for plotting. Here is how to do this cleanly:

```
grid.list<- list( x= 1:10, y=1:15)
xg<- make.surface.grid(grid.list)
# look at a surface depending on
 xg locations
z<- xg[,1] + 2*xg[,2]
out<- list( x=grid.list$x, y= grid.list$y, z=matrix( z, nrow=10, ncol=15))
# now for example
image.plot( out)
```

The key here is that xg and `matrix` both organize the grid in the same order.

Some fields internal functions that support interpreting grid list format are:

`fields.x.to.grid`: Takes an "x" matrix of locations or independent variables and creates a reasonable grid list. This is used to evaluate predicted surfaces when a grid list is not explicitly given to predictSurface. The variables (i.e., columns of x) that are not part of the grid are set to the median values. The x grid values are nx equally spaced points in the range `x[, xy[1]]`. The y grid values are ny equally spaced points in the range `x[, xy[2]]`.

`parse.grid.list`: Takes a grid list and returns the information in a more expanded list form that is easy to use. This is used, for example, by predictSurface to figure out what to do!

`fields.convert.grid`: Takes a vector of n values assumed to be midpoints of a grid and returns the n+1 boundaries. See how this is used in discretize.image with the cut function. This function will handle unequally spaced grid values.

`discretize.image`: Takes a vector of locations and a 2-d grid and figures out to which boxes they belong. The output matrix ind has the grid locations. If boundary.grid is FALSE, then the grid list (grid) is assumed to be grid midpoints. The grid boundaries are taken to be the point halfway between these midpoints. The first and last boundary points are determined by extrapolating so that the first and last boxes have their midpoints in their centers. (See the code in fields.convert.grid for details.) If grid is NULL, then midpoints are found from m and n and the range of the x matrix.

`unrollXMatGrid` Checks that the XMatGrid object is compatible with the grid.list and concatenates the grid arrays into vectors. This version of the covariates is used by the usual predict function.

See Also

as.surface, predictSurface, plot.surface, surface, expand.grid, as.image

Examples

```
#Given below are some examples of grid.list objects and the results
#when they are used with make.surface.grid. Note that
#make.surface.grid returns a matrix that retains the grid.list
#information as an attribute.

grid.l<- list( 1:3, 2:5)
make.surface.grid(grid.l)


grid.l <- list( 1:3, 10, 1:3)
make.surface.grid(grid.l)
```

```
#The next  example shows how the grid.list can be used to
#control surface plotting and evaluation of an estimated function.
# first create a test function

set.seed( 124)

X<- 2*cbind( runif(30), runif(30), runif(30)) -1

dimnames( X)<- list(NULL, c("X1","X2","X3"))
y<- X[,1]**2 + X[,2]**2 + exp(X[,3])

# fit an  interpolating thin plate spline
out<- Tps( X,y)

grid.l<- list( X1= seq( 0,1,,20), X2=.5, X3=seq(0,1,,25))
surface( out, grid.list=grid.l)
#  surface plot based on a 20X25 grid in X1 an X3
#                       over the square [0,2] and [0,2]
#                       holding X2 equal to 1.0.
#


# test of discretize to make sure points on boundaries are counted right
set.seed(123)
x<- matrix( runif(200), 100,2)
look<- discretize.image( x, m=2,n=2)
xc<- seq(min(x[,1]), max(x[,1]),,5)
xc<- xc[2:4]
yc<- seq(min(x[,2]), max(x[,2]),,5)
yc<- yc[2:4]
grid <-  list( x= xc, y= yc)
look2<- discretize.image( x, m=2,n=2)

table( look$index )
table( look2$index )

# indicator image of discretized locations
data(RMprecip)
look<- discretize.image( RMprecip$x, m=15, n=15)
image.plot( look$grid$x, look$grid$y,look$hist )
# actual locations
points( RMprecip$x,col="magenta", pch=".")
```

---

image.cov *Exponential, Matern and general covariance functions for 2-d gridded locations.*

Description

Given two sets of locations defined on a 2-d grid, efficiently multiplies a cross covariance with a vector. The intermediate computations (the setup) can also be used for fast simulation of the processes on a grid using the circulant embedding technique.

Usage

```
stationaryImageCov(ind1 = NULL, ind2 = NULL, Y, covObject = NULL, gridList = NULL,
    mKrigObject = NULL, setup = FALSE, M = NULL, cov.function = "stationary.cov",
     delta = NULL, cov.args = NULL, ...)

stationary.image.cov(ind1, ind2, Y, cov.obj = NULL, setup = FALSE,
grid, M=NULL,N=NULL,cov.function="stationary.cov", delta = NULL, cov.args = NULL, ...)

Exp.image.cov(ind1, ind2, Y, cov.obj = NULL, setup = FALSE, grid, ...)

Rad.image.cov(ind1, ind2, Y, cov.obj = NULL, setup = FALSE, grid, ...)

matern.image.cov(ind1, ind2, Y, cov.obj = NULL, setup = FALSE, grid,
M=NULL,N=NULL,aRange= 1.0, smoothness=.5, theta=NULL)

wendland.image.cov(ind1, ind2, Y, cov.obj = NULL,
    setup = FALSE, grid, M = NULL, N = NULL, cov.args=NULL, ...)
```

Arguments

ind1	Matrix of indices for first set of locations. This is a two column matrix where each row is the row/column index of the image element. If missing, the default is to use all grid locations.
ind2	Matrix of indices for second set of locations. If missing, this is taken to be ind1. If ind1 is missing, ind2 is coerced to be all grid locations.
Y	Vector to multiply by the cross covariance matrix. Y must be at the same locations as those referred to by ind2.
cov.args	Any additional arguments or parameters to the covariance function.
cov.obj	A list with the information needed to do the multiplication by convolutions. This is usually found by using the returned list when setup=T.
covObject	Same as cov.obj above.
cov.function	Name of the (stationary) covariance function.
setup	If TRUE, do not do the multiplication; just return the covariance object required by this function.
delta	A distance that indicates the range of the covariance when it has compact support. For example, this is the aRange parameter in the Wendland covariance.
grid	A grid list giving the X and Y grids for the image. (See example below.) This is only required if setup is TRUE.

gridList	Same as grid above.
M	For stationaryImageCov, a vector of two integers that are the desired size for the larger grid to facilitate the computation. If NULL, then this is chosen to be at least twice the grid size and powers of 2 and 3. For stationary.image.cov, the size of the x-grid used to compute multiplication (see notes on image.smooth for details) by the FFT. If NULL, the default for M is the largest power of 2 greater than or equal to 2*m where m is the length of the x component of the grid. In either case, this will give an exact result, but smaller values of M will yield an approximate, faster result.
mKrigObject	The fit object from spatialProcess or mKrig. This is in place of specifying the covariance parameters individually.
N	Size of y-grid used to compute multiplication by the FFT for stationary.image.cov
aRange	Scale parameter for Matern.
theta	Same as aRange.
smoothness	Smoothness parameter for Matern (.5=Exponential)
...	Any arguments to pass to the covariance function in setting up the covariance object. This is only required if setup is TRUE. For stationary.image.cov, one can include V, a matrix reflecting a rotation and scaling of coordinates. See stationary.cov for details.

Details

This function was provided to do fast computations for large numbers of spatial locations. In doing so, the observations can be irregularly spaced but their coordinates must be 2-dimensional and be restricted to grid points. (The function as.image will take irregular, continuous coordinates and overlay a grid on them.)

Returned value: If ind1 and ind2 are matrices where nrow(ind1)=m and nrow(ind2)=n, then the cross covariance matrix, Sigma, is an mXn matrix whose (i,j) element is the covariance between the grid locations indexed at ind1[i,] and ind2[j,]. The returned result is Sigma multiplied by Y. Note that one can always recover the coordinates themselves by evaluating the grid list at the indices. E.g., if x and y are the grids for the X and Y dimensions, cbind(x[ind1[,1]], y[ind1[,2]]) will give the coordinates associated with ind1. Clearly it is better just to work with ind1!

Functional Form: Following the same form as Exp.cov and stationary.cov for irregular locations, the covariance is defined as phi(D.ij) where D.ij is the Euclidean distance between x1[i,] and x2[j,] but having first been scaled by aRange. Specifically,

D.ij = sqrt(sum.k ((x1[i,k] - x2[j,k]) /aRange[k])**2).

See Matern for the version of phi for the Matern family.

Note that if aRange is a scalar, then this defines an isotropic covariance function.

Implementation: This function does the multiplication on the full grid efficiently by a 2-d FFT. The irregular pattern in Y is handled by padding with zeroes, and once that multiplication is done, only the appropriate subset is returned.

NOTE: stationaryImageCov is the current version and should be used instead of the older stationary.image.cov

As an example, assume that the grid is 100X100, let big.Sigma denote the big covariance matrix among all grid points (if the parent grid is 100x100, then big.Sigma is 10K by 10K!). Here are the algorithmic steps:

temp<- matrix(0, 100,100)

temp[ind2] <- Y

temp2<- big.Sigma%*% temp

temp2[ind1]

Here, the matrix multiplication is effected through fast convolution/FFT exploiting stationarity and avoids creating and multiplying big.Sigma explicitly. It is often faster to multiply the regular grid and throw away the parts we do not need than to deal directly with the irregular set of locations.

Note: In this entire discussion, Y is treated as a vector. However, if one has complete data, then Y can also be interpreted as an image matrix conformed to correspond to spatial locations. See the last example for this distinction.

Value

A vector that is the multiplication of the cross covariance matrix with the vector Y.

See Also

as.image,stationary.cov, circulantEmbeddingSetup, mKrigFastPredict

Examples

```
# multiply 2-d isotropic exponential with aRange=4 by a random vector

junk<- matrix(rnorm(50*100), 50,100)
gridList<- list( x= seq(0,1,length.out=50),
     y= seq(-1,1,length.out=100) )

covObject<- stationaryImageCov( setup=TRUE,
Covariance="Matern",
smoothness=1.0,
aRange=.08,
             gridList=gridList)

result<-  stationaryImageCov(Y=junk, covObject=covObject)

imagePlot(gridList$x,gridList$y, result ) # NOTE that is also a smoother!

# to do it again, no setup is needed
#  e.g.
#  junk2<- matrix(rnorm(100**2, 100,100))
#  result2<-  stationaryImageCov(Y=junk2, cov.obj=cov.obj)

# generate a grid and set of indices based on discretizing the locations
# in the precip dataset
data(RMprecip)
 out<-as.image( RMprecip$y, x= RMprecip$x)
```

```
 ind1<- out$ind
 gridList<- list( x= out$x, y=out$y)

#
# discretized x locations  to use for comparison
  xd<- cbind( out$x[ out$ind[,1]], out$y[ out$ind[,2]] )

# setup to create cov.obj for exponential covariance with range= 1.25

 covObject<- stationaryImageCov( setup=TRUE, gridList=gridList, aRange=1.25)

# multiply covariance matrix by an arbitrary vector
 junk<-  rnorm(nrow( ind1))
 result<- stationaryImageCov( ind1, ind1, Y= junk,covObject=covObject)

# The brute force way would be
#   result<- stationary.cov( xd, xd, aRange=1.25, C=junk)
# or
#   result<- stationary.cov( xd, xd, aRange=1.25) %*% junk
# both of these take much longer


# evaluate the covariance between all grid points and the center grid point
mGrid<- covObject$m
 Y<- matrix(0,mGrid[1], mGrid[2])
 Y[32,32]<- 1
 result<- stationaryImageCov( Y= Y,covObject=covObject)
# covariance surface with respect to the grid point at (32,32)
#
# reshape "vector" as an image
 temp<-  matrix( result, mGrid[1],mGrid[2])
 imagePlot( temp)
```

`image.plot` *Draws an image plot with a legend strip for the color scale based on either a regular grid or a grid of quadrilaterals.*

Description

This function combines the R image function with some automatic placement of a legend. This is done by splitting the plotting region into two parts, putting the image in one and the legend in the other. After the legend is added, the plot region is reset to the image plot. This function also allows for plotting quadrilateral cells in the image format, which often arise from regular grids transformed with a map projection or a scaling and rotation of coordinates. See the example where this function can create a similar graphic to the ggplot package. `image.plot` functionality has been frozen along with documentation; see the more recent function `imagePlot` which is backwardly compatible with this function. Also see `help(imagePlot)` for updated documentation.

Usage

```
## S3 method for class 'plot'
image(...,
                  add = FALSE,
                  breaks= NULL, nlevel = 64, col = NULL,
    horizontal = FALSE, legend.shrink = 0.9, legend.width = 1.2,
    legend.mar = ifelse(horizontal, 3.1, 5.1), legend.lab = NULL,
    legend.line= 2,
    graphics.reset = FALSE, bigplot = NULL, smallplot = NULL,
    legend.only = FALSE,  lab.breaks = NULL,
    axis.args = NULL, legend.args = NULL, legend.cex=1.0,
    midpoint = FALSE, border = NA,
    lwd = 1,verbose = FALSE )
```

Arguments

...	The usual arguments to the image function as x, y, or z or as a list with x,y,z as components. One can also include a breaks argument for an unequally spaced color scale with color scale boundaries at the breaks (see example below). If a "quadrilateral grid", arguments must be explicitly x, y and z with x and y being matrices of dimensions equal to, or one more than, z giving the grid locations. The basic concept is that the coordinates of x and y still define a grid, but the image cells are quadrilaterals rather than being restricted to rectangles. NOTE: graphical arguments passed here will only have impact on the image plot. To change the graphical defaults for the legend, use the individual legend arguments and/or legend.arg listed below.
add	If TRUE, add the image and a legend strip to the existing plot.
bigplot	Plot coordinates for image plot. If not passed, these will be determined within the function.
border	This only works if x and y are matrices – if NA, the quadrilaterals will have a border color that is the same as the interior color. Otherwise, this specifies the color to use.
breaks	Break points in sorted order to indicate the intervals for assigning the colors. Note that if there are nlevel colors, there should be (nlevel+1) breakpoints. If breaks is not specified, (nlevel+1) equally spaced breaks are created where the first and last bin have their midpoints at the minimum and maximum values in z or at zlim.
col	Color table to use for image (see help file on image for details). Default is a pleasing range of 64 divisions suggested by Tim Hoar and is similar to the MATLAB (TM) jet color scheme. Note that if breaks is specified, there must be one less color specified than the number of breaks.
graphics.reset	If FALSE (default), the plotting region (plt in par) will not be reset and one can add more information onto the image plot. (e.g., using functions such as points or lines). If TRUE, will reset plot parameters to the values before entering the function.
horizontal	If FALSE (default), the legend will be a vertical strip on the right side. If TRUE, the legend strip will be along the bottom.

lab.breaks	If breaks are supplied, these are text string labels to put at each break value. This is intended to label the axis on a transformed scale such as logs.
axis.args	Additional arguments for the axis function used to create the legend axis. (See example below adding a log scaling.)
legend.only	If TRUE, just add the legend to the plot in the plot region defined by the coordinates in smallplot. In the absence of other information, the range for the legend is determined from the zlim argument.
legend.args	Arguments for a complete specification of the legend label, e.g., if you need to rotate text or other details. This is in the form of a list and is just passed to the mtext function, and you will need to give both the side and line arguments for positioning. This usually will not be needed. (See example below.)
legend.cex	Character expansion to change the size of the legend label.
legend.line	Distance in units of character height (as in mtext) of the legend label from the color bar. Make this larger if the label collides with the color axis labels.
legend.mar	Width in characters of legend margin that has the axis. Default is 5.1 for a vertical legend and 3.1 for a horizontal legend.
legend.lab	Label for the axis of the color legend. Default is no label, as this is usually evident from the plot title.
legend.shrink	Amount to shrink the size of the legend relative to the full height or width of the plot.
legend.width	Width in characters of the legend strip. Default is 1.2, a little bigger than the width of a character.
lwd	Line width of bordering lines around pixels. This might need to be set less than 1.0 to avoid visible rounding of the pixel corners.
midpoint	This option is for the case of unequally spaced grids with x and y being matrices of grid point locations. If FALSE (default), the quadrilaterals will be extended to surround the z locations as midpoints. If TRUE, z values will be averaged to yield a midpoint value and the original grid points will be used to define the quadrilaterals. (See help on poly.image for details.) In most cases, midpoint should be FALSE to preserve exact values for z and let the grid polygons be modified.
nlevel	Number of color levels used in legend strip
smallplot	Plot coordinates for legend strip. If not passed, these will be determined within the function. Be sure to leave room for the axis labels. For example, if the legend is on the right side, smallplot= c(.85,.9,0,1) will leave (.1 in plot coordinates) for the axis labels to the right of the color strip. This argument is useful for drawing a plot with the legend that is the same size as the plots without legends.
verbose	If TRUE, prints intermediate information about setting up plots (for debugging).

Details

This is a function using the basic R graphics. The coding was done to make it easier for users to see how this function works and to modify it.

How this function works: The strategy for `image.plot` is simple: divide the plotting region into two smaller regions, `bigplot` and `smallplot`. The image goes in one and the legend in the other. This way, there is always room for the legend. Some adjustments are made to this rule by not shrinking the `bigplot` if there is already room for the legend strip and also sticking the legend strip close to the image plot. One can specify the plot regions explicitly by `bigplot` and `smallplot` if the default choices do not work. (Note that these are in figure coordinates.) There may be problems with small plotting regions in fitting both of these elements into the plot region, and one may have to change the default character sizes or margins to make things fit. Sometimes this function will not reset the type of margins correctly, and the "null" call `par(mar = par("mar"))` may help to fix this issue.

The text is too small! This always seems to happen as one is rushing to finish a talk and the figures have tiny default axis labels. Try just calling the function `fields.style` before plotting. List out this function to see what is changed; however, all text is increased by 20% in size.

Why "image.plot" and not "image"? The R base function `image` is very useful, but it is awkward to place a legend quickly. However, that said, if you are drawing several image plots and want a common legend, use the `image` function and just use `image.plot` to add the legend. See the example in the help file. Note that you can use `image` to draw a bunch of images and then follow with `image.plot` and `legend.only=TRUE` to add a common legend. (See examples below.)

Almost choropleths too: It should be noted that this image function is slightly different from a choropleth map because the legend is assuming that a continuous scale has been discretized into a series of colors. To use the image.plot function as a choropleth graphic, one would of course use the `breaks` option and for clarity perhaps code the different regions as different integer values. In addition, for publication quality, one would want to use the `legend.args` to add more descriptive labels at the midpoints in the color strip.

Relationship of x, y and z: If the z component is a matrix, then the user should be aware that this function locates the matrix element z[i,j] at the grid locations (x[i], y[j]). This is very different from simply listing out the matrix in the usual row/column tabular form. See the example below for details on the difference in formatting. What does one do if you do not really have the "z" values on a regular grid? See the functions `quilt.plot` and `as.image` to discretize irregular observations to a grid. If the values make sense as points on a smooth surface, see `Tps` and `fastTps` for surface interpolation.

Adding separate color to indicate the grid box boundaries. Sometimes you want to show the grid box borders to emphasize this is not a smooth surface. To our knowledge there is no easy way to do this as an option in `image`. But if your image is formatted in the "poly image" style where x and y are also matrices, you can use the poly.image (see the `border` argument above) option to draw in boundaries.

Grids with unequal spacing – quadrilateral pixels: If x and y are matrices that are a smooth transformation of a regular grid, then z[i,j] can be interpreted as representing the average value in a quadrilateral that is centered at x[i,j] and y[i,j] (`midpoint` TRUE). The details of how this cell is found are buried in `poly.image`, but it is essentially found using midpoints between the centers. If `midpoint` is FALSE, then x and y are interpreted as the corners of the quadrilateral cells. But what about z? The four values of z are now averaged to represent a value at the midpoint of the cell and this is what is used for plotting. Quadrilateral grids were added to help with plotting the gridded output of geophysical models where the regular grid is defined according to one map projection but the image plotting is required in another projection. Typically, the regular grid becomes distorted in a smooth way when this happens. See the regional climate example for an illustration of this

application. One can add border colors in this case easily because these choices are just passed on to the polygon function.

Adding the pixel grid for rectangular images: For adding the grid of pixel borders to a rectangular image, try this example after calling `image.plot`.

```
dx <- x[2] - x[1]
dy <- y[2] - y[1]
xtemp<- seq(  min( x)- dx/2, max(x)+ dx/2,
        length.out = length(x) +1)
ytemp<- seq(  min( y)- dy/2, max(y)+ dy/2,
        length.out = length(y) +1)
xline( xtemp, col="grey", lwd=2)
yline( ytemp, col="grey", lwd=2)
```

Here x and y are the x and y grid values from the image list.

Fine tuning color scales: This function gives some flexibility in tuning the color scale to fit the rendering of z values. This can either be a specially designed color scale with specific colors (see help on `designer.colors`), positioning the colors at specific points on the [0,1] scale, or mapping distinct colors to intervals of z. The examples below show how to do each of these. In addition, by supplying `lab.break` strings or axis parameters, one can annotate the legend axis in an informative manner.

Adding just the legend strip: Note that to add just the legend strip, all the numerical information one needs is the `zlim` argument and the color table! See examples for tricks in positioning.

About color tables: We like `tim.colors` as a default color scale, so if this is what you use, this can be omitted. Unfortunately, this is not the default for the `image` function. Another important color scale is `viridis()` from the viridis package. By and large, everyone seems to react positively to viridis – guess that is the point!

The topographic color scale (`topo.colors`) is also a close second, showing our geophysical bias. Users may find `larry.colors` useful for coding distinct regions in the style of a choropleth map. See also `terrain.colors` for a subset of the topo ones and `designer.colors` to "roll your own" color table. One nice option in this last function is to fix color transitions at particular quantiles of the data rather than at equally spaced intervals. For color choices, see how the `nlevels` argument figures into the legend and main plot number of colors. Also see the `colors` function for a listing of all the colors that come with the R base environment.

The details of placing the legend and dividing up the plotting real estate: It is surprising how hard it is to automatically add the legend! All "plotting coordinates" mentioned here are in device coordinates. The plot region is assumed to be [0,1]X[0,1] and plotting regions are defined as rectangles within this square. We found these easier to work with than user coordinates.

`legend.width` and `legend.mar` are in units of character spaces. These units are helpful in thinking about axis labels that will be put into these areas. To add more or less space between the legend and the image plot, alter the mar parameters. The default mar settings (5.1,5.1,5.1,2.1) leave 2.1 spaces for vertical legends and 5.1 spaces for horizontal legends.

There are always problems with default solutions to placing information on graphs, but the choices made here may be useful for most cases. The most annoying thing is that after using image.plot and adding information, the next plot that is made may have the slightly smaller plotting region set by the image plotting. The user should set `graphics.reset=TRUE` to avoid the plotting size

from changing. The disadvantage, however, of resetting the graphics is that one can no longer add additional graphics elements to the image plot. Note that filled.contour always resets the graphics but provides another mechanism to pass through plotting commands. Apparently `filled.contour`, while very pretty, does not work for multiple plots.

Multiple images: By keeping the `zlim` argument the same across images, one can generate the same color scale. (See the `image` help file.) One useful technique for a panel of images is to just draw the images with good old `image` and then use image.plot to add a legend to the last plot. (See example below for messing with the outer margins to make this work.) Usually a square plot (`pty="s"`) done in a rectangular plot region will have room for the legend stuck to the right side without any other adjustments. See the examples below for more complicated arrangements of multiple image plots and a summary legend. The reader is also referred to the package `autoimage` as a set of functions in base R to help with drawing multiple images and also more support for geographic coordinates.

Side Effects

After exiting, the plotting region may be changed to make it possible to add more features to the plot. To be explicit, `par()\$plt` may be changed to reflect a smaller plotting region that has accommodated room for the legend subplot.

If `xlim` and `ylim` are specified, the pixels may overplot the axis lines. Just use the `box` function to redraw them.

See Also

imagePlot, image,poly.image, filled.contour, quilt.plot, bubblePlot, plot.surface, add.image, colorBar, tim.colors, designer.colors

Examples

```
  x<- 1:10
  y<- 1:15
  z<- outer( x,y,"+")
  image.plot(x,y,z)

# or
  obj<- list( x=x,y=y,z=z)
  image.plot(obj, legend.lab="Sverdrups")

#################################################################
# the next sequence of examples explain how to quickly
# adapt this basic plot to include more features
# In another direction see the very last example where
# we use many of the setting in base R graphic to mimic a
# (beautiful) ggplot version.
#################################################################
#
# add some points on diagonal using standard plot function
#(with some clipping beyond 10 anticipated)

  points( 5:12, 5:12, pch="X", cex=3)
```

```
# in general image.plot will reset the plot window so you
# can add any feature that normally works in base R
# e.g. lines, text, contour, boxplots, ....
#
# adding breaks and distinct colors for intervals of z
# with and without lab.breaks

  brk<- quantile( c(z))
  image.plot(x,y,z, breaks=brk, col=rainbow(4))

# annotate legend strip with the  break point values and add a label

  image.plot(x,y,z, breaks=brk, col=rainbow(4),
                       lab.breaks=names(brk))
#
# compare to

  zp <-quantile(c(z), c( .05, .1,.5, .9,.95))
  image.plot(x,y,z,
     axis.args=list( at=zp, labels=names(zp) ) )

# a log scaling for the colors

  ticks<- c( 1, 2,4,8,16,32)
  image.plot(x,y,log(z), axis.args=list( at=log(ticks), labels=ticks))

# see help file for designer.colors to generate a color scale that adapts to
# quantiles of z.
# Add some color scales together here is an example of  5 blues to white to
# 5 reds
# with white being a specific size.
 colorTable<- designer.colors(11, c( "blue","white", "red") )
# breaks with a gap of 10 to 17 assigned the white color
 brks<- c(seq( 1, 10,,6), seq( 17, 25,,6))
 image.plot( x,y,z,breaks=brks, col=colorTable)
#
#fat (5 characters wide) and short (50% of figure)  color bar on the bottom
   image.plot( x,y,z,legend.width=5, legend.shrink=.5, horizontal=TRUE)

# adding a label with all kinds of additional arguments.
# use side=4 for vertical legend and side= 1 for horizontal legend
# to be parallel to axes. See help(mtext).

image.plot(x,y,z,
       legend.args=list( text="unknown units",
     col="magenta", cex=1.5, side=4, line=2))

# and finally add some grid lines
 dx <- x[2] - x[1]
 dy <- y[2] - y[1]
 xtemp<- seq(  min( x)- dx/2, max(x)+ dx/2,
         length.out = length(x) +1)
```

```
 ytemp<- seq(  min( y)- dy/2, max(y)+ dy/2,
         length.out = length(y) +1)
 xline( xtemp, col="grey", lwd=2)
 yline( ytemp, col="grey", lwd=2)

#################################################################
#### example using an irregular quadrilateral grid
#################################################################
data( RCMexample)

image.plot( RCMexample$x, RCMexample$y, RCMexample$z[,,1])
ind<- 50:75 # make a smaller image to show bordering lines
image.plot( RCMexample$x[ind,ind], RCMexample$y[ind,ind],
          RCMexample$z[ind,ind,1],border="grey50", lwd=2)

#################################################################
#### multiple images with a common legend
#################################################################
set.panel()

# Here is quick but quirky way to add a common legend to several plots.
# read about outer margin parameters in the \code{\link{par}} function.
# The idea is to leave some room in the margin and then at the end
# overplot the legend in this margin

par(oma=c( 0,0,0,4)) # margin of 4 spaces width at right hand side
set.panel( 2,2) # 2X2 matrix of plots

# now draw all your plots using usual image command
for (  k in 1:4){
  data<- matrix( rnorm(150), 10,15)
  image( data, zlim=c(-4,4), col=tim.colors())
# and just for fun add a contour plot
  contour( data, add=TRUE)
}

par(oma=c( 0,0,0,1))# reset margin to be much smaller.
image.plot( legend.only=TRUE, zlim=c(-4,4))

# image.plot tricked into  plotting in margin of old setting

set.panel() # reset plotting device


# combining the  usual image function and adding a legend
# first change margin for some more room
## Not run:
par( mar=c(10,5,5,5))
image( x,y,z, col=topo.colors(64))
image.plot( zlim=c(0,25), nlevel=64,legend.only=TRUE, horizontal=TRUE,
col=topo.colors(64))

## End(Not run)
```

```
#

#########################################################
##### Comparison to ggplot
#########################################################
# the following example was created as way avoid doing more important
# things
# Note how close base graphics can get to reproducing the ggplot style.

## Not run:
library(viridisLite)
library(ggplot2)

x<- 1:20
y<-  1:24
z<- outer( x, y, "+")


# ggplot version
  mesh<- expand.grid( x= x, y=y)
  mesh$z <- c(z)
  ggplot( data=mesh, aes( x=x, y=y, fill=z)) +
    geom_raster(interpolate= FALSE)  +
    scale_fill_continuous(type = "viridis")  +
    theme_bw()


# inflate range to give a margin around image
  xr<- range(x) +  c(-.08, .08)* diff( range(x))
  yr<- range(y) +  c(-.08, .08)* diff( range(y))

# changing these graphics parameters tends to push
# text closer to the axes.
  par( mgp=c(1.5,.5,0),mar=c(2.5,2.5,.5,1), cex=.8)

  image.plot(x,y,z,
             col = viridis(128),
   legend.shrink = .27,
            xlim = xr,
            ylim = yr,
    legend.width = 1.5,
      legend.mar = 3,
     legend.args = list( text = "z",
                         cex = .8,
                        side = 3,
                        line = .5)
   )


## End(Not run)
```

`image.smooth` *Kernel smoother for irregular 2-d data*

Description

Takes an image matrix and applies a kernel smoother to it. Missing values are handled using the Nadaraya/Watson normalization of the kernel.

Usage

```
## S3 method for class 'smooth'
image(x, wght = NULL, dx = 1, dy = 1,
    kernel.function = double.exp,
    aRange = 1, grid = NULL, tol = 1e-08, xwidth = NULL, ywidth = NULL,
    weights = NULL, theta=NULL, ...)

setup.image.smooth(nrow = 64, ncol = 64, dx = 1, dy = 1,
                   kernel.function = double.exp,
                   aRange = 1, xwidth = nrow * dx, ywidth = ncol * dx,
     lambda=NULL,theta=NULL, ...)
```

Arguments

`x`	A matrix image. Missing values can be indicated by NAs.
`wght`	FFT of smoothing kernel. If this is NULL, the default is to compute this object.
`grid`	A list with x and y components. Each is equally spaced, and they define the rectangular grid (see grid.list).
`dx`	Grid spacing in x direction
`dy`	Grid spacing in y direction
`kernel.function`	An R function that takes as its argument the *squared* distance between two points divided by the bandwidth. The default is exp(-abs(x)) yielding a normal kernel
`aRange`	the bandwidth or scale parameter.
`theta`	Same as aRange.
`xwidth`	Amount of zero padding in horizontal dimension in units of the grid spacing. If NULL, the default value is equal to the width of the image, the most conservative value, but possibly inefficient for computation. Set this equal to zero to get periodic wrapping of the smoother. This is useful to smooth a Mercator map projection.
`ywidth`	Same as xwidth but for the vertical dimension.
`weights`	Weights to apply when smoothing.
`tol`	Tolerance for the weights of the N-W kernel. This avoids kernel estimates that are "far" away from data. Grid points with weights less than tol are set to NA.

nrow	X dimension of image in setting up smoother weights
ncol	Y dimension of image
lambda	Smoothing parameter if smoother is interpreted in a spline-like way.
...	Other arguments to be passed to the kernel function

Details

The function works by taking convolutions using an FFT. The missing pixels are taken into account and the kernel smoothing is correctly normalized for the edge effects following the classical Nadaraya-Watson estimator. For this reason, the kernel does not have to be a density, as it is automatically normalized when the kernel weight function is found for the data. If the kernel has limited support, then the width arguments can be set to reduce the amount of computation. (See example below.) For multiple smoothing, compute the FFT of the kernel just once using `setup.image.smooth` and pass this as the wght argument to image.smooth. This will save an FFT in computations.

Value

The smoothed image in R image format. (A list with components x, y and z.) `setup.image.smooth` returns a list with components W, a matrix being the FFT of the kernel, dx, dy, xwidth and ywidth.

See Also

as.image, sim.rf, image.plot

Examples

```
# first convert precip data to the 128X128 discretized image format ( with
# missing  values to indicate where data is not observed)
#
data(RMprecip)
out<- as.image( RMprecip$y, x= RMprecip$x, nx=128, ny=128)
# out$z is the image matrix

dx<- out$x[2]- out$x[1]
dy<-  out$y[2] - out$y[1]

#
# grid scale in degrees and choose kernel bandwidth to be .25 degrees.

look<- image.smooth( out, aRange= .25)

# pass in a tophat kernel
topHat<- function( dd, h ){ ifelse( dd <= h^2, 1, 0)}
## dd is the distance squared
look2<- image.smooth( out, kernel.function=topHat, h=.8)

image.plot(look)
points( RMprecip$x)
US( add=TRUE, col="grey", lwd=2)

# to save on computation, decrease the padding with zeroes
```

```
# only pad 32 grid points around the margins of the image.

look<- image.smooth(out$z, dx=dx, dy=dy, aRange= .25, xwidth=32*dx,ywidth=32*dy)

# the range of these data is ~ 10 degrees  and so
# with a padding of 32 grid points  32*( 10/128) =  2.5
# about 10 standard deviations of the normal kernel so there is still
# lots of room for padding
# a minimal choice might be  xwidth = 4*(.25)= 1  4 SD for the normal kernel
# creating weighting object outside the call
# this is useful when one wants to smooth different data sets but on the
# same grid with the same kernel function
#

#
#  random fields from smoothing white noise with this filter.
#
set.seed(123)
test.image<- matrix( rnorm(128**2),128,128)
dx<- .1
dy<- .8

wght<- setup.image.smooth( nrow=128, ncol=128,  dx=dx, dy=dy,
             aRange=.25, xwidth=2.5, ywidth=2.5)
#
look<- image.smooth( test.image, dx=dx, dy=dy, wght)

# NOTE:   this is the same as using
#
#     image.smooth( test.image , 128,128), xwidth=2.5,
#                          ywidth=2.5, dx=dx,dy=dy, aRange=.25)
#
#   but the call to image.smooth is faster because the fft of kernel
#   has been precomputed.



# periodic smoothing in the horizontal dimension

look<- image.smooth( test.image , xwidth=1.5,
                        ywidth=2.5, dx=dx,dy=dy, aRange=1.5)
look2<- image.smooth( test.image , xwidth=0,
                        ywidth=2.5, dx=dx,dy=dy, aRange=1.5)
# compare these two
set.panel( 1,2)
image.plot( look, legend.mar=7.1)
title("free boundaries")
image.plot( look2, legend.mar=7.1) # look for periodic continuity at edges!
title("periodic boundary in horizontal")
set.panel(1,1)
```

image2lz *Some simple functions for subsetting images*

Description

These functions help in subsetting images or reducing their size by averaging adjacent cells.

Usage

```
crop.image(obj, loc=NULL,...)
which.max.matrix(z)
which.max.image(obj)
get.rectangle()
average.image(obj, Q=2)
half.image(obj)
in.poly( xd, xp, convex.hull=FALSE, inflation=1e-07)
in.poly.grid( grid.list,xp, convex.hull=FALSE, inflation=1e-07)
```

Arguments

obj	A list in image format with the usual x,y defining the grid and z a matrix of image values.
loc	A 2 column matrix of locations within the image region that define the subset. If not specified, then the image is plotted and the rectangle can be specified interactively.
Q	Number of pixels to average.
xd	A 2 column matrix of locations that are the points to check for being inside a polygon.
xp	A 2 column matrix of locations that are vertices of a polygon. The last point is assumed to be connected to the first.
convex.hull	If TRUE, then the convex hull of xp is used instead of the polygon.
grid.list	A list with components x and y specifying the 2-d grid values. (See help(grid.list) for more details.)
inflation	A small expansion factor to ensure that points precisely on the boundaries and vertices of the convex hull are included as members.
z	A matrix of numerical values
...	Graphics arguments passed to image.plot. This is only relevant when loc is NULL and the locator function is called via get.rectangle.

Details

If `loc` has more than 2 rows, then the largest rectangle containing the locations is used.

crop.image Creates a subset of the image `obj` by using the largest rectangle in the locations `loc`. This is useful if one needs to extract an image that is no bigger in extent than some data location. If locations are omitted, the parent image is plotted and the locations are taken from two mouse clicks on the image. Returned value is an image with appropriate `x,y` and `z` components.

get.rectangle Given an image, plots it and waits for two mouse clicks that are returned.

which.max.image Returns a list with components `x, y, z` , and `ind` giving the location of the maximum and value of the maximum in the image based on the grid values and also on the indices of the image matrix.

average.image, half.image Takes the passed image and averages the pixel values and adjusts the grid to create an image that has a smaller number of elements. If `Q=2` in `average.image`, it has the same effect as `half.image` but might be slower – if the original image is mXn then half image will be an image (m/2)X(n/2). This raises the question of what happens when m or n is odd or when (m/Q) or (n/Q) are not integers. In either case, the largest rows or columns are dropped. (For large `Q` the function might be modified to drop about half the pixels at both edges.)

in.poly, in.poly.grid Determines whether the points xd,yd are inside a polygon or outside. Return value is a logical vector with TRUE being inside or on boundary of polygon. The test expands the polygon slightly in size (on the order of single precision zero) to include points that are at the vertices. `in.poly` does not really depend on an image format; however, the grid version `in.poly.grid` is more efficient for considering the locations on a regular grid. See also `in.land.grid` that is hard-coded to work with the fields world map.

Author(s)

Doug Nychka

See Also

drape.plot, image.plot, interp.surface, interp.surface.grid, in.land.grid

Examples

```
data(RMelevation)

# region defining Colorado Front Range

  loc<- rbind( c(-106.5, 40.8),
               c(-103.9, 37.5))

# extract elevations for just CO frontrange.
   FR<- crop.image(RMelevation, loc)
   image.plot( FR, col=terrain.colors(256))

   which.max.image( FR)
```

```
# average cells  4 to 1 by doing this twice!
   temp<-  half.image( RMelevation)
   temp<- half.image( temp)

# or in one step
   temp<-  average.image( RMelevation, Q=4)-> temp
   image.plot( temp, col=terrain.colors(256))

# a polygon (no special meaning entered with just locator)
x1p<- c(
 -106.2017, -104.2418, -102.9182, -102.8163, -102.8927, -103.3254, -104.7763,
 -106.5581, -108.2889, -109.1035, -109.3325, -108.7980)

x2p<- c(
  43.02978, 42.80732, 41.89727, 40.84566, 39.81427, 38.17618, 36.53810, 36.29542,
  36.90211, 38.29752, 39.45025, 41.02767)
xp<- cbind( x1p,x2p)

 image.plot( temp)
 polygon( xp[,1], xp[,2], lwd=2)

# find all grid points inside poly
 fullset<- make.surface.grid( list( x= temp$x, y= temp$y))
 ind<-  in.poly( fullset,xp)

# take a look
 plot( fullset, pch=".")
 polygon( xp[,1], xp[,2], lwd=2)
 points( fullset[ind,], pch="o", col="red", cex=.5)

# masking out the image NA == white in the image plot
 temp$z[!ind] <- NA
 image.plot( temp)
 polygon( xp[,1], xp[,2], lwd=2)

# This is more efficient for large grids:
# because the large number of grid location ( xg above) is
# never explicitly created.

 ind<- in.poly.grid( list( x= temp$x, y= temp$y), xp)

# now use ind in the same way as above to mask points outside of polygon
```

`imagePlot` *Draws an image plot with a legend strip for the color scale based on either a regular grid or a grid of quadrilaterals.*

Description

These functions combine the R image function with some automatic placement of a colorbar legend. `imagePlot` is backwardly compatible with the older function `image.plot`. `plotMatrix` gives an image plot but shuffles the matrix so that it is in its usual tabular order. There are also some supporting functions to place color bars on other kinds of plots.

Usage

```
imagePlot(..., add = FALSE, breaks = NULL, nlevel = 64, col =
                  NULL, horizontal = FALSE, legend.shrink = 0.9,
                  legend.width = 1.2, legend.mar = ifelse(horizontal,
                  3.1, 5.1), legend.lab = NULL, legend.line = 2,
                  graphics.reset = FALSE, oldPar = NULL, bigplot = NULL,
                  smallplot = NULL, legend.only = FALSE, lab.breaks =
                  NULL, axis.args = NULL, legend.args = NULL, legend.cex
                  = 1, midpoint = FALSE, border = NA, lwd = 1,
                  lowerTriangle = FALSE, upperTriangle = FALSE, asp =
                  NA, verbose = FALSE)

divMap(x, y, z, zForce = NULL, zCap = NULL, col = NULL,
                  worldCol = "magenta1", lty = 1, lwd = 1, brewerPal =
                  "Spectral", rev = TRUE, wrapVal = c(0, 360), ...)

colorBar(breaks = NULL, smallplot, colorTable = tim.colors(),
                  horizontal = FALSE, lab.breaks = NULL, axis.args =
                  NULL, legend.lab = NULL, legend.line = 2, legend.args
                  = NULL, legend.cex = 1, lowerTriangle = FALSE,
                  upperTriangle = FALSE, zlim = NULL)

addColorBar(zlim, col = turbo(256), ...)

addLegend()

addColorBarTriangle(lowerColor=NULL,
                                  upperColor=NULL,
                                  horizontal=TRUE)

plotMatrix(A, M = 5, N = 5, ...)
```

Arguments

`...` **imagePlot** The usual arguments to the `image` function as x, y, or z or as a list with x,y,z as components. One can also include a `breaks` argument for an unequally spaced color scale with color scale boundaries at the breaks (see example below). If a "quadrilateral grid", arguments must be explicitly x, y and z with x and y being matrices of dimensions equal to, or one more than, z giving the grid locations. The basic concept is that the coordinates of x and y still define a grid, but the image cells are quadrilaterals rather than being restricted to rect-

angles. NOTE: graphical arguments passed here will only have impact on the image plot. To change the graphical defaults for the legend, use the individual legend arguments, `axis.args`, or `legend.arg` listed below.

addColorBar, plotMatrix Additional arguments for the `imagePlot` function for setting up the legend.

`add` If TRUE, add the image and a legend strip to the existing plot.

`asp` Plot aspect. The same function as in the `plot` or `image` functions.

`axis.args` A list giving additional arguments for the axis function used to create the legend axis. Some typical ones are `at` and `labels` and specify `cex` to control the size of the labels. (See example below adding a log scaling.)

`A` Matrix to plot as an image.

`bigplot` Plot coordinates for image plot. If not passed, these will be determined within the function.

`border` This only works if x and y are matrices – if NA, the quadrilaterals will have a border color that is the same as the interior color. Otherwise, this specifies the color to use.

`breaks` Break points in sorted order to indicate the intervals for assigning the colors. Note that if there are nlevel colors, there should be (nlevel+1) breakpoints. If `breaks` is not specified, (nlevel+1) equally spaced breaks are created where the first and last bin have their midpoints at the minimum and maximum values in `z` or at `zlim`.

`brewerPal` Name of RColorBrewer palette used to create color scale. Default for `divMap` is `'spectral'`.

`col` Color table to use for image (see help file on image for details). Default is a pleasing range of 64 divisions suggested by Tim Hoar and is similar to the MATLAB (TM) jet color scheme. Note that if `breaks` is specified, there must be one less color specified than the number of breaks. Or use `viridisLite::rocket`. For `divMap`, the default scale is the reversed spectral from RColorBrewer.

`colorTable` The colors for the color scale – the same as col.

`graphics.reset` If FALSE (default), the plotting region (plt in par) will not be reset and one can add more information onto the image plot. (e.g., using functions such as points or lines). If TRUE, will reset plot parameters to the values before entering the function.

`horizontal` If FALSE (default), the legend will be a vertical strip on the right side. If TRUE, the legend strip will be along the bottom.

`lab.breaks` If breaks are supplied, these are text string labels to put at each break value. This feature is useful, for example, to label the legend axis on a transformed scale such as logs. It is also used in the `bubblePlot` function to label the colors when z is categorical.

`legend.only` If TRUE, just add the legend to the plot in the plot region defined by the coordinates in smallplot. In the absence of other information, the range for the legend is determined from the `zlim` argument.

`legend.args` Arguments for a complete specification of the legend label, e.g., if you need to rotate text or other details. This is in the form of a list and is just passed to the

	mtext function, and you will need to give both the side and line arguments for positioning. This usually will not be needed. (See example below.)
legend.cex	Character expansion to change the size of the legend label.
legend.line	Distance in units of character height (as in mtext) of the legend label from the color bar. Make this larger if the label collides with the color axis labels.
legend.mar	Width in characters of legend margin that has the axis. Default is 5.1 for a vertical legend and 3.1 for a horizontal legend.
legend.lab	Label for the axis of the color legend. Default is no label, as this is usually evident from the plot title.
legend.shrink	Amount to shrink the size of the legend relative to the full height or width of the plot.
legend.width	Width in characters of the legend strip. Default is 1.2, a little bigger than the width of a character.
lowerColor	The name of a color for the triangle added to the lower end of the color bar.
lowerTriangle	If TRUE, a triangle will be added to the lower end of the color bar using the first color in the color table. (i.e., the color of col[1]). If FALSE, a normal color strip will be drawn.
lwd	Line width of bordering lines around pixels. This might need to be set less than 1.0 to avoid visible rounding of the pixel corners.
lty	Line type for the world map in divMap following the usual base R graphics styles.
M	Number of rows to label in plotMatrix
midpoint	This option is for the case of unequally spaced grids with x and y being matrices of grid point locations. If FALSE (default), the quadrilaterals will be extended to surround the z locations as their midpoints. If TRUE, z values will be averaged to yield a midpoint value and the original grid points will be used to define the quadrilaterals. (See help on poly.image for details.) In most cases, midpoint should be FALSE to preserve exact values for z and let the grid polygons be modified.
N	Number of columns to label in plotMatrix
nlevel	Number of color levels used in legend strip
oldPar	par arguments that the function should reset to after drawing the figure.
smallplot	Plot coordinates for legend strip. If not passed, these will be determined within the function. Be sure to leave room for the axis labels. For example, if the legend is on the right side, smallplot= c(.85,.9,0,1) will leave (.1 in plot coordinates) for the axis labels to the right of the color strip. This argument is useful for drawing a plot with the legend that is the same size as the plots without legends.
rev	Reverse the order of the color scale.
worldCol	Color of world map outline in divMap.
wrapVal	How the image is oriented. Typically c(-180,180) for a centering around the prime meridian or c(0,360), the default, centered on the Pacific Ocean.

upperColor	The color for the triangle added to the upper end of the color bar.
upperTriangle	If TRUE, a triangle is added to the upper end of the color bar using the last color in the color table. (i.e., the col argument). If FALSE, a normal color strip will be drawn.
x	Horizontal grid values.
y	Vertical grid values.
verbose	If TRUE, prints intermediate information about setting up plots (for debugging).
z	A matrix of image values.
zCap	Maximum range for color scale. All image values in absolute value greater than this are reset to this maximum times their sign.
zForce	z limit in addition to the image and zCap value. The image color scale range is found as +/- max(abs(z),zForce,zCap).
zlim	For addLegend, the range of the color scale. Default is to use the range set by most recent call to color.scale

Details

imagePlot is a function using base R graphics. We like working directly with the basic functions for teaching because they make the R user think about building a complicated figure using simpler elements. Moreover, the coding was done to make it easier for users to see how this function works and to modify it. Note that the comments for these functions are stripped out when the package is loaded. Commented code can be found by going to the fields GitHub repository. (See the repository README file for more directions.) This will have the same functionality as the older image.plot but calls the colorBar function in place of direct coding in the main function. The functions colorBar and addColorBarTriangle are used internally and included for completeness.

How this function works:

This function works by splitting the plotting region into two parts, putting the image in one and the legend in the other. After the legend is added, the plot region is reset to the main image plot. This function also allows for plotting quadrilateral cells in the image format that often arise from regular grids transformed with a map projection or a scaling and rotation of coordinates. Finally, see the last example where this function can create a similar graphic to the ggplot package but using all base R graphics. Two additional functions are provided to add a color scale to other kinds of figures.

The strategy for imagePlot is simple: divide the plotting region into two smaller regions, bigplot and smallplot. The image goes in one and the legend in the other. This way, there is always room for the legend. Some adjustments are made to this rule by not shrinking the bigplot if there is already room for the legend strip and also sticking the legend strip close to the image plot. The functionality has evolved over many uses of image plots for geophysical data and the examples given below show the flexibility of this function. Also, this function is backwardly compatible with the older image.plot but includes adding endcap triangles to the color bar to indicate values beyond the range of the color scale. Also, the function uses colorBar to add the legend strip, a function that is useful on its own in other applications.

Figure real estate One can specify the plot regions explicitly by bigplot and smallplot if the default choices do not work. (Note that these are in figure coordinates.) There may be problems with small plotting regions in fitting both of these elements into the plot region, and one may have

to change the default character sizes or margins to make things fit. Sometimes this function will not reset the type of margins correctly, and the "null" call `par(mar = par("mar"))` may help to fix this issue.

The text is too small! This always seems to happen as one is rushing to finish a talk and the figures have tiny default axis labels. Try just calling the function `fields.style` before plotting and list out this function to see how to make further changes. Also, there are several calling arguments in the function that you can change to adjust the placement and size of the axes and labels.

Why "imagePlot" and not "image"? The R base function `image` is very useful, but it is awkward to place a legend quickly. However, that said, if you are drawing several image plots and want a common legend, use the `image` function and just use `imagePlot` to add the legend. See the example in the help file. Note that you can use `image` to draw a bunch of images and then follow with `imagePlot` and `legend.only=TRUE` to add a common legend. `addColorBar` is a convenient wrapper to this addition. (See examples below.)

Almost choropleths too! This image function is slightly different from a choropleth map because the legend is assuming that a continuous scale has been discretized into a series of colors. Using the imagePlot function to create a choropleth graphic involves an extra step of first recoding the image matrix into a set of integers that correspond to the colors in the color table. With this new matrix as the image, use the `legend.args` list to place labels for each of the colors beside the color strip. For example, if one has used `N` colors in `col`, then the locations of the color strip run from 1 to N, and one could place the `N` color labels at the positions `(1:N)` along the axis. Note that this two-step technique could also be adapted if one had a nonlinear color scale and wanted to identify the values.

Relationship of x, y and z: If the z component is a matrix, then the user should be aware that this function locates the matrix element z[i,j] at the grid locations (x[i], y[j]). This is very different from simply listing out the matrix in the usual row/column tabular form. See the example below for details on the difference in formatting. What does one do if you do not really have the "z" values on a regular grid? See the functions `quilt.plot`, `bubblePlot` and `as.image` to handle irregular observations. If the values make sense as points on a smooth surface, see `Tps` and `fastTps` for surface interpolation.

Adding separate color to indicate the grid box boundaries. Sometimes you want to show the grid box borders to emphasize this is not a smooth surface. To our knowledge there is no easy way to do this as an option in `image`. But if your image is formatted in the "poly image" style where x and y are also matrices, you can use the poly.image (see the `border` argument above) option to draw in boundaries.

Grids with unequal spacing – quadrilateral pixels: If x and y are matrices that are a smooth transformation of a regular grid, then z[i,j] can be interpreted as representing the average value in a quadrilateral that is centered at x[i,j] and y[i,j] (`midpoint` TRUE). The details of how this cell is found are buried in `poly.image`, but it is essentially found using midpoints between the centers. If `midpoint` is FALSE, then x and y are interpreted as the corners of the quadrilateral cells. But what about z? The four values of z are now averaged to represent a value at the midpoint of the cell and this is what is used for plotting. Quadrilateral grids were added to help with plotting the gridded output of geophysical models where the regular grid is defined according to one map projection but the image plotting is required in another projection. Typically, the regular grid becomes distorted in a smooth way when this happens. See the regional climate example for an illustration of this application. One can add border colors in this case easily because these choices are just passed on to the polygon function.

Adding the pixel grid for rectangular images: For adding the grid of pixel borders to a rectangular image, try this example after calling `imagePlot`.

```
dx <- x[2] - x[1]
dy <- y[2] - y[1]
xtemp<- seq(  min( x)- dx/2, max(x)+ dx/2,
        length.out = length(x) +1)
ytemp<- seq(  min( y)- dy/2, max(y)+ dy/2,
        length.out = length(y) +1)
xline( xtemp, col="grey", lwd=2)
yline( ytemp, col="grey", lwd=2)
```

Here x and y are the x and y grid values from the image list.

Fine tuning color scales: This function gives some flexibility in tuning the color scale to fit the rendering of z values. This can either be a specially designed color scale with specific colors (see help on `designer.colors`), positioning the colors at specific points on the [0,1] scale, or mapping distinct colors to intervals of z. The examples below show how to do each of these. In addition, by supplying `lab.break` strings or axis parameters, one can annotate the legend axis in an informative manner.

Adding just a legend strip/color bar: Note that to add just the legend strip, all the numerical information one needs is the `zlim` argument and the color table! `addColorBar` has the same function as using `imagePlot` with `legend.only=TRUE` – and might be easier to remember! See the examples for tricks in positioning.

About color tables/scales: We like the `turbo` color scale (the default) as a default color scale, so if this is what you use, this can be omitted. Unfortunately, this is not the default for the `image` function. Another important color scale is `viridis()` from the viridisLite package. By and large, everyone seems to react positively to viridis – guess that is the point!

The topographic color scale (`topo.colors`) is also a close second (and also see `snow.colors`), showing our geophysical bias. Users may find `larry.colors` useful for coding distinct regions in the style of a choropleth map. See also `terrain.colors` for a subset of the topo ones and `designer.colors` to "roll your own" color table. One nice option in this last function is to fix color transitions at particular quantiles of the data rather than at equally spaced intervals. For color choices, see how the `nlevels` argument figures into the legend and main plot number of colors. Also see the `colors` function for a listing of all the colors that come with the R base environment.

The details of placing the legend and dividing up the plotting real estate: It is surprising how hard it is to automatically add the legend! All the more reason to have base R graphics code where one can see what is going on and possibly edit the choices. All "plotting coordinates" mentioned here are in device coordinates. The plot region is assumed to be [0,1]X[0,1] and plotting regions are defined as rectangles within this square. We found these easier to work with than user coordinates.

`legend.width` and `legend.mar` are in units of character spaces. Same as the mar choices for a plot region. These units are helpful in thinking about axis labels that will be put into these areas. To add more or less space between the legend and the image plot, alter the mar parameters. The default mar settings (5.1,5.1,5.1,2.1) leave 2.1 spaces for vertical legends and 5.1 spaces for horizontal legends.

There are always problems with default solutions to placing information on graphs, but the choices made here may be useful for most cases. The most annoying thing is that after using imagePlot and adding information, the next plot that is made may have the slightly smaller plotting region set by the image plotting. The user should set `graphics.reset=TRUE` to avoid the plotting size from changing. The disadvantage, however, of resetting the graphics is that one can no longer add additional graphics elements to the image plot. Note that filled.contour always resets the graphics

but provides another mechanism to pass through plotting commands. Apparently `filled.contour`, while very pretty, does not work for multiple plots.

About the `addLegend` **function** This is a more recent idea and came about to create a panel of plots and easily add a legend in the outer margin. Note that this function simply resets the plot region to be [0,1]X[0,1] to make it easier to place a legend. You will have to use the legend function to do this.

See `panelPlotCommonAxes` for examples.

More complete graphics languages, such as that in ggplot, do not need such functions because the entire graphics segment is parsed to create the complete figure. In this way, room for a color scale can be created automatically because the plot function is informed of the need for space for the legend. The functions proposed here are a simple workaround to create these figures using base R graphics.

Other packages `levelplot` that is part of the lattice package has a very similar function to imagePlot and a formula syntax in the call. The `geom_raster` for setting up a graphics object within `ggplot` is another alternative for image plots with legends. See the last example to compare the steps in creating an image plot using `imagePlot` that is close to the ggplot version. Mostly this involves resetting base graphics parameters using the `par` function, but gives an appreciation of the complexity built into the ggplot default graphic choices.

Multiple images, single color bar: By keeping the `zlim` argument the same across images, one can generate the same color scale. (See the `image` help file.) One useful technique for a panel of images is to just draw the images with good old `image` and then use `imagePlot` or `addColorBar` to add a legend to the last plot. (See example below for messing with the outer margins to also make this work.) Moreover, a square plot (`pty="s"`) done in a rectangular plot region will usually have room for the legend stuck to the right side without any other adjustments. See the examples below for more complicated arrangements of multiple image plots and a summary color bar. The reader is also referred to the package `autoimage` as a set of functions in base R to help with drawing multiple images and also more support for geographic coordinates.

Side Effects

After exiting, the plotting region may be changed to make it possible to add more features to the plot. To be explicit, `par()\$plt` may be changed to reflect a smaller plotting region that has accommodated room for the legend subplot.

If `xlim` and `ylim` are specified, the pixels may overplot the axis lines. Just use the `box` function to redraw them.

`addLegend` purposefully messes with the graphics coordinates to make it easier to locate additional legends and other features. To reset, use `oldPar<- par()` before `addLegend()`, and then, after drawing legends, `par(oldPar)` to reset to the original coordinates.

See Also

image, poly.image, filled.contour, quilt.plot, plot.surface, add.image, colorbar.plot, tim.colors, designer.colors

Examples

```
x<- 1:10
```

```
  y<- 1:15
  z<- outer( x,y,"+")
  imagePlot(x,y,z)

# to view z in the usual  matrix indexing:
  plotMatrix( z)

# or
  obj<- list( x=x,y=y,z=z)
  imagePlot(obj, legend.lab="Sverdrups")

# to test add some points on diagonal using standard plot function
# (with some clipping beyond 10 anticipated)

  points( 5:12, 5:12, pch="X", cex=3)

# in general imagePlot will reset the plot window so you
# can add any feature that normally works in base R
# e.g. lines, text, contour, boxplots, ....
#

###################################################################
# the next sequence of examples explain how to quickly
# adapt this basic plot to include more features
###################################################################
#
# adding explicit breaks and distinct colors for intervals of z

  brk<- quantile(c(z), c(0, .1,.25, .5, .75, .9, 1.0))
  imagePlot(x,y,z, breaks=brk, col=topo.colors(6))

# last bins are depicted as end triangles on the color bar.
  imagePlot(x,y,z, breaks=brk, col=topo.colors(6),
           lowerTriangle=TRUE, upperTriangle=TRUE)

# NOTE:  the image  function does not use -Inf and Inf for the breaks argument
# and so if one  wants the triangles at ends to indicate values beyond the
# range of the color bar one has to create the  breaks "by hand" when the
# two outer bins will define the values for the triangles.

# annotate legend strip with the  break point values and add a label
  imagePlot(x,y,z, breaks=brk, col=rainbow(6),
                       lab.breaks=names(brk))
#
# compare to default color scale and special labels on color bar

  zp <-quantile(c(z), c( .05, .1,.25, .5,.75, .9,.95))
  imagePlot(x,y,z,
     axis.args=list( at=zp, labels=names(zp) ) )

# a log scaling for the colors

  ticks<- c( 1, 2,4,8,16,32)
```

```
  imagePlot(x,y,log(z), axis.args=list( at=log(ticks), labels=ticks))

# see help(designer.colors) to generate a color scale that adapts to
# quantiles of z.
# Add some color scales together here is an example of5 blues to white to 5 reds
# with white being a specific size.

 colorTable<- designer.colors(11, c( "blue","white", "red") )

# breaks with a gap of 10 to 17 assigned the white color
 brks<- c(seq( 1, 10,,6), seq( 17, 25,,6))
 imagePlot( x,y,z,breaks=brks, col=colorTable)
#
#fat (5 characters wide) and short (50% of figure)  color bar on the bottom
   imagePlot( x,y,z,legend.width=5, legend.shrink=.5, horizontal=TRUE)

# adding a label and all kinds of additional arguments to color bar
# use side=4 for vertical legend and side= 1 for horizontal legend
# to be parallel to axes. See help(mtext).

imagePlot(x,y,z,
       legend.args=list( text="unknown units",
     col="magenta", cex=1.5, side=4, line=2))

# and finally add some grid lines
 dx <- x[2] - x[1]
 dy <- y[2] - y[1]
 xtemp<- seq(  min( x)- dx/2, max(x)+ dx/2,
         length.out = length(x) +1)
 ytemp<- seq(  min( y)- dy/2, max(y)+ dy/2,
         length.out = length(y) +1)
 xline( xtemp, col="grey", lwd=2)
 yline( ytemp, col="grey", lwd=2)

##################################################################
#### example using an irregular quadrilateral grid
##################################################################
data( RCMexample)

imagePlot( RCMexample$x, RCMexample$y, RCMexample$z[,,1])
ind<- 50:75 # make a smaller image to show bordering lines
imagePlot( RCMexample$x[ind,ind], RCMexample$y[ind,ind], RCMexample$z[ind,ind,1],
                                     border="grey50", lwd=2)

##################################################################
#### multiple images with a common legend
##################################################################
set.panel()

# Here is quick but quirky way to add a common legend to several plots.
# The idea is leave some room in the margin and then at the end
# overplot the legend in this margin
```

```
par(oma=c( 0,0,0,5)) # outer margin of 5 spaces at right hand side

set.panel( 2,2) # 2X2 matrix of plots

# now draw all your plots using usual image command
for (  k in 1:4){
  data<- matrix( rnorm(150), 10,15)
  image( data, zlim=c(-4,4), col=tim.colors())
# and just for fun add a contour plot
  contour( data, add=TRUE)
}

addColorBar( zlim=c(-4,4), col=turbo(256), legend.shrink=.5)

# equivalent to:  imagePlot( legend.only=TRUE, zlim=c(-4,4),
 #.  col=turbo(256), legend.shrink=.5)


## Not run:
# drawing and saving the plot to a pdf file.
pdf("test.pdf", width=6, height=4)
par( mar=c( 10,5,2,2) ) # you may have to adjust these.
plot( 1:10, 1:10)
imagePlot( zlim=c(0,25), legend.only=TRUE, horizontal = TRUE)
dev.off()

## End(Not run)


# combining the  usual image function and adding a legend
# first change margin for some more room
## Not run:
par( mar=c(10,5,5,5))
image( x,y,z, col=topo.colors(64))
imagePlot( zlim=c(0,25), nlevel=64,legend.only=TRUE,
            horizontal=TRUE,
            col=topo.colors(64))

## End(Not run)


#########################################################
##### Nathan's div plot
#########################################################

data(LENSExample)
# default RBrewer spectral color scale.
divMap(LENSExample$x, LENSExample$y,
LENSExample$z,worldCol="white", lwd=.5,
zCap=2, wrapVal=c(-180, 180))


#########################################################
```

```
##### Comparison to ggplot
#########################################################
# The following example was created as way avoid doing more important
# things
# Note how close base graphics can get to reproducing the ggplot style.
# and how a first cut imagePlot(x,y,z, col = viridis(128) ) is probably
# acceptable in most cases for fast EDA!

# Surprisingly the hardest feature is to add the grey grid lines behind
# the image. This can be done but adds more complexity to the imagePlot code.
#

## Not run:
library( viridis)
library( ggplot2)

x<- 1:20
y<-  1:24
z<- outer( x, y, "+")


# ggplot version
  mesh<- expand.grid( x= x, y=y)
  mesh$z <- c(z)
  ggplot( data=mesh, aes( x=x, y=y, fill=z)) +
    geom_raster(interpolate= FALSE)  +
    scale_fill_continuous(type = "viridis")  +
    theme_bw()
# reset to a single plot
 set.panel()

# inflate range to give a margin around image
  xr<- range(x) +  c(-.08, .08)* diff( range(x))
  yr<- range(y) +  c(-.08, .08)* diff( range(y))

# changing these graphics parameters tends to push
# text closer to the axes.
  par( mgp=c(1.5,.5,0), mar=c(2.5,2.5,.5,1), cex=.8)

  imagePlot(x,y,z,
             col = viridis(128),
   legend.shrink = .27,
            xlim = xr,
            ylim = yr,
    legend.width = 1.5,
      legend.mar = 3,
     legend.args = list( text = "z",
                         cex = .8,
                        side = 3,
                        line = .5)
   )
```

```
## End(Not run)
```

`interp.surface` *Fast bilinear interpolator from a grid.*

Description

Uses bilinear weights to interpolate values on a rectangular grid to arbitrary locations or to another grid.

Usage

```
interp.surface(obj, loc)
interp.surface.grid(obj, grid.list)
interp.surface.FFT(obj, M)
fillGrid( gridList, M)
```

Arguments

`obj`	A list with components x,y, and z in the same style as used by contour, persp, image, etc. x and y are the X and Y grid values and z is a matrix with the corresponding values of the surface. For the FFT method there must be an odd number of grid points.
`loc`	A matrix of (irregular) locations to interpolate. First column of loc is the X coordinates and second is the Y's.
`grid.list`	A list with components x and y describing the grid to interpolate. The grids do not need to be equally spaced.
`gridList`	Just a newer name for a grid list object.
`M`	A multiple of the source grid to interpolate to create the target grid. E.g., if M=5, then a 10X20 source grid in `obj` will give an interpolated grid of (10*5)X(20*5). See the `fillGrid` function that creates the larger grid.

Details

interp.surface Here is a brief explanation of the interpolation: Suppose that the location, (locx, locy) lies in between the first two grid points in both x and y. That is, locx is between x1 and x2 and locy is between y1 and y2. Let ex= (locx-x1)/(x2-x1) and ey= (locy-y1)/(y2-y1). The interpolant is

(1-ex)(1-ey)*z11 + (1- ex)(ey)*z12 + (ex)(1-ey)*z21 + (ex)(ey)*z22

Where the z's are the corresponding elements of the Z matrix.

Note that bilinear interpolation can produce some artifacts related to the grid and not reproduce higher behavior in the surface. For example, the extrema of the interpolated surface will always be at the parent grid locations. There is nothing special about interpolating to another grid; this function just includes a `for` loop over one dimension and a call to the function for irregular locations. It was included in fields for convenience, since the grid format is so common.

See also the akima package for fast interpolation from irregular locations. Many thanks to Jean-Olivier Irisson for making this code more efficient and concise.

interp.surface.FFT This version does the interpolation via the usual sine /cosine basis and the FFT. This method only makes sense for interpolating to a target grid that is a refinement of the source grid. E.g., if M=5, then a 10X20 source grid in `obj` will give an interpolated grid of (10*5)X(20*5) = 50X200. The endpoints of this grid match the endpoints of the source.

Note the FFT interpolation is "C infinity" accurate, which means in practice that this method will do well for smooth fields. Sharp changes will induce the usual Gibbs oscillations around the large changes. The interpolation is also periodic in both dimensions (a torus) and so might give strange results for non-periodic fields. Pad the edges to mitigate this artifact.

The algorithm in brief is: 1) FFT of the source image/matrix. 2) Stuff this into corners of a larger matrix of zeroes and of the size of the refined grid. 3) Inverse FFT of stuffed matrix.

Value

interp.surface A vector of interpolated values. NAs are returned for regions of the z matrix that are NA and also for locations outside of the range of the parent grid.

interp.surface.grid Interpolated values using bilinear interpolation in the list/image format with components: `x`, `y`, `z`.

interp.surface.FFT Interpolated values using the FFT method in an image format with the grid refined by the factor `M`.

See Also

image.smooth, as.surface, as.image, imagePlot, image.plot, fastTps

Examples

```
#
# evaluate an image object  at a finer grid
#

data( lennon)
# create an example in the right list format like image or contour or persp.
obj<- list( x= 1:21, y=1:21, z= lennon[ 201:221, 201:221])

set.seed( 123)
# lots of random points
N<- 500
loc<- cbind( runif(N)*20, runif(N)*20)
z.new<- interp.surface( obj, loc)
# compare the image with bilinear interpolation at scattered points
set.panel(2,2)
image.plot( obj)
quilt.plot( loc, z.new)

# sample at 100X100 equally spaced points on a grid

grid.list<- list( x= seq( 1,20,,100), y=  seq( 1,20,,100))
```

```
interp.surface.grid( obj, grid.list)-> look

# this will give an error in the FFT version because
# there are an even number of grid points for x.
# objTest<- list( x= 1:20, y=1:21, z= lennon[ 201:220, 201:221])
# look2<- interp.surface.FFT( objTest, M=10)

look2<- interp.surface.FFT( obj, M=20)-> look2
# take a look
set.panel(2,2)
image.plot( obj)
image.plot( look)
image.plot( look2)
```

`Krig` *Kriging surface estimate*

Description

Fits a surface to irregularly spaced data. The Kriging model assumes that the unknown function is a realization of a Gaussian random spatial process. The assumed model is additive Y = P(x) + XMat(X) + e, where P is a low order polynomial and XMat is a mean zero, Gaussian stochastic process with a covariance that is unknown up to a scale constant. The main advantages of this function are the flexibility in specifying the covariance as an R language function and also the supporting functions plot, predict, predictSE, surface for subsequent analysis. Krig also supports a correlation model where the mean and marginal variances are supplied.

Usage

```
Krig(x, Y, cov.function = "stationary.cov", lambda = NA, df = NA, GCV = FALSE,
                XMat = NULL,
                Z = NULL,
                cost = 1, weights = NULL,
                m = 2, nstep.cv = 200, scale.type = "user", x.center =
                rep(0, ncol(x)), x.scale = rep(1, ncol(x)), sigma =
                NA, tau2 = NA, method = "REML", verbose = FALSE,
                null.function = "Krig.null.function", wght.function =
                NULL, offset = 0, na.rm = TRUE, cov.args = NULL,
                chol.args = NULL, null.args = NULL, wght.args = NULL,
                W = NULL, give.warnings = TRUE, mean.obj = NA, sd.obj
                = NA, ...)

## S3 method for class 'Krig'
fitted(object,...)

## S3 method for class 'Krig'
```

```
coef(object,...)

resid.Krig(object,...)
```

Arguments

chol.args	Arguments to be passed to the Cholesky decomposition in Krig.engine.fixed. The default if NULL, assigned at the top level of this function, is list(pivot=FALSE). This argument is useful when working with the sparse matrix package.
cov.args	A list with the arguments to call the covariance function (in addition to the locations).
cov.function	Covariance function for data in the form of an R function (see Exp.simple.cov as an example). Default assumes that correlation is an exponential function of distance. See also `stationary.cov` for more general choices of covariance shapes. `exponential.cov` will be faster if only the exponential covariance form is needed.
cost	Cost value used in GCV criterion. Corresponds to a penalty for increased number of parameters. The default is 1.0 and corresponds to the usual GCV function.
df	The effective number of parameters for the fitted surface. Conversely, N - df, where N is the total number of observations, is the degrees of freedom associated with the residuals. This is an alternative to specifying lambda and much more interpretable. NOTE: GCV argument defaults to TRUE if this argument is used.
GCV	If TRUE, matrix decompositions are done to allow estimating lambda by GCV or REML and specifying smoothness by the effective degrees of freedom. So the GCV switch does more than just supply a GCV estimate. Also, if lambda or df are passed, the estimate will be evaluated at those values, not at the GCV/REML estimates of lambda. If FALSE, the Kriging estimate is found under a fixed lambda model.
give.warnings	If TRUE, warnings are given about GCV grid search limits. If FALSE, warnings are not given. Best to leave this TRUE! This argument is set to FALSE if warn is less than zero in the top level, R options function. See options()$warn
lambda	Smoothing parameter that is the ratio of the error variance (tau**2) to the scale parameter of the covariance function (sigma). If omitted, this is estimated by GCV (see method below).
method	Determines what "smoothing" parameter should be used. The default is to estimate standard GCV. Other choices are: GCV.model, GCV.one, RMSE, pure error and REML. The differences are explained below.
mean.obj	Object to predict the mean of the spatial process. This is used when fitting a correlation model with varying spatial means and varying marginal variances. (See details.)
m	A polynomial function of degree (m-1) will be included in the model as the drift (or spatial trend) component. The "m" notation is from thin-plate splines where m is the derivative in the penalty function. With m=2 as the default, a linear model in the locations will be fit as a fixed part of the model.

na.rm	If TRUE, NAs will be removed from the y vector and the corresponding rows of x – with a warning. If FALSE, Krig will just stop with a message. Once NAs are removed, all subsequent analysis in fields does not use those data.
nstep.cv	Number of grid points for the coarse grid search to minimize the GCV RMLE and other related criteria for finding lambda, the smoothing parameter. Default is 200, fairly large to avoid some cases of closely spaced local minima. Evaluations of the GCV and related objective functions are cheap given the matrix decompositions described below.
null.args	Extra arguments for the null space function null.function. If fields.mkpoly is passed as null.function then this is set to a list with the value of m. So the default is to use a polynomial of degree m-1 for the null space (fixed part) of the model.
null.function	An R function that creates the matrices for the null space model. The default is fields.mkpoly, an R function that creates a polynomial regression matrix with all terms up to degree m-1. (See Details)
offset	The offset to be used in the GCV criterion. Default is 0. This would be used when Krig is part of a backfitting algorithm and the offset is other model degrees of freedom from other regression components.
sigma	Scale factor for covariance.
scale.type	This is a character string among: "range", "unit.sd", "user", "unscaled". The independent variables and knots are scaled to the specified scale.type. By default, no scaling is done. This usually makes sense for spatial locations. Scale type of "range" scales the data to the interval (0,1) by forming (x-min(x))/range(x) for each x. Scale type of "unit.sd" scales each variable to have mean 0 and standard deviation 1. Scale type of "user" allows specification of an x.center and x.scale by the user. The default for "user" is mean 0 and standard deviation 1. Scale type of "unscaled" does not scale the data.
sd.obj	Object to predict the marginal standard deviation of the spatial process.
tau2	Variance of the errors, often called the nugget variance. If weights are specified, then the error variance is tau2 divided by weights. Note that lambda is defined as the ratio tau2/sigma.
verbose	If TRUE, will print out all kinds of intermediate stuff. Default is FALSE, of course, as this is used mainly for debugging.
weights	Weights are proportional to the reciprocal variance of the measurement error. The default is equal weighting, i.e., a vector of unit weights.
wght.function	An R function that creates a weights matrix for the observations. This is only needed if the weight matrix has off-diagonal elements. The default is NULL, indicating that the weight matrix is a diagonal, based on the weights argument. (See details)
W	The observation weight matrix.
wght.args	Optional arguments to be passed to the weight function (wght.function) used to create the observation weight matrix.
x	Matrix of independent variables. These could be the locations for spatial data or the independent variables in a regression.

x.center	Centering values to be subtracted from each column of the x matrix.
x.scale	Scale values that are divided into each column after centering.
Y	Vector of dependent variables. These are the values of the surface (perhaps with measurement error) at the locations or the dependent response in a regression.
XMat	A vector or matrix of covariates to be included in the fixed part of the model. If NULL (default), no additional covariates are included.
Z	This is the old argument for the fixed part of the model and is deprecated. It has been changed to XMat above.
...	Optional arguments that appear are assumed to be additional arguments to the covariance function, or are included in method functions (resid, fitted, coef) as a required argument.
object	A Krig object

Details

This function produces an object of class Krig. With this object it is easy to subsequently predict with this fitted surface, find standard errors, alter the y data (but not x), etc. It is the original version of Kriging implemented in fields and we recommend using the more current function spatialProcess instead.

The Kriging model is: Y.k= f(x.k) = P(x.k) + XMat(x.k) + e.k

where ".k" means subscripted by k, Y is the dependent variable observed at location x.k, P is a low order polynomial, XMat is a mean zero, Gaussian field with covariance function K and e is assumed to be independent normal errors. The estimated surface is the best linear unbiased estimate (BLUE) of f(x)= P(x) + XMat(x) given the observed data. For this estimate, K is taken to be sigma*cov.function and the errors have variance tau**2. In more conventional geostatistical terms, sigma is the "sill" if the covariance function is actually a correlation function and tau**2 is the nugget variance or measurement error variance (the two are confounded in this model). If the weights are given, then the variance of e.k is tau**2/ weights.k . In the case that the weights are specified as a matrix, W, using the wght.function option, then the assumed covariance matrix for the errors is tau**2 Wi, where Wi is the inverse of W. It is straightforward to show that the estimate of f only depends on tau and sigma through the ratio lambda = tau**2/ sigma. This parameter, termed the smoothing parameter, plays a central role in the statistical computations within Krig. See also the help for thin plate splines (Tps) to get another perspective on the smoothing parameter.

This function also supports a modest extension of the Generalized Kriging model to include other covariates as fixed regression type components. In matrix form, Y = XMat b + F + E where XMat is a matrix of covariates and b a fixed parameter vector, F the vector of function values at the observations and E a vector of errors. The XMat argument in the function is the way to specify this additional component.

If the parameters sigma and tau2 are omitted in the call, then they are estimated in the following way. If lambda is given, then tau2 is estimated from the residual sum of squares divided by the degrees of freedom associated with the residuals. Sigma squared, the process variance, is found as the difference between the sums of squares of the predicted values having subtracted off the polynomial part and tau2. These estimates are the MLEs under Gaussian assumptions on the process and errors. If lambda is also omitted, it is estimated from the data using a variety of approaches and then the values for tau and sigma are found in the same way from the estimated lambda.

A useful extension of a stationary correlation to a nonstationary covariance is what we term a correlation model. If mean and marginal standard deviation objects are included in the call, then the observed data is standardized based on these functions. The spatial process is then estimated with respect to the standardized scale. However, for predictions and standard errors the mean and standard deviation surfaces are used to produce results in the original scale of the observations.

The GCV function has several alternative definitions when replicate observations are present or if one uses a reduced set of knots. Here are the choices based on the method argument:

GCV: leave-one-out GCV. But if there are replicates, it is leave-one-group-out. (Wendy and Doug prefer this one.)

GCV.one: Really leave-one-out GCV even if there are replicate points. This is what the old tps function used in FUNFITS.

rmse: Match the estimate of tau**2 to an external value (called rmse).

pure error: Match the estimate of tau**2 to the estimate based on replicated data (pure error estimate in ANOVA language).

GCV.model: Only considers the residual sums of squares explained by the basis functions.

REML: The process and errors are assumed to be Gaussian and the likelihood is concentrated (or profiled) with respect to lambda. The MLE of lambda is found from this criterion. Restricted means that the likelihood is formed from a linear transformation of the observations that is orthogonal to the column space of P(x).

WARNING: The covariance functions often have nonlinear parameter(s) that control the strength of the correlations as a function of separation, usually referred to as the range parameter. This parameter must be specified in the call to Krig and will not be estimated.

Value

An object of class Krig. This includes the predicted values in fitted.values and the residuals in residuals. The results of the grid search to minimize the generalized cross validation function are returned in gcv.grid.

The coef.Krig function only returns the coefficients, "d", associated with the fixed part of the model (also known as the null space or spatial drift).

`call`	Call to the function
`y`	Vector of dependent variables.
`x`	Matrix of independent variables.
`weights`	Vector of weights.
`knots`	Locations used to define the basis functions.
`transform`	List of components used in centering and scaling data.
`np`	Total number of parameters in the model.
`nt`	Number of parameters in the null space.
`matrices`	List of matrices from the decompositions (D, G, u, X, qr.T).
`gcv.grid`	Matrix of values from the GCV grid search. The first column is the grid of lambda values used in the search, the second column is the trace of the A matrix, the third column is the GCV values and the fourth column is the estimated value of tau conditional on the value of lambda.

lambda.est	A table of estimated smoothing parameters with corresponding degrees of freedom and estimates of tau found by different methods.
cost	Cost value used in GCV criterion.
m	Order of the polynomial space: highest degree polynomial is (m-1). This is a fixed part of the surface often referred to as the drift or spatial trend.
eff.df	Effective degrees of freedom of the model.
fitted.values	Predicted values from the fit.
residuals	Residuals from the fit.
lambda	Value of the smoothing parameter used in the fit. Lambda is defined as tau**2/sigma. See discussion in details.
yname	Name of the response.
cov.function	Covariance function of the model.
beta	Estimated coefficients in the ridge regression format
d	Estimated coefficients for the polynomial basis functions that span the null space
fitted.values.null	Fitted values for just the polynomial part of the estimate
trace	Effective number of parameters in model.
c	Estimated coefficients for the basis functions derived from the covariance.
coefficients	Same as the beta vector.
just.solve	Logical describing if the data has been interpolated using the basis functions.
tauHat	Estimated standard deviation of the measurement error (nugget effect).
tau2	Estimated variance of the measurement error (tauHat**2).
sigma	Scale factor for covariance. COV(h(x),h(x)) = sigma*cov.function(x,x) If the covariance is actually a correlation function then sigma is also the "sill".
mean.var	Normalization of the covariance function used to find sigma.
best.model	Vector containing the value of lambda, the estimated variance of the measurement error and the scale factor for covariance used in the fit.

References

See "Additive Models" by Hastie and Tibshirani, "Spatial Statistics" by Cressie and the FIELDS manual.

See Also

summary.Krig, predict.Krig, predictSE.Krig, predictSurfaceSE, predictSurface, plot.Krig, surface.Krig

Examples

```
# a 2-d example
# fitting a surface to ozone

# measurements. Exponential covariance, range parameter is 20 (in miles)
data(ChicagoO3)
fit <- Krig(ChicagoO3$x, ChicagoO3$y, aRange=20)

summary( fit) # summary of fit
set.panel( 2,2)
plot(fit) # four diagnostic plots of fit
set.panel()
surface( fit, type="C") # look at the surface

# predict at data
predict( fit)

# predict using 7.5 effective degrees of freedom:
predict( fit, df=7.5)


# predict on a grid ( grid chosen here by defaults)
 out<- predictSurface( fit)
 surface( out, type="C") # option "C" our favorite

# predict at arbitrary points (10,-10) and (20, 15)
 xnew<- rbind( c( 10, -10), c( 20, 15))
 predict( fit, xnew)

# standard errors of prediction based on covariance model.
 predictSE( fit, xnew)

# surface of standard errors on a default grid
 predictSurfaceSE( fit)-> out.p # this takes some time!
 surface( out.p, type="C")
 points( fit$x)

## Not run:
# Using another stationary covariance.
# smoothness is the shape parameter for the Matern.

fit <- Krig(ChicagoO3$x, ChicagoO3$y,
Covariance="Matern", aRange=10, smoothness=1.0)
summary( fit)

#
# Roll your own: creating very simple user defined Gaussian covariance
#

test.cov <- function(x1,x2,aRange,marginal=FALSE,C=NA){
   # return marginal variance
     if( marginal) { return(rep( 1, nrow( x1)))}
```

```
    # find cross covariance matrix
      temp<- exp(-(rdist(x1,x2)/aRange)**2)
      if( is.na(C[1])){
          return( temp)}
      else{
          return( temp%*%C)}
      }
#
# use this and put in quadratic polynomial fixed function

data(flame)
fit.flame<- Krig(flame$x, flame$y, cov.function="test.cov", m=3, aRange=.5)

#
# note how range parameter is passed to Krig.
# BTW:  GCV indicates an interpolating model (nugget variance is zero)
# This is the content of the warning message.

# take a look ...
 surface(fit.flame, type="I")

## End(Not run)

#
# Thin plate spline fit to ozone data using the radial
# basis function as a generalized covariance function
#
# p=2 is the power in the radial basis function (with a log term added for
# even dimensions)
# If m is the degree of derivative in penalty then p=2m-d
# where d is the dimension of x. p must be greater than 0.
#  In the example below p = 2*2 - 2 = 2
#

 out<- Krig( ChicagoO3$x, ChicagoO3$y,cov.function="Rad.cov",
                      m=2,p=2,scale.type="range")

# See also the Fields function Tps
# out  should be identical to  Tps( ChicagoO3$x, ChicagoO3$y)
#

## Not run:
#
#
# explore some different values for the range and lambda using GCV
  data(ozone2)
  aRange <- seq(200,600,,40)
  GCV<- matrix( NA, 40,80)
# the loop
  for( k in 1:40){
# call to Krig with different ranges
# also turn off warnings for GCV search
```

```
# to avoid lots of messages. (not recommended in general!)
    obj<-Krig( ozone2$lon.lat,ozone2$y[16,],
             cov.function="stationary.cov",
             aRange=aRange[k],
             Covariance="Matern",smoothness=1.0,
             Distance="rdist.earth", nstep.cv=80,
             give.warnings=FALSE, na.rm=TRUE)
    GCV[k,]<-obj$gcv.grid[,3]
  }
# get lambda grid  from looping
   k<- 1
   lam<-  Krig( ozone2$lon.lat,ozone2$y[16,],
             cov.function="stationary.cov",
             aRange=aRange[k],
             Covariance="Matern",smoothness=.5,
             Distance="rdist.earth", nstep.cv=80,
             give.warnings=FALSE, na.rm=TRUE)$gcv.grid[,1]
 matplot( log10(lam), t(GCV),type="l",lty=1)


## End(Not run)
```

`Krig.Amatrix` *Smoother (or "hat") matrix relating predicted values to the dependent (Y) values.*

Description

For a fixed value of the smoothing parameter or the covariance function, some nonparametric curve estimates are linear functions of the observed data. This is an intermediate level function that computes the linear weights to be applied to the observations to estimate the curve at a particular point. For example, the predicted values can be represented as AY where A is an N X N matrix of coefficients and Y is the vector of observed dependent variables. For linear smoothers the matrix A may depend on the smoothing parameter (or covariance function) and the independent variables (X) but NOT on Y.

Usage

```
Krig.Amatrix(object, x0 = object$x, lambda=NULL,
      eval.correlation.model = FALSE,...)
```

Arguments

Output object from fitting a data set using a FIELD regression method. Currently this is supported only for Krig (and Tps) functions.

`object`	A Krig object produced by the Krig (or Tps) function.
`x0`	Locations for prediction; the default is the observation locations.
`lambda`	Value of the smoothing parameter.

`eval.correlation.model`
: This applies to a correlation model where the observations have been standardized – e.g. y standardized = (yraw - mean) / (standard deviation). If TRUE, the prediction in the correlation scale is transformed by the standard deviation and mean to give a prediction in the raw scale. If FALSE, predictions are left in the correlation scale.

`...`
: Other arguments that can be used by predict.Krig.

Details

The main use of this function is in finding prediction standard errors.

For the Krig (and Tps) functions, the A matrix is constructed based on the representation of the estimate as a generalized ridge regression. The matrix expressions are explained in the references from the FIELDS manual. For linear regression, the matrix that gives predicted values is often referred to as the "hat" matrix and is useful for regression diagnostics. For smoothing problems, the effective number of parameters in the fit is usually taken to be the trace of the A matrix. Note that while the A matrix is usually constructed to predict the estimated curve at the data points, Amatrix.Krig does not have such restrictions. This is possible because any value of the estimated curve will be a linear function of Y.

The actual calculation in this function is simple. It involves looping through the unit vectors at each observation and computation of the prediction for each of these delta functions. This approach makes it easy to handle different options such as including covariates.

Value

A matrix where the number of rows is equal to the number of predicted points and the number of columns is equal to the length of the Y vector.

References

Nychka (2000) "Spatial process estimates as smoothers."

See Also

Krig, Tps, predict.Krig

Examples

```
# Compute the A matrix or "hat" matrix for a thin plate spline
# check that this gives the same predicted values
data(ChicagoO3)
tps.out<-Tps( ChicagoO3$x, ChicagoO3$y)
A<-Krig.Amatrix( tps.out, ChicagoO3$x)
test<- A%*%ChicagoO3$y
# now compare this to predict( tps.out) or tps.out$fitted.values
#                    they should be the same
stats( test- tps.out$fitted.values)
```

`Krig.null.function` *Default function to create fixed matrix part of spatial process model.*

Description

Constructs a matrix of terms representing a low order polynomial and binds additional columns due to covariates (the XMat matrix).

Usage

```
Krig.null.function(x, XMat = NULL, drop.XMat = FALSE, m)
```

Arguments

`x`	Spatial locations
`XMat`	Other covariates to be associated with each location.
`drop.XMat`	If TRUE, only the low order polynomial part is created.
`m`	The polynomial order is (m-1).

Details

This function can be modified to produce a different fixed part of the spatial model. The arguments x, XMat and drop.XMat are required, but other arguments can be passed as part of a list in null.args in the call to Krig.

Value

A matrix where the first columns are the polynomial terms and the following columns are from XMat.

Author(s)

Doug Nychka

See Also

Krig

`Krig.replicates` *Collapse repeated spatial locations into unique locations*

Description

In case several observations are available for a single spatial location, find the group means and replicate variability.

Usage

```
Krig.replicates(out = NULL, x, y, XMat = NULL, weights = rep(1,
                 length(y)), digits = 8, verbose = FALSE)
```

Arguments

`out`	A list with components `x`, `y`, `weights`, and possibly `XMat`.
`x`	Spatial locations.
`y`	Spatial observations
`XMat`	Spatial covariates.
`weights`	Weights proportional to reciprocal variances of observations.
`digits`	Number of significant digits to consider in determining a replicate location.
`verbose`	If TRUE, print out details for debugging.

Details

This function figures out which locations are the same and within the function fast.1way uses `tapply` to find replicate group means and standard deviations. NOTE: it is assumed the XMat covariates are unique at the locations. Currently these functions cannot handle a model with common spatial locations but different values for the XMat covariates.

Value

A list with components:

`yM`	Data at unique locations; where more than one observation is available, this is the mean of the replicates.
`xM`	Unique spatial locations.
`weightsM`	Weights matching the unique locations proportional to reciprocal variances. This is found as a combination of the original weights at each location.
`XMatM`	Values of the covariates at the unique locations.
`uniquerows`	Index for unique rows of `x`.
`tauHat.rep`, `tauHat.pure.error`	Standard deviation of pure error estimate based on replicate groups (and adjusting for possibly different weights).
`rep.info`	Integer tags indicating replicate groups.

Author(s)

Douglas Nychka

Examples

```
#create  some spatial replicates
 set.seed( 123)
 x0<- matrix( runif(10*2), 10,2)
 x<-  x0[ c(rep(1,3), 2:8, rep( 9,5),10) , ]
 y<-  rnorm( 16)

 out<- Krig.replicates( x=x, y=y)
# compare
# out$yM[1] ;  mean( y[1:3])
# out$yM[9] ; mean( y[11:15])
# mean( y[ out$rep.info==9])
```

KrigFindLambda *Finds profile likelihood and GCV estimates of smoothing parameters for splines and Kriging.*

Description

This is a secondary function that will use the computed Krig object and find various estimates of the smoothing parameter lambda. These are several different flavors of cross-validation, a moment matching strategy and the profile likelihood. This function can also be used independently with different data sets (the y's) if the covariates (the x's) are the same and thus reduce the computation.

Usage

```
KrigFindLambda(
out, lambda.grid = NA, cost = 1, nstep.cv = 200, rmse
                 = NA, verbose = FALSE, tol = 1e-05, offset = 0, y =
                 NULL, give.warnings = TRUE)

gcv.sreg (
out, lambda.grid = NA, cost = 1, nstep.cv = 80, rmse =
                 NA, offset = 0, trmin = NA, trmax = NA, verbose =
                 FALSE, tol = 1e-05, give.warnings = TRUE)
```

Arguments

out	A Krig or sreg object.
lambda.grid	Grid of lambdas for coarse search. The default is equally spaced on effective degree of freedom scale.
cost	Cost used in GCV denominator

nstep.cv	Number of grid points in coarse search.
rmse	Target root mean squared error to match with the estimate of tau**2
verbose	If TRUE, prints intermediate results.
tol	Tolerance in declaring convergence of golden section search or bisection search.
offset	Additional degrees of freedom to be added into the GCV denominator.
y	A new data vector to be used in place of the one associated with the Krig object (obj)
give.warnings	If FALSE, will suppress warnings about grid search being out of range for various estimates based on GCV and REML.
trmin	Minimum value of lambda for grid search specified in terms of effective degrees of freedom.
trmax	Maximum value for grid search.

Details

This function finds several estimates of the smoothing parameter using first a coarse grid search followed by a refinement using a minimization (in the case of GCV or maximum likelihood) or bisection in the case of matching the rmse. Details of the estimators can be found in the help file for the Krig function.

The Krig object passed to this function has some matrix decompositions that facilitate rapid computation of the GCV and ML functions and do not depend on the independent variable. This makes it possible to compute the Krig object once and to reuse the decompositions for multiple data sets. (But keep in mind that if the x values change, then the object must be recalculated.) The example below shows how this can be used for a simulation study on the variability for estimating the smoothing parameter.

Value

A list giving a summary of estimates and diagnostic details with the following components:

gcv.grid	A matrix describing results of the coarse search; rows are values of lambda and the columns are lambda= value of smoothing parameter, trA=effective degrees of freedom, GCV=Usual GCV criterion, GCV.one=GCV criterion leave-one-out, GCV.model= GCV based on average response in the case of replicates, tauHat= Implied estimate of tau, -Log Profile= negative log of profile likelihood for the lambda.
lambda.est	Summary table of all estimates. Rows index different types of estimates: GCV, GCV.model, GCV.one, RMSE, pure error, -Log Profile and the columns are the estimated values for lambda, trA, GCV, tauHat.

Author(s)

Doug Nychka

See Also

Krig, Tps, predict.Krig

Examples

```
#
data(ChicagoO3)
obj<- Tps( ChicagoO3$x, ChicagoO3$y) # default is to find lambda by GCV
summary( obj)

out<- KrigFindLambda( obj)
print( out$lambda.est) # results agree with Tps summary

data(rat.diet)
out<-sreg( rat.diet$t, rat.diet$trt)
gcv.sreg( out, tol=1e-10) # higher tolerance search for minimum
## Not run:
# a simulation example
x<- seq( 0,1,,150)
f<-  x**2*( 1-x)
f<- f/sqrt( var( f))

set.seed(123) # let's all use the same seed
tau<- .1
y<- f + rnorm( 150)*tau

Tps( x,y)-> obj # create Krig object

hold<- hold2<- matrix( NA, ncol=6, nrow=200)

for( k in 1:200){
# look at GCV estimates of lambda
# new data simulated
   y<- f + rnorm(150)*tau
# save GCV estimates
lambdaTable<- KrigFindLambda(obj,  y=y, give.warnings=FALSE)$lambda.est
hold[k,]<-  lambdaTable[1,]
hold2[k,]<-  lambdaTable[6,]
}
matplot( cbind(hold[,2], hold2[,2]),cbind( hold[,4],hold2[,4]),
 xlab="estimated eff. df", ylab="tau hat", pch=16, col=c("orange3", "green2"), type="p")
yline( tau, col="grey", lwd=2)


## End(Not run)
```

---

lennon *Gray image of John Lennon.*

Description

A 256X256 image of John Lennon. Try:

`image(lennon, col=grey(seq(0,1,,256)) )`

LENSExample *Surface temperature sensitivity from the NCAR Climate model.*

Description

Based on the NCAR Earth System Model (CESM). The local change in surface temperature (C) expected by a 1 degree change in the global average. This statistic is referred to as the climate sensitivity and was found by OLS regression on a 30-member ensemble of climate runs.

Format

The usual image format with components x, y and z.

x Equally spaced longitude values.

y Equally spaced latitude values.

z Climate sensitivity on a 288X192 grid.

Source

The CESM Large Ensemble Project, led by Clara Deser and Jennifer Kay, is a publicly available set of climate model simulations intended for understanding variability in the Earth's climate system and climate change. See https://www.cesm.ucar.edu/community-projects/lens for background.

All simulations are performed with the nominal 1-degree latitude/longitude version of the Community Earth System Model version 1 (CESM1) with CAM5.2 as its atmospheric component. Each member is subject to the same radiative forcing scenario: historical up to 2005 and RCP8.5 thereafter. Here, the main component of these scenarios is the observed or projected greenhouse gas emissions leading to global warming. Each climate run begins from a slightly different initial atmospheric state, created by randomly perturbing temperatures at the level of round-off error. Even this subtle difference, due to nonlinear dynamics, leads to climate simulations that can be assumed to be independent realizations.

Here, 30 members of the ensemble are summarized to give local sensitivity for the Northern Hemisphere summer months, June, July, and August (aka JJA). This involves a linear regression on the summer mean time series for each model grid box with the global summer average and for each ensemble member. The 30 slopes from the regressions for each grid box are averaged to get the sensitivity in this data set.

See the fields package GitHub repository for the R script that creates this data set from the model output.

Examples

```
data(LENSExample)
imagePlot( LENSExample)

# Nathan Lenssen's version

divMap(LENSExample$x, LENSExample$y, LENSExample$z,
      worldCol="white", lwd=.5, zCap=2, wrapVal=c(-180, 180))
```

minitri *Mini triathlon results*

Description

Results from a mini triathlon sponsored by Bud Light, held in Cary, NC, June 1990. Times are in minutes for the male 30-34 group. Man was it hot and humid! (DN)

The events, in order, were: swim: (1/2 mile) bike: (15 miles) run: (4 miles)

Usage

```
data(minitri)
```

Format

This is a data frame. Row names are the place within this age group based on total time.

Data frame comonents `"swim" "bike" "run"`

swim swim times

bike bike times

run run times

Examples

```
data(minitri)
pairs( minitri)
```

mKrig *"micro Krig" Spatial process estimate of a curve or surface, "kriging" with a known covariance function.*

Description

This is an implementation of the frequentist spatial process estimate known as "Kriging" that is optimized for large data sets, sparse linear algebra, and a clear exposition of the computations. Lambda, the smoothing parameter, must be fixed. This function is called by higher level functions for maximum likelihood estimates of covariance parameters and so often serves to compute the objective function.

Usage

```
mKrig(x, y, weights = rep(1, nrow(x)), XMat = NULL, XMatCommon =
                 NULL, cov.function = "stationary.cov", cov.args =
                 NULL, lambda = NA, m = 2, chol.args = NULL, find.trA =
                 TRUE, NtrA = 20, iseed = NA, na.rm = FALSE,
                 collapseFixedEffect = TRUE, tau = NA, sigma2 = NA,
                 verbose = FALSE, simpleKriging = FALSE, Z = NULL,
                 ZCommon = NULL, ...)
## S3 method for class 'mKrig'
predict( object, xnew=NULL,ynew=NULL, grid.list = NULL,
derivative=0,
XMat=NULL,drop.XMat=FALSE,
 Z = NULL, drop.Z = NULL,
                 just.fixed=FALSE,
collapseFixedEffect = object$collapseFixedEffect, ...)

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

## S3 method for class 'mKrig'
print( x, digits=4,... )
## S3 method for class 'mKrigSummary'
print( x, digits=4,... )

mKrig.coef(object, y, collapseFixedEffect=TRUE)

mKrig.trace( object, iseed, NtrA)

mKrigCheckXY(x, y, weights, XMat, XMatCommon, na.rm)
```

Arguments

collapseFixedEffect	If replicated fields are given to mKrig (i.e., y has more than one column), there is the choice of estimating the fixed effect coefficients (d in the returned object) separately for each replicate or pooling across replicates and deriving a single estimate. If collapseFixedEffect is TRUE (default), the estimates are pooled.
chol.args	A list of optional arguments (pivot, nnzR) that will be used with the call to the Cholesky decomposition. Pivoting is done by default to make use of sparse matrices when they are generated. This argument is useful in some cases for sparse covariance functions to reset the memory parameter nnzR. (See example below.)
cov.args	A list of optional arguments that will be used in calls to the covariance function.
cov.function	The name, a text string, of the covariance function.
derivative	If zero, the surface will be evaluated. If not zero, the matrix of partial derivatives will be computed.
digits	Number of significant digits used in printed output.

drop.XMat	If TRUE, the fixed part will only be evaluated at the polynomial part of the fixed model. The contribution from the other covariates will be omitted.
drop.Z	An older name for drop.XMat. If used, a warning will be printed.
find.trA	If TRUE, will estimate the effective degrees of freedom using a simple Monte Carlo method. This will add to the computational burden by approximately NtrA solutions of the linear system, but the Cholesky decomposition is reused.
grid.list	A grid.list to evaluate the surface in place of specifying arbitrary locations.
iseed	Random seed (using set.seed(iseed)) used to generate iid normals for Monte Carlo estimate of the trace. Set this to an integer to ensure the same random draws are used to compute the approximate trace and be consistent across different values of the covariance parameters.
just.fixed	If TRUE, only the predictions for the fixed part of the model will be evaluated.
lambda	Smoothing parameter or, equivalently, the ratio between the nugget and process variances.
m	The degree of the polynomial used in the fixed part. This follows the thin plate spline convention that the degree is m-1. Default is m=2 for a linear function, m=1 for a constant and, as a quick logical switch, m=0 will result in no fixed part in the spatial model.
na.rm	If TRUE, NAs in y are omitted along with corresponding rows of x.
NtrA	Number of Monte Carlo samples for the trace. But if NtrA is greater than or equal to the number of observations, the trace is computed exactly.
object	Object returned by mKrig. (Same as "x" in the print function.)
sigma2	Value of the process variance.
simpleKriging	If TRUE, there is no fixed linear part in the spatial model. This is a more rational way than setting m=0.
tau	Value of the measurement error (aka nugget) standard deviation.
verbose	If TRUE, prints out intermediate values of objects for debugging.
weights	Precision (1/variance) of each observation
x	Matrix of unique spatial locations (or, in print or surface, the returned mKrig object).
xnew	Locations for predictions.
y	Vector or matrix of observations at spatial locations, missing values are not allowed! Or in mKrig.coef a new vector of observations. If y is a matrix, the columns are assumed to be independent replicates of the spatial field, i.e., observation vectors generated from the same covariance and measurement error model but independent from each other.
ynew	New observation vector. mKrig will reuse matrix decompositions and find the new fit to these data.
XMat	Linear covariates to be included in the fixed part of the model that are distinct from the default low order polynomial in x. (NOTE: the order of the polynomial is determined by m.)

XMatCommon	Linear covariates to be included in the fixed part of the model. This option is only relevant for independent realizations of the fields (i.e., y is a matrix). The number of rows of XMatCommon should be equal to the number of observations times the number of realizations.
Z	This argument is deprecated and a warning will be printed. Should use the new argument name XMat.
ZCommon	This argument is deprecated and a warning will be printed. Should use the new argument name XMatCommon.
...	In mKrig and predict, additional arguments that will be passed to the covariance function.

Details

This function is an abridged version of Krig and uses a more direct computation for the linear algebra that facilitates compactly supported covariance functions. The m stands for micro and is done for a fixed lambda parameter and other covariance parameters (e.g., aRange) and for unique spatial locations. This is also the basic computational element of the top-level spatialProcess function for finding MLEs. See the source code for a commented version that describes the computation. These restrictions simplify the code for reading. Note also that little checking is done and the spatial locations are not transformed before the estimation.

Repeated locations To keep the coding simple, x must be a set of unique locations. When there are multiple observations at a location – also known as replicated locations – one strategy is to use the average spatial value for this location in place of multiple observations. One can also adjust the weights accordingly for this location if that makes sense. To do this, use the function Krig.replicates. For example

```
out<- Krig.replicates( x=x, y=y)
```

and outxM, outyM, out$weightsM will be the unique locations with averaged values and the number of observations.

```
out<- Krig.replicates( x=x, y=y, weights= weights)
```

will also combine the weight vector based on the replications and be returned as out$weightsM.

Multiple fields Because most of the operations are linear algebra, this code has been written to handle multiple data sets. Specifically, if the spatial model is the same except for different observed values (the y's), one can pass y as a matrix and the computations are done efficiently for each set. The likelihood across all replicates is combined and denoted with FULL at the end. Also note the collapseFixedEffect switch to determine if the regression part is found separately for each replicate or combined into a single model. Note that this is not a multivariate spatial model – just an efficient computation over several data vectors without explicit looping. A big difference in the computations is that an exact expression for the trace of the smoothing matrix (trace A(lambda)) is computationally expensive and a Monte Carlo approximation is supplied instead.

Supporting functions

See predictSE.mKrig for prediction standard errors and sim.mKrig.approx to quantify the uncertainty in the estimated function using conditional simulation.

predict.mKrig will evaluate the derivatives of the estimated function if derivatives are supported in the covariance function. For example, the wendland.cov function supports derivatives.

summary.mKrig creates a list of class mKrigSummary along with a table of standard errors for the fixed linear parameters.

print.mKrigSummary prints the mKrigSummary object and adds some more explanation about the model and results

print.mKrig prints a summary for the mKrig object that combines the summary and print methods.

mKrig.coef finds "beta" and "c" coefficients representing the solution using the previous Cholesky decomposition but for a new data vector. This is used in computing the prediction standard error in predictSE.mKrig and can also be used to evaluate the estimate efficiently at new vectors of observations provided the locations, fixed part, and the covariance matrix all remain the same.

Sparse covariance functions Sparse matrix methods are handled through overloading the usual linear algebra functions with sparse versions. But to take advantage of some additional options in the sparse methods, the list argument chol.args is a device for changing some default values. The most important of these is nnzR, the number of nonzero elements anticipated in the Cholesky factorization of the positive definite linear system used to solve for the basis coefficients. The sparsity of this system is essentially the same as the covariance matrix evaluated at the observed locations. As an example of resetting nnzR to 450000, one would use the following argument for chol.args in mKrig:

```
chol.args=list(pivot=TRUE,memory=list(nnzR= 450000))
```

Approximation trace estimate mKrig.trace This is an internal function called by mKrig to estimate the effective degrees of freedom. The Kriging surface estimate at the data locations is a linear function of the data and can be represented as A(lambda)y. The trace of A is one useful measure of the effective degrees of freedom used in the surface representation. In particular, this figures into the GCV estimate of the smoothing parameter. It is computationally intensive to find the trace explicitly, but there is a simple Monte Carlo estimate that is often very useful. If E is a vector of iid N(0,1) random variables, then the trace of A is the expected value of t(E)AE. Note that AE is simply predicting a surface at the data location using the synthetic observation vector E. This is done for NtrA independent N(0,1) vectors and the mean and standard deviation are reported in the mKrig summary. Typically, as the number of observations is increased, this estimate becomes more accurate. If NtrA is as large as the number of observations (np), then the algorithm switches to finding the trace exactly based on applying A to np unit vectors.

Value

summary	A named array with the values of covariance parameters and log likelihoods.
beta	Coefficients of the polynomial fixed part and, if present, the covariates (XMat). To determine which is which, the logical vector ind.drift, also part of this object, is TRUE for the polynomial part.
c.coef	Coefficients of the nonparametric part.
nt	Dimension of fixed part.
np	Dimension of c.coef.
nXMat	Number of columns of XMat covariate matrix (can be zero).
ind.drift	Logical vector that indicates polynomial coefficients in the d coefficients vector. This is helpful to distinguish between the polynomial part and the extra covariate coefficients associated with XMat.

`lambda.fixed` The fixed lambda value

`x` Spatial locations used for fitting.

`knots` The same as x

`cov.function.name`
Name of covariance function used.

`args` A list with all the covariance arguments that were specified in the call.

`m` Order of fixed part polynomial.

`chol.args` A list with all the Cholesky arguments that were specified in the call.

`call` A copy of the call to mKrig.

`non.zero.entries`
Number of nonzero entries in the covariance matrix for the process at the observation locations.

`lnDetCov` Log determinant of the covariance matrix for the observations having factored out sigma.

`Omega` GLS covariance for the estimated parameters in the fixed part of the model (d coefficients).

`qr.VT, Mc` QR and Cholesky matrix decompositions needed to recompute the estimate for new observation vectors.

`fitted.values, residuals`
Usual predictions from fit.

`eff.df` Estimate of effective degrees of freedom. Either the mean of the Monte Carlo sample or the exact value.

`trA.info` If NtrA is less than `np`, then the individual members of the Monte Carlo sample and `sd(trA.info)/ sqrt(NtrA)` is an estimate of the standard error. If NtrA is greater than or equal to `np`, then these are the diagonal elements of A(lambda).

`GCV` Estimated value of the GCV function.

`GCV.info` Monte Carlo sample of GCV functions

Author(s)

Doug Nychka, Reinhard Furrer, John Paige

References

https://github.com/dnychka/fieldsRPackage

See Also

Krig, surface.mKrig, Tps, fastTps, predictSurface, predictSE.mKrig, sim.mKrig.approx, `mKrig.grid`

Examples

```
#
# Midwest ozone data  'day 16' stripped of missings
  data( ozone2)
  y<- ozone2$y[16,]
  good<- !is.na( y)
  y<-y[good]
  x<- ozone2$lon.lat[good,]
# nearly interpolate using defaults (Exponential covariance range = 2.0)
# see also mKrigMLEGrid to choose lambda by maximum likelihood
  out<- mKrig( x,y, aRange = 2.0, lambda=.01)
  out.p<- predictSurface( out)
  surface( out.p)
#
# NOTE this should be identical to
# Krig( x,y, aRange=2.0, lambda=.01)

###############################################################################
# an example using a "XMat" covariate and the Matern family
#  again see mKrigMLEGrid to choose parameters by MLE.
data(COmonthlyMet)
yCO<- CO.tmin.MAM.climate
good<- !is.na( yCO)
yCO<-yCO[good]
xCO<- CO.loc[good,]
XMat<- CO.elev[good]
out<- mKrig( xCO,yCO, XMat=XMat, cov.function="stationary.cov", Covariance="Matern",
                   aRange=4.0, smoothness=1.0, lambda=.1)
set.panel(2,1)
# quilt.plot with elevations
quilt.plot( xCO, predict(out))
# Smooth surface without elevation linear term included
surface( out)
set.panel()


#############################################################################
# here is a series of examples with bigger datasets
# using a compactly supported covariance directly

set.seed( 334)
N<- 1000
x<- matrix( 2*(runif(2*N)-.5),ncol=2)
y<- sin( 1.8*pi*x[,1])*sin( 2.5*pi*x[,2]) + rnorm( 1000)*.1

look2<-mKrig( x,y, cov.function="wendland.cov",k=2, aRange=.2,
            lambda=.1)

# take a look at fitted surface
predictSurface(look2)-> out.p
surface( out.p)
```

```
# this works because the number of nonzero elements within distance aRange
# are less than the default maximum allocated size of the
# sparse covariance matrix.
#  see  options() for the default values. The names follow the convention
# spam.arg where arg is the name of the spam component
#   e.g. spam.nearestdistnnz

# The following will give a warning for aRange=.9 because
# allocation for the  covariance matrix storage is too small.
# Here aRange controls the support of the covariance and so
# indirectly the  number of nonzero elements in the sparse matrix

## Not run:
 look2<- mKrig( x,y, cov.function="wendland.cov",k=2, aRange=.9, lambda=.1)

## End(Not run)

# The warning resets the memory allocation  for the covariance matrix
# according the to values   options(spam.nearestdistnnz=c(416052,400))'
# this is inefficient because the preliminary pass failed.

# the following call completes the computation in "one pass"
# without a warning and without having to reallocate more memory.

options( spam.nearestdistnnz=c(416052,400))
  look2<- mKrig( x,y, cov.function="wendland.cov",k=2,
                   aRange=.9, lambda=1e-2)
# as a check notice that
#   print( look2)
# reports the number of nonzero elements consistent with the specific allocation
# increase in spam.options


# new data set of 1500 locations
  set.seed( 234)
  N<- 1500
  x<- matrix( 2*(runif(2*N)-.5),ncol=2)
  y<- sin( 1.8*pi*x[,1])*sin( 2.5*pi*x[,2]) + rnorm( N)*.01

## Not run:
# the following is an example of where the allocation  (for nnzR)
# for the Cholesky factor is too small. A warning is issued and
# the allocation is increased by 25
#
 look2<- mKrig( x,y,
            cov.function="wendland.cov",k=2, aRange=.1, lambda=1e2  )

## End(Not run)
# to avoid the warning
 look2<-mKrig( x,y,
            cov.function="wendland.cov", k=2, aRange=.1,
            lambda=1e2, chol.args=list(pivot=TRUE, memory=list(nnzR= 450000)))
```

```
##############################################################################
# fitting multiple data sets
#
#\dontrun{
  y1<- sin( 1.8*pi*x[,1])*sin( 2.5*pi*x[,2]) + rnorm( N)*.01
  y2<- sin( 1.8*pi*x[,1])*sin( 2.5*pi*x[,2]) + rnorm( N)*.01
  Y<- cbind(y1,y2)
  look3<- mKrig( x,Y,cov.function="wendland.cov",k=2, aRange=.1,
            lambda=1e2  )
# note slight difference in summary because two data sets have been fit.
  print( look3)
#}

## Not run:
#####################################################################
# finding a good choice for aRange as a taper

# Suppose the target is a spatial prediction using roughly 50 nearest neighbors
# (tapering covariances is effective for roughly 20 or more in the situation of
#  interpolation) see Furrer, Genton and Nychka (2006).
# take a look at a random set of 100 points to get idea of scale
# and saving  computation time by not  looking at the complete set
# of points
# NOTE: This could also be done directly using the  FNN package for finding nearest
# neighbors
  set.seed(223)
  ind<- sample( 1:N,100)
  hold<- rdist( x[ind,], x)
  dd<- apply( hold, 1, quantile, p= 50/N )
  dguess<- max(dd)
# dguess is now a reasonable guess at finding cutoff distance for
# 50 or so neighbors
# full distance matrix excluding distances greater than dguess
  hold2<- nearest.dist( x, x, delta= dguess )
# here is a trick to find the number of nonzero rows for a matrix in spam format.
  hold3<-  diff( hold2@rowpointers)
#  min( hold3) = 43   which we declare close enough. This also counts the diagonal
# So there are a minimum of 42 nearest neighbors  ( median is 136)
# see  table( hold3) for the distribution
# now the following will use no less than 43 - 1  nearest neighbors
# due to the tapering.

  mKrig( x,y, cov.function="wendland.cov",k=2, aRange=dguess,
            lambda=1e2) ->  look2

## End(Not run)

##############################################################################
# use precomputed distance matrix
#
## Not run:
  y1<- sin( 1.8*pi*x[,1])*sin( 2.5*pi*x[,2]) + rnorm( N)*.01
  y2<- sin( 1.8*pi*x[,1])*sin( 2.5*pi*x[,2]) + rnorm( N)*.01
```

```
  Y<- cbind(y1,y2)
  #precompute distance matrix in compact form
  distMat = rdist(x, compact=TRUE)
  look3<- mKrig( x,Y,cov.function="stationary.cov", aRange=.1,
           lambda=1e2, distMat=distMat )
  #precompute distance matrix in standard form
  distMat = rdist(x)
  look3<- mKrig( x,Y,cov.function="stationary.cov", aRange=.1,
           lambda=1e2, distMat=distMat )

## End(Not run)
```

`mKrigMLE` *Maximizes likelihood for the process marginal variance (sigma) and nugget standard deviation (tau) parameters (e.g., lambda) over many covariance models or covariance parameter values.*

Description

These functions are designed to explore the likelihood surface for different covariance parameters with the option of maximizing over tau and sigma. They use the `mKrig` base and are designed for computational efficiency.

Usage

```
mKrigMLEGrid(x, y, weights = rep(1, nrow(x)), XMat = NULL,XMatCommon=NULL,
                        mKrig.args = NULL,
                           cov.function = "stationary.cov",
                          cov.args = NULL,
                            na.rm = TRUE,
                          par.grid = NULL,
                            reltol = 1e-06,
                              REML = FALSE,
                              GCV  = FALSE,
                         optim.args = NULL,
                  cov.params.start = NULL,
                           verbose = FALSE,
                             iseed = NA)


mKrigMLEJoint(x, y, weights = rep(1, nrow(x)), XMat = NULL,XMatCommon=NULL,
                  mKrig.args = NULL,
   na.rm = TRUE, cov.function = "stationary.cov",
                  cov.args = NULL, cov.params.start = NULL, optim.args =
                  NULL, reltol = 1e-06, parTransform = NULL, REML =
                  FALSE, GCV = FALSE, hessian = FALSE, iseed = 303,
   verbose = FALSE)
```

```
profileCI(obj, parName, CIlevel = 0.95, REML = FALSE)

mKrigJointTemp.fn(parameters, mKrig.args, cov.args, parTransform,
                  parNames, REML = FALSE, GCV = FALSE, verbose =
                  verbose, capture.env)
```

Arguments

capture.env	For the ML objective function, the frame to save the results of the evaluation. This should be the environment of the function calling optim.
CIlevel	Confidence level.
cov.function	The name, a text string, of the covariance function.
cov.args	The arguments that would also be included in calls to the covariance function to specify the fixed part of the covariance model. This is in the form of a list, e.g., cov.args= list(aRange = 3.5).
cov.params.start	A list of initial starts for covariance parameters to perform likelihood maximization. The list contains the names of the parameters as well as the values. It usually makes sense to optimize over the important lambda parameter (tau^2/ sigma^2) in most spatial applications, and so if lambda is omitted, then the component lambda = .5 is added to this list.
hessian	If TRUE, return the BFGS approximation to the Hessian matrix at convergence.
iseed	Sets the random seed in finding the approximate Monte Carlo based GCV function and the effective degrees of freedom. This will not affect random number generation outside these functions.
mKrig.args	A list of additional parameters to supply to the mKrig function. E.g., Use mKrig.args= list(m=1) to set the regression function to be a constant function. mKrig function that are distinct from the covariance model. For example, mKrig.args= list(m=1) will set the polynomial to be just a constant term (degree = m - 1 = 0). Use mKrig.args= list(m = 0) to omit a fixed model and assume the observations have an expectation of zero.
na.rm	Remove NAs from data.
optim.args	Additional arguments that would also be included in calls to the optim function in joint likelihood maximization. If NULL, this will be set to use the "BFGS" optimization method. See optim for more details. The default value is: optim.args = list(method = "BFGS", control=list(fnscale = -1, ndeps = rep(log(1.1), length(cov.params.start)+1), abstol=1e-04, maxit=20)) Note that the first parameter is lambda and the others are the covariance parameters in the order they are given in cov.params.start. Also note that the optimization is performed on a transformed scale (based on the function parTransform), and this should be taken into consideration when passing arguments to optim.
parameters	The parameter values to evaluate the likelihood.

par.grid	A list or data frame with components being parameters for different covariance models. A typical component is "aRange" comprising a vector of scale parameters to try. If par.grid is "NULL", then the covariance model is fixed at values that are given in ....
obj	List returned from mKrigMLEGrid
parName	Name of parameter to find confidence interval.
parNames	Names of the parameters to optimize over.
parTransform	A function that maps the parameters to a scale for optimization or performs the inverse map from the transformed scale into the original values. See below for more details.
reltol	Optim BFGS convergence criterion.
REML	If TRUE, use REML instead of the full log likelihood.
GCV	NOT IMPLEMENTED YET! A placeholder to implement optimization using an approximate cross-validation criterion.
verbose	If TRUE, print out interesting intermediate results.
weights	Precision (1/variance) of each observation
x	Matrix of unique spatial locations (or, in print or surface, the returned mKrig object).
y	Vector or matrix of observations at spatial locations, missing values are not allowed! Or in mKrig.coef a new vector of observations. If y is a matrix, the columns are assumed to be independent observation vectors generated from the same covariance and measurement error model.
XMat	Linear covariates to be included in the fixed part of the model that are distinct from the default low order polynomial in x
XMatCommon	Linear covariates to be included in the fixed part of the model where a common parameter is estimated across all realizations. This option only makes sense for multiple realizations (y is a matrix).

Details

The observational model is the same as that described in the Krig function, and thus the two primary covariance parameters for a stationary model are the nugget standard deviation (tau) and the marginal variance of the process (sigma). It is useful to reparametrize as sigma and lambda = tau^2/sigma. The likelihood can be maximized analytically over sigma and the parameters in the fixed part of the model; this estimate of sigma can be substituted back into the likelihood to give an expression that is just a function of lambda and the remaining covariance parameters. This operation is called concentrating the likelihood by maximizing over a subset of parameters.

For these kinds of computations, there has to be some device to identify parameters that are fixed and those that are optimized. For mKrigMLEGrid and mKrigMLEJoint, the list cov.args should have the fixed parameters. For example, this is how to fix a lambda value in the model. The list cov.params.start should be a list with all parameters to optimize. The values for each component are used as the starting values. This is how the optim function works.

These functions may compute the effective degrees of freedom (see mKrig.trace) using the random trace method and so need to generate some random normals. The iseed argument can be used to

set the seed for this with the default being the seed 303. Note that the random number generation internal to these functions is coded so that it does not affect the random number stream outside these function calls.

For mKrigMLEJoint, the default transformation of the parameters is set up for a log/exp transformation:

```
parTransform <- function(ptemp, inv = FALSE) {
           if (!inv) {
               log(ptemp)
           }
           else {
               exp(ptemp)
           }
       }
```

Value

mKrigMLEGrid returns a list with the components:

summary	A matrix with each row giving the results of evaluating the likelihood for each covariance model.
par.grid	The par.grid argument used. A matrix where rows are the combination of parameters considered.
call	The calling arguments to this function.

mKrigMLEJoint returns a list with components:

summary	A vector giving the MLEs and the log likelihood at the maximum
lnLike.eval	A table containing information on all likelihood evaluations performed in the maximization process.
optimResults	The list returned from the optim function. Note that the parameters may be transformed values.
par.MLE	The maximum likelihood estimates.
parTransform	The transformation of the parameters used in the optimization.

Author(s)

Douglas W. Nychka, John Paige

References

https://github.com/dnychka/fieldsRPackage

See Also

mKrig, Krig, stationary.cov, optim

Examples

```
## Not run:
#perform joint likelihood maximization over lambda and aRange.
# NOTE: optim can get a bad answer with poor initial starts.
  data(ozone2)
  s<- ozone2$lon.lat
  z<- ozone2$y[16,]
  gridList<- list( aRange = seq( .4,1.0,length.out=20),
                lambda = 10**seq( -1.5,0,length.out=20)
                )
  par.grid<- make.surface.grid( gridList)
  out<- mKrigMLEGrid( s,z, par.grid=par.grid,
                          cov.args= list(smoothness=1.0,
                                    Covariance="Matern" )
                          )
  outP<- as.surface( par.grid, out$summary[,"lnProfileLike.FULL"])
  image.plot( outP$x, log10(outP$y),outP$z,
              xlab="aRange", ylab="log10 lambda")


## End(Not run)

 ## Not run:
  N<- 50
  set.seed(123)
  x<- matrix(runif(2*N), N,2)
  aRange<- .2
  Sigma<-  Matern( rdist(x,x)/aRange , smoothness=1.0)
  Sigma.5<- chol( Sigma)
  tau<- .1
  #  250 independent spatial data sets but a common covariance function
  #    -- there is little overhead in
  #        MLE across independent realizations and a good test of code validity.
  M<-250
  F.true<- t( Sigma.5) %*% matrix( rnorm(N*M), N,M)
  Y<-  F.true +  tau* matrix( rnorm(N*M), N,M)

# find MLE for lambda with grid of ranges
# and smoothness fixed in Matern
 par.grid<- list( aRange= seq( .1,.35,,8))
  obj1b<- mKrigMLEGrid( x,Y,
     cov.args = list(Covariance="Matern", smoothness=1.0),
     cov.params.start=list( lambda = .5),
        par.grid = par.grid
                  )
  obj1b$summary # take a look
# profile over aRange
  plot( par.grid$aRange, obj1b$summary[,"lnProfileLike.FULL"],
    type="b", log="x")

## End(Not run)
  ## Not run:
```

```
# m=0 is a simple switch to indicate _no_ fixed spatial drift
# (the default and highly recommended  is linear drift, m=2).
# However, m=0 results in MLEs that are less biased, being the correct model
# -- in fact it nails it !

  obj1a<- mKrigMLEJoint(x,Y,
                   cov.args=list(Covariance="Matern", smoothness=1.0),
                   cov.params.start=list(aRange =.5, lambda = .5),
                    mKrig.args= list( m=0))

 test.for.zero( obj1a$summary["tau"], tau, tol=.007)
 test.for.zero( obj1a$summary["aRange"], aRange, tol=.015)


## End(Not run)


##############################################################################
# A bootstrap example
# Here is a example of a more efficient (but less robust) bootstrap using
# mKrigMLEJoint and tuned starting values
##############################################################################
## Not run:
data( ozone2)
obj<- spatialProcess( ozone2$lon.lat,ozone2$y[16,] )

######### boot strap
  set.seed(123)
  M<- 25
# create M independent copies of the observation vector
  ySynthetic<- simSpatialData( obj, M)
  bootSummary<- NULL

 aRangeMLE<- obj$summary["aRange"]
 lambdaMLE<- obj$summary["lambda"]

  for(  k in 1:M){
  cat( k, " " )
# here the MLEs are found using the easy top level level wrapper
# see mKrigMLEJoint for a more efficient strategy
  out <- mKrigMLEJoint(obj$x, ySynthetic[,k],
                  weights = obj$weights,
              mKrig.args = obj$mKrig.args,
                 cov.function = obj$cov.function.name,
                cov.args = obj$cov.args,
        cov.params.start = list( aRange = aRangeMLE,
                                 lambda = lambdaMLE)
                      )
  newSummary<- out$summary
  bootSummary<- rbind( bootSummary, newSummary)
  }

  cat(  " ", fill=TRUE )
```

```
  obj$summary
  stats( bootSummary)


## End(Not run)
## Not run:
#perform joint likelihood maximization over lambda, aRange, and smoothness.
#note: finding smoothness is not a robust optimization
#      can get a bad answer with poor initial guesses.
obj2<- mKrigMLEJoint(x,Y,
                     cov.args=list(Covariance="Matern"),
                     cov.params.start=list( aRange = .18,
                                      smoothness = 1.1,
                                          lambda = .08),
                      )

#look at lnLikelihood  evaluations
obj2$summary
#compare to REML
obj3<- mKrigMLEJoint(x,Y,
                     cov.args=list(Covariance="Matern"),
                     cov.params.start=list(aRange = .18,
                                     smoothness = 1.1,
                                         lambda = .08),
                      , REML=TRUE)
obj3$summary

## End(Not run)
## Not run:
#look at lnLikelihood  evaluations

# check convergence of MLE to true fit with no fixed part
#
obj4<- mKrigMLEJoint(x,Y,
                     mKrig.args= list( m=0),
                     cov.args=list(Covariance="Matern", smoothness=1),
                     cov.params.start=list(aRange=.2, lambda=.1),
                      REML=TRUE)
#look at lnLikelihood  evaluations
obj4$summary
# nails it!

## End(Not run)
```

`naturalSplineBasis` *Natural spline basis.*

Description

This simple function relies on the splines package and implements de Boor's algorithm to create a B-spline basis. The "natural boundary" conditions constrain higher order derivatives to be zero at the endpoints.

Usage

```
naturalSplineBasis(sGrid, sKnots, degree = 3, derivative = 0)
```

Arguments

sGrid	Values to evaluate the basis functions.
sKnots	Knot points. There will be as many basis functions as knot points, and they must be unique.
degree	Degree of spline; degree = 3 gives a basis that is piecewise cubic polynomial at the knots.
derivative	Order of derivative to evaluate at sGrid. Default is to evaluate the basis functions.

Details

This implements the famous and elegant De Boor algorithm to find the basis functions. Following De Boor's analysis, natural boundary conditions are enforced by repeating knots at the boundaries. With the natural boundary conditions, there are as many basis functions as knots. Although piecewise polynomials with the segments defined by the knots, the functions are constrained to be smooth. For the cubic case, all basis functions will be continuous and have continuous first and second derivatives. At the end points (first and last knot), the functions will have third and second derivatives equal to zero. Thus, extrapolation beyond the end points is linear. For cubic natural splines, the support is over 5 knot values away from the boundaries.

The natural spline basis is also the basis for a smoothing spline and is distinct from more arbitrary choices such as regression splines.

For the curious reader, one can back out the cubic coefficients for this basis by examining the derivatives in each interval – and using your calculus!

Value

If m is the length of sGrid and n is the length of sKnots, the returned value is an m X n matrix with i,j element being the j-th basis function evaluated at sGrid[i].

Author(s)

Douglas Nychka

See Also

Tps, Tps.cov, sreg, splint

Examples

```
library( splines)
sGrid<- seq( 0,1, length.out=200)
sKnots<- seq( 0,1, length.out=8) # 8 knot points
BasisMatrix<- naturalSplineBasis(sGrid, sKnots)
fields.style()
matplot(sGrid, BasisMatrix, lty=1, type="l" )
rug( sKnots, lwd=3)
# Note symmetry but also different shapes near endpoints.
```

NorthAmericanRainfall *Observed North American summer precipitation from the historical climate network.*

Description

Average rainfall in tenths of millimeters for the months of June, July and August (JJA) for the period 1950-2010 (old version, 1700+ locations) or 1971-2023 (new version, 4800+ locations) and for North America. The old version uses monthly data from the Global Historical Climate Network (GHCN) version 2 and the new version uses Version 4.

Format

For both versions, the format is a list with components: "longitude" "latitude" "precip" "elevation" "precipSE" "trend" "trendSE" "type" "x.s" "sProjection" with elevation in meters, longitude as (-180,180), latitude as (-90, 90) and precipitation in 1/10 mm (precip/254 converts to inches of rainfall). Note that the usual convention is to report a total amount of rainfall over the season. Divide by 3 to get the mean.

`precip` is the intercept for 1980.5 (1997 for new version) when a straight line least squares regression is fit to each station's record. SE is the companion standard error from the least squares fit. If the station is complete, then `precip` and `precipSE` will just be the mean and standard deviation adjusted for a linear trend. The estimated trend `trend` and its standard error `trendSE` are also included.

Also, due to the centering, for complete data the intercept and trend estimate will be uncorrelated. The component `type` indicates whether the station has been "adjusted" (see below) or is still in "unadjusted" form.

`x.s` is a useful transformation of locations into stereographic coordinates that reduces the inflation of northern Canada due to the usual lon/lat coordinates. Specifically, it is found by:

```
 library(mapproj)
xStereo<- mapproject( NorthAmericanRainfall$lon,NorthAmericanRainfall$lat, projection="stereographi
 NorthAmericanRainfall$x.s<- cbind( xStereo$x, xStereo$y)
 hold<- .Last.projection()
  NorthAmericanRainfall$orientation<- hold$orientation
  NorthAmericanRainfall$projection <- hold$projection
```

Source

The monthly data used to construct NorthAmericanRainfall was generously provided by Xuebin Zhang; however, the original source is freely available as the Global Historical Climate Network Version 2 Precipitation quality controlled, curated and served by the US National Centers for Environmental Information. The adjusted data from this archive has been modified from its raw form to make the record more homogeneous. Heterogeneities can come from a variety of sources such as moving the station a short distance or changes in instruments. See the National Centers for Environmental Information, then Access Data, and then GHCN.

The new version NorthAmericanRainfall2 uses the GHCN "Version 4" data product. There are substantially more stations reported in this data set. To reduce the size, stations for this R data set were selected if they had more than 80 percent of the months reported over the period 1971-2023. This subset amounts to 4893 locations. A simple linear regression was fit to each station record and the OLS trend, trend SE, and trend line at 1997 are evaluated to get a "mean" rainfall estimate for this location.

See the fields GitHub repository for the source code and source data to create this data set.

Examples

```
data(NorthAmericanRainfall)
x<- cbind(NorthAmericanRainfall$longitude,  NorthAmericanRainfall$latitude)
y<- NorthAmericanRainfall$precip
quilt.plot( x,y)
world( add=TRUE)

XMatstat<- NorthAmericanRainfall$trend / NorthAmericanRainfall$trendSE
quilt.plot( x, XMatstat)

## Not run:
# with a better projection
# note that you may need to install the mapproj package
library( maps)
library( mapproj)
bubblePlot( NorthAmericanRainfall$x.s,
NorthAmericanRainfall$precip,
size=.3, col=rev(tim.colors()) )
map("world",
projection=NorthAmericanRainfall$projection,
orientation= NorthAmericanRainfall$orientation,
add=TRUE, col="grey30")

## End(Not run)
# note station sparsity outside CONUS and lower CA.

# comparing older and newer versions

data(NorthAmericanRainfall)
bubblePlot( NorthAmericanRainfall$lon, NorthAmericanRainfall$lat,
NorthAmericanRainfall$precip,
size=.3, col=rev(tim.colors()) )
world( add=TRUE, col="grey30")
```

```
data(NorthAmericanRainfall2)
bubblePlot( NorthAmericanRainfall2$lon,NorthAmericanRainfall2$lat,
NorthAmericanRainfall2$precip,
size=.3, col=rev(tim.colors()), xlim= c(-130,-50))
world( add=TRUE, col="grey30")
```

`offGridWeights` *Utilities for fast spatial prediction.*

Description

Based on a stationary Gaussian process model, these functions support fast prediction onto a grid using a sparse matrix approximation. They also allow for fast prediction to off-grid values (aka interpolation) from an equally spaced rectangular grid using a spatial model. The sparsity comes about because only a fixed number of neighboring grid points (NNSize) are used in the prediction. The prediction variance for off-grid locations is also given in the returned object. These functions are used as the basis for approximate conditional simulation for large spatial datasets and also for fast spatial prediction from irregular locations onto a grid.

Usage

```
offGridWeights(s, gridList, NNSize = 2, mKrigObject = NULL, Covariance = NULL,
   covArgs = NULL, aRange = NULL, sigma2 = NULL, giveWarnings = TRUE,
   debug=FALSE,verbose = FALSE)

   offGridWeights1D(s, gridList, NNSize = 2, mKrigObject = NULL,
   Covariance = NULL,
   covArgs = NULL, aRange = NULL, sigma2 = NULL, giveWarnings = TRUE,
   debug=FALSE, verbose = FALSE)

   offGridWeights2D(s, gridList, NNSize = 2, mKrigObject = NULL,
                Covariance = NULL, covArgs = NULL, aRange = NULL,
                sigma2 = NULL, giveWarnings = TRUE, debug = FALSE,
                findCov = TRUE, verbose = TRUE)

  approximateCovariance2D(s, gridList, NNSize = 4, mKrigObject = NULL,
                Covariance = "Matern", aRange = NULL, sigma2 = 1,
                covArgs = NULL, giveWarnings = TRUE, debug = FALSE,
                verbose = FALSE)



augmentPredictionGrid(gridList=NULL, s, nx=NULL, ny=NULL, NNSize,
                                  verbose=FALSE)
```

```
findGridBox(s, gridList)

mKrigFastPredictSetup(mKrigObject,
                          gridList,
                          NNSize=4,
     giveWarnings = TRUE, verbose = FALSE)
```

Arguments

aRange	The range parameter.
covArgs	If mKrigObject is not specified, a list giving any additional arguments for the covariance function.
Covariance	The stationary covariance function (taking pairwise distances as its first argument).
debug	If TRUE, returns intermediate calculations and structures for debugging and checking.
findCov	If TRUE, the prediction variances (via the Kriging model and assumed covariance function) for the off-grid locations are found. If FALSE, just the prediction weights are returned. This option is a bit faster. This switch is only implemented for the 2-D case, since the extra computational demands for 1-D are modest.
giveWarnings	If TRUE, will warn if two or more observations are in the same grid box. See details below.
gridList	A list in the gridList format (x and y components) that describes the rectangular grid. The grid must have at least NNSize extra grid points beyond the range of the points in s. See grid.list.
mKrigObject	The output object (a list with some specific components) from either mKrig or spatialProcess. This has the information about the covariance function used to do the Kriging. If this object is not supplied, the following items are used instead. See the example below for more details.
nx	Number of grid points in X
ny	Number of grid points in Y
NNSize	Number of nearest neighbor grid points to use for prediction. NNSize = 1 will use the 4 grid points that bound the off-grid point. NNSize = 2 will be a 4X4 subgrid that defines a 3X3 set of boxes and the middle grid box containing the off-grid point. In general, there will be (2*NNSize)^2 neighboring points used.
s	Off-grid spatial locations
sigma2	Marginal variance of the process.
verbose	If TRUE, intermediate output is printed for debugging.

Details

offGridWeights This function creates the interpolation weights taking advantage of some efficiency in the covariance function being stationary, use of a fixed configuration of nearest neighbors, and Kriging predictions from a rectangular grid.

The returned matrix is in spam sparse matrix format. See example below for the "one-liner" to make the prediction once the weights are computed. Although created primarily for (approximate) conditional simulation of a spatial process, this function is also useful for interpolating to off-grid locations from a rectangular field. It is also used for fast, but approximate, prediction for Kriging with a stationary covariance.

The function `offGridWeights` is a simple wrapper to call either the 1D or 2D functions.

In most cases, one would not use these approximations for a 1D problem. However, the 1D algorithm is included as a separate function for testing and also because it is easier to read and to understand the conversion between the Kriging weights for each point and the sparse matrix encoding of them.

The interpolation errors are also computed based on the nearest neighbor predictions. This is returned as a sparse matrix in the component SE. If all observations are in different grid boxes, then `SE` is diagonal and agrees with the square root of the component `predictionVariance`, but if multiple observations are in the same grid box, then SE has blocks of upper triangular matrices that can be used to simulate the prediction error dependence among observations in the same grid box. Explicitly, if `obj` is the output object and there are `nObs` observations, then

```
error <- obj$SE%*% rnorm( nObs)
```

will simulate a prediction error that includes the dependence. Note that in the case that all observations are in separate grid boxes, this code line is the same as

```
error <- sqrt(obj$predictionVariance)*rnorm( nObs)
```

It is always true that the prediction variance is given by `diag( obj$SE%*% t( obj$SE))`.

The user is also referred to the testing scripts `offGridWeights.test.R` and `offGridWeights.testNEW.R` in `tests` where the Kriging predictions and standard errors are computed explicitly and tested against the sparse matrix computation. This is helpful in defining exactly what is being computed.

Returned value below pertains to the offGridWeights function.

approximateCovariance2d This is used to approximate a stationary covariance for fast prediction. The weights required are the same as those used for the local simulation and so this function is included here. See `predictSurface` with the `fast` option for more details.

augmentPredictionGrid This is a supporting function that adds extra grid points to a gridList so that every irregular point has a complete number of nearest neighbors. Typically, the initial grid has as its limits the ranges of the observations. This function adds more grid points at the edges so that there are `NNSize` neighbors around every data location.

findGridBox This is a handy function that finds the indices of the lower left grid box that contains the points in `s`. If the point is not contained within the range of the grid, then NA is returned for the index. This function assumes that the grid points are equally spaced.

Value

`B` A sparse matrix that is of dimension mXn with m the number of locations (rows) in `s` and n being the total number of grid points. `n = length(gridList$x)*length(gridList$y)`

predictionVariance
: A vector of length equal to the rows of s with the Kriging prediction variance based on the nearest neighbor prediction and the specified covariance function.

SE
: A sparse matrix that can be used to simulate dependence among prediction errors for observations in the same grid box. (See explanation above.)

Author(s)

Douglas Nychka and Maggie Bailey

References

Bailey, Maggie D., Soutir Bandyopadhyay, and Douglas Nychka. "Adapting conditional simulation using circulant embedding for irregularly spaced spatial data." Stat 11.1 (2022): e446.

See Also

interp.surface, mKrigFastPredict

Examples

```
# an M by M  grid
M<- 400
xGrid<- seq( -1, 1, length.out=M)
gridList<- list( x= xGrid,
                 y= xGrid
                 )
NNSize<- 3
 n<- 100
# sample n locations but avoid margins so that every point has a full set of nearest neighbors

set.seed(123)
s<- matrix(    runif(n*2, xGrid[(NNSize+1)],xGrid[(M-NNSize)]),
               n, 2 )


obj<- offGridWeights( s, gridList, NNSize=3,
                   Covariance="Matern",
                   aRange = .1, sigma2= 1.0,
                   covArgs= list( smoothness=1.0)
                   )

# make the predictions  by obj$B%*%c(y)
# where y is the matrix of values on the grid

# try it out on a simulated  Matern field
CEobj<- circulantEmbeddingSetup( gridList,
                  cov.args=list(
                  Covariance="Matern",
                   aRange = .1,
                    smoothness=1.0)
                    )
```

```
 set.seed( 333)
Z<- circulantEmbedding(CEobj)

#
# Note that grid values are "unrolled" as a vector
# for multiplication
# predOffGrid<- obj$B%*% c( Z)

predOffGrid<- obj$B%*% c( Z)

set.panel( 1,2)
zr<- range( c(Z))
image.plot(gridList$x, gridList$y, Z, zlim=zr)
bubblePlot( s[,1],s[,2], z= predOffGrid , size=.5,
highlight=FALSE, zlim=zr)
 set.panel()
```

---

ozone2 *Daily 8-hour ozone averages for sites in the Midwest*

Description

The response is 8-hour average (surface) ozone (from 9AM-4PM) measured in parts per billion (PPB) for 153 sites in the Midwestern US over the period June 3, 1987 through August 31, 1987, 89 days. This season of high ozone corresponds with a large modeling experiment using the EPA Regional Oxidant Model.

Usage

```
data(ozone2)
```

Format

The data list has components:

y An 89X153 matrix of ozone values. Rows are days and columns are the sites.

lon.lat Site locations in longitude and latitude as a 153X2 table.

chicago.subset Logical vector indicating stations that form the smaller Chicagoland subset. (See the fields ChicagoO3 data set.)

Reference: Nychka, D., Cox, L., Piegorsch, W. (1998) Case Studies in Environmental Statistics Lecture Notes in Statistics, Springer Verlag, New York

Examples

```
data( ozone2)

# pairwise correlation among all stations
# ( See cover.design to continue this example)
cor.mat<- cor( ozone2$y, use="pairwise")

#raw data image for day number 16
good<- !is.na( ozone2$y[16,])
out<- as.image( ozone2$y[16,good], x=ozone2$lon.lat[good,])
image.plot( out)
```

---

panelPlot *Creates a panel of plots with common axes using base R graphics.*

Description

This function creates panels of plots in a row/column layout that can have common axes and a single legend or color bar. Its main benefit is that one can customize a single plot in the panel to add additional features. All the code is in base R graphics, so it is useful as a graphics tutorial or as a starting point for modifying the code, and for the user comfortable with base R coding. This function is similar to the lattice package, but the intent is that it be easier to use and modify.

Usage

```
panelPlotCommonAxes(m, n, xr, yr,
                                plotCode,
                                xlabels= "",
                                ylabels="",
                                outerXLabel=NULL,
                                outerYLabel=NULL,
                                PANESPACE=0,
                                LEGENDSPACE=c(0,0,0,0),
                                axes=TRUE,
                                addAxis=c(TRUE, TRUE, TRUE, TRUE),
                                alternateAxes=TRUE,
                                axisMargin=c( 3.1,3.1,3.1,3.1),
                                axisList=list( line=0),
                                axisLabelList=list(line=2),
                                outerLabelMargin= c(3.1,3.1,0,0),
                                outerLabelList=list(line=2),
                                colBg="grey90",
                                colGrid="grey98",
                                boxLwd= 1,
                                titleBoxHeight= 0,
                                titleBoxList= list(
                                boxColor = "lightgreen",
```

```
                              boxBorder= "black",
                              lwd=boxLwd),
                              verbose=FALSE
                              )

      addTitleBox(  boxHeight = 1.2,
                    boxColor="green3",
                    boxBorder="black",
                    lwd=1.0
                    )
```

Arguments

m	Number of panel rows.
n	Number of panel columns.
xr	Range of the horizontal axis/axes. If a two-column matrix, row J of the matrix gives the axis range for column J of the panel.
yr	Range of the vertical axis/axes. If a two-column matrix, row I of the matrix gives the axis range for row I of the panel.
plotCode	A list of expressions for plotting in each panel. See Examples and Details.
xlabels	Labels for the horizontal axes.
ylabels	Labels for the vertical axes.
outerXLabel	Outside horizontal label.
outerYLabel	Outside vertical label.
PANESPACE	Spacing between and around panes in units of a character width.
LEGENDSPACE	A vector of four line spaces in the same order as the mar and oma margins. This is the spacing added around the panel in units of a character width.
axes	If TRUE, add axes to the panel.
addAxis	A logical vector of four values specifying which sides should get axes.
alternateAxes	If TRUE, only every other axis is labeled with numerical values. This makes for less busy panel margins.
axisMargin	Space added for tick marks and the axis labeling.
axisList	A list of other axis arguments. See mtext for the possible choices. The default is a list with just a line component, list(line=0), but the user can add to this for more modifications.
axisLabelList	Same as axisList but for the outer labels.
outerLabelMargin	Same as axisMargin but for the outer label.
outerLabelList	Same as axisLabelList but for the outer label.
colBg	Background color for the plots.
colGrid	Color of grid lines

titleBoxHeight	Height of the title box in units of "lines". This argument is passed to addTitleBox
titleBoxList	A list with other arguments to send to the addTitleBox function.
boxHeight	Height of the title box in units of "lines" (the same units used to position text and axes in the margins).
verbose	If TRUE, print some information about the layout. This is for debugging.
boxColor	Fill color of the box.
boxBorder	Color of the box border.
boxLwd	Line width for the box around each plot and for the title box.
lwd	Thickness of the border lines.

Details

This function was created to make plots in an mXn panel with consistent spacing and to add common axes on the sides. The idea is that if two plots side by side have a common vertical axis, then it should not be replotted in between the two plots. Instead, the second axis should be added on the right side – or omitted. We have resisted making this into a more polished function for two reasons. First, panel plots typically need customization, and so handling all kinds of flexibility would make a general function complex and its code more difficult to follow. Even so, we are not totally happy with how this version reads! There are useful frameworks such as ggplot2 and lattice that provide this kind of added functionality, but in doing so they require some effort to deviate from the default settings.

The second reason is our realization that the basic elements of grouping plots together and adding common axes on the outsides are really not too complicated to code up. The goal of this function was to make the whole process accessible using just base R graphics. In this way, it is easier to modify if one wants some additional customization.

Difference from the lattice package The lattice package is an elegant framework following Cleveland's work on statistical graphics. The main idea is that the same graphics function is applied to subsets of a data set, for example, making separate plots for different categories. In this way, the "for" loop over the separate plots is implicit – determined by the number of different categories. The challenge is that customizing a specific plot in this group, e.g., adding an arrow or an additional line for explanation, is hard. And, of course, while the default spacing and layout are thoughtful, inevitably they will need to be tweaked for the final version in a publication or for a talk. This function takes the opposite point of view in that you specify each plot in the panel separately to have full control. This can be tedious for a panel with many plots and is a disadvantage. However, see the note at the end of this section on how to mimic lattice and draw multiple plots with a single graphics function.

Space The many arguments for spacing are included because, inevitably, to get a final figure, one needs to tweak the spaces around a plot. Sometimes this happens simply when converting the plot viewed in an interactive session to a pdf for a paper or talk.

Basic idea of the function The figure panel is mXn – m rows and n columns – with each plot referred to as a pane. The main part of the function loops through plotting these *row by row*. So if the panel is 3X2, then it plots in the order (1,1), (1,2), (2,1), ..., (3,2). Each pane of the panel is initially set up blank with a generic call to the plot function using the limits in xr and yr, without any axes or plotting. The expressions in plotCode then plot into this pane. Just use R graphics functions that *add* to a plot, e.g., points, lines, matpoints. Functions such as image,

imagePlot, contour, and map with the argument add=TRUE also avoid creating a new plot. The real estate is already set up for you!

Specifying the plotting code So what about the plotting code? This is, at first glance, a bit strange (maybe at second glance too). It is a list with each component being an *expression*. Each expression can have multiple plotting/R code lines separated by commas. So, for example, if one planned to have 4 scatterplots with common axes, based on xr and yr, and using points:

```
plotCode<- list(
                expression( points(x1,y1)),
                expression( points(x2,y2)),
                expression( points(x3,y3)),
                expression( points(x4,y4),
                      title("fourth plot"))
                )
```

Here we added another plot function to the fourth plot – adding a title to show how this is done. If a component of the list is NULL, then that plot is skipped.

Exporting the for loop indices The function has the variables I (row index), J (column index), and counter (plots in order) to keep track of the current plot. These can be used to avoid creating separate plotting code for each plot or to customize a particular case. For example, a single expression can be used with these variables to create the different plots. This is more in the style of the lattice package; see the example below for its use.

Adding legends, color bars and other annotation. Finally, we note that often one wants to draw a panel and then also add a common legend, color bar, or both. Again, we want to do this using base R graphics in a simple way that is easy to customize.

We suggest the following steps: 1) Use LEGENDSPACE to reserve space for the legends, typically to the right or at the bottom. 2) Draw your plot. 3) Use addLegend() to reset your figure region to a single [0,1]X[0,1] plotting region.

4) Add your extra legend or color bar using the [0,1] ranges to place these elements. The reason for the figure reset is to make it easier to think about plotting coordinates for locating the legend. See the examples below. The advantage of step 3 is that both the color bar and legend placement work nicely with a unit square figure region.

Adding a strip of color to the top Lastly, we have included a handy function, addTitleBox, intended for the specific purpose of adding a strip of a different color at the top of a plot to hold a title. This is the style of the lattice package, and it would be annoying to keep coding this if the function were not around. See the examples below. BTW, for a humble statistician this was very tricky to code up! In this function, note how the line spacing in the margin is translated to user coordinates.

This function is also a good example of the care needed using base R graphics. The function turns off the clipping that prevents plotting outside of the plot region. Then it adds a colored rectangle, still using the plot coordinate system. After drawing the rectangle, the clipping is turned back on. In other circumstances, this strategy of plotting outside the normal plot rectangle might easily go awry and give strange results.

Author(s)

Douglas Nychka

See Also

imagePlot,addLegend, addColorBar

Examples

```
# A 2x2 panel to show sampling variability of the N(0,1)
# scatter plots  from 4 samples

set.seed(123)
x<- matrix( rnorm(50*6), ncol=6,nrow=50)
y<- matrix( rnorm(50*6), ncol=6, nrow=50)

# begin list of expressions
plotCode<-  list(
expression( points(x[,1], y[,1],pch=1)
),
expression(
points(x[,2], y[,2],pch=2)
),
expression(
points(x[,3], y[,3],pch=3)
),
expression(
points(x[,4], y[,4], pch=4)
)
)# end list

#
xr<- range( c(x))
yr<- range( c(y))

############################################################
# a basic plot relying on all the default choices
############################################################
panelPlotCommonAxes(
m=2,n=2,
xr,yr,
plotCode
)
############################################################
# same plot but using the indexing within the for loop
############################################################
plotCode1<-  list(
expression( points(x[,counter], y[,counter],pch=counter, col=counter)
)
)
# plotting code reused for each pane of the panel
# because this list has length 1
fields.style() # handy reset of some  default graphics sizes and colors
panelPlotCommonAxes(
m=2,n=2,
xr,yr,
```

```
plotCode1
)

############################################################
# add some titles in a color strip
############################################################

plotCode2<-  list(
expression( points(x[,1], y[,1],pch=1),
title("Case 1", adj=.2, line=.2)
),
expression(
points(x[,2], y[,2],pch=2),
title("Case 2", adj=.2, line=.2)
),
expression(
points(x[,3], y[,3],pch=3),
title("Case 3", adj=.2, line=.2)
),
expression(
points(x[,4], y[,4], pch=4),
title("Case 4", adj=.2, line=.2)
)
)# end list

panelPlotCommonAxes(
m=2,n=2,
xr,yr,
plotCode2,
titleBoxHeight=1.2
)


############################################################
# adding a legend, turning off some axes, and adding labels
# also mess with the x ranges so there are different axes
############################################################

# different ranges for x -- something harder to do in the lattice package
xr<- rbind( c( -3,3),
            c(-4,4))
#

panelPlotCommonAxes(m=2,n=2,
# ranges
xr,yr,
# the code for the four plots
plotCode2,
# outer labels
outerXLabel ="X",
outerYLabel= "Y",
# picky about which axes are plotted
addAxis=c(TRUE, TRUE, FALSE, FALSE),
```

```
alternateAxes=FALSE,
# room for a title
titleBoxHeight=1.2,
# amount of space (in lines) for the axes on the sides of the panel
axisMargin=c( 4.1,4.1,0,0),
# extra room for outer labels
# tight margin
outerLabelMargin= c( 1,1,0,0),
# 5 extra spaces on the right reserved for the legend
LEGENDSPACE=c(0,0,0,5),
boxLwd=1
)
#
# Now the base R magic of adding the legend
# reset the figure region to (0,1) X (0,1) coordinates
addLegend()
cXY<- par()$cxy # character size for these dimensions
# this is useful for thinking in units of "lines"
# now call the usual legend function ...
legend(
"topright",
# set the number of lines in from the top-right corner
# adjust as needed ...
inset=c(cXY[1]*1.5,cXY[2]*3.5), #  about 1.5 and 3.5 lines from corner
legend=c("do","re","mi", "fa"),
pch= 1:4, cex=1.2)

###

## Not run:
dataLattice<- data.frame( X=c(x), Y=c(y), cases=
as.factor(rep(1:6, rep(nrow(x),6))) )

library( lattice)

# clean call to lattice using all its defaults
# but plot in reverse order  ( a table)

xyplot( Y~X|cases, data=dataLattice,
main= "lattice panel plot ", as.table=TRUE )

# use the model of lattice/trellis: the same graphics applied to each subset
plotCode3<-  list(
expression( points(x[,counter], y[,counter],
col="#0080ff", #medium sky blue default for lattice
cex=1.5),
title(format(counter),  line=.2))
)

xr<- range( c( x))
yr<- range( c( y))

# quick with defaults
```

```
 panelPlotCommonAxes(m=2,n=3,
                     xr,yr,
                     plotCode3,
                     outerXLabel ="X",
                     outerYLabel= "Y",
                     titleBoxHeight=1.2
 )

# matching lattice
# note some futzing around with spacing and color
# grid and background are turned off -- not our first choice
#
 panelPlotCommonAxes(m=2,n=3,
                     xr,yr,
                     plotCode3,
                     outerXLabel ="X",
                     outerYLabel= "Y",
                     axisMargin=c(1,1,1,1) ,
                     outerLabelMargin= c(4,4,4,4), #tight margin
                     titleBoxHeight=1.2,
                     outerLabelList=list(line=3),
                     titleBoxList= list(
                       boxColor = "grey90",
                       boxBorder= "black",
                       lwd=1),
                       colBg="white",
                       colGrid="white",
                     axisLabelList=list(line=3),
 )

 # this is complicated but most shows how to control
 # exactly where this will go
 mtext( text= "fields panel plot ", outer=TRUE,
 side =3, line = 3,
 font = 2, cex = 1.2)
# BTW typically in a publication the caption subsumes any title in the figure


# and doing this in ggplot2
# not all the feature are easily reproduciable
#-- we  suspect the ggplot folks would prefer the plot the way it is!

library(ggplot2)
ggplot(dataLattice, aes(x = Y, y = X)) +
  geom_point(shape = 1, colour = "#0080ff" ) +
  facet_wrap(~ cases, nrow = 3, ncol = 2) +
  theme_bw() +
  theme(panel.spacing = unit(0, "lines"),
        strip.background = element_rect(fill = "grey80"))

## End(Not run)
```

`plot.Krig` *Diagnostic and summary plots of a Kriging, spatialProcess or spline object.*

Description

Plots a series of four diagnostic plots that summarize the fit.

Usage

```
## S3 method for class 'Krig'
plot(x, digits=4, which= 1:4,...)
## S3 method for class 'sreg'
plot(x, digits = 4, which = 1:4, ...)
```

Arguments

`x`	A Krig or an sreg object
`digits`	Number of significant digits for the RMSE label.
`which`	A vector specifying by number which of the four plots to draw. 1:4 plots all four.
`...`	Optional graphics arguments to pass to each plot.

Details

This function creates four summary plots of the Krig or sreg object. The default is to put these on separate pages. However, if the screen is already divided in some other fashion the plots will just be added according to that scheme. This option is useful to compare several different model fits.

The first is a scatterplot of predicted value against observed.

The second plot is "standardized" residuals against predicted value. Here we mean that the residuals are divided by the GCV estimate for tau and multiplied by the square root of any weights that have been specified. In the case of a "correlation model" the residuals are also divided by the marginal standard deviation from this model.

The third plot shows the values of the GCV function against the effective degrees of freedom. When there are replicate points, several versions of the GCV function may be plotted. The GCV function is with respect to the standardized data if a correlation model is specified. A vertical line indicates the minimum found.

For `Krig` and `sreg` objects, the fourth plot is a histogram of the standardized residuals. For sreg, if multiple lambdas are given, boxplots of the residuals for each fit are plotted.

For a `spatialProcess` object, the fourth plot is the profile likelihood for the aRange parameter. Points are the actual evaluated log likelihoods and the dashed line is just a spline interpolation to help with visualization.

See Also

Krig, spatialProcess, summary.Krig, Tps, set.panel

Examples

```
data( ozone2)
x<- ozone2$lon.lat
y<- ozone2$y[16,]
fit1<-Krig(x,y, aRange=200)
# fitting a surface to ozone
# measurements
set.panel( 2,2)
plot(fit1)

# fit rat data
data(rat.diet)
fit3<-sreg(rat.diet$t,rat.diet$con)
set.panel(2,2)
plot(fit3)

set.panel(1,1) # reset graphics window.
```

---

plot.surface *Plots a surface*

Description

Plots a surface object in several different ways to give 3-d information, e.g., a contour plot or perspective plot.

Usage

```
## S3 method for class 'surface'
plot(x, main = NULL, type = "C", zlab = NULL, xlab = NULL,
    ylab = NULL, levels = NULL, zlim = NULL, graphics.reset = NULL,
     labcex = 0.6, add.legend=TRUE, ...)
```

Arguments

x	A surface object. At a minimum, a list with components x,y and z in the same form as the input list for the standard contour, persp or image functions. This can also be an object from predictSurface.
main	Title for plot.
type	type="p" for a perspective/drape plot (see drape.plot), type="I" for an image plot with a legend strip (see image.plot), type="c" draws a contour plot, type="C" is the "I" option but with contour lines added. type="b" gives both "p" and "C" as a 2X1 panel.

zlab	z-axis label
xlab	x-axis label
ylab	y-axis label
levels	Vector of levels to be passed to contour function.
graphics.reset	Reset to original graphics parameters after function plotting. Default is to reset if type ="b", but not for the single plot options.
zlim	Sets z limits on perspective plot.
labcex	Label sizes for axis labeling etc.
add.legend	If TRUE, adds a legend to the draped perspective plot
...	Other graphical parameters that are passed along to either drape.plot or image.plot

See Also

surface, predictSurface, as.surface, drape.plot, image.plot

Examples

```
x<- seq( -2,2,,80)
y<- seq( -2,2,,80)
# test image
z<- outer( x,y, "+")
# basic image/surface object
obj<- list(x=x, y=y,z=z)

#  contour plot
# note how graphical parameters appropriate to contour are passed
plot.surface( obj, type="c", col="red")

# using a fields function to fit a surface and evaluate as surface object.
data(BD)
fit<- Tps( BD[,1:4], BD$lnya) # fit surface to data
# surface of variables 2 and 3 holding 1 and 4 fixed at their median levels
 out.p<-predictSurface(fit, xy=c(2,3))

 plot.surface(out.p) # surface plot
```

---

poly.image *Image plot for cells that are irregular quadrilaterals.*

Description

Creates an image using polygon filling based on a grid of irregular quadrilaterals. This function is useful for a regular grid that has been transformed to another nonlinear or rotated coordinate system. This situation comes up in lon-lat grids created under different map projections. Unlike the usual image format, this function requires the grid to be specified as two matrices x and y that give the grid x and y coordinates explicitly for every grid point.

Usage

```
poly.image(x, y, z, col = tim.colors(64), breaks,
                 transparent.color = "white", midpoint = FALSE, zlim =
                 range(z, na.rm = TRUE), xlim = range(x), ylim =
                 range(y), add = FALSE, border = NA, lwd.poly = 1, asp
                 = NA, ...)

poly.image.regrid(x)
```

Arguments

x	A matrix of the x locations of the grid.
y	A matrix of the y locations of the grid.
z	Values for each grid cell. Can either be the value at the grid points or interpreted as the midpoint of the grid cell.
col	Color scale for plotting.
breaks	Numerical breaks to match to the colors. If missing, breaks are equally spaced on the range zlim.
transparent.color	Color to plot cells that are outside the range specified in the function call.
midpoint	Only relevant if the dimensions of x,y, and z are the same. If TRUE, the z values will be averaged and then used as the cell midpoints. If FALSE, the x/y grid will be expanded and shifted to represent grid cell corners. (See poly.image.regrid.)
zlim	Plotting limits for z.
xlim	Plotting limits for x.
ylim	Plotting limits for y.
add	If TRUE, will add the image onto current plot.
border	Color of the edges of the quadrilaterals; the default is no color.
lwd.poly	Line width for the mesh surface, i.e., the outlines of the quadrilateral facets. This might have to be set smaller than one if rounded corners on the facets are visible.
asp	The plot aspect with similar function to that in the plot function.
...	If add is FALSE, additional graphical arguments that will be supplied to the plot function.

Details

This function is straightforward except in the case when the dimensions of x,y, and z are equal. In this case, the relationship of the values to the grid cells is ambiguous and the switch midpoint gives two possible solutions. The z values at 4 neighboring grid cells can be averaged to estimate a new value interpreted to be at the center of the grid. This is done when midpoint is TRUE. Alternatively, the full set of z values can be retained by redefining the grid. This is accomplished by finding the midpoints of x and y grid points and adding two outside rows and columns to complete the grid. The new result is a new grid that is (M+1)X (N+1) if z is MXN. These new grid points define cells

that contain each of the original grid points as their midpoints. Of course, the advantage of this alternative is that the values of z are preserved in the image plot, a feature that may be important for some uses.

The function image.plot uses this function internally when image information is passed in this format and can add a legend. In most cases, just use image.plot.

The function `poly.image.regrid` does a simple averaging and extrapolation of the grid locations to shift from midpoints to corners. In the interior, grid corners are found by the average of the 4 closest midpoints. For the edges, the corners are just extrapolated based on the separation of neighboring grid cells.

Author(s)

Doug Nychka

See Also

image.plot

Examples

```
data(RCMexample)
set.panel( 1,2)
par(pty="s")
# plot with grid modified
poly.image( RCMexample$x, RCMexample$y, RCMexample$z[,,1])

# use midpoints of z
poly.image( RCMexample$x, RCMexample$y, RCMexample$z[,,1],midpoint=TRUE)

  set.panel()
# an example with quantile breaks

 brk<- quantile(  RCMexample$z[,,1], c( 0, .9,.95,.99,1.0) )
 poly.image( RCMexample$x, RCMexample$y, RCMexample$z[,,1], breaks=brk, col=
    rainbow(4))


# images are very similar.
  set.panel()
# Regridding of x and y
  l1<- poly.image.regrid( RCMexample$x)
  l2<- poly.image.regrid( RCMexample$y)

# test that this works
  i<- 1:10
  plot( l1[i,i], l2[i,i])
  points( RCMexample$x[i,i], RCMexample$y[i,i],col="red")
```

predict.Krig *Evaluation of Krig spatial process estimate.*

Description

Provides predictions from the Krig spatial process estimate at arbitrary points, new data (Y) or other values of the smoothing parameter (lambda) including a GCV estimate.

Usage

```
## S3 method for class 'Krig'
predict(
object,
       x = NULL,
    XMat = NULL, drop.XMat = FALSE,
       Z = NULL, drop.Z = NULL,
       just.fixed = FALSE, lambda = NA, df = NA, model = NA,
                 eval.correlation.model = TRUE, y = NULL, yM = NULL,
                 verbose = FALSE, ...)
predictDerivative.Krig(object, x = NULL,  verbose = FALSE,...)

## S3 method for class 'Tps'
predict(object, ... )

## S3 method for class 'fastTps'
predict(object, xnew = NULL, grid.list = NULL, ynew = NULL,
                 derivative = 0, XMat = NULL, drop.XMat = FALSE, just.fixed =
                 FALSE, xy = c(1, 2), ...)
```

Arguments

derivative	The degree of the derivative to be evaluated. Default is 0 (evaluate the function itself), 1 is supported by some covariance functions. Higher derivatives are not supported in this version and for mKrig.
df	Effective degrees of freedom for the predicted surface. This can be used in place of lambda (see the function Krig.df.to.lambda)
eval.correlation.model	If TRUE (the default), will multiply the predicted function by the marginal SDs and add the mean function. This is usually what one wants. If FALSE, will return predicted surface in the standardized scale. The main use of this option is a call from Krig to find MLEs of sigma and tau2
grid.list	A grid.list specifying a grid of locations to evaluate the fitted surface.
just.fixed	Only the fixed part of the model is evaluated
lambda	Smoothing parameter. If omitted, the value in the lambda of the object will be used. (See also df and gcv arguments.)

model	Generic argument that may be used to pass a different lambda.
object	Fit object from the Krig, Tps, mKrig, or fastTps functions.
verbose	Print out all kinds of intermediate stuff for debugging
xy	The column positions that locate the x and y variables for evaluating on a grid. This is mainly useful if the surface has more than 2 dimensions.
y	Evaluate the estimate using the new data vector y (in the same order as the old data). This is equivalent to recomputing the Krig object with this new data but is more efficient because many pieces can be reused. Note that the x values are assumed to be the same.
x	Matrix of x values on which to evaluate the kriging surface. If omitted, the data x values will be used.
xnew	Same as x above.
ynew	Same as y above.
yM	If not NULL, evaluate the estimate using this vector as the replicate mean data. That is, assume the full data has been collapsed into replicate means in the same order as xM. The replicate weights are assumed to be the same as the original data. (weightsM)
XMat	Vector/Matrix of additional covariates to be included in the fixed part of the spatial model
drop.XMat	If TRUE, only the spatial fixed part of the model is evaluated, i.e., XMat covariates are not used.
Z	This argument is deprecated and a warning will be printed. Should use the new argument name XMat.
drop.Z	An older name for drop.XMat. If used, a warning will be printed.
...	Other arguments passed to the covariance function. In the case of fastTps, these are the same arguments as predict.mKrig. This argument is usually not needed.

Details

The main goal in this function is to reuse the Krig object to rapidly evaluate different estimates. Thus, there is flexibility in changing the value of lambda and also the independent data without having to recompute the matrices associated with the Krig object. The reason this is possible is that most of the calculations depend on the observed locations and not on lambda or the observed data. Note that the version for evaluating partial derivatives does not provide the same flexibility as predict.Krig and makes some assumptions about the null model (as a low order polynomial) and cannot handle the correlation model form.

Value

Vector of predicted responses or a matrix of the partial derivatives.

See Also

Krig, predictSurface, gcv.Krig

Examples

```
  data(ChicagoO3)
  Krig(ChicagoO3$x,ChicagoO3$y, aRange=50) ->fit
  predict( fit) # gives predicted values at data points should agree with fitted.values
                #  in fit object

# predict at the coordinate (-5,10)
  x0<- cbind( -5,10) # has to be a  1X2 matrix
  predict( fit,x= x0)

# redoing predictions at data locations:
   predict( fit, x=ChicagoO3$x)

# only the fixed part of the model
  predict( fit, just.fixed=TRUE)

# evaluating estimate at a grid of points
  grid<- make.surface.grid( list( seq( -40,40,,15), seq( -40,40,,15)))
  look<- predict(fit,grid) # evaluate on a grid of points

# some useful graphing functions for these gridded predicted values
  out.p<- as.surface( grid, look) # reformat into $x $y $z image-type object
  contour( out.p)

# see also the functions predictSurface and surface
# for functions that combine these steps


# refit with 10 degrees of freedom in surface
  look<- predict(fit,grid, df=15)
# refit with random data
  look<- predict( fit, grid, y= rnorm( 20))


# finding partial derivatives of the estimate
#
# find the partial derivatives at observation locations
# returned object is a two column matrix.
# this does not make sense for the exponential covariance
# but can illustrate this with a thin plate spline with
# a high enough order ( i.e. need m=3 or greater)
#
  data(ozone2)
# the 16th day of this ozone spatial dataset
  fit0<- Tps( ozone2$lon.lat, ozone2$y[16,], m=3)
  look1<- predictDerivative.Krig( fit0)
# for extra credit compare this to
  look2<- predictDerivative.Krig( fit0, x=ozone2$lon.lat)
# (why are there more values in look2)
```

predictSE *Standard errors of predictions for Krig spatial process estimate*

Description

Finds the standard error (or covariance) of prediction based on a linear combination of the observed data. The linear combination is usually the "Best Linear Unbiased Estimate" (BLUE) found from the Kriging equations. This statistical computation is done under the assumption that the covariance function is known.

Usage

```
predictSE(object, ...)
## S3 method for class 'Krig'
predictSE(object, x = NULL, cov = FALSE, verbose = FALSE,...)
## S3 method for class 'mKrig'
predictSE(object, xnew = NULL, XMat = NULL, verbose = FALSE, drop.XMat
                 = FALSE, ...)
```

Arguments

drop.XMat	If FALSE, find standard error without including the additional spatial covariates described by XMat. If TRUE, find full standard error with spatial covariates if they are part of the model.
object	A fitted object that can be used to find prediction standard errors. This is usually from fitting a spatial model to data, e.g., a Krig or mKrig object.
xnew	Points to compute the prediction standard error or the prediction cross covariance matrix.
x	Same as xnew – points to compute the prediction standard error or the prediction cross covariance matrix.
cov	If TRUE, the full covariance matrix for the predicted values is returned. Make sure this will not be big if this option is used. (e.g., a 50X50 grid will return a matrix that is 2500X2500!) If FALSE, just the marginal standard deviations of the predicted values are returned. Default is FALSE – of course.
verbose	If TRUE, will print out various information for debugging.
...	Additional arguments passed to the predictSE function.
XMat	Additional matrix of spatial covariates used for prediction. These are used to determine the additional covariance contributed in the fixed part of the model.

Details

The predictions are represented as a linear combination of the dependent variable, Y. Call this LY. Based on this representation, the conditional variance is the same as the expected value of (P(x) + XMat(X) - LY)**2, where P(x)+XMat(x) is the value of the surface at x and LY is the linear

combination that estimates this point. Finding this expected value is straightforward given the unbiasedness of LY for P(x) and the covariance for XMat and Y.

In these calculations, it is assumed that the covariance parameters are fixed. This is an approximation, since in most cases they have been estimated from the data. It should also be noted that if one assumes a Gaussian field and known parameters in the covariance, the usual Kriging estimate is the conditional mean of the field given the data. This function finds the conditional standard deviations (or full covariance matrix) of the fields given the data.

There are two useful extensions supported by this function: adding the variance to the estimate of the spatial mean if this is a correlation model (see help file for Krig), and calculating the variances under covariance misspecification. The function `predictSE.KrigA` uses the smoother matrix (A(lambda)) to find the standard errors or covariances directly from the linear combination of the spatial predictor. Currently this is also the calculation in `predictSE.Krig`, although a shortcut is used in `predictSE.mKrig` for mKrig objects.

Value

A vector of standard errors for the predicted values of the Kriging fit.

See Also

spatialProcess, mKrig, Krig, predict.Krig, predictSurfaceSE

Examples

```
#
 data(ozone2)
 sOzone<- ozone2$lon.lat
 yOzone<- ozone2$y[16,]
# Matern smoothness =1.0 sigma2 and tau found by ML
  fit0<- spatialProcess( sOzone,yOzone, aRange=10)
# prediction SEs at  miss data locations
  missingObs<- is.na(yOzone)
  missingSE<- predictSE(fit0, sOzone[missingObs,])
# NOTE based on dispatching predictSE.mKrig is called.
# 95% CI for  ozone surface
   fHat<- predict(fit0, sOzone[missingObs,])
   cbind(fHat - 1.96*missingSE, fHat + 1.96*missingSE )
# 95% CI for  ozone _observation_
   fHat<- predict(fit0, sOzone[missingObs,])
   tau2<- fit0$summary["tau"]^2
   cbind(fHat - 1.96*sqrt(missingSE + tau2) ,
         fHat + 1.96*sqrt(missingSE + tau2) )

# SE on  a  grid of locations
  xg<-make.surface.grid(
  list(x=seq(-91,-83,,30),
       y=seq( 37, 45,,30))
  )
  gridSE<- predictSE(fit0,xg)   # std errors of predictions

#at the grid points out is a vector of length 900
```

```
#reshape the grid points into a 30X30 matrix etc.

   out.p<-as.surface( xg, gridSE)
   surface( out.p, type="C")
   points( sOzone[!missingObs,], col="magenta", pch=16)

# this is equivalent to the higher level functions
# (We are lazy and the grid will be a bit different here. See the
# help file!)
 out<- predictSurfaceSE( fit0, nx=30, ny=30, extra=TRUE)
 imagePlot( out, col=viridis(256))
 contour( out, add=TRUE, levels=c( 10,20, 20))
 points( sOzone[!missingObs,], col="magenta", pch=16)
```

---

predictSurface *Evaluates a fitted function or the prediction error as a surface that is suitable for plotting with the image, persp, or contour functions.*

Description

Evaluates a fitted model or the prediction error on a 2-D grid, keeping any other variables constant. The resulting object is suitable for use with functions for viewing 3-d surfaces such as image, imagePlot and contour.

Usage

```
## Default S3 method:
predictSurface(object, grid.list = NULL,
                     extrap = FALSE, chull.mask = NA, nx = 80, ny = 80,
                        xy = c(1,2),  verbose = FALSE, ...)

## S3 method for class 'mKrig'
predictSurface(
object, gridList = NULL, grid.list = NULL, ynew =
                 NULL, extrap = FALSE, chull.mask = NA, nx = 80, ny =
                 80, xy = c(1, 2), verbose = FALSE,
                 XMatGrid = NULL, drop.XMat = FALSE,
                 ZGrid = NULL, drop.Z = NULL,
                 just.fixed = FALSE, fast = FALSE,
                 NNSize = 4, setupObject = NULL, giveWarnings = FALSE,
                 derivative = 0, ...)

mKrigFastPredict(object, gridList, ynew = NULL, derivative = 0, XMatGrid =
                 NULL, drop.XMat = FALSE, NNSize = 4, setupObject = NULL,
                 giveWarnings = TRUE, verbose = FALSE)
```

```
## S3 method for class 'Krig'
predictSurface(object, grid.list = NULL, extrap = FALSE, chull.mask = NA,
nx = 80, ny = 80, xy = c(1, 2), verbose = FALSE,
XMatGrid = NULL,  drop.XMat = FALSE,
ZGrid = NULL, drop.Z = NULL,
 just.fixed=FALSE,  ...)

## S3 method for class 'fastTps'
predictSurface(object, gridList = NULL,
       extrap = FALSE, chull.mask = NA, nx = 80, ny = 80,
       xy = c(1,2),  verbose = FALSE, ...)


## Default S3 method:
predictSurfaceSE(object, grid.list = NULL, extrap = FALSE, chull.mask =
                 NA, nx = 80, ny = 80, xy = c(1, 2), verbose = FALSE,
                 XMatGrid = NULL, just.fixed = FALSE, ...)
```

Arguments

object	An object from fitting a function to data. In fields this is usually a Krig, mKrig, or fastTps object.
gridList	A list with as many components as variables describing the surface. All components should have a single value except the two that give the grid points for evaluation. If the matrix or data frame has column names, these must appear in the grid list. See the grid.list help file for more details. If this is omitted and the fit just depends on two variables, the grid will be made from the ranges of the observed variables. (See the function `fields.x.to.grid`.)
grid.list	Alternative to the `gridList` argument.
giveWarnings	If TRUE, will warn when more than one observation is in a grid box.
extrap	Extrapolation beyond the range of the data. If `FALSE` (the default), the predictions will be restricted to the convex hull of the observed data or the convex hull defined from the points from the argument chull.mask. This function may be slightly faster if this logical is set to `TRUE` to avoid checking the grid points for membership in the convex hull. For more complicated masking, a low-level creation of a bounding polygon and testing for membership with `in.poly` may be useful.
chull.mask	Whether to restrict the fitted surface to be on a convex hull; NAs are assigned to values outside the convex hull. chull.mask should be a sequence of points defining a convex hull. Default is to form the convex hull from the observations if this argument is missing (and extrap is FALSE).
nx	Number of grid points in X axis.
ny	Number of grid points in Y axis.
xy	A two-element vector giving the positions for the "X" and "Y" variables for the surface. The positions refer to the columns of the x matrix used to define the multidimensional surface. This argument is provided in lieu of generating the

grid list. If a 4-dimensional surface is fit to data, then `xy= c(2,4)` will evaluate a surface using the second and fourth variables with variables 1 and 3 fixed at their median values. NOTE: this argument is ignored if a grid.list argument is passed.

`drop.XMat` If TRUE, the fixed part of the model depending on covariates is omitted.

`drop.Z` This argument is deprecated and a warning will be printed. Should use the new argument name drop.XMat.

`just.fixed` If TRUE, the nonparametric surface is omitted.

`fast` If TRUE, approximate predictions for stationary models are made using the FFT. For large grids (e.g., nx, ny > 200) this can be substantially faster and still accurate to several decimal places.

`NNSize` Order of nearest neighborhood used for fast prediction. The default, `NNSize = 5`, means an 11X11=121 set of grid points/covariance kernels are used to approximate the off-grid covariance kernel.

`setupObject` The object created explicitly using `mKrigFastPredictSetup`. Useful for predicting multiple surfaces with the same observation locations. If NULL, the setup is done each time the function is called. (See `approximateCovariance2D`.)

`derivative` Predict the estimated derivatives of order `derivative`.

`ynew` New data to use to refit the spatial model. Locations must be the same, but if so, this is efficient because the matrix decompositions are reused.

`...` Any other arguments to pass to the predict function associated with the fit object. Some of the usual arguments for several of the fields fitted objects include:

ynew New values of y used to reestimate the surface.

XMat A matrix of covariates for the fixed part of the model.

`XMatGrid` An array or list form of covariates to use for prediction. This must match the same dimensions from the `grid.list` / `gridList` argument.

If XMatGrid is an array, then the first two indices are the x and y locations in the grid. The third index, if present, indexes the covariates. E.g., for evaluation on a 10X15 grid and with 2 covariates, `dim( XMatGrid) == c(10,15, 2)`. If XMatGrid is a list, then the components x and y should match those of the grid list and the z component follows the shape described above for the non-list case.

`ZGrid` This argument is deprecated and a warning will be printed. Should use the new argument name XMatGrid.

`XMat` The covariates for the grid unrolled as a matrix. Columns index the variables and rows index the grid locations. E.g., for evaluation on a 10X15 grid and with 2 covariates, `dim( XMatGrid) == c(10,15, 2)` and so `dim( XMat) = c(150, 2)` and `XMat[,1] <- c( XMatGrid[,,1])`

`verbose` If TRUE, prints out some intermediate results for debugging.

Details

These functions evaluate the spatial process or thin plate spline estimates on a regular grid of points. The grid can be specified using the grid.list/gridList information or just the sizes.

For the standard Krig and mKrig versions, the steps are to create a matrix of locations that represent the grid, call the predict function for the object with these points and also adding any extra arguments passed in the ... section, and then reform the results as a surface object (as.surface). To determine what parts of the prediction grid are in the convex hull of the data the function `in.poly` is used. The argument inflation in this function is used to include a small margin around the outside of the polygon so that points on the convex hull are included. This potentially confusing modification is to prevent excluding grid points that fall exactly on the ranges of the data. Also note that, as written, there is no computational savings for evaluating only the convex subset compared to the full grid.

For the "fast" option, a stationary covariance function and resulting surface estimate is approximated by the covariance kernel restricted to the grid locations. In this way, the approximate problem becomes a 2-d convolution. The evaluation of the approximate prediction surface uses a fast Fourier transform to compute the predicted values at the grid locations.

The nearest neighbor argument `NNSize` controls the number of covariance kernels only evaluated at grid locations used to approximate a covariance function at an off-grid location. We have found good results with `NNSize=5`.

`predictSurface.fastTps` is a specific version (m=2 and k=2) of Kriging with a compact covariance kernel (Wendland) that can be much more efficient because it takes advantage of a low level FORTRAN call to evaluate the covariance function. Use `predictSurface` or `predict` for other choices of m and k.

`predictSurface.Krig` is designed to also include covariates for the fixed part in terms of grids.

`predictSurface.mKrig` Similar in function to the Krig prediction function but is more efficient using the `mKrig` fit object.

`mKrigFastPredict` Although this function might be called at the top level, it is easier to use through the wrapper `predictSurface.mKrig` and `fast=TRUE`.

NOTE: `predict.surface` has been deprecated and just prints out a warning when called.

Value

The usual list components for making image, contour, and perspective plots (x,y,z) along with labels for the `x` and `y` variables. For `predictSurface.derivative`, the component `z` is a three dimensional array with values(`nx`, `ny`, 2)

See Also

Tps, Krig, predict, grid.list, make.surface.grid, as.surface, surface, in.poly

Examples

```
data( ozone2)

  x<- ozone2$lon.lat
  y<- ozone2$y[16,]

  obj<- Tps( x,y)

  # or try the alternative model:
```

```
  #  obj<- spatialProcess(x,y)

  fit<- predictSurface( obj, nx=40, ny=40)
  imagePlot( fit)

# predicting a 2d surface holding other variables fixed.
data(BD)
fit<- Tps( BD[,1:4], BD$lnya)  # fit surface to data

# evaluate fitted surface for  first two
# variables holding other two fixed at median values

out.p<- predictSurface(fit)
surface(out.p, type="C")

#
# plot surface for second and fourth variables
# on specific grid.

glist<- list( KCL=29.77, MgCl2= seq(3,7,,25), KPO4=32.13,
                    dNTP=seq( 250,1500,,25))

out.p<- predictSurface(fit, glist)
surface(out.p, type="C")

out.p<- predictSurfaceSE(fit, glist)
surface(out.p, type="C")

## a test of the fast prediction algorithm for use with
# mKrig/spatialProcess objects.
# for explanation of the fit see the MLEs from
# fitObject<- spatialProcess( x,y)
#  fitObject$summary

## Not run:
data(NorthAmericanRainfall)

x<- cbind(NorthAmericanRainfall$longitude,
          NorthAmericanRainfall$latitude)
y<- log10(NorthAmericanRainfall$precip)

mKrigObject<- mKrig( x,y,
                    lambda=.024,
                    cov.args= list(     aRange= 5.87,
                                   Covariance="Matern",
                                   smoothness=1.0),
                                       sigma2=.157
                                 )
gridList<- list( x = seq(-134, -51, length.out = 100),
                 y = seq( 23,   57, length.out = 100))

# exact prediction
system.time(
```

```
gHat<- predictSurface( mKrigObject, gridList)
)

# approximate
system.time(
gHat1<- predictSurface( mKrigObject, gridList,
                                fast = TRUE)
)

# don't worry about the warning ...
# just indicates some observation locations are located
# in the same grid box.

# approximation error omitting the NAs from outside the convex hull
 stats( log10(abs(c(gHat$z - gHat1$z))) )

 image.plot(gHat$x, gHat$y,  (gHat$z - gHat1$z) )
 points( x,  pch=".", cex=.5)
 world( add=TRUE )


## End(Not run)
```

`print.Krig` *Print kriging fit results.*

Description

Prints the results from fitting a spatial process estimate (Krig)

Usage

```
## S3 method for class 'Krig'
print(x,digits=4,...)
```

Arguments

`x`	Object from Krig function.
`digits`	Number of significant digits in printed output. Default is 4.
`...`	Other arguments to print.

Value

Selected summary results from Krig.

See Also

print, summary.Krig, Krig

Examples

```
data(ChicagoO3)
fit<- Krig(ChicagoO3$x,ChicagoO3$y, aRange=100)
print(fit) # print the summary
fit # this will work too
```

---

pushpin *Adds a "push pin" to an existing 3-d plot*

Description

Adds to an existing 3-d perspective plot a push pin to locate a specific point.

Usage

```
pushpin( x,y,z,p.out, height=.05,col="black",text=NULL,adj=-.1,cex=1.0,...)
```

Arguments

x	x location
y	y location
z	z location
p.out	Projection information returned by persp
height	Height of pin in device coordinates (default is about 5% of the vertical distance).
col	Color of pin head.
text	Optional text to go next to pin head.
adj	Position of text relative to pin head.
cex	Character size for pin head and/or text
...	Additional graphics arguments that are passed to the text function.

Details

See help(text) for the conventions on the adj argument and other options for placing text.

Author(s)

Doug Nychka

See Also

drape.plot, persp

Examples

```
# Dr. R's favorite New  Zealand Volcano!
     data( volcano)
     M<- nrow( volcano)
     N<- ncol( volcano)
     x<- seq( 0,1,,M)
     y<- seq( 0,1,,N)

     drape.plot( x,y,volcano, col=terrain.colors(128))-> pm

     max( volcano)-> zsummit
     xsummit<- x[ row( volcano)[volcano==zsummit]]
     ysummit<- y[ col( volcano)[volcano==zsummit]]

pushpin( xsummit,ysummit,zsummit,pm, text="Summit")
```

---

qsreg *Quantile or robust spline regression*

Description

Uses a penalized likelihood approach to estimate the conditional quantile function for regression data. This method is only implemented for univariate data. For the pairs (X,Y), the conditional quantile, f(x), is P(Y<f(x)| X=x) = alpha. This estimate is useful for determining the envelope of a scatterplot or assessing departures from a constant variance with respect to the independent variable.

Usage

```
qsreg(x, y, lam = NA, maxit = 50, maxit.cv = 10, tol =
                 1e-07, offset = 0, sc = sqrt(var(y)) * 1e-05, alpha =
                 0.5, wt = rep(1, length(x)), cost = 1, nstep.cv = 80,
                 hmin = NA, hmax = NA, trmin = 2 * 1.05, trmax = 0.95
                 * length(unique(x)))
```

Arguments

x	Vector of the independent variable in y = f(x) + e
y	Vector of the dependent variable
lam	Values of the smoothing parameter. If omitted, it is found by GCV based on the quantile criterion
maxit	Maximum number of iterations used to estimate each quantile spline.
maxit.cv	Maximum number of iterations to find GCV minimum.
tol	Tolerance for convergence when computing quantile spline.
cost	Cost value used in the GCV criterion. Cost=1 is the usual GCV denominator.

offset	Constant added to the effective degrees of freedom in the GCV function.
sc	Scale factor for rounding out the absolute value function at zero to a quadratic. Default is a small scale to produce something more like quantiles. Scales on the order of the residuals will result in a robust regression fit using the Huber weight function. The default is 1e-5 of the variance of the Y's. The larger this value, the better behaved the problem is numerically and requires fewer iterations for convergence at each new value of lambda.
alpha	Quantile to be estimated. Default is to find the median.
wt	Weight vector; default is constant values. Passing nonconstant weights is a pretty strange thing to do.
nstep.cv	Number of points used in CV grid search
hmin	Minimum value of log(lambda) used for GCV grid search.
hmax	Maximum value of log(lambda) used for GCV grid search.
trmin	Minimum value of effective degrees of freedom in model used for specifying the range of lambda in the GCV grid search.
trmax	Maximum value of effective degrees of freedom in model used for specifying the range of lambda in the GCV grid search.

Details

This is an experimental function to find the smoothing parameter for a quantile or robust spline using a more appropriate criterion than mean squared error prediction. The quantile spline is found by an iterative algorithm using weighted least squares cubic splines. At convergence, the estimate will also be a weighted natural cubic spline, but the weights will depend on the estimate. Alternatively, at convergence, the estimate will be a least squares spline applied to the empirical pseudo data. The user is referred to the paper by Oh and Nychka (2002) for the details and properties of the robust cross-validation using empirical pseudo data. Of course, these weights are crafted so that the resulting spline is an estimate of the alpha quantile instead of the mean. CV as a function of lambda can be strange, so it should be plotted.

Value

trmin trmax	Define the minimum and maximum values for the CV grid search in terms of the effective number of parameters. (see hmin, hmax). Object of class qsreg with many arguments similar to a sreg object. One difference is that cv.grid has five columns, the last being the number of iterations for convergence at each value of lambda.

See Also

sreg and QTps

Examples

```
## Not run:
     # fit a quantile spline using CV
```

```
data(rat.diet)
fit50<- qsreg(rat.diet$t,rat.diet$con)
# (default is .5 so this is an estimate of the conditional median)
# control group of rats.
plot( fit50)
predict( fit50)
# predicted values at data points
xg<- seq(0,110,,50)
plot( fit50$x, fit50$y)
lines( xg, predict( fit50, xg))

# A robust fit to rat diet data
#
SC<- .5* median(abs((rat.diet$con- median(rat.diet$con))))
fit.robust<- qsreg(rat.diet$t,rat.diet$con, sc= SC)
plot( fit.robust)

# The global GCV function suggests little smoothing so
# try the local
# minima with largest lambda instead of this default value.
# (One should should consider redoing the three quantile fits in this
# example after looking at the CV functions and choosing a better value for
# lambda.)
lam<- fit50$cv.grid[,1]
tr<- fit50$cv.grid[,2]
# find lambda close to df=6
lambda.good<- max(lam[tr>=6])
fit50.subjective<-qsreg(rat.diet$t,rat.diet$con, lam= lambda.good)
fit10<-qsreg(rat.diet$t,rat.diet$con, alpha=.1, nstep.cv=200)
fit90<-qsreg(rat.diet$t,rat.diet$con, alpha=.9, nstep.cv=200)
# spline fits at 50 equally spaced points
sm<- cbind(

predict( fit10, xg),
predict( fit50.subjective, xg),predict( fit50, xg),
predict( fit90, xg))

# and now zee data ...
plot( fit50$x, fit50$y)
# and now zee quantile splines at 10
#
matlines( xg, sm, col=c( 3,3,2,3), lty=1) # the spline

## End(Not run)
```

QTps *Robust and quantile smoothing using a thin-plate spline*

Description

This function uses the standard thin plate spline function Tps and an algorithm based on pseudo data to compute robust smoothers based on the Huber weight function. By modifying the symmetry of the Huber function and changing the scale, one can also approximate a quantile smoother. This function is experimental in that it is not clear how efficient the pseudo-data algorithm is at achieving convergence to a solution.

Usage

```
QTps(x, Y, ..., f.start = NULL, psi.scale = NULL, C = 1, alpha = 0.5, Niterations = 100,
                tolerance = 0.001, verbose = FALSE)
QSreg(x, Y, lambda = NA, f.start = NULL, psi.scale = NULL,
    C = 1, alpha = 0.5, Niterations = 100, tolerance = 0.001,
    verbose = FALSE)
```

Arguments

x	Locations of observations.
Y	Observations
lambda	Value of the smoothing parameter. If NA, it is found by an approximate cross-validation criterion.
...	Any other arguments to pass to the Tps function, which are then passed to the Krig function. give.warnings =FALSE can be used to turn off pesky warnings when not important (see example below).
C	Scaling for Huber robust weighting function. (See below.) Usually it is better to leave this at 1 and just modify the scale psi.scale according to the size of the residuals.
f.start	The initial value for the estimated function. If NULL, then the constant function at the median of Y will be used. NOTE: This may not be a very good starting vector, and a more robust method would be to use a local robust smoother.
psi.scale	The scale value for the Huber function. When C=1, this is the point where the Huber weight function will change from quadratic to linear. Default is to use the scale .05*mad(Y) and C=1 . Very small scales relative to the size of the residuals will cause the estimate to approximate a quantile spline. Very large scales will yield the ordinary least squares spline.
alpha	The quantile that is estimated by the spline. Default is .5, giving a median. Equivalently, this parameter controls the slope of the linear wings in the Huber function: 2*alpha for the positive wing and 2*(1-alpha) for the negative wing.
Niterations	Maximum number of iterations of the pseudo data algorithm
tolerance	Convergence criterion based on the relative change in the predicted values of the function estimate. Specifically, if the criterion mean(abs(f.hat.new - f.hat))/mean(abs(f.hat)) is less than tolerance, the iterations are stopped.
verbose	If TRUE, intermediate results are printed out.

Details

These are experimental functions that use the pseudo-value algorithm to compute a class of robust and quantile problems. `QTps` uses the `Tps` function as its least squares base smoother while `QSreg` uses the efficient `sreg` for 1-D cubic smoothing spline models. Currently, for the 1-d spline problem, we recommend using (or at least comparing to) the old `qsreg` function. `QSreg` was created to produce a more readable version of the 1-d method that follows the thin plate spline format.

The Thin Plate Spline/ Kriging model through fields is: Y.k= f(x.k) = P(x.k) + XMat(x.k) + e.k

with the goal of estimating the smooth function: f(x)= P(x) + XMat(x)

The extension in this function is that e.k can be heavy-tailed or have outliers, and one would still like a robust estimate of f(x). In the quantile approximation (very small scale parameter), f(x) is an estimate of the alpha quantile of the conditional distribution of Y given x.

The algorithm is iterative and involves at each step tapering the residuals in a nonlinear way. Let psi.wght be this tapering function; then, given an initial estimate of f, f.hat, the new data for smoothing is

```
Y.pseudo<- f.hat + psi.scale* psi.wght( Y - f.hat, psi.scale=psi.scale, alpha=alpha)
```

A thin plate spline is now estimated for these data and a new prediction for f is found. This new vector is used to define new pseudo values. Convergence is achieved when the subsequent estimates of f.hat do not change between iterations. The advantage of this algorithm is that at every step a standard "least squares" thin plate spline is fit to the pseudo data. Because only the observation vector is changing at each iteration, some matrix decompositions need only be found once and the computations at each subsequent iteration are efficient. At convergence there is some asymptotic theory to suggest that the pseudo data can be fit using the least squares spline and the standard smoothing techniques are valid. For example, one can consider looking at the cross-validation function for the pseudo-data as a robust version to select a smoothing parameter. This approach is different from the weighted least squares algorithm used in the `qsreg` function. Also, `qsreg` is only designed to work with 1-d cubic smoothing splines.

The "sigma" function indicating the departure from a pure quadratic loss function has the definition

```
qsreg.sigma<-function(r, alpha = 0.5, C = 1)
     temp<- ifelse( r< 0, ((1 - alpha) * r^2)/C ,  (alpha * r^2)/C)
     temp<- ifelse( r >C,   2 * alpha * r - alpha * C, temp)
     temp<- ifelse( r < -C, -2 * (1 - alpha) * r - (1 - alpha) * C, temp)
     temp
```

The derivative of this function "psi" is

```
 qsreg.psi<- function(r, alpha = 0.5, C = 1)
     temp <- ifelse( r < 0, 2*(1-alpha)* r/C, 2*alpha * r/C )
     temp <- ifelse( temp > 2*alpha, 2*alpha, temp)
     temp <- ifelse( temp < -2*(1-alpha), -2*(1-alpha), temp)
     temp
```

Note that if C is very small and if alpha = .5, then psi will essentially be 1 for r > 0 and -1 for r < 0. The key feature here is that outside a certain range the residual is truncated to a constant value. This is similar to the Winsorizing operation in classical robust statistics.

Another advantage of the pseudo data algorithm is that at convergence one can just apply all the usual generic functions from Tps to the pseudo data fit. For example, predict, surface, print, etc. Some additional components are added to the Krig/Tps object, however, for information about the iterations and original data. Note that currently these are not reported in the summaries and printing of the output object.

Value

A `Krig` object with additional components:

`yraw`	Original Y values
`conv.info`	A vector giving the convergence criterion at each iteration.
`conv.flag`	If TRUE then convergence criterion was less than the tolerance value.
`psi.scale`	Scaling factor used for the psi.wght function.
`value`	Value of alpha.

Author(s)

Doug Nychka

References

Oh, Hee-Seok, Thomas CM Lee, and Douglas W. Nychka. "Fast nonparametric quantile regression with arbitrary smoothing methods." Journal of Computational and Graphical Statistics 20.2 (2011): 510-526.

See Also

qsreg

Examples

```
data(ozone2)
x<- ozone2$lon.lat
y<- ozone2$y[16,]



# Smoothing fixed at 50 df
    look1<- QTps( x,y, psi.scale= 15, df= 50)


# Least squares spline (because scale is so large)
    look2<- QTps( x,y, psi.scale= 100, df= 50)
#
    y.outlier<- y
# add in a huge outlier.
    y.outlier[58]<- 1e5
    look.outlier1<- QTps( x,y.outlier, psi.scale= 15, df= 50,
                          give.warnings= FALSE)
```

```
# least squares spline.
    look.outlier2<- QTps( x,y.outlier, psi.scale=100 , df= 50,
                          give.warnings= FALSE)
#
    set.panel(2,2)
    par( mar=c(3,3,1,1))
    surface( look1,
    xlab="",ylab="")
    title("robust spline")
    surface( look2)
    title("least squares spline")
    surface( look.outlier1,  zlim=c(0,250),
    xlab="",ylab="")
    title("robust spline w/outlier")
    points( rbind(x[58,]), pch="+", cex=2,col="white")
    surface( look.outlier2, zlim=c(0,250),
    xlab="",ylab=""
    )
    title("least squares spline w/outlier")
    points( rbind(x[58,]), pch="+",  cex=2,col="white")
    set.panel()

## Not run:
# some quantiles
look50 <- QTps( x,y, psi.scale=.5,)
look75 <- QTps( x,y,f.start= look50$fitted.values, alpha=.75)

# a simulated example that finds some different quantiles.
set.seed(123)
N<- 400
x<- matrix(runif( N), ncol=1)
true.g<- x *(1-x)*2
true.g<- true.g/ mean( abs( true.g))
y<-  true.g + .2*rnorm( N )

look0 <- QTps( x,y, psi.scale=10, df= 15)
look50 <- QTps( x,y, df=15)
look75 <- QTps( x,y,f.start= look50$fitted.values, df=15, alpha=.75)

## End(Not run)
## Not run:
# this example tests the quantile estimate by Monte Carlo
# by creating many replicate points to increase the sample size.
# Replicate points are used because the computations for the
# spline are dominated by the number of unique locations not the
# total number of points.
set.seed(123)
N<- 80
M<- 200
x<- matrix( sort(runif( N)), ncol=1)
x<- matrix( rep( x[,1],M), ncol=1)

true.g<- x *(1-x)*2
```

```
true.g<- true.g/ mean( abs( true.g))
errors<- .2*(rexp( N*M) -1)
y<- c(matrix(true.g, ncol=M, nrow=N) + .2 *  matrix( errors, ncol=M, nrow=N))

look0 <- QTps( x,y, psi.scale=10, df= 15)
look50 <- QTps( x,y, df=15)
look75 <- QTps( x,y, df=15, alpha=.75)


bplot.xy(x,y, N=25)
xg<- seq(0,1,,200)
lines( xg, predict( look0, x=xg), col="red")
lines( xg, predict( look50, x=xg), col="blue")
lines( xg, predict( look75, x=xg), col="green")

## End(Not run)
## Not run:
# A comparison with qsreg
data(rat.diet)
  qsreg.fit50<- qsreg(rat.diet$t,rat.diet$con, sc=.5)
  lam<- qsreg.fit50$cv.grid[,1]
  df<- qsreg.fit50$cv.grid[,2]
  M<- length(lam)
  CV<-rep( NA, M)
  M<- length( df)
  fhat.old<- NULL
  for ( k in M:1){
   temp.obj<- QTps(rat.diet$t,rat.diet$con, f.start=fhat.old,  psi.scale=.5, tolerance=1e-6,
     verbose=FALSE, df= df[k],
     give.warnings=FALSE)
     # avoids warnings from Krig search on lambda
     cat(k, " ")
     CV[k] <- temp.obj$Qinfo$CV.psuedo
     fhat.old<- temp.obj$fitted.values
  }
  plot( df, CV, type="l", lwd=2)
# pseudo data estimate
  points( qsreg.fit50$cv.grid[,c(5,6)], col="blue")
# alternative CV estimate via reweighted LS
  points( qsreg.fit50$cv.grid[,c(2,3)], col="red")

## End(Not run)
```

---

quantilePlot *Quantile plots of general distributions and with uncertainty bounds*

Description

This function extends the usual normal QQ plot to other distributions and also uses Monte Carlo sampling to get an idea of the finite sample uncertainty if the sample was indeed from the target distribution. Uncertainty is characterized by a simultaneous coverage envelope around the expected QQ plot.

Usage

```
quantilePlot(y, quantileFunction = "qnorm",
quantileArgs = NULL, alpha = 0.05, M = 200,
smoothQuantiles = FALSE, dfSmooth = 80,
col = "thistle1",
lineCol = "thistle3", cex = 1, ...)
```

Arguments

y	The data
quantileFunction	The target distribution in the form of a quantile function.
quantileArgs	Other arguments to be used in the quantile distribution function.
alpha	Coverage probability level.
M	Number of Monte Carlo draws to approximate the coverage envelope.
smoothQuantiles	If TRUE, smooth out the Monte Carlo results to improve the plotting.
dfSmooth	Effective degrees of freedom to use for the quantile envelope smoothing. Default is the minimum of 80 degrees of freedom or .95* sample size.
col	Fill color of the envelope.
lineCol	Boundary color of the envelope.
cex	Character size for main plot.
...	Any additional arguments to pass to the scatterplot function (plot)

Details

If p= F(y) is the distribution for a random variable Y, then the quantile function is y = Q(p). A QQ plot plots a sorted sample, YS, against YTarget with YTarget[i] being Q((i- (1/2))/n) and n the sample size. The coverage envelope is found by generating M Monte Carlo samples of size n from F. These are sorted and the mean, m[i], and standard deviation, s[i], are found for each i. The coverage envelope is the value of B so that m[i] +/- B*s[i] contains (1-alpha) percent of the samples. The smoothing option just smooths this envelope boundary slightly using a cubic smoothing spline for a cleaner figure.

Note

We are not sure why this kind of function, with the uncertainty bounds, was not part of base R. QQ plots have all kinds of hidden dependencies, and it is useful to take this into account.

Author(s)

Douglas Nychka

See Also

qqnorm

Examples

```
set.seed( 123)
n<- 100
y<- rnorm(n)
quantilePlot( y)
# note the width of envelope at the ends!

set.seed( 222)
yE<- rexp(n)
# standardize
yEStandardized<- (yE- mean( yE))/ sd( yE)
quantilePlot( yEStandardized)
# suggests yE is not normally distributed N(0,1)

quantilePlot( yE,quantileFunction = "qexp" ,M=200)
# now within bounds.

quantilePlot( y, smoothQuantiles=TRUE, dfSmooth=20)
# removes some kinks
```

---

quilt.plot *Useful plots for visualizing irregular spatial data.*

Description

Given a vector of z values associated with 2-d locations, this function produces an image-like plot where the locations are discretized to a grid and the z values are coded as a color level from a color scale.

Usage

```
quilt.plot(x, y, z, nx = 64, ny = 64, grid = NULL, add.legend =
                  TRUE, add = FALSE, nlevel = 64, col =
                  tim.colors(nlevel), nrow = NULL, ncol = NULL, FUN =
                  NULL, plot = TRUE, na.rm = FALSE, boundary.grid =
                  FALSE, ...)

bubblePlot(x, y, z, col = viridisLite::viridis(256), zlim = NULL,
                  noLegend = FALSE, horizontal = FALSE, legend.cex = 1,
```

```
legend.lab = NULL, legend.line = 2, legend.shrink =
0.8,
legend.width = 1.2, legend.mar =
ifelse(horizontal, 3.1, 5.1), axis.args = NULL,
legend.args = NULL, size = 1, add = FALSE, highlight =
FALSE, highlight.color = "grey30", bubbleType =
"circle", ...)
```

Arguments

x	A vector of the x coordinates of the locations -or- a 2-column matrix of the x-y coordinates.
y	A vector of the y coordinates -or- if the locations are passed in x, the z vector
z	Values of the variable to be plotted. For bubblePlot, if z is a character or factor, it will be assigned colors based on the order of appearance.
nlevel	Number of color levels.
nx	Number of grid boxes in x if a grid is not specified.
ny	Number of grid boxes in y.
nrow	Deprecated, same as nx.
ncol	Deprecated, same as ny.
grid	A grid in the form of a `grid list`.
add.legend	If TRUE, a legend color strip is added.
add	If TRUE, add to existing plot.
col	Color function or the color scale for the image, the default is the tim.colors function – a pleasing spectrum for quilt.plot and the modern and versatile viridis for bubblePlot
boundary.grid	If FALSE, the passed grid is considered to be the midpoints of the grid cells. If TRUE, these are assumed to define corners and the number of boxes is one less in each dimension.
plot	If FALSE, just returns the image object instead of plotting it.
FUN	The function to apply to values that are common to a grid box. The default is to find the mean. (see `as.image`).
na.rm	If FALSE, NAs are not removed from z, and so a grid box with even one of these values may be an NA. (See details below.)
size	Size of bubble dots in cex units. Can be a vector to produce different size points in the scatterplot.
zlim	Numerical range to determine the colorscale. If omitted, the range of z is used.
...	For quilt.plot, arguments to be passed to the `image.plot` function. For bubblePlot, arguments to be passed to the `plot` function if `add=FALSE` (the default) and to the `points` function if `add=TRUE`.
horizontal	If FALSE (default), the legend will be a vertical strip on the right side. If TRUE, the legend strip will be along the bottom.
bubbleType	Symbol for the bubble as a text string. Either "circle" (default) or "square".

highlight	If TRUE, will add an outline color around the "bubble" with the color highlight.color. This is useful when the bubbles are distinct.
highlight.color	Color of circle, default is a darker grey.
legend.cex	Character expansion to change the size of the legend label.
legend.lab	Label for the axis of the color legend. Default is no label, as this is usually evident from the plot title.
legend.line	Distance in units of character height (same as in mtext) of the legend label from the color bar. Make this larger if the label collides with the color axis labels.
legend.mar	Width in characters of legend margin that has the axis. Default is 5.1 for a vertical legend and 3.1 for a horizontal legend.
legend.shrink	Amount to shrink the size of the legend relative to the full height or width of the plot.
legend.width	Width in characters of the legend strip. Default is 1.2, a little bigger than the width of a character.
axis.args	A list giving additional arguments for the axis function used to create the legend axis. (See examples in image.plot.)
legend.args	The nuclear option: arguments for a complete specification of the legend label, e.g., if you need to rotate text or other details. This is in the form of a list and is just passed to the mtext function, and you will need to give both the side and line arguments for positioning. This usually will not be needed.
noLegend	If TRUE, just the bubble plot is drawn without adding a legend strip for the color bar. Useful if multiple plots are drawn with a common legend or color bar.

Details

quilt.plot This function combines the discretization to an image by the function as.image, which is then graphed by image.plot. By default, locations that fall into the same grid box will have their z values averaged. This also means that observations that are NA will result in the grid box average also being NA and can produce unexpected results because the NA patterns can dominate the figure. If you are unsure of the effect, try na.rm = TRUE for a comparison.

A similar function exists in the lattice package and produces spiffy looking plots. The advantage of this fields version is that it uses the standard R graphics functions and is written in R code. Also, the aggregation to average values for z values in the same grid box allows for different choices of grids. If two locations are very close, separating them could result in very small boxes.

Legend placement is never completely automatic. Place the legend independently for more control, perhaps using image.plot in tandem with split.screen or enlarging the plot margin. See help(image.plot) for examples of this function and these strategies.

bubblePlot Why was this function written for fields? Certainly ggplot has many options for this kind of figure. To quote Tim Hoar, a gifted data scientist and software engineer at NCAR: "because we could". It is a crisp implementation of this type of plot using lower level fields functions. The user may choose simply to use the source code as the basis for a more detailed function. However, this is also a quick plot to introduce in teaching.

This function is experimental in the sense that it explores setting out a plotting region in advance of the actual plotting using R base graphics. See the function addLegend for more details. Other

graphics approaches in R, such as ggplot, determine the plotting regions and layout based on having the entire figure specification at hand. Although this is a comprehensive solution, it also seems like overkill to just add a lone color bar to annotate a plot. Moreover, the lower-level graphics functions to add the color bar legend are already available from the image.plot function.

Author(s)

D. Nychka

See Also

as.image, discretize.image, image.plot, lattice, persp, drape.plot

Examples

```
data( ozone2)
# plot 16 day of ozone data set

quilt.plot( ozone2$lon.lat, ozone2$y[16,])
US( add=TRUE, col="grey", lwd=2)

bubblePlot( ozone2$lon.lat, ozone2$y[16,] )
US( add=TRUE, col="magenta", lwd=2)

# colors based on a factor or character vector
Label<-c("low","bckgrd","med", "high")
O3Levels<- cut( ozone2$y[16,], c( 0,40,60,80,Inf),
labels= Label
)

# check this
table( O3Levels)

# quick and simple plot using defaults
bubblePlot( ozone2$lon.lat,
O3Levels)

# a more informative legend matching categories

colTable<- viridis(4)

bubblePlot( ozone2$lon.lat, O3Levels,
axis.args= list( at=1:4, labels=Label),
col= colTable
)


### adding a common legend strip "by hand"
## to a panel of plots
## and a custom color table

coltab<- two.colors( 256, middle="grey50" )
```

```
# save some room for
# the legend on the right
par( oma=c( 0,0,0,5),
mar=c( 3,3,1,1))

set.panel(2,2)
zr<- range( ozone2$y, na.rm=TRUE)

for( k in 1:4){
quilt.plot( ozone2$lon.lat, ozone2$y[15+k,], add.legend=FALSE,
 zlim=zr, col=coltab, nx=40, ny=40)
US( add=TRUE)
}
addLegend()
imagePlot(zlim=zr,legend.only=TRUE, legend.shrink=.5,
legend.mar=4, col=coltab)

# may have to adjust number of spaces in oma to make this work.
#  adding some grid lines  and using the boundary.grid option
# note that in this case grid boxes drawn to match lon/lats
data( ozone2)
lon<- ozone2$lon.lat[,1]
lat<- ozone2$lon.lat[,2]
z<- ozone2$y[16,]

gridList<- list( x=-94 :-81,
                 y= 36:45 )

quilt.plot( lon, lat, z, grid= gridList, boundary.grid = TRUE,
             col=viridis(256), legend.mar=4 )
# add some gird lines
xline( gridList$x , col="grey", lwd=1, lty=1)
yline( gridList$y, col="grey", lwd=1, lty=2)
```

`rat.diet` *Experiment studying an appetite suppressant in rats.*

Description

The 'rat.diet' data frame has 39 rows and 3 columns. These are data from a study of an appetite suppressant given to young rats. The suppressant was removed from the treatment group at around 60 days. The responses are the median food intake, and each group had approximately 10 animals.

Usage

```
data(rat.diet)
```

Format

This data frame contains the following columns:

t Time in days

con Median food intake of the control group

trt Median food intake of the treatment group

Examples

```
data(rat.diet)
matplot( rat.diet$t,
cbind(rat.diet$trt,rat.diet$con),
type="l", lty=1 )
```

`RCMexample` *3-hour precipitation fields from a regional climate model*

Description

These are a few model output fields from the North American Regional Climate Change and Assessment Program (NARCCAP). The images are transformed surface precipitation fields simulated by the WRFP regional climate model (RCM) over North America forced by observation data. The fields are 3 hour precipitation for 8 time periods on January 1, 1979. The grid is unequally spaced in longitude and latitude due to an appropriate projection centered on the model domain. The grid points are nearly equally spaced in great circle distance due to this projection. Precipitation is in a log 10 scale where values smaller than 4.39e-5 (the .87 quantile) have been set to this value. Longitudes have been shifted from the original coordinates (0-360) to the range (-180-180) that is assumed by the R `map` function.

Usage

```
data(RCMexample)
```

Format

The format is a list of three arrays:

- x: 123X101 matrix of the longitude locations
- y: 123X101 matrix of the latitude locations
- z: 123X101X8 transformed matrix of precipitation

Spatial units are degrees with longitude being -180,180 with the prime meridian at 0. Precipitation is log 10 of cm / 3 hour period.

Details

This is primarily an example of a regular grid that is not equally spaced and is due to transforming an equally spaced grid from one map projection into longitude/latitude coordinates. This model is one small part of an extensive series of numerical experiments in the North American Regional Climate Change and Assessment Program (NARCCAP). NARCCAP has used 4 global climate models and observational data to supply the atmospheric boundary conditions for 6 different regional climate models. In the current data, the forcing is the observations derived from the NCEP reanalysis data and is for January 1, 1979. The full simulation runs for 20 years from this starting date. See the NARCCAP web page for more information about these data.

To facilitate a better representation of these fields, the raw precipitation values have been transformed to the log scale with all values below 4.39E-5 cm/3 hours set to this lower bound.

Examples

```
data(RCMexample)
# second time period

image.plot( RCMexample$x, RCMexample$y, RCMexample$z[,,2])
world( add=TRUE,  lwd=2, col="grey")
```

---

rdist *Euclidean distance matrix or vector*

Description

Given two sets of locations, `rdist` and `fields.rdist.near` compute the full Euclidean distance matrix among all pairings or a sparse version for points within a fixed threshold distance. `rdist.vec` computes a vector of pairwise distances between corresponding elements of the input locations and is used in empirical variogram calculations.

Usage

```
rdist(x1, x2 = NULL, compact = FALSE)

fields.rdist.near(x1,x2, delta, max.points= NULL, mean.neighbor = 50)

rdist.vec(x1, x2)
```

Arguments

`x1`	Matrix of first set of locations where each row gives the coordinates of a particular point.
`x2`	Matrix of second set of locations where each row gives the coordinates of a particular point. If this is not passed or given as NULL, x1 is used.

`compact`	Whether or not to return a distance matrix in compact form inheriting class "dist" (as returned by the `dist` function in base R). Only values for one triangle of the symmetric distance matrix are returned. This saves time evaluating the returned matrix and the covariance. Note that this option is ignored when `x2` is not NULL.
`delta`	Threshold distance. All pairs of points that are separated by more than delta in distance are ignored.
`max.points`	Size of the expected number of pairs less than or equal to delta. The default is set to the nrow(x1)*mean.neighbor.
`mean.neighbor`	Sets the temp space for max.points

Details

More about fields.rdist.near:

The sparse version is designed to work with the sparse covariance functions in fields and anticipates that the full matrix, D, is too large to store. The argument max.points is set as a default to nrow(x1)*100 and allocates the space to hold the sparse elements. If there are more points within delta, the function stops with an error but lists the offending rows. Just rerun the function with a larger choice for max.points.

It is possible that for certain x1 points there are no x2 points within a distance delta. This situation will cause an error if the list is converted to spam format.

Returned values

Let D be the mXn distance matrix, with m= nrow(x1) and n=nrow(x2). The elements are the Euclidean distances between all locations x1[i,] and x2[j,]. That is,

D.ij = sqrt(sum.k ((x1[i,k] - x2[j,k]) **2).

`rdist` The distance matrix D is returned.

`fields.rdist.near` The elements of D that are less than or equal to delta are returned in the form of a list.

List components:

ind Row and column indices of elements

ra Distances (D.ij)

da Dimensions of full distance matrix.

This is a simple sparse format that can be manipulated by several fields functions. E.g., ind2spam will convert this list to the format used by the spam sparse matrix package. ind2full will convert this to an ordinary matrix with zeroes.

Author(s)

Doug Nychka, John Paige

See Also

stationary.cov, Exp.cov, rdist.earth, dist, ind2spam, ind2full

Examples

```
data(ChicagoO3)
out<- rdist( ChicagoO3$x)
# out is a 20X20 matrix.

out2<- rdist( ChicagoO3$x[1:5,], ChicagoO3$x[11:20,])
#out2 is a 5X10 matrix

set.seed(123)
x1<- matrix( runif( 20*2), 20,2)
x2<-  matrix( runif( 15*2), 15,2)

out3<- fields.rdist.near( x1,x2, delta=.5)
# out3 is a sparse structure in list format

# or to "save"  work space decrease size of temp array

 out3<- fields.rdist.near( x1,x2, delta=.5,max.points=20*15)

# explicitly reforming as a full matrix
temp<- matrix( NA, nrow=out3$da[1], ncol= out3$da[2])
temp[ out3$ind] <- out3$ra

#       or just use

  temp<- spind2full( out3)
  image( temp)

# this is  identical to
 temp2<- rdist( x1,x2)
 temp2[ temp2<= .5] <- NA

#compute pairwise distance vector
x1 = 1:10
x2 = seq(from=10, to=1)
rdist.vec(x1, x2)

#calculate output matrix in compact form:
distOut = rdist(1:10, compact=TRUE)
distOut
as.vector(distOut)
```

`rdist.earth` *Great circle distance matrix or vector*

Description

Given two sets of longitude/latitude locations, `rdist.earth` computes the great circle (geographic) distance matrix among all pairings and `rdist.earth.vec` computes a vector of pairwise great circle distances between corresponding elements of the input locations using the Haversine method and is used in empirical variogram calculations.

Usage

```
rdist.earth(x1, x2, miles = TRUE, R = NULL)
RdistEarth(x1, x2=NULL, miles=TRUE, R=NULL)
rdist.earth.vec(x1, x2, miles = TRUE, R = NULL)
```

Arguments

x1	Matrix of first set of lon/lat coordinates; the first column is the longitudes and second is the latitudes.
x2	Matrix of second set of lon/lat coordinates; the first column is the longitudes and second is the latitudes. If missing or NULL, x1 is used.
miles	If TRUE, distances are in statute miles; if FALSE, distances are in kilometers.
R	Radius to use for sphere to find spherical distances. If NULL, the radius is either in miles or kilometers depending on the value of the miles argument. If R=1, then distances are of course in radians.

Details

Surprisingly, the distance matrix is computed efficiently in R by dot products of the direction cosines. This is the calculation in `rdist.earth`. Thanks to Qing Yang for pointing this out a long time ago. A more efficient version has been implemented in C with the R function `RdistEarth` by Florian Gerber, who has also experimented with parallel versions of fields functions. The main advantage of `RdistEarth` is the largely reduced memory usage. The speed seems similar to `rdist.earth`. As Florian writes:

"The current fields::rdist.earth() is surprisingly fast. In the case where only the argument 'x1' is specified, the new C implementation is faster. In the case where 'x1' and 'x2' are given, fields::rdist.earth() is a bit faster. This might be because fields::rdist.earth() does not check its input arguments and uses a less complicated (probably numerically less stable) formula."

Value

The great circle distance matrix. If nrow(x1)=m and nrow(x2)=n, then the returned matrix will be mXn.

Author(s)

Doug Nychka, John Paige, Florian Gerber

See Also

rdist, stationary.cov, fields.rdist.near

Examples

```
data(ozone2)
out<- rdist.earth ( ozone2$lon.lat)
#out is a 153X153 distance matrix
```

```
out2<- RdistEarth ( ozone2$lon.lat)
all.equal(out, out2)

upper<-  col(out)> row( out)
# histogram of all pairwise distances.
hist( out[upper])

#get pairwise distances between first 10 and second 10 lon/lat points
x1 = ozone2$lon.lat[1:10,]
x2 = ozone2$lon.lat[11:20,]
dists = rdist.earth.vec(x1, x2)
print(dists)
```

`registeringCode` *Information objects that register C and FORTRAN functions.*

Description

These are objects of class `CallRoutine` or `FortranRoutine` and also `NativeSymbolInfo`. They provide information for compiled functions called with `.Call`, or `.Fortran`. Ordinarily, one would not need to consult these, and they are used to make the search among dynamically loaded libraries (in particular the fields library) have less ambiguity and also be faster. These are created when the package/library is loaded and have their definitions from the compilation of `init.c` in the package source (src) directory.

Format

The format is a list with components:

name The (registration ?) name of the C function.

address See NativeSymbolInfo.

dll Dynamically linked library information.

numParameters Number of calling arguments in function.

Details

addToDiagC Adds diagonal elements to a matrix. See `mKrig`.

ExponentialUpperC Fills in upper triangle of a matrix with the exponential covariance function. See `ExponentialUpper`

compactToMatC Converts compact format to full matrix format. See `compactToMat`.

multebC Multiplies a vector/matrix with an exponential covariance function. See `exp.cov`

multwendlandg This has been mysteriously included but it is not a function!

mltdrb Evaluates the derivatives of thin plate spline radial basis functions. See `rad.cov`.

RdistC Euclidean distance function between sets of coordinates. See `rdist`.

distMatHaversin Used in `RdistEarth`.

distMatHaversin2 Used in RdistEarth.

See package_native_routine_registration_skeleton for the utility used to create these data objects.

It is not clear why these routines have been flagged as needing documentation while other routines have not.

References

For background on registering C, C++ and Fortran functions, see Section 5.4 of Writing R Extensions. For this package refer to the C code in src/init.c as an example.

Examples

```
print(addToDiagC)
```

---

ribbon.plot *Adds to an existing plot, a ribbon of color, based on values from a color scale, along a sequence of line segments.*

Description

Given a series of 2-d points and values at these segments, the function colors the segments according to a color scale and the segment values. This is essentially an image plot restricted to line segments.

Usage

```
ribbon.plot(x,y,z,zlim=NULL, col=tim.colors(256),
               transparent.color="white",...)
```

Arguments

x	x locations of line segments
y	y locations of line segments
z	Values associated with each segment.
zlim	Range for z values to determine color scale.
col	Color table used for strip. Default is our favorite tim.colors being a scale from a dark blue to dark red.
transparent.color	Color used for missing values. Default is that missing values make the ribbon transparent.
...	Optional graphical arguments that are passed to the segments plotting function. A favorite is lwd to make a broad ribbon.

Details

Besides possible 2-d applications, this function is useful to annotate a curve on a surface using colors. The values mapped to a color scheme could indicate a feature other than the height of the surface. For example, this function could indicate the slope of the surface.

Author(s)

Doug Nychka

See Also

image.plot, arrow.plot, add.image, colorbar.plot

Examples

```
plot( c(-1.5,1.5),c(-1.5,1.5), type="n")
temp<- list( x= seq( -1,1,,40), y= seq( -1,1,,40))
temp$z <- outer( temp$x, temp$y, "+")
contour( temp, add=TRUE)

t<- seq( 0,.5,,50)
y<- sin( 2*pi*t)
x<- cos( pi*t)
z<- x + y

ribbon.plot( x,y,z, lwd=10)

persp(  temp, phi=15, shade=.8, col="grey")-> pm
trans3d( x,y,z,pm)-> uv
ribbon.plot( uv$x, uv$y, z**2,lwd=5)
```

RMprecip *Monthly total precipitation (mm) for August 1997 in the Rocky Mountain Region and some gridded 4km elevation data sets (m).*

Description

RMprecip is a useful spatial data set of moderate size consisting of 806 locations. PRISMelevation and RMelevation are gridded elevations for the continental US and a subset of the Rocky Mountain region at 4km resolution. Note that the gridded elevations from the PRISM data product are different from the exact station elevations. (See example below.)

Usage

```
data(RMprecip)
data(RMelevation)
data(PRISMelevation)
```

Format

The data set RMprecip is a list containing the following components:

x Longitude-latitude position of monitoring stations. Row names are station id codes consistent with the US Cooperative observer network. The ranges for these coordinates are [-111, -99] for longitude and [35,45] for latitude.

elev Station elevation in meters.

y Monthly total precipitation in millimeters for August 1997

The data sets PRISMelevation and RMelevation are lists in the usual R grid format for images and contouring.

They have the following components:

x Longitude grid at approximately 4km resolution

y Latitude grid at approximately 4km resolution

z Average elevation for grid cell in meters

This data set was created the historical data taken from Observed monthly precipitation, min and max temperatures for the coterminous US 1895-1997

These elevations and the companion grid formed the basis for the 103-Year High-Resolution Precipitation Climate Data Set for the Conterminous United States (see https://prism.oregonstate.edu/documents/PRISM_downloads_FTP.pdf and also archived at the National Climate Data Center). This work is authored by Chris Daly https://prism.oregonstate.edu and his PRISM group but has some contribution from the Geophysical Statistics Project at NCAR and is an interpolation of the observational data to a 4km grid that takes into account topography such as elevation and aspect.

Details

Contact Doug Nychka for the binary file RData.USmonthlyMet.bin and information on its source.

```
#  source code to create the RMprecip data
dir <- "" # include path to data file
load(paste(dir, "RData.USmonthlyMet.bin", sep="/")
#year.id<-  1963- 1895
year.id<- 103
#pptAUG63<- USppt[ year.id,8,]
loc<- cbind(USpinfo$lon, USpinfo$lat)
xr<- c(-111, -99)
yr<- c( 35, 45)
station.subset<-  (loc[,1]>= xr[1]) & (loc[,1] <= xr[2]) & (loc[,2]>= yr[1]) & (loc[,2]<= yr[2])
ydata<-  USppt[ year.id,8,station.subset]
ydata <- ydata*10 #  cm -> mm conversion
xdata<- loc[station.subset,]
dimnames(xdata)<- list( USpinfo$station.id[station.subset], c( "lon", "lat"))
xdata<- data.frame( xdata)
good<- !is.na(ydata)
```

```
ydata<- ydata[good]
xdata<- xdata[good,]

test.for.zero.flag<- 1
test.for.zero( unlist(RMprecip$x), unlist(xdata), tag="locations")
test.for.zero( ydata, RMprecip$y, "values")
```

Examples

```
data(RMprecip)
# plot it!

# gridded
quilt.plot(RMprecip$x, RMprecip$y)
US( add=TRUE, col=2, lty=2)

# or as points
bubblePlot(RMprecip$x, RMprecip$y )
US( add=TRUE, col=2, lty=2)

# comparison of station elevations with PRISM gridded values

data(RMelevation)

interp.surface( RMelevation, RMprecip$x)-> test.elev

plot( RMprecip$elev, test.elev, xlab="Station elevation",
ylab="Interpolation from PRISM grid")
abline( 0,1,col="blue")

# some differences  with high elevations probably due to complex
# topography!

#
# view of Rockies looking from theSoutheast

save.par<- par(no.readonly=TRUE)

par( mar=c(0,0,0,0))

# fancy use of persp with shading and lighting.
persp( RMelevation, theta=75, phi= 15,
          box=FALSE, axes=FALSE, xlab="", ylab="",
         border=NA,
         shade=.95, lphi= 10, ltheta=80,
         col= "wheat4",
         scale=FALSE, expand=.00025)

# reset graphics parameters and a more conventional image plot.
par( save.par)
image.plot(RMelevation, col=topo.colors(256))
US( add=TRUE, col="grey", lwd=2)
```

```
title("PRISM elevations (m)")
```

`set.panel` *Specify a panel of plots*

Description

Divides up the graphics window into a matrix of plots.

Usage

```
set.panel(m=1, n=1, relax=FALSE)
```

Arguments

`m`	Number of rows in the panel of plots
`n`	Number of columns in the panel.
`relax`	If TRUE and the par command is already set for multiple plots, then the set.panel command is ignored. The default is relax set to FALSE.

Details

After set.panel is called, the graphics screen is reset to put plots according to an m x n table. Plotting starts in the upper left-hand corner and proceeds row by row. After m x n plots have been drawn, the next plot will erase the window and start in the 1,1 position again. This function is just a repackaging for specifying the mfrow argument to par. Setting up a panel of plots is a quick way to change the aspect ratio of the graph (ratio of height to width) or the size. For example, plotting 2 plots to a page produces a useful size graph for including in a report. You can print out the graphs at any stage without having to fill up the entire window with plots. This function, except for the "relax" option, is equivalent to the S sequence: par(mfrow=c(m,n)).

Side Effects

The function will echo your choice of m and n to the terminal.

See Also

`par`

Examples

```
set.panel(5,2) #divide screen to hold 10 plots where there are 5 rows
        #and 2 columns
plot( 1:10)
plot( 2:8)

set.panel() #reset screen to one plot per screen
```

`sim.spatialProcess` *Unconditional and conditional simulation of a spatial process*

Description

Generates exact (or approximate) random draws from the unconditional or conditional distribution of a spatial process given specific observations. Draws from the conditional distribution, known as conditional simulation in geostatistics, are a useful way to characterize the uncertainty in the predicted process from data. Note that exact simulation is limited by the number of locations, but there are approximate strategies to handle simulation for large grids of locations.

Usage

```
simSpatialData(object,  M = 1, verbose = FALSE)
sim.spatialProcess(object, xp,  M = 1, verbose = FALSE, ...)
sim.Krig(object, xp, M = 1, verbose = FALSE, ...)

simLocal.spatialProcess(
mKrigObject, gridList = NULL, M = 1, nx = 80, ny = 80,
                 delta = NULL, fast = FALSE, NNSize = 4, NNSizePredict
                 = 4, giveWarnings = FALSE, verbose = FALSE, ...)

ensembleEnvelope(object=NULL,ensemble= NULL, alpha.level =.95,
                            ...)
```

Arguments

`alpha.level`	alpha level for simultaneous coverage of confidence envelope.
`delta`	If the covariance has compact support, the simulation method can take advantage of this. This is the amount of buffer added for the simulation domain in the circulant embedding method. A minimum size would be `aRange` for the Wendland, but a multiple of this may be needed to obtain a positive definite circulant covariance function.
`ensemble`	The result of `sim.spatialProcess`
`fast`	If TRUE, will use approximate, fast spatial prediction on grids.
`giveWarnings`	If TRUE, will warn when more than one observation is in a grid box. This is instead of giving an error and stopping.
`gridList`	A grid list that is the grid for predictions. If NULL (default), a grid will be created based on the location ranges and `nx` and `ny`.
`mKrigObject`	An mKrig Object (or spatialProcess object)
`M`	Number of draws from conditional distribution.
`NNSize`	Degree of nearest neighbors to use. In the 2D case, the default `NNSize=2` uses 16 points in a 4X4 grid for prediction of the off-grid point. This is for both the rapid prediction and the local simulation. (See `predictSurface.mKrig` with `fast= TRUE`.)

NNSizePredict	Same definition of nearest neighbor size but applied to the rapid prediction approximation for the covariance. (See approximateCovariance2D.)
nx	Number of grid points in prediction locations for x coordinate.
ny	Number of grid points in prediction locations for y coordinate.
object	The spatial fit object.
in the	
xp	Same as predictionPoints above.
verbose	If TRUE, prints out intermediate information for debugging.
...	Any other arguments to be passed to the predict function. Usually this is the XMat or drop.XMat argument when there are additional covariates in the fixed part of the model. (See example below.) For ensembleEnvelope, any arguments to pass to sim.spatialProcess

Details

These functions generate samples from an unconditional or conditional multivariate (spatial) distribution, or an approximate one. The **unconditional** simulation function, simSpatialData, is a handy way to generate synthetic observations from a fitted model. Typically, one would use these for a parametric bootstrap. The functions that simulate **conditional** distributions are much more involved in their coding. They are useful for describing the uncertainty in predictions using the estimated spatial process under Gaussian assumptions. An important assumption throughout these functions is that all covariance parameters are fixed at their estimated or prescribed values from the passed object. Although these functions might be coded up easily by the users, these versions have the advantage that they take the mKrig, spatialProcess or Krig objects as a way to specify the model in an unambiguous way.

Given a spatial process h(x)= P(x) + g(x) observed at

Y.k = XMat(x.k)d + P(x.k) + g(x.k) + e.k

where P(x) is a low order, fixed polynomial and g(x) a Gaussian spatial process and XMat(x.k) is a vector of covariates that are also indexed by space (such as elevation). XMat(x.k)d is a linear combination of the covariates with the parameter vector d being a component of the fixed part of the model and estimated in the usual way by generalized least squares.

With Y= Y.1, ..., Y.N, the goal is to sample the conditional distribution of the process.

[h(x) | Y] or the full prediction XMat(x)d + h(x)

For a fixed covariance, this is just a multivariate normal sampling problem. sim.spatialProcess samples this conditional process at the points xp and is exact for fixed covariance parameters.

The outline of the conditional simulation algorithm is given below and has the advantage that it only depends on the unconditional simulation of the spatial process and being able to make a spatial prediction – the conditional mean of the process given the observations and covariance function.

Conditional Simulation Algorithm:

0) Find the spatial prediction at the unobserved locations based on the actual data. Call this h.hat(x); this is also the conditional mean.

1) Generate a realization that includes both prediction and observation locations from the unconditional spatial process and from this process simulate synthetic observations.

2) Use the spatial prediction model (using the true covariance) to estimate the spatial process at unobserved locations.

3) Find the difference between the simulated process and its prediction based on synthetic observations. Call this e(x).

4) h.hat(x) + e(x) is a draw from [h(x) | Y].

The approximations for this simulation come in at step 1). Here, the field at the observation locations is approximated using a local conditional simulation from the nearest grid points.

NOTE: A fixed part in the model is handled easily by simply making the prediction from the synthetic observations that have mean zero but include estimation of the fixed part as part of the prediction. Because the regression estimates are unbiased, this gives a valid draw from the correct multivariate distribution even though the synthetic observations do not include a fixed part.

`sim.spatialProcess` follows this algorithm exactly. For the case of an additional covariate, this, of course, needs to be included. For a model with covariates use `drop.XMat=TRUE` for the function to ignore prediction using the covariate and generate conditional samples for just the spatial process and any low order polynomial. Finally, it should be noted that this function will also work with an `mKrig` object because the essential prediction information in the mKrig and spatialProcess objects is the same. The naming is for convenience.

`sim.Krig` also follows this algorithm exactly, but for the older `Krig` object. Note that the inclusion of drop.XMat=TRUE or FALSE will determine whether the conditional simulation includes the covariates XMat or not. (See example below.)

`simLocal.spatialProcess` This function is designed for conditional simulation for a large prediction grid and for a large number of observations. The approximation will be accurate for fine grids that separate clusters of observation locations. E.g., multiple observations in a single gridbox are treated exactly. If observation locations are separated by a grid box, then, due to the screening effect, the approximation error will be negligible, especially if a nugget component (tau) is present.

A 1D version of this function will not be much more efficient than an exact computation, so it has not been included.

See the utility function `offGridWeights2D` for the function that creates weights used to generate the (approximate) conditional sample. The functions `checkPredictGrid` and `makePredictionGridList` are utilities to set up the grids when not specified.

`ensembleEnvelope` This is a simple function to find the simultaneous confidence envelope based on conditionally simulated samples. For 1-D curve fitting, use this with envelopePlot to add a spiffy confidence region that has simultaneous coverage instead of just pointwise. This function will, in general, give simultaneous upper and lower bounds for the estimated surface, however, it becomes more difficult to visualize and present beyond the 1-D case. (See example below.)

Value

`sim.Krig` and `sim.spatialProcess` a matrix with rows indexed by the locations in `xp` and columns being the `M` independent draws.

`simLocal.spatialProcess` a list with components `x`, `y` and `z` being the simulations at the prediction grid. The `x` and `y` are the typical image format specifying a regular grid. If `nx` and `ny` are the lengths of the grid values in X and Y, then `z` is an array with dimension `c(nx, ny, M)`. For the 1D case ny is set to 1. The component `hHat` is the conditional mean (as an `nx` X `ny` matrix) and the remaining arguments are various timing results for parts of the computation.

Author(s)

Doug Nychka

See Also

sim.rf, Krig, spatialProcess

Examples

```
## 10 member ensemble for the O3 data

## Not run:
data( "ozone2")
fitObject<- spatialProcess( ozone2$lon.lat, ozone2$y[16,],
                              smoothness=.5)

nx<- 35
ny<- 25

xGridList<- fields.x.to.grid( fitObject$x, nx=nx, ny=ny)

xGrid<- make.surface.grid( xGridList)

allTime0<- system.time(
 look0<- sim.spatialProcess(fitObject, xp=xGrid, M=5)
)
print( allTime0)

# for M=5 this does not make much sense ... however here are the
# Monte Carlo based prediction standard deviations.

predictSE<- apply( look0, 1, sd)

# compare to  predictSE(fitObject, xp=xGrid)

## Local simulation and same grid size for prediction
## this runs much faster compared to the exact method above
## as nx, ny are increased  e.g. nx= 128, ny=128 has dramatic difference

allTime1<- system.time(
  look<- simLocal.spatialProcess(fitObject, M=5,nx=nx, ny=ny,

                     NNSize=5)
)
print( allTime1)
print( look$timing)

allTime2<- system.time(
  look<- simLocal.spatialProcess(fitObject, M=5,nx=nx, ny=ny,
                     NNSize=5,
                     NNSizePredict=6,
                     fast=TRUE)
```

```
)
print( allTime2)
print( look$timing)

## End(Not run)

#### 1 D fit to the AudiA4 data
#### with confidence envelope

data(AudiA4)
AudiA4<- na.omit(AudiA4)

s<- cbind(AudiA4$mileage)/1000
y<- AudiA4$price/1000
obj1<- spatialProcess( s, y,  gridN=30,
smoothness= 1.5, aRange=50)
sGrid<- cbind(seq( 0, 170, length.out=100))
simOut<- sim.spatialProcess(obj1, xp=sGrid, M=500)
CIEnvelope<- ensembleEnvelope( ensemble= simOut)
plot( s,y,  type="n", xlab="mileage (1000)", ylab="price (1000)")
envelopePlot( sGrid,CIEnvelope)
lines( sGrid, predict(obj1, sGrid), col="blue")
points( s,y, pch=16 )

## Not run:
## A simple example for setting up a bootstrap
## M below should be
## set to a much larger sample size, however, ( e.g. M <- 200) for better
## statistics

data( ozone2)
obj<- spatialProcess( ozone2$lon.lat,ozone2$y[16,] )
aHat<- obj$summary["aRange"]
lambdaHat<- obj$summary["lambda"]

######### boot strap
# create M independent copies of the observation vector
# here we just grab the model information from the
# spatialProcess object above.
#
# However, one could just create the list
#  obj<- list( x= ozone2$lon.lat,
#       cov.function.name="stationary.cov",
#     summary= c( tau= 9.47, sigma2= 499.79, aRange= .700),
#    cov.args= list( Covariance="Matern", smoothness=1.0),
#     weights= rep( 1, nrow(ozone2$lon.lat) )
# )
# Here summary component  has the parameters
# tau, sigma2 and aRange
# and cov.args component has the remaining ones.

set.seed(223)
M<- 25
```

```
ySynthetic<- simSpatialData( obj, M)

bootSummary<- NULL

for(  k in 1:M){
cat( k, " ")
# here the MLEs are found using the easy top level level wrapper
# see mKrigMLEJoint for a more efficient strategy
  newSummary<- spatialProcess(obj$x,ySynthetic[,k],
                    cov.params.start= list(
                      aRange = aHat,
                     lambda = lambdaHat)
                                )$summary
  bootSummary<- rbind( bootSummary, newSummary)
  }
cat( fill= TRUE)
# the results and 95
  stats( bootSummary )

  obj$summary
  tmpBoot<- bootSummary[,c("lambda", "aRange") ]
  confidenceInterval <- apply(tmpBoot, 2,
                              quantile, probs=c(0.025,0.975) )
# compare to estimates used as the "true" parameters
  obj$summary[2:5]
  print( t(confidenceInterval) )
# compare to confidence interval using large sample theory
  print( obj$CITable)

## End(Not run)

## Not run:
# conditional simulation with covariates
# colorado climate example
# note how elevation enters either as a vector or image/grid

  data(COmonthlyMet)
  fit1E<- spatialProcess(CO.loc,CO.tmin.MAM.climate, XMat=CO.elev   )
# conditional simulation application for filling missing data
  good<- !is.na(CO.tmin.MAM.climate )
  infill<- sim.spatialProcess( fit1E, xp=CO.loc[!good,],
                XMat= CO.elev[!good], M= 10)
# conditional simulation on a grid  with covariates.
# get an elevation grid  ... NGRID<- 50 gives a nicer image but takes longer
 NGRID <- 25
 # get elevations on a grid
   COGrid<- list( x=seq( -109.75, -100.5, ,NGRID), y= seq(36, 41.75,,NGRID) )
   COGridPoints<- make.surface.grid( COGrid)
 # elevations are a bilinear interpolation from the 4km
 # Rocky Mountain elevation fields data set.
   data( RMelevation)
   # COElevTmp<- interp.surface.grid(RMelevation,COGrid )
   #  check that predicted surface is OK
```

```
   #look<- predictSurface( fit1E, gridList=COGrid, XMat=COElevTmp )
# elevations as a vector
  COElevGrid<- interp.surface( RMelevation, COGridPoints )
# NOTE call to sim.spatialProcess treats the grid points as just a matrix
# of locations the plot has to "reshape" these into a grid
# to use with image.plot
   SEout<- sim.spatialProcess( fit1E, xp=COGridPoints,  XMat= COElevGrid, M= 30)
# for just the smooth surface in lon/lat
#  SEout<- sim.spatialProcess( fit1E, xp=COGridPoints,
#                            drop.XMat=TRUE, M= 30)
# of course,  M should be larger to reduce Monte Carlo error.
   surSE<- apply( SEout, 1, sd )
   image.plot( as.surface( COGridPoints, surSE))
   points( fit1E$x, col="magenta", pch=16)


## End(Not run)

### Approximate conditional simulation
## Not run:

 # create larger lon/lat grid
 # see NNSize and NNSizePredict for additional arguments to control the
 # size of nearest neighborhoods used.

   NGRID <-200
   COGrid2<- list( x=seq( -109.75, -100.5, ,NGRID),
                   y= seq(36, 41.75,,NGRID) )
 # interpolation elevations to this grid.

   COElevGrid2<- interp.surface.grid( RMelevation, COGrid2 )

   #  check that predicted surface
   # 1. Create a sample plot
plot(1:10, 1:10, main = "Plot with Margin Text Box")


  look<- predictSurface.mKrig( fit1E, gridList=COGrid2,
   XMatGrid=COElevGrid2, fast=TRUE )

   set.seed(222)
  system.time(
    SEout0<- simLocal.spatialProcess( fit1E,COGrid2,
    XMatGrid= COElevGrid2$z,
    M= 10)
    )

# now with rapid, approximate prediction
    set.seed(222)
    system.time(
    SEout1<- simLocal.spatialProcess( fit1E,COGrid2 ,
    XMatGrid= COElevGrid2$z,
    M= 50, fast = TRUE)
```

```
    )

   look <- apply( SEout1$z,c(1,2), sd)
  imagePlot(SEout1$x, SEout1$y, look, col=viridis(256) )
  points( fit1E$x, pch=16, cex=.5, col="magenta")
  title("Monte Carlo prediction SE")


## End(Not run)

#### example using Krig object and exact conditional simulation.
## Not run:
data( ozone2)
set.seed( 399)
# fit to day 16 from Midwest ozone data set.
  out<- Krig( ozone2$lon.lat, ozone2$y[16,], Covariance="Matern",
            aRange=1.0,smoothness=1.0, na.rm=TRUE)

# NOTE aRange =1.0 is not the best choice but

# the six missing data locations
 xp<-  ozone2$lon.lat[ is.na(ozone2$y[16,]),]

# 30 draws from process at xp given the data

 sim.out<- sim.Krig( out,xp, M=30)


## End(Not run)
```

`smooth.2d` *Fast kernel-based smoother for irregular 2-d data*

Description

An approximate Nadaraya-Watson kernel smoother is obtained by first discretizing the locations to a grid and then using convolutions to find and to apply the kernel weights. The main advantage of this function is a smoother that avoids explicit looping and is fast using the FFT for the image convolution.

Usage

```
smooth.2d(Y, ind = NULL, weight.obj = NULL, setup = FALSE, grid = NULL,
    x = NULL, nrow = 64, ncol = 64, surface = TRUE, cov.function =
gauss.cov, Mwidth = NULL, Nwidth = NULL, ...)
```

Arguments

Y	A vector of data to be smoothed
ind	Row and column indices that correspond to the locations of the data on a regular grid. This is most useful when smoothing the same locations many times. (See also the x argument.)
weight.obj	An object that has the FFT of the convolution kernel and other information (i.e., the result from calling this with setup=TRUE).
setup	If TRUE, creates a list that includes the FFT of the convolution kernel. In this case, the function will return this list. Default is FALSE.
grid	A list with components x and y being equally spaced values that define the grid. Default is integers 1:nrow, 1:ncol. If x is given, the ranges will be used to define the grid.
x	Actual locations of the Y values. Not needed if ind is specified.
nrow	Number of points in the horizontal (x) axis of the grid. Not needed if grid is specified; the default is 64.
ncol	Number of points in the vertical (y) axis of the grid. Not needed if grid list is specified; the default is 64.
surface	If TRUE (the default), a surface object is returned suitable for use by image, persp or contour functions. If FALSE, then just the nrowXncol matrix of smoothed values is returned.
cov.function	S function describing the kernel function. To be consistent with the other spatial functions, this is in the form of a covariance function. The only assumption is that this be stationary. Default is the (isotropic) Gaussian.
Nwidth	The size of the padding regions of zeroes when computing the (exact) convolution of the kernel with the data. The most conservative values are 2*nrow and 2*ncol, the default. If the kernel has support of say 2L+1 grid points then the padding region needs only be of size L+1.
Mwidth	See Nwidth.
...	Parameters that are passed to the smoothing kernel. (e.g., the scale parameter aRange for the exponential or Gaussian).

Details

The irregular locations are first discretized to a regular grid (using as.image), then a 2-d FFT is used to compute a Nadaraya-Watson type kernel estimator. Here we take advantage of two features. The kernel estimator is a convolution, and by padding the regular grid by zeroes where data is not observed, one can sum the kernel over irregular sets of locations. A second convolution is used to find the normalization of the kernel weights.

The kernel function is specified by a function that should evaluate with the kernel for two matrices of locations. Assume that the kernel has the form: K(u-v) for two locations u and v. The function given as the argument to cov.function should have the call myfun(x1,x2) where x1 and x2 are matrices of 2-d locations; if nrow(x1)=m and nrow(x2)=n, then this function should return a mXn matrix where the (i,j) element is K(x1[i,]- x2[j,]). Optional arguments that are included in the ... arguments are passed to this function when it is used. The default kernel is the Gaussian and the argument aRange is the bandwidth. It is easy to write other kernels; just use Exp.cov.simple as a template.

Value

Either a matrix of smoothed values or a surface object. The surface object also has a component 'ind' that gives the subscripts of the image matrix where the data is present.

Examples

```
# Normal kernel smooth of the precip data with bandwidth of .5 ( degree)
#
data(RMprecip)
look<- smooth.2d( RMprecip$y,  x=RMprecip$x, aRange=.25)

# finer resolution used in computing the smooth
look3<-smooth.2d( RMprecip$y, x=RMprecip$x, aRange=.25, nrow=256,
ncol=256,Nwidth=32,
Mwidth=32)
# if the width arguments were omitted the padding would create a
# 512X 512 matrix with the data filled in the upper 256X256 part.
# with a bandwidth of .25 degrees the normal kernel is essentially zero
# beyond 32 grid points from its center ( about 6 standard deviations)
#
# take a look:

#set.panel(2,1)
#image( look3, zlim=c(-8,12))
#points( RMprecip$x, pch=".")
#image( look, zlim =c(-8,12))
#points( RMprecip$x, pch=".")


# bandwidth changed to .25, exponential kernel
look2<- smooth.2d( RMprecip$y, x=RMprecip$x, cov.function=Exp.cov,aRange=.25)
#
```

---

spam2lz *Conversion of formats for sparse matrices*

Description

Some supporting functions that are internal to fields top-level methods. These are used to convert between the efficient but opaque format used by spam and a more easily checked format based directly on the row and column indices of nonzero elements.

Usage

```
spind2full(obj)
```

```
spam2full(obj)

spind2spam(obj, add.zero.rows=TRUE)

spam2spind(obj)
```

Arguments

obj	Either a list with the sparse index components (spind) or an object of class spam.
add.zero.rows	If TRUE and an entire row is zero, add a hard zero value to the element in the first column for each zero row. The spam format requires at least one element in each row to have an explicit value. It is OK if this value is zero, but one must be specified.

Details

The difference in formats is best illustrated by an example:

A 4X5 sparse matrix:

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    9    0    0   33
[2,]    0    0    0   26   34
[3,]    3   11    0   27   35
[4,]    0   12   20    0   36
```

spind format is a list with components "ind", "ra" and "da". Here is how the matrix above would be encoded:

```
ind
      I
 [1,] 1 1
 [2,] 1 2
 [3,] 1 5
 [4,] 2 4
 [5,] 2 5
 [6,] 3 1
 [7,] 3 2
 [8,] 3 4
 [9,] 3 5
[10,] 4 2
[11,] 4 3
[12,] 4 5

da
[1] 4 5
```

```
ra
 [1]  1  9 33 26 34  3 11 27 35 12 20 36
```

spam format is an S4 class with slot names "entries", "colindices", "rowpointers" and "dimension".

entries

[1] 1 9 33 26 34 3 11 27 35 12 20 36

colindices

[1] 1 2 5 4 5 1 2 4 5 2 3 5

rowpointers

[1] 1 4 6 10 13

dimension

[1] 4 5

The row pointers are the positions in the array of entries where the next row starts.

NOTE: It is possible for the spind format to have a missing row of all zeroes, but this is not allowed in spam format and produces an error message.

Author(s)

Doug Nychka

See Also

as.spam

`spatialProcess` *Estimates a spatial process model.*

Description

For a given covariance function, estimates the covariance parameters by maximum likelihood and then evaluates the spatial model with these estimated parameters. The returned object can be used for spatial prediction, conditional simulation, and profiling the likelihood function. For fixed values of the covariance parameters, this process estimate is also known as Kriging.

Usage

```
spatialProcess(x, y,  weights = rep(1, nrow(x)),
                 XMat = NULL, XMatCommon=NULL,
                 Z = NULL, ZCommon=NULL,
                  mKrig.args = NULL,
                         tau = NA,
                cov.function = NULL,
                    cov.args = NULL,
                     parGrid = NULL,
```

```
                       reltol = 1e-4,
                        na.rm = TRUE,
                      verbose = FALSE,
                         REML = FALSE,
              cov.params.start = NULL,
                         gridN = 5,
                 profileLambda = FALSE,
                 profileARange = FALSE,
                 profileGridN  = 15,
                    gridARange = NULL,
                    gridLambda = NULL,
                       CILevel = .95,
                         iseed = 303,
  collapseFixedEffect = TRUE,
   simpleKriging = FALSE,
                             ...)
## S3 method for class 'spatialProcess'
summary(object,  ...)
## S3 method for class 'spatialProcess'
print(x, digits = 4, ...)
## S3 method for class 'spatialProcessSummary'
print(x, digits = 4, ...)
## S3 method for class 'spatialProcess'
plot(x, digits = 4, which = 1:4, ...)


spatialProcessSetDefaults(x, cov.function, cov.args, cov.params.start,tau,
                 parGrid,
                 mKrig.args, extraArgs = NULL, gridN = 5,
                 collapseFixedEffect = TRUE, simpleKriging = FALSE,
                 verbose = FALSE)



confidenceIntervalMLE( obj, CILevel, verbose=FALSE)

profileMLE (obj, parName, parGrid=NULL, gridN=15,
                        cov.params.start=NULL, GCV=FALSE, REML=FALSE,
                        verbose=FALSE)
```

Arguments

x	Observation locations
y	Observation values
weights	Weights for the error term (nugget) in units of reciprocal variance.

`XMat` A matrix of extra covariates for the fixed part of the spatial model, e.g., elevation for fitting climate data over space.

`XMatCommon` A matrix of extra covariates for the fixed part of the spatial model that pertain to parameters that hold across all realizations. These covariates only make sense for multiple realizations (i.e., y is a matrix with more than one column).

`Z` For backward compatibility, this argument is still included, but one should use XMat. The change was motivated by the fact that Z and z are used for other components of the model and so, as the fixed part matrix, this is confusing!

`ZCommon` For backward compatibility, this argument is still included, but one should use XMatCommon.

`CILevel` Confidence level for intervals for the estimated parameters.

`collapseFixedEffect`
If TRUE, a single vector of parameters is found for the fixed part for all realizations, e.g., if the fixed part includes a linear function of the locations, the parameters for the linear function are the same for all realizations. If FALSE, parameters are estimated separately for each realization. This option only makes sense for multiple realizations of the field, i.e., y is a matrix with more than one column.

`cov.args` A list specifying parameters and other components of the covariance function. Default is that no extra arguments are required. (But see the next item.)

`cov.function` A character string giving the name of the covariance function for the spatial component. If NULL (the default), this is filled in as `stationary.cov` and then if `cov.args` is also NULL, this is filled in as `list(Covariance = "Matern", smoothness = 1.0)` by the spatialProcessSetDefaults function.

`cov.params.start`
A list where the names are parameter names that appear in the covariance function. The values of each component are assumed to be the starting values when optimizing to find MLEs. If lambda does not appear as an additional argument when calling spatialProcess, it is added internally to this list with the starting value .5.

`digits` Number of significant digits in printed summary

`extraArgs` Extra arguments passed using the ... device in R. Typically these are extra covariance parameters specified in `spatialProcess` and then passed to `spatialProcessSetDefaults`

`GCV` A future argument not currently implemented. If TRUE, will find parameters by minimizing an approximate generalized cross-validation function.

`gridARange` A grid for profiling over the range parameter. If omitted, the default is based on a grid of profileGridN points centered at the MLE.

`gridLambda` A grid for profiling over lambda.

`gridN` Number of grid points for initial grid search to find starting values.

`na.rm` If TRUE, NAs are removed from the data.

`mKrig.args` Arguments as a list passed to the mKrig function. For example, use mKrig.args=list(m = 1) to set the fixed part of the model to just a constant function, or 0 to omit any fixed part. (The default is m=2, a linear function, which is recommended for most data analysis.) See `mKrig` for more details.

obj	A spatialProcess object returned from the spatialProcess function.
object	See obj.
parGrid	A data frame with the values of covariance parameters to use as an initial grid search for starting values.
parName	Text string that is the name of the parameter to profile.
profileARange	If TRUE, profile likelihood on aRange. Default is TRUE if aRange is omitted.
profileGridN	Number of grid points to use for profiling.
profileLambda	If TRUE, profile likelihood on lambda. This takes extra time and is not necessary, so the default is FALSE.
reltol	Relative tolerance used in optim for convergence.
REML	If TRUE, the parameters are found by restricted maximum likelihood.
simpleKriging	If TRUE, then no fixed part of the spatial model is included. When FALSE, the default is to include a linear function in the spatial coordinates. A higher degree polynomial can be chosen using the mKrig.args argument
tau	Tau is the nugget standard deviation. If NA, this is estimated by maximum likelihood. See Details below for how various combinations of passing lambda, tau, and sigma2 are handled.
verbose	If TRUE, print out intermediate information for debugging.
iseed	A seed to fix the random number stream used to compute the effective degrees of freedom using the random trace method. Setting this seed will not affect any random number generation outside this function.
...	Any other arguments that will be passed to the mKrig function and interpreted as additional arguments to the covariance function. This is a lazy way of specifying these. E.g., aRange = .1 will set the covariance argument aRange to .1.
which	The vector 1:4 or any subset of 1:4, giving the plots to draw. See the description of these plots below.

Details

This function makes many choices for the user in terms of defaults, and it is important to be aware of these. **Spatial Model** The spatial model is

Y.k= P(x.k) + XMat(x.k)%*%beta2 + g(x.k) + e.k

where ".k" means subscripted by k, Y.k is the dependent variable observed at location x.k. P is a low degree polynomial (default is a linear function in the spatial coordinates, m=2) and XMat is a matrix of covariates (optional) that enter the fixed part as a linear model. g is a mean zero, Gaussian stochastic process with a marginal variance of sigma2 and a scale (or range) parameter, aRange. By default, the covariance function is a stationary Matern with smoothness (nu) of 1.0. The measurement errors, e.k, are assumed to be uncorrelated, normally distributed with mean zero and standard deviation tau. If weights are supplied, then the variance of e is assumed to be tau^2/ weights. The polynomial, if specified, and extra covariates define the fixed part of this spatial model, and the coefficients are found by generalized least squares (GLS).

Note that tau, the nugget standard deviation, is used throughout while sigma2 is used for the process variance. To make matters more confusing, lambda = tau^2/ sigma2. If both tau and

sigma2 are specified in spatialProcess, then no estimation is done for these two parameters and lambda. If any two are given, then the third is filled in through their functional relationships.

Variance parameters Perhaps the most important aspect of this function is that the range parameter (aRange), nugget (tau**2) and process variance (sigma) parameters for the covariance are estimated by maximum likelihood, and this is the model that is then used for spatial prediction. Geostatistics usually refers to tau^2 + sigma^2 as the "sill" and often these parameters are estimated by variogram fitting rather than maximum likelihood. To be consistent with spline models and to focus on the key part of the model we reparametrize as lambda= tau**2/ sigma^2 and sigma. Thinking about h as the spatial signal and e as the noise, 1/lambda can be interpreted as the "signal-to-noise" ratio in this spatial context. (See also the comparison with fitting the geoR model in the examples section.) Other covariance parameters can also be estimated in this function. For example, the smoothness (aka nu) in the Matern family is a popular parameter to determine from data; see the example below. However, in our opinion, smoothness is often best left fixed, as it is difficult to estimate and complicates the interpretation of the range parameter.

About the estimation The optimization to obtain the MLEs is kept simple using the optim function with some good empirical starting values from coarse grid searching and using a log scale for positive parameters. The default is to find tau, sigma and aRange by maximum likelihood.

For a fixed set of covariance and variance parameters, the log Likelihood and the estimated model are computed using the lower level function mKrig, and the user is encouraged to consult this code for numerical details. When spatialProcess is given a fixed set of all the parameters, then it becomes a pass-through wrapper for mKrig and essentially returns the mKrig object. Also, at convergence for the MLEs, spatialProcess computes the final model with a call to mKrig and returns this result. See the example below for comparing the call to spatialProcess with mKrig for fixed parameters.

For an isotropic covariance function, the likelihood and the cross-validation function can be concentrated to only depend on lambda and aRange and so in reporting the optimization of these two criteria we focus on this form of the parameters. Once lambda and aRange are found, the MLE for sigma has a closed form, and, of course, tau is then determined from lambda and sigma. The estimates of the coefficients for the fixed part of the model, determined by GLS, will also be the MLEs.

Often the lambda parameter is difficult to interpret when covariates and a linear function of the coordinates are included and also when the range becomes large relative to the size of the spatial domain. For this reason, it is convenient to report the effective degrees of freedom (also referred to as trA in R code and the output summaries) associated with the predicted surface or curve. This measure has a one-to-one relationship with lambda and is easier to interpret. For example, an effective degrees of freedom that is very small suggests that the surface is well represented by a low order polynomial. Degrees of freedom close to the number of locations indicate a surface that is close to interpolating the observations and suggest a small or zero value for the nugget variance.

The default covariance model is assumed to follow a Matern with smoothness set to 1.0. This is implemented using the stationary.cov covariance that can take an argument for the form of the covariance, sill and range parameters and possibly additional parameters that might control the shape.

See the example below for how to switch to another model. (Note that the exponential is also part of the Matern family with smoothness set to .5.)

The parameter estimation is done by MLESpatialProcess and the returned list from this function is added to the Krig output object that is returned by this function. The estimate is a version of

maximum likelihood where the observations are transformed to remove the fixed linear part of the model. If the user just wants to fix the range parameter aRange, then `Krig` can be used.

NOTE: The defaults for the `optim` function used in MLESpatialProcess are:

```
list(method = "BFGS",
     control=list(fnscale = -1,
                   ndeps = rep(log(1.1),length(cov.params.start)+1),
                  reltol = reltol,
                   maxit = 20))
```

There is always a hazard in providing a simple-to-use method that makes many default choices for the spatial model. As in any analysis, be aware of these choices and try alternative models and parameter values to assess the robustness of your conclusions. Also examine the residuals to check the adequacy of the fit. See the examples below for some help in how to do this easily in fields. Also see quilt.plot to get a quick plot of a spatial field to discern obvious spatial patterns.

summary method forms a list of class `spatialProcessSummary` that has a subset of information from the output object and also creates a table of the estimates of the linear parameters in the fixed part of the model. With replicated fields, there is an option to estimate different linear parameters for each field (`collapseFixedEffect = FALSE`), and in this case, a table is not created because there is more than one estimate. See (`Omega` and `fixedEffectsCov`) in the `mKrig` object to build the standard errors.

plot method provides potentially four diagnostic plots of the fit. Use the `which` argument to pick and choose among them or use `set.panel` to see them all. The third and fourth plots, however, are only available if the profile computations have been done. If lambda is profiled (`lambdaProfile` is not `NULL`), the third plot is the profile log likelihood for lambda with the GCV function on a second vertical scale. This is based on the grid evaluations in the component `lambdaProfile\$MLEProfileLambda`. The fourth plot is a profile log likelihood trace for aRange based on `aRangeProfile\$MLEProfileLambda`.

print method prints the `spatialProcessSummary` object of the fit, adding some details and explanations.

spatialProcessSetDefaults This is a useful way to fill in defaults for the function in one place. The main choices are choosing the Matern family, smoothness and a default fixed model (aka spatial drift). The grids for profiling are also created if they have not been supplied.

Value

An object of classes `mKrig` and `SpatialProcess`. The differences from mKrig are some extra components. The more useful ones are listed below.

MLESummary A named array that has the fixed and estimated parameters along with likelihood values and some optim info.

profileSummaryLambda and profileSummaryARange The output list from mKrigMLEGrid for searching over a grid of lambda and aRange.

CITable Approximate confidence intervals based on the inverse Hessian of the log likelihood function.

MLEInfo A list that has a full documentation of the maximization including all parameters and likelihood values that were tried by the optim function.

InitialGridSearch Results from initial grid search to get good starting values for lambda and/or aRange.

Author(s)

Doug Nychka

See Also

Tps, mKrigMLEGrid, mKrigMLEJoint, plot.Krig, predict.mKrig, predictSE.mKrig

Examples

```
data( ozone2)
# x is a two column matrix where each row is a location in lon/lat
# coordinates
  x<- ozone2$lon.lat
# y is a vector of ozone measurements at day 16. Note some missing values.
  y<- ozone2$y[16,]

# lots of default choices  made here -- see gridN to increase
# the number of points in grid searches for MLEs
# without specifying  lambda or aRange both are found in a robust
# way uses grid searches
# the covariance is assumed to be  stationary Matern with a smoothness of
# 1.0 (aka Whittle).
# profiling over lambda and aRange  is not required but completes the full
# example.  Omit these for a faster computation
# e.g. obj<- spatialProcess( x, y)

  obj<- spatialProcess( x, y, profileLambda=TRUE, profileARange=TRUE)

# summary of model
  summary( obj)

# diagnostic plots
  set.panel(2,2)
  plot(obj)

# plot 1 data vs. predicted values
# plot 2 residuals vs. predicted
# plot 3 criteria to select the smoothing
#        parameter lambda = tau^2 / sigma
#        the x axis has log10 lambda
#        Note that here the GCV function is minimized
#        while the log profile likelihood  is maximzed.
# plot 4 the log profile likelihood used to
#        determine range parameter aRange.
#

set.panel()
# predictions on a grid
```

```
surface( obj, xlab="longitude", ylab="latitude")
US( add=TRUE, col="grey", lwd=2)
title("Predicted ozone (in PPB)  June 18, 1987 ")
#(see also predictSurface for more control on evaluation grid, predicting
# outside convex hull of the data. and plotting)

# prediction standard errors, note two steps now to generate
# and then plot surface
look<- predictSurfaceSE( obj)
surface( look, xlab="longitude", ylab="latitude")
points( x, col="magenta", pch=16, cex=.8)
title("prediction standard errors (PPB)")

# here  is a sanity check -- call spatialProcess with the MLEs found
# above, better get the same predictions!

objTest<- spatialProcess( x, y,
                          lambda=obj$MLESummary["lambda"],
                          aRange=obj$MLESummary["aRange"]
                          )

test.for.zero(objTest$fitted.values, obj$fitted.values,
            tag="sanity check" )


# switching to an exponential function
# See Exponential for the definition.
#
objExponential <- spatialProcess( x, y,
              Covariance="Exponential" )

objExponential$summary


############## spatialProcess vs mKrig
# another sanity check to show how spatialProcess and mKrig give the same result
# when parameters are fixed.
# Note the call to mKrig reveals all the default choices that are made by
# spatialProcess
# omit NAs so mKrig can be called directly
sOzone<- x
yOzone<- y
good<- !is.na( yOzone)
sOzone<- sOzone[good, ]
yOzone<- yOzone[good]

obj1<- spatialProcess( sOzone, yOzone,
                       tau= obj$summary["tau"],
                     sigma2=obj$summary["sigma2"],
                      aRange=obj$summary["aRange"]
                     )
```

```
objTmp<- mKrig(sOzone, yOzone,
                 m= obj$mKrig.args$m,
               tau= obj$summary["tau"],
               sigma2=obj$summary["sigma2"],
               cov.function="stationary.cov",
               cov.args=list(
                 Covariance="Matern",
                 aRange=obj$summary["aRange"],
                 nu=obj$cov.args$smoothness))
#
test.for.zero(objTmp$summary, obj1$summary)


####### simple Kriging
# this is a approximate spatial analysis to show how the
# simpleKriging=TRUE might be used -- but not recommended!
# the exact estimator can be found through backfitting ...
#
# first linear fit in spatial coordinates
yFit<- lm( y~x)
good<- !is.na(y)
# spatial fit to residuals
objTest2<- spatialProcess( x[good,], yFit$residuals, simpleKriging=TRUE)
# how close is it to the text book estimator?
stats( objTest2$residuals-obj$residuals )
plot(objTest2$residuals,obj$residuals )
abline( 0,1, col="red")

## Not run:
#################################
# working with covariates and filling in missing station data
# using an ensemble method
# see the example under  help(sim.spatialProcess) to see how to
# handle a conditional simulation on a grid of predictions with
# covariates.
data(COmonthlyMet)
  fit1E<- spatialProcess(CO.loc,CO.tmin.MAM.climate, XMat=CO.elev,
                  profileLambda=TRUE, profileARange=TRUE
          )
  set.panel( 2,2)
  plot( fit1E)

  set.panel(1,2)

# plots of the fitted surface and surface of prediction standard errors
  out.p<-predictSurface( fit1E,CO.Grid,
                              XMatGrid= CO.elevGrid, extrap=TRUE)

  imagePlot( out.p, col=larry.colors())
  US(add=TRUE, col="grey")
  contour( CO.elevGrid, add=TRUE, levels=seq(1000,3000,,5), col="black")
  title("Average Spring daily min. temp in CO")
```

```
  out.p2<-predictSurfaceSE( fit1E,CO.Grid,
                        XMatGrid= CO.elevGrid,
                        extrap=TRUE, verbose=FALSE)

  imagePlot( out.p2, col=larry.colors())
  US(add=TRUE, col="grey")
  points( fit1E$x, pch=".")
  title("Prediction SE")
  set.panel()

## End(Not run)

## Not run:
###################################
# conditional simulation
###################################
#  first a small application  for fill in  missing data
  notThere<- is.na(CO.tmin.MAM.climate )
  xp <- CO.loc[notThere,]
  XMatp <- CO.elev[notThere]
  infill<- sim.spatialProcess( fit1E, xp=xp,
                      XMat= XMatp, M= 10)
  dim( infill)
#
# interpretation is that these infilled values are all equally plausible
# given the observations and also given the estimated covariance model
#
# EXTRA CREDIT:  standardize the infilled values to have
# conditional mean and variance from the exact computations
#  e.g. predict( fit1E, xp=CO.loc[!good,],  XMat= CO.elev[!good])
#  and  predictSE(fit1E, xp=CO.loc[!good,],  XMat= CO.elev[!good])
# with these standardization one would still preserve the correlations
# among the infilled values that is also important for considering them as a
# multivariate prediction.

# conditional simulation on a grid but not using the covariate of elevation

 fit2<- spatialProcess(CO.loc,CO.tmin.MAM.climate,
                       gridARange= seq(.25, 2.0, length.out=10)
                      )
# note larger range parameter
# create 2500=50X50  grid points using a handy fields function
  gridList <- fields.x.to.grid( fit2$x, nx=50,ny=50)
  xGrid<- make.surface.grid( gridList)
  ensemble<- sim.spatialProcess( fit2, xp=xGrid, M = 6)

# this is an "n^3" computation so increasing the grid size
# can slow things down for computation
# The 6 ensemble members
   set.panel( 3,2)
   for( k in 1:6){
    imagePlot( as.surface( xGrid, ensemble[,k]))
   }
```

```
  set.panel()

## End(Not run)

## Not run:
## changing the covariance model.

data(ozone2)

  x<- ozone2$lon.lat
  y<- ozone2$y[16,]

# a comparison to using an exponential and Wendland covariance function
# and great circle distance -- just to make range easier to interpret.
  obj <- spatialProcess( x, y,
                              Distance = "rdist.earth")
 obj2<- spatialProcess( x, y,
         cov.args = list(Covariance = "Exponential"),
                              Distance = "rdist.earth" )
 obj3<- spatialProcess( x, y,
         cov.args = list(Covariance = "Wendland",
                         dimension  = 2,
                                  k = 2),
                           Distance = "rdist.earth")
# obj2 could be also be fit using the argument:
#   cov.args = list(Covariance = "Matern", smoothness=.5)
#
# Note very different range parameters - BTW these are in miles
# but similar nugget variances.
 rbind( Whittle= obj$summary,
 Exp= obj2$summary,
Wendland= obj3$summary
)

# since the exponential is Matern with smoothness == .5 the first two
# fits can be compared in terms of their likelihoods
# the ln likelihood   value is slightly higher for obj verses obj2 (-613.9 >  -614.9)
# these are the _negative_ log  likelihoods so suggests a preference for the
# smoothness = 1.0 (Whittle)  model
#
# does it really matter in terms of spatial prediction?
set.panel( 3,1)
surface( obj)
US( add=TRUE)
title("Matern sm= 1.0")
surface( obj2)
US( add=TRUE)
title("Matern sm= .5")
surface( obj3)
US( add=TRUE)
title("Wendland k =2")
# prediction standard errors
# these take a while because prediction errors are based
```

```
# directly on the Kriging weight matrix
# see mKrig for an alternative.
set.panel( 2,1)
out.p<- predictSurfaceSE( obj, nx=40,ny=40)
surface( out.p)
US( add=TRUE)
title("Matern sm= 1.0")
points( x, col="magenta")
#
out.p<- predictSurfaceSE( obj, nx=40,ny=40)
surface( out.p)
US( add=TRUE)
points( x, col="magenta")
title("Matern sm= .5")
set.panel(1,1)

## End(Not run)
```

---

splint *Cubic spline interpolation*

Description

A fast, FORTRAN-based function for cubic spline interpolation.

Usage

```
splint(x, y, xgrid, wt = NULL, derivative = 0, lam = 0, df =
                NA, lambda = NULL, nx = NULL, digits = 8)
```

Arguments

x	The x values that define the curve or a two-column matrix of x and y values.
y	The y values that are paired with the x's.
xgrid	The grid to evaluate the fitted cubic interpolating curve.
derivative	Indicates whether the function or a first or second derivative should be evaluated.
wt	Weights for different observations in the scale of reciprocal variance.
lam	Value for smoothing parameter. Default value is zero giving interpolation.
lambda	Same as lam, just to make this easier to remember.
df	Effective degrees of freedom. Default is to use lambda =0 or a df equal to the number of observations.
nx	If not NULL, this should be the number of points to evaluate on an equally spaced grid in the range of x
digits	Number of significant digits used to determine what is a replicate x value.

Details

Fits a piecewise interpolating or smoothing cubic polynomial to the x and y values. This code is designed to be fast but does not have the many options in `sreg` or other more statistical implementations. To make the solution well-posed, the second and third derivatives are set to zero at the limits of the x values. Extrapolation outside the range of the x values will be a linear function.

It is assumed that there are no repeated x values; use sreg followed by predict if you do have replicated data.

Value

A vector consisting of the spline evaluated at the grid values in `xgrid`.

References

See Additive Models by Hastie and Tibshirani.

See Also

sreg, Tps

Examples

```
x<- seq( 0, 120,,200)

data(rat.diet)
# an interpolation
splint(rat.diet$t, rat.diet$trt,x )-> y

plot( rat.diet$t, rat.diet$trt)
lines( x,y)
#( this is weird and not appropriate!)

# the following two smooths should be the same

splint( rat.diet$t, rat.diet$con,x, df= 7)-> y1

# sreg function has more flexibility than splint but will
# be slower for larger data sets.

sreg( rat.diet$t, rat.diet$con, df= 7)-> obj
predict(obj, x)-> y2

# in fact predict.sreg interpolates the predicted values using splint!

# the two predicted lines (should) coincide
lines( x,y1, col="red",lwd=2)
lines(x,y2, col="blue", lty=2,lwd=2)
```

sreg *Cubic smoothing spline regression*

Description

Fits a cubic smoothing spline to univariate data. The amount of smoothness can be specified or estimated from the data by GCV.

Usage

```
sreg(x, y, lambda = NA, df = NA, offset = 0, weights =
                 rep(1, length(x)), cost = 1, nstep.cv = 80, tol =
                 1e-05, find.diagA = TRUE, trmin = 2.01, trmax = NA,
                 lammin = NA, lammax = NA, verbose = FALSE, do.cv =
                 TRUE, method = "GCV", rmse = NA, na.rm = TRUE, digits
                 = 8)


## S3 method for class 'sreg'
predict(object, x, derivative = 0, model = 1,...)
```

Arguments

x	Vector of x values
y	Vector of y values
lambda	Single smoothing parameter or a vector of values. If omitted, the smoothing parameter is estimated by GCV. NOTE: lam here is equivalent to the value lambda*N in Tps/Krig, where N is the number of unique observations. See example below.
object	An sreg object.
derivative	Order of derivative to evaluate. Must be 0, 1, or 2.
df	Amount of smoothing in terms of effective degrees of freedom for the spline
offset	An offset added to the term cost*degrees of freedom in the denominator of the GCV function. (This would be used for adjusting the df from fitting other models such as in back-fitting additive models.)
model	Specifies which model parameters to use.
weights	A vector that is proportional to the reciprocal variances of the errors.
cost	Cost value to be used in the GCV criterion.
nstep.cv	Number of grid points of smoothing parameter for GCV grid search.
tol	Tolerance for convergence in minimizing the GCV or other criteria to estimate the smoothing parameter.

find.diagA	If TRUE, calculates the diagonal elements of the smoothing matrix. The effective number of degrees of freedom is the sum of these diagonal elements. Default is TRUE. This requires more storage if a grid of smoothing parameters is passed. (See returned values below.)
trmin	Sets the minimum of the smoothing parameter range for the GCV grid search in terms of effective degrees of freedom.
trmax	Sets the maximum of the smoothing parameter range for the GCV grid search in terms of effective degrees of freedom. If NA, the range is set to .99 of the number of unique locations.
lammin	Same function as trmin but in the lambda scale.
lammax	Same function as trmax but in the lambda scale.
verbose	Print out all sorts of debugging info. Default is FALSE, of course!
do.cv	Evaluate the spline at the GCV minimum. Default is TRUE.
method	A character string giving the method for determining the smoothing parameter. Choices are "GCV", "GCV.one", "GCV.model", "pure error", "RMSE". Default is "GCV".
rmse	Value of the root mean square error to match by varying lambda.
na.rm	If TRUE, NAs are removed from y before analysis.
digits	Number of significant digits used to determine replicate x values.
...	Other optional arguments to pass to the predict function.

Details

MODEL: The assumed model is Y.k=f(x.k) +e.k where e.k should be approximately normal and independent errors with variances tau**2/w.k

ESTIMATE: A smoothing spline is a locally weighted average of the y's based on the relative locations of the x values. Formally, the estimate is the curve that minimizes the criterion:

(1/n) sum(k=1,n) w.k(Y.k - f(X.k))**2 + lambda R(f)

where R(f) is the integral of the squared second derivative of f over the range of the X values. Because of the inclusion of the (1/n) in the sum of squares, the lambda parameter in sreg corresponds to a value of lambda*n in the Tps function and in the Krig function.

The solution to this minimization is a piecewise cubic polynomial with the join points at the unique set of X values. The polynomial segments are constructed so that the entire curve has continuous first and second derivatives, and the second and third derivatives are zero at the boundaries. The smoothing has the range [0,infinity]. Lambda equal to zero gives a cubic spline interpolation of the data. As lambda diverges to infinity (e.g., lambda =1e20) the estimate will converge to the straight line estimated by least squares.

The values of the estimated function at the data points can be expressed in the matrix form:

predicted values= A(lambda)Y

where A is an nXn symmetric matrix that does NOT depend on Y. The diagonal elements are the leverage values for the estimate and the sum of these (trace(A(lambda))) can be interpreted as the effective number of parameters that are used to define the spline function. If there are replicate points, the A matrix is the result of finding group averages and applying a weighted spline to the

means. The A matrix is also used to find "Bayesian" confidence intervals for the estimate, see the example below.

CROSS-VALIDATION: The GCV criterion with no replicate points for a fixed value of lambda is

(1/n)(Residual sum of squares)/((1-(tr(A)-offset)*cost + offset)/n)**2,

Usually offset =0 and cost =1. Variations on GCV with replicate points are described in the documentation help file for Krig. With an appropriate choice for the smoothing parameter, the estimate of tau**2 is found by (Residual sum of squares)/tr(A).

COMPUTATIONS: The computations for 1-d splines exploit the banded structure of the matrices needed to solve for the spline coefficients. Banded structure also makes it possible to get the diagonal elements of A quickly. This approach is different from the algorithms in Tps and tremendously more efficient for larger numbers of unique x values (say > 200). The advantage of Tps is getting "Bayesian" standard errors at predictions different from the observed x values. This function is similar to the S-Plus smooth.spline. The main advantages are more information and control over the choice of lambda, and also the FORTRAN source code is available (css.f).

See also the function `splint`, which is designed to be a bare-bones but fast smoothing spline.

Value

Returns a list of class sreg. Some of the returned components are

`call`	Call to the function
`yM`	Vector of dependent variables. If replicated data is given, these are the replicate group means.
`xM`	Unique x values matching the y's.
`weights`	Proportional to reciprocal variance of each data point.
`weightsM`	Proportional to reciprocal pooled variance of each replicated mean data value (xM).
`x`	Original x data.
`y`	Original y data.
`method`	Method used to find the smoothing parameter.
`pure.ss`	Pure error sum of squares from replicate groups.
`tauHat.pure.error`	Estimate of tau from replicate groups.
`tauHat.GCV`	Estimate of tau using estimated lambda from GCV minimization
`trace`	Effective degrees of freedom for the spline estimate(s)
`gcv.grid`	Values of trace, GCV, tauHat, etc. for a grid of smoothing parameters. If lambda (or df) is specified, those values are used.
`lambda.est`	Summary of various estimates of the smoothing parameter
`lambda`	If lambda is specified, the passed vector; if missing, the estimated value.
`residuals`	Residuals from spline(s). If lambda or df is specified, the residuals from these values. If lambda and df are omitted, then the residuals from the spline with the estimated lambda. This will be a matrix with as many columns as the values of lambda.

fitted.values	Matrix of fitted values. See notes on residuals.
predicted	A list with components x and y. x is the unique values of xraw in sorted order. y is a matrix of the spline estimates at these values.
eff.df	Same as trace.
diagA	Matrix containing diagonal elements of the smoothing matrix. Number of columns is the number of lambda values. WARNING: If there is replicated data, the diagonal elements are those for smoothing the group means at the unique x locations.

See Also

Krig, Tps, splint

Examples

```
# fit a GCV spline to
# control group of rats.
data(rat.diet)
fit<- sreg(rat.diet$t,rat.diet$con)
summary( fit)

set.panel(2,2)
plot(fit)                      # four diagnostic plots of  fit
set.panel()

predict( fit) # predicted values at data points

xg<- seq(0,110,,50)
sm<-predict( fit, xg) # spline fit at 50 equally spaced points
der.sm<- predict( fit, xg, deriv=1) # derivative of spline fit
set.panel( 2,1)
plot( fit$x, fit$y) # the data
lines( xg, sm) # the spline
plot( xg,der.sm, type="l") # plot of estimated derivative
set.panel() # reset panel to 1 plot


# the same fit using  the thin plate spline numerical algorithms
# sreg does not scale the obs so instruct Tps not to scale either
# this will make lambda comparable within factor of n.

   fit.tps<-Tps( rat.diet$t,rat.diet$con, scale="unscaled")
   summary( fit.tps)

# compare sreg and Tps results to show the adjustment to lambda.

   predict( fit)-> look
   predict( fit.tps, lambda=fit$lambda*fit$N)-> look2
   test.for.zero( look, look2) # silence means it checks to 1e-8

# finding approximate standard errors at observations
```

```
SE<- fit$tauHat.GCV*sqrt(fit$diagA)

# compare to predictSE( fit.tps) differences are due to
# slightly different lambda values and using tauHat.MLE instead of tauHat.GCV
#

# 95% pointwise prediction intervals
Zvalue<-  qnorm(.0975)
upper<- fit$fitted.values + Zvalue* SE
lower<- fit$fitted.values - Zvalue* SE
#
# conservative, simultaneous Bonferroni bounds
#
ZBvalue<-  qnorm(1- .025/fit$N)
upperB<- fit$fitted.values + ZBvalue* SE
lowerB<- fit$fitted.values - ZBvalue* SE
#
# take a look

plot( fit$x, fit$y, type="n")
envelopePlot(fit$x, lowerB,fit$x, upperB, col = "grey90",
             lineCol="grey")
envelopePlot(fit$x, lower,fit$x, upper, lineCol="grey")
lines( fit$predicted, col="red",lwd=2)
points( fit$x, fit$y,pch=16)

title( "95 pct pointwise  and simultaneous intervals")

# or try the more visually  honest not connecting points
plot( fit$x, fit$y, type="n")
segments(  fit$x, lowerB, fit$x, upperB, col="grey",lwd=3)
segments(  fit$x, lower, fit$x, upper, col="thistle3", lwd=6)
lines( fit$predicted, lwd=2,col="red")
points( fit$x, fit$y,pch=16)
title( "95 pct pointwise  and simultaneous intervals")

set.panel( 1,1)
```

`stats` *Calculate summary statistics*

Description

Various summary statistics are calculated for different types of data.

Usage

```
stats(x, by)
```

Arguments

x	The data structure to compute the statistics. This can either be a vector, matrix (data sets are the columns), or a list (data sets are the components).
by	If x is a vector, an optional vector (either character or numerical) specifying the categories to divide x into separate data sets.

Details

Stats breaks x up into separate data sets and then calls describe to calculate the statistics. Statistics are found by columns for matrices, by components for a list and by the relevant groups when a numeric vector and a by vector are given. The default set of statistics is the number of (non-missing) observations, mean, standard deviation, minimum, lower quartile, median, upper quartile, maximum, and number of missing observations. If any data set is nonnumeric, missing values are returned for the statistics. The by argument is a useful way to calculate statistics on parts of a data set according to different cases.

Value

A matrix where rows index the summary statistics and the columns index the separate data sets.

See Also

stats.bin, stats.bplot, describe

Examples

```
#Statistics for 8 normal random samples:
zork<- matrix( rnorm(200), ncol=8)
stats(zork)

zork<- rnorm( 200)
id<- sample( 1:8, 200, replace=TRUE)
stats( zork, by=id)
```

---

stats.bin *Bins data and finds some summary statistics.*

Description

Cuts up a numeric vector based on binning by a covariate and applies the fields stats function to each group

Usage

```
stats.bin(x, y, N = 10, breaks = NULL,prettyBins=FALSE)
```

Arguments

x	Values to use to decide bin membership
y	A vector of data
N	Number of bins. If breaks is missing, there are N bins equally spaced on the range of x.
breaks	The bin boundaries. If there are N+1 of these, there will be N bins. The bin widths can be unequal.
prettyBins	If FALSE, creates exactly N-1 bins. If TRUE, the number of bins is determined by N and the pretty function.

Value

A list with several components. stats is a matrix with columns indexing the bins and rows being summary statistics found by the stats function. These are: number of obs, mean, sd, min, quartiles, max and number of NAs. (If there are no data for a given bin, NAs are filled in.) breaks are the breaks passed to the function and centers are the bin centers.

See Also

bplot, stats

Examples

```
u<- rnorm( 2000)
v<- rnorm( 2000)
x<- u
y<- .7*u + sqrt(1-.7**2)*v

look<- stats.bin( x,y)
look$stats["Std.Dev.",]

data( ozone2)
# make up a variogram day 16 of Midwest daily ozone ...
look<- vgram( ozone2$lon.lat, c(ozone2$y[16,]), lon.lat=TRUE)

# break points
brk<- seq( 0, 250,,40)

out<-stats.bin( look$d, look$vgram, breaks=brk)
# plot bin means, and some quantiles  Q1, median, Q3
matplot( out$centers, t(out$stats[ c("mean", "median","Q1", "Q3"),]),
type="l",lty=c(1,2,2,2), col=c(3,4,3,4), ylab="ozone PPB")
```

summary.Krig *Summary for Krig or spatialProcess estimated models.*

Description

Creates a list of summary results including estimates for the nugget variance (tau) and the smoothing parameter (lambda). This list is usually printed using a "print.summary" function for nice formatting.

Usage

```
## S3 method for class 'Krig'
summary(object, digits=4,...)
```

Arguments

object	A Krig or spatialProcess object.
digits	Number of significant digits in summary.
...	Other arguments to summary

Details

This function is a method for the generic function summary for class Krig. The results are formatted and printed using print.summary.Krig.

Value

Gives a summary of the Krig object. The components include the function call, number of observations, effective degrees of freedom, residual degrees of freedom, root mean squared error, R-squared and adjusted R-squared, log10(lambda), cost, GCV minimum and a summary of the residuals.

See Also

Krig, summary, print.summary.Krig, summary.spatialProcess

Examples

```
data(ChicagoO3)
fit<- Krig(ChicagoO3$x, ChicagoO3$y, aRange=100)
summary(fit)                          # summary of fit
```

summary.ncdf *Summarizes a netCDF file handle*

Description

Provides a summary of the variable names and sizes from the handle returned from a netCDF file.

Usage

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

Arguments

object	The "handle" returned by the read.ncdf function from the ncdf package.
...	Other arguments to pass to this function. Currently, no other arguments are used.

Details

This function is out of place in fields, but was included because large geophysical data sets are often in netCDF format and the ncdf R package is also needed. To date, the summary capability in the ncdf package is limited and this function is used to supplement its use. The function is also a useful device to see how the ncdf object is structured.

Author(s)

D. Nychka

See Also

ncdf

supportsArg *Tests if a function supports a given argument*

Description

Tests if the given function supports the given argument. Commonly used in fields code for determining if a covariance function supports precomputation of the distance matrix and evaluation of the covariance matrix over only the upper triangle.

Usage

```
supportsArg(fun=stationary.cov, arg)
```

Arguments

fun	The function to test for whether it supports the argument arg as input
arg	The argument to check if fun supports using as input

Details

Currently, only stationary.cov and Exp.cov support evaluation of the covariance matrix over the upper triangle (and diagonal) only via the onlyUpper argument and distance matrix precomputation via the distMat argument.

Value

A logical indicating whether the given function supports use of the given argument

Author(s)

John Paige

See Also

stationary.cov, Exp.cov These covariance functions have the onlyUpper option allowing the user to evaluate the covariance matrix over the upper triangle and diagonal only and to pass a precomputed distance matrix

Examples

```
################
#Test covariance function to see if it supports evaluation of
#covariance matrix over upper triangle only
################

supportsArg(Rad.cov, "distMat")
supportsArg(Rad.cov, "onlyUpper")
supportsArg(stationary.cov, "distMat")
supportsArg(stationary.cov, "onlyUpper")
supportsArg(Exp.cov, "distMat")
supportsArg(Exp.cov, "onlyUpper")
```

---

surface.Krig *Plots a surface and contours*

Description

Creates different plots of the fitted surface of a Krig object. This is a quick way to look at the fitted function over reasonable default ranges.

Usage

```
## S3 method for class 'Krig'
surface(object, grid.list = NULL, extrap = FALSE,
                 graphics.reset = NULL, xlab = NULL, ylab = NULL, main
                 = NULL, zlab = NULL, zlim = NULL, levels = NULL, type
                 = "C", nx = 80, ny = 80, col = viridisLite::viridis(256), ...)
## S3 method for class 'mKrig'
surface(
object, grid.list = NULL, extrap = FALSE,
                 graphics.reset = NULL, xlab = NULL, ylab = NULL, main
                 = NULL, zlab = NULL, zlim = NULL, levels = NULL, type
                 = "C", nx = 80, ny = 80, col = viridisLite::viridis(256), ...)
```

Arguments

object	A Krig object or an mKrig object.
grid.list	A list with as many components as variables describing the surface. All components should have a single value except the two that give the grid points for evaluation. If the matrix or data frame has column names, these must appear in the grid list. If grid.list is missing and the surface has just two dimensions, the grid is based on the ranges of the observed data.
extrap	Extrapolation beyond the range of the data. If FALSE, only the convex hull of the observations is plotted. Default is FALSE.
graphics.reset	Reset to original graphics parameters after function plotting.
type	Type of plot as a character. "p" perspective plot (persp). "c" contour plot (contour). "b" a two-panel figure with perspective and contour plots. "I" image plot with legend strip (image.plot). "C" image plot with contours overlaid. Image with contour is the default.
main	Title of plot
xlab	x axis label
ylab	y axis label
zlab	z axis label if "p" or "b" type is used.
zlim	z limits passed to persp
levels	Contour levels passed to contour.
nx	Number of grid points to evaluate surface on the horizontal axis (the x-axis).
ny	Number of grid points to evaluate surface on the vertical axis (the y-axis).
col	Color scale.
...	Any other plotting options.

Details

This function is essentially a combination of predictSurface and plot.surface. It may not always give a great rendition, but is easy to use for checking the fitted surface. The default of extrap=F is designed to discourage looking at the estimated surface outside the range of the observations.

NOTE: any XMat covariates will be dropped and only the spatial part of the model will be evaluated.

See Also

Krig, predictSurface, plot.surface, image.plot

Examples

```
data(ChicagoO3)
fit<- Krig(ChicagoO3$x,ChicagoO3$y, aRange=30)  # krig fit

#Image plot of surface with nice, smooth  contours and shading

surface(fit, type="C", nx=128, ny=128)
```

The Engines: *Basic linear algebra utilities and other computations supporting the Krig function.*

Description

These are internal functions to Krig that compute the basic matrix decompositions or solve the linear systems needed to evaluate the Krig/Tps estimate. Others listed below do some simple housekeeping and formatting. Typically they are called from within Krig but can also be used directly if passed a Krig object list.

Usage

```
Krig.engine.default(out, verbose = FALSE)
Krig.engine.fixed( out, verbose=FALSE, lambda=NA)

Krig.coef(out, lambda = out$lambda, y = NULL, yM = NULL, verbose = FALSE)
Krig.make.u(out, y = NULL, yM = NULL, verbose = FALSE)
Krig.check.xY(x, Y,XMat, weights, na.rm, verbose = FALSE)
Krig.transform.xY(obj, knots, verbose = FALSE)

Krig.make.W( out, verbose=FALSE)
Krig.make.Wi ( out, verbose=FALSE)
```

Arguments

out	A complete or partial Krig object. If partial, it must have all the information accumulated to this calling point within the Krig function.
obj	Same as out.
verbose	If TRUE, prints out intermediate results for debugging.
lambda	Value of smoothing parameter "hard wired" into decompositions. Default is NA, i.e., use the value in out\$lambda.

y	New y vector for recomputing coefficients. OR for %d*% a vector or matrix.
yM	New y vector for recomputing coefficients but the values have already been collapsed into replicate group means.
Y	raw data Y vector
x	raw x matrix of spatial locations OR In the case of %d*%, y is either a matrix or a vector. As a vector, y is interpreted to be the elements of a diagonal matrix.
weights	Raw weights vector passed to Krig
XMat	Raw vector or matrix of additional covariates.
na.rm	NA action logical values passed to Krig
knots	Raw knots matrix passed to Krig

Details

ENGINES:

The engines are the code modules that handle the basic linear algebra needed to compute the estimated curve or surface coefficients. All the engines work on the data that has been reduced to unique locations and possibly replicate group means with the weights adjusted accordingly. All information needed for the decomposition is contained in the Krig object passed to these functions.

Krig.engine.default finds the decompositions for a Universal Kriging estimator by simultaneously diagonalizing the linear system for the coefficients of the estimator. The main advantage of this form is that it is fairly stable numerically, even with ill-conditioned covariance matrices with lambda > 0. (i.e., provided there is a "nugget" or measurement error). Also, the eigendecomposition allows for rapid evaluation of the likelihood, GCV and coefficients for new data vectors under different values of the smoothing parameter, lambda.

Krig.engine.knots This code has been omitted from versions >= 12.0. See 11.6 to recover this functionality.

Finds the decompositions in the case that the covariance is evaluated at arbitrary locations possibly different from the data locations (called knots). The intent of these decompositions is to facilitate the evaluation at different values for lambda.

Krig.engine.fixed are specific decompositions based on the Cholesky factorization assuming that the smoothing parameter is fixed. This is the only case that works in the sparse matrix. Both knots and the full set of locations can be handled by this case. The difference from the "knots" engine above is that only a single value of lambda is considered in the fixed engine.

OTHER FUNCTIONS:

Krig.coef Computes the "c" and "d" coefficients to represent the estimated curve. These coefficients are used by the predict functions for evaluations. Krig.coef can be used outside of the call to Krig to recompute the fit with different Y values and possibly with different lambda values. If new y values are not passed to this function, then the yM vector in the Krig object is used. The internal function Krig.ynew sorts out the logic of what to do and use based on the passed arguments.

Krig.make.u Computes the "u" vector, a transformation of the collapsed observations that allows for rapid evaluation of the GCV function and prediction. This only makes sense when the decomposition is WBW or DR, i.e., an eigendecomposition. If the decomposition is Cholesky based, then this function returns NA for the u component in the list.

Krig.check.xY Checks for and removes missing values (NAs).

`Krig.cor.Y` This code has been omitted from versions >= 12.0. See 11.6 to recover this functionality.

`Krig.transform.xY` Finds all replicates and collapses to unique locations and mean response and pooled variances and weights. These are the xM, yM and weightsM used in the engines. Also scales the x locations and the knots according to the transformation.

`Krig.make.W` and `Krig.make.Wi` These functions create an off-diagonal weight matrix and its symmetric square root or the inverse of the weight matrix based on the information passed to Krig. If `out$nondiag` is TRUE, W is constructed based on a call to the passed function wght.function along with additional arguments. If this flag is FALSE, then W is just `diag(out$weightsM)` and the square root and inverse are computed directly.

`%d*%` Is a simple way to implement efficient diagonal multiplications. x%d*%y is interpreted to mean diag(x)%*% y if x is a vector. If x is a matrix, then this becomes the same as the usual matrix multiplication.

Returned Values

ENGINES:

The returned value is a list with the matrix decompositions and other information. These are incorporated into the complete Krig object.

Common to all engines:

decomp Type of decomposition

nt dimension of T matrix

np number of knots

`Krig.engine.default`:

u Transformed data using eigenvectors.

D Eigenvalues

G Reduced and weighted matrix of the eigenvectors

qr.T QR decomposition of fixed regression matrix

V The eigenvectors

`Krig.engine.fixed`:

d estimated coefficients for the fixed part of model

c estimated coefficients for the basis functions derived from the covariance function.

Using all data locations

qr.VT QR decomposition of the inverse Cholesky factor times the T matrix.

MC Cholesky factor

Using knot locations

qr.Treg QR decomposition of regression matrix modified by the estimate of the nonparametric (or spatial) component.

lambda.fixed Value of lambda used in the decompositions

OTHER FUNCTIONS:

`Krig.coef`

yM Y values as replicate group means

tauHat.rep Sample standard deviation of replicates

tauHat.pure.error Same as tauHat.rep

pure.ss Pure error sums of squares based on replicates

c The "c" basis coefficients associated with the covariance or radial basis functions.

d The "d" regression type coefficients that are from the fixed part of the model or the linear null space.

u When the default decomposition is used, the data vector transformed by the orthogonal matrices. This facilitates evaluating the GCV function at different values of the smoothing parameter.

`Krig.make.W`

W The weight matrix

W2 Symmetric square root of weight matrix

`Krig.make.Wi`

Wi The inverse weight matrix

W2i Symmetric square root of inverse weight matrix

Author(s)

Doug Nychka

See Also

`Krig`, `Tps`

Examples

```
data( ChicagoO3)
out <- Krig( ChicagoO3$x, ChicagoO3$y, aRange=100)

stuff <- Krig.engine.default( out)

# compare "stuff" to components in out$matrices

look1<- Krig.coef( out)
look1$c
# compare to out$c

look2<- Krig.coef( out, yM = ChicagoO3$y)
look2$c
# better be the same even though we pass as new data!
```

`tim.colors` *Some useful color tables for images and tools to handle them.*

Description

Several color scales useful for image plots: a pleasing rainbow-style color table patterned after that used in Matlab by Tim Hoar and also some simple color interpolation schemes between two or more colors. There is also a function that converts between colors and a real-valued vector.

Usage

```
tim.colors(n = 64, alpha=1.0)

larry.colors()

snow.colors(n=256, alpha=1)

data(Mines.colors)

two.colors(n=256, start="darkgreen", end="red", middle="white",
alpha=1.0)

designer.colors( n=256, col= c("darkgreen", "white", "darkred"), x=
                              seq(0,1,, length(col)) ,alpha=1.0)

color.scale(z, col = tim.colors, NC = 256, zlim = NULL,
                 transparent.color = "white", eps = 1e-08)

fieldsPlotColors( col,...)
```

Arguments

`alpha`	The transparency of the color – 1.0 is opaque and 0 is transparent. This is useful for overlays of color and still being able to view the graphics that are covered.
`n`	Number of color levels. The setting `n=64` is the original definition.
`start`	Starting color for lowest values in color scale
`end`	Ending color.
`middle`	Color scale passes through this color at halfway
`col`	A list of colors (names or hex values) to interpolate. But for the `color.scale` function this can also be a function that returns `NC` colors when called with just the `NC` argument.
`x`	Positions of colors on a [0,1] scale. Default is to assume that the x values are equally spaced from 0 to 1.
`z`	Real vector to encode in a color table.

zlim	Range to use for color scale. Default is the range(z) inflated by 1- eps and 1+eps.
transparent.color	Color value to use for NAs or values outside zlim
eps	A small inflation of the range to avoid the boundary values of z being coded as NAs.
NC	The number of colors to return from calling the function passed in the col argument. This is only used if col is a function.
...	Additional plotting arguments that are passed to image.plot.

Details

The color in R can be represented as three vectors in RGB coordinates and these coordinates are interpolated separately using a cubic spline to give color values that are intermediate to the specified colors.

Ask Tim Hoar about tim.colors! As a Matlab black belt, this was his favorite scale at NCAR in that system. The viridis package has a similar version (turbo) that is a greyscale-friendly alternative.

two.colors is really about three different colors. For other colors, try fields.color.picker to view possible choices. start="darkgreen", end="azure4" are the options used to get a nice color scale for rendering aerial photos of ski trails. (See https://github.com/dnychka/MJProject.)

larry.colors is a 13-color palette used by Larry McDaniel (retired software engineer from NCAR) and is particularly useful for visualizing fields of climate variables.

snow.colors is the scale used by Will Klieber's team for visualizing snow cover from remotely sensed data products. See the commented code for the script as to how this was formed from an original raw 256-level scale. Note that the first color in this table is grey and is designed to represent the minimum value of the range (e.g., 0). If the image is in percent snow cover, then zlim=c(0,100) would make sense as a range to fit grey pixels to zero and white to 100 percent.

See data(Mines.colors) to define the data sets that are web colors for Colorado School of Mines. Note that since these are defined as text strings, do _not_ quote them when using them in the R graphics functions – they are already character data. E.g., plot(x,y, col=blasterBlue)

designer.colors is the master function for the other scales. It can be useful if one wants to customize the color table to match quantiles of a distribution. E.g., if the median of the data is at .3 with respect to the range, then set x equal to c(0,.3,1) and specify two colors to provide a transition that matches the median value. In fields language, this function interpolates between a set of colors at locations x. While you can be creative about these colors, just using another color scale as the basis is easy. For example,

```
designer.colors( 256, rainbow(4), x= c( 0,.2,.8,1.0))
```

leaves the choice of the colors to Dr. R after a thunderstorm. See also colorBrewer to choose sequences of colors that form a good palette.

color.scale assigns colors to a numerical vector in the same way as the image function. This is useful to keep the assignment of colors consistent across several vectors by specifying a common zlim range.

plotColorScale is a simple function to plot a vector of colors to examine their values.

Value

A vector giving the colors in a hexadecimal format; two extra hex digits are added for the alpha channel.

Or, in the case of `Mines.colors` a set of character strings loaded as separate objects with the color name. Use `ls()` to list them out.

See Also

topo.colors, terrain.colors, image.plot, imagePlot, quilt.plot, grey.scale, fields.color.picker

Examples

```
tim.colors(10)
# returns an array of 10 character strings encoding colors in hex format

# e.g. (red, green,  blue) values of   (16,255, 239)
#   translates to "#10FFEF"
# rgb( 16/255, 255/255, 239/255, alpha=.5)
#   gives   "#10FFEF80"  note extra "alpha channel"

# view some color table choices
set.panel( 4,1)
fieldsPlotColors( tim.colors())
title("tim.colors")
fieldsPlotColors( larry.colors())
title("larry.colors")
fieldsPlotColors( two.colors())
title("two.colors")
fieldsPlotColors( snow.colors())
title("snow.colors")

# a bubble plot with some transparency for overlapping dots
set.seed(123)
loc<- matrix( rnorm( 200), 100,2)
Z<- loc[,1] + loc[,2]
colorMap<- color.scale( Z, col=tim.colors(10, alpha=.8))
par( mar=c(5,5,5,5)) # extra room on right for color bar
plot( loc, col=colorMap, pch=16, cex=2)
#  add a color scale
 image.plot(legend.only=TRUE, zlim=range( Z), col=tim.colors(10))

# using transparency without alpha, the image
# plot would cover points

obj<- list( x= 1:8, y=1:10, z= outer( 1:8, 1:10, "+") )
plot( 1:10,1:10)

image(obj, col=two.colors(alpha=.5), add=TRUE)

coltab<- designer.colors(col=c("blue", "grey", "green"),
                  x= c( 0,.3,1) )
```

```
image( obj, col= coltab )

# peg colors at some desired quantiles  of data.
# NOTE need 0 and 1 for the color scale to make sense
x<- quantile( c(obj$z), c(0,.25,.5,.75,1.0) )
# scale these to [0,1]
zr<- range( c(obj$z))
x<- (x-zr[1])/ (zr[2] - zr[1])

coltab<- designer.colors(256,rainbow(5), x)
image( obj$z, col= coltab )
# see image.plot for adding all kinds of legends


data(Mines.colors)
plot( 1:10, 101:110, col=blasterBlue, pch=19)

set.panel()
```

Tps *Thin plate spline regression*

Description

Fits a thin plate spline surface to irregularly spaced data. The smoothing parameter is chosen as a default by generalized cross-validation. The assumed model is additive Y = f(X) +e where f(X) is a d-dimensional surface .

Usage

```
Tps(x, Y, m = NULL, p = NULL, scale.type = "range", lon.lat = FALSE,
         miles = TRUE, method = "GCV", GCV = TRUE, ...)

fastTps(x, Y, m = NULL, p = NULL, aRange, lon.lat = FALSE,
                 find.trA = FALSE, REML = FALSE,theta=NULL,  ...)
```

Arguments

x	Matrix of independent variables. Each row is a location or a set of independent covariates.
Y	Vector of dependent variables.
m	A polynomial function of degree (m-1) will be included in the model as the drift (or spatial trend) component. Default is the value such that 2m-d is greater than zero, where d is the dimension of x.

p — Polynomial power for Wendland radial basis functions. Default is 2m-d where d is the dimension of x.

scale.type — The independent variables and knots are scaled to the specified scale.type. By default, the scale type is "range", whereby the locations are transformed to the interval (0,1) by forming (x-min(x))/range(x) for each x. Scale type of "user" allows specification of an x.center and x.scale by the user. The default for "user" is mean 0 and standard deviation 1. Scale type of "unscaled" does not scale the data.

aRange — The tapering range that is passed to the Wendland compactly supported covariance. The covariance (i.e., the radial basis function) is zero beyond range aRange. The larger aRange, the closer this model will approximate the standard thin plate spline.

lon.lat — If TRUE, locations are interpreted as longitude and latitude, and great circle distance is used to find distances among locations. The aRange scale parameter for `fast.Tps` (setting the compact support of the Wendland function) in this case is in units of miles (see example and caution below).

method — Determines what "smoothing" parameter should be used. The default is to estimate standard GCV. Other choices are: GCV.model, GCV.one, RMSE, pure error and REML. The differences are explained in the Krig help file.

GCV — If TRUE, the decompositions are done to efficiently evaluate the estimate, GCV function and likelihood at multiple values of lambda.

miles — If TRUE, great circle distances are in miles; if FALSE, distances are in kilometers

find.trA — If TRUE, will estimate the effective degrees of freedom using a simple Monte Carlo method (random trace). This will add to the computational burden by approximately `NtrA` solutions of the linear system, but the Cholesky decomposition is reused.

REML — If TRUE, find the MLE for lambda using restricted maximum likelihood instead of the full version.

theta — Same as aRange.

... — For `Tps`, any argument that is valid for the `Krig` function. Some of the main ones are listed below.

For `fastTps`, any argument that is suitable for the `mKrig` function; see help on mKrig for these choices. The most common would be `lambda` fixing the value of this parameter (tau^2/sigma^2), `XMat` linear covariates or `mKrig.args= list( m=1)` setting the regression model to be just a constant function.

Arguments for Tps:

lambda Smoothing parameter that is the ratio of the error variance (tau**2) to the scale parameter of the covariance function. If omitted, this is estimated by GCV.

XMat Linear covariates to be included in the fixed part of the model that are distinct from the default low order polynomial in `x`

df The effective number of parameters for the fitted surface. Conversely, N - df, where N is the total number of observations, is the degrees of freedom associated with the residuals. This is an alternative to specifying lambda and much more interpretable.

cost Cost value used in GCV criterion. Corresponds to a penalty for increased number of parameters. The default is 1.0 and corresponds to the usual GCV.

weights Weights are proportional to the reciprocal variance of the measurement error. The default is no weighting, i.e., a vector of unit weights.

nstep.cv Number of grid points for minimum GCV search.

x.center Centering values are subtracted from each column of the x matrix. Must have scale.type="user".

x.scale Scale values that are divided into each column after centering. Must have scale.type="user".

sigma Scale factor for covariance.

tau2 Variance of errors or, if weights are not equal to 1, the variance is tau**2/weight.

verbose If TRUE, will print out all kinds of intermediate stuff.

mean.obj Object to predict the mean of the spatial process.

sd.obj Object to predict the marginal standard deviation of the spatial process.

null.function An R function that creates the matrices for the null space model. The default is fields.mkpoly, an R function that creates a polynomial regression matrix with all terms up to degree m-1. (See Details)

offset The offset to be used in the GCV criterion. Default is 0. This would be used when Krig/Tps is part of a backfitting algorithm and the offset has to be included to reflect other model degrees of freedom.

Details

Overview

This is the classic nonparametric curve/surface estimate pioneered in statistics by Grace Wahba. The computational algorithm is based around a QR decomposition followed by an eigendecomposition on a reduced matrix derived from the spline radial basis functions. This ensures a stable computation – basically bombproof – but is not the fastest. See the function mKrig and examples for fitting a thin plate spline using spatialProcess that uses a different linear algebra approach. This is implemented in the "fast" version, albeit an approximate version, to exploit sparse matrices.

This function also works for just a single dimension and reproduces the well-known cubic smoothing spline for `m ==2`. Finally, we note that a thin plate spline is a limiting case of a Gaussian process estimate (Kriging) as the range parameter in the Matern family increases to infinity.

A "fast" version of this function uses a compactly supported Wendland covariance and sparse linear algebra for handling larger data sets. Although a good approximation to Tps for sufficiently large aRange, its actual form is very different from the textbook thin-plate definition. The user will see that `fastTps` is largely a wrapper for a call to `spatialProcess` with the Wendland covariance function.

Background on the thin plate spline function, Tps A thin plate spline is the result of minimizing the residual sum of squares subject to a constraint that the function have a certain level of smoothness (or roughness penalty). Roughness is quantified by the integral of squared m-th order derivatives. For one dimension and m=2, the roughness penalty is the integrated square of the second derivative of the function. For two dimensions, the roughness penalty is the integral of

(Dxx(f))**2 + 2(Dxy(f))**2 + (Dyy(f))**2

(where Duv denotes the second partial derivative with respect to u and v). Besides controlling the order of the derivatives, the value of m also determines the base polynomial that is fit to the data. The degree of this polynomial is (m-1).

The smoothing parameter controls the amount that the data is smoothed. In the usual form, this is denoted by lambda, the Lagrange multiplier of the minimization problem. Although this is an awkward scale, lambda = 0 corresponds to no smoothness constraints and the data is interpolated. lambda=infinity corresponds to just fitting the polynomial base model by ordinary least squares.

This estimator is implemented by passing the right generalized covariance function based on radial basis functions to the more general function Krig. One advantage of this implementation is that once a Tps/Krig object is created, the estimator can be found rapidly for other data and smoothing parameters, provided the locations remain unchanged. This makes simulation within R efficient (see example below). Tps does not currently support the knots argument where one can use a reduced set of basis functions. This is mainly to simplify the code, and a good alternative using knots would be to use a valid covariance from the Matern family and a large range parameter.

Using a great circle distance function The option to use great circle distance to define the radial basis functions (`lon.lat=TRUE`) is very useful for small geographic domains where the spherical geometry is well approximated by a plane. However, for large domains, the spherical distortion can be large enough that the radial basis functions no longer define a positive definite system and Tps will report a numerical error. An alternative is to switch to a three dimensional thin plate spline with the locations being the direction cosines. This will give approximate great circle distances for locations that are close, and the numerical methods will always have positive definite matrices. There are other radial basis functions that are specific to a spherical geometry, but these are not implemented in fields.

Here is an example using this idea for `RMprecip` and also some examples of building grids and evaluating the Tps results on them:

```
# a useful function:
  dircos<- function(x1){
              coslat1 <- cos((x1[, 2] * pi)/180)
              sinlat1 <- sin((x1[, 2] * pi)/180)
              coslon1 <- cos((x1[, 1] * pi)/180)
              sinlon1 <- sin((x1[, 1] * pi)/180)
              cbind(coslon1*coslat1, sinlon1*coslat1, sinlat1)}
# fit in 3-d to direction cosines
  data(RMpecip)
  out<- Tps(dircos(RMprecip$x),RMprecip$y)
  xg<-make.surface.grid(fields.x.to.grid(RMprecip$x))
  fhat<- predict( out, dircos(xg))
# coerce to image format from prediction vector and grid points.
  out.p<- as.surface( xg, fhat)
  surface( out.p)
# compare to the automatic
  out0<- Tps(RMprecip$x,RMprecip$y, lon.lat=TRUE)
  surface(out0)
```

The function `fastTps` is really a convenient wrapper function that calls `spatialProcess` with a suitable Wendland covariance function. This means one can use all the additional functions for

prediction and simulation built for the spatialProcess and mKrig objects. Some care needs to be exercised in specifying the support aRange – a Goldilocks problem – where aRange is large enough so that every location has a reasonable number (say 10 or more) of neighboring locations that have non-zero covariances, but also the number of neighbors is not so large that the sparsity of the covariance matrix is compromised. To figure out the neighborhood pattern of the spatial locations, one can use the function nearest.dist, the sparse matrix format, and the table function.

```
set.seed(222)
s<- cbind( runif(1e4),runif(1e4))
look<- nearest.dist(s,s, delta = .03)
look<- spam2spind(look)

stats( table( look$ind[,1]))
```

Here one has a summary of the number of nearest neighbors within a distance of .03 for these (randomly generated) locations. I see a minimum of 7 for at least one location, so aRange should be larger than .03. Trial and error with different deltas can lead to a better choice.

Note that, unlike Tps, the locations are not scaled to unit range, and this can cause havoc in smoothing problems with variables in very different units. So rescaling the locations x<- scale(x) is a good idea for putting the variables on a common scale for smoothing. A conservative rule of thumb is to make aRange large enough so that about 50 nearest neighbors are within this distance for every observation location.

This function does have the potential to approximate estimates of Tps for very large spatial data sets. See wendland.cov and help on the SPAM package, and the friendly spind format for more background. Also, the function predictSurface.fastTps has been made more efficient for the case of k=2 and m=2 – a common choice for parameters.

Value

A list of class Krig. This includes the fitted values, the predicted surface evaluated at the observation locations, and the residuals. The results of the grid search minimizing the generalized cross validation function are returned in gcv.grid. Note that the GCV/REML optimization is done even if lambda or df is given. Please see the documentation on Krig for details of the returned arguments.

References

See "Nonparametric Regression and Generalized Linear Models"

by Green and Silverman. See "Additive Models" by Hastie and Tibshirani.

See Wahba, Grace. "Spline models for observational data." Society for Industrial and Applied Mathematics, 1990.

See Also

Krig, mKrig, spatialProcess, Tps.cov, sim.spatialProcess, summary.Krig, predict.Krig, predictSE.Krig, predictSurface, predictSurface.fastTps, plot.Krig, surface.Krig, sreg

Examples

```
#2-d example
data(ozone2)

x<- ozone2$lon.lat
y<- ozone2$y[16,]

fit<- Tps(x,y)  # fits a surface to ozone measurements.

set.panel(2,2)
plot(fit) # four diagnostic plots of  fit and residuals.
set.panel()

# summary of fit and estimates of lambda, the smoothing parameter
summary(fit)

surface( fit) # Quick image/contour plot of GCV surface.

# NOTE: the predict function is quite flexible:

     look<- predict( fit, lambda=2.0)
#  evaluates the estimate at lambda =2.0  _not_ the GCV estimate
#  it does so very efficiently from the Krig fit object.

     look<- predict( fit, df=7.5)
#  evaluates the estimate at the lambda values such that
#  the effective degrees of freedom is 7.5


# compare this to fitting a thin plate spline with
# lambda chosen so that there are 7.5 effective
# degrees of freedom in estimate
# Note that the GCV function is still computed and minimized
# but the lambda values used correspond to 7.5 df.

fit1<- Tps(x, y,df=7.5)

set.panel(2,2)
plot(fit1) # four diagnostic plots of  fit and residuals.
          # GCV function (lower left) has vertical line at 7.5 df.
set.panel()

# The basic matrix decompositions are the same for
# both fit and fit1 objects.

# predict( fit1) is the same as predict( fit, df=7.5)
# predict( fit1, lambda= fit$lambda) is the same as predict(fit)


# predict onto a grid that matches the ranges of the data.

out.p<-predictSurface( fit)
```

```
imagePlot( out.p)

# the surface function (e.g. surface( fit))  essentially combines
# the two steps above

# predict at different effective
# number of parameters
out.p<-predictSurface( fit, df=10)

## Not run:
# predicting on a grid along with a covariate
  data( COmonthlyMet)
# predicting average daily minimum temps for spring in Colorado
# NOTE to create an  4km  elevation grid:
# data(PRISMelevation); CO.elev1 <- crop.image(PRISMelevation, CO.loc )
# then use same grid for the predictions: CO.Grid1<- CO.elev1[c("x","y")]
  obj<- Tps( CO.loc, CO.tmin.MAM.climate, XMat= CO.elev)
  out.p<-predictSurface( obj,
              CO.Grid, XMatGrid= CO.elevGrid)
  imagePlot( out.p)
  US(add=TRUE, col="grey")
  contour( CO.elevGrid, add=TRUE, levels=c(2000), col="black")

## End(Not run)
## Not run:
#A 1-d example  with confidence intervals
data(rat.diet)
  out<-Tps( rat.diet$t, rat.diet$trt) # lambda found by GCV
  out
  plot( out$x, out$y)
  xgrid<- seq(  min( out$x), max( out$x),,100)
  fhat<- predict( out,xgrid)
  lines( xgrid, fhat,)
  SE<- predictSE( out, xgrid)
  lines( xgrid,fhat + 1.96* SE, col="red", lty=2)
  lines(xgrid, fhat - 1.96*SE, col="red", lty=2)

#
# compare to the ( much faster) B spline algorithm
#  sreg(rat.diet$t, rat.diet$trt)

# Here is a 1-d example with 95 percent  CIs  where sreg would not
# work:
#  sreg would give the right estimate here but not the right CI's
  x<- seq( 0,1,,8)
  y<- sin(3*x)
  out<-Tps( x, y) # lambda found by GCV
  plot( out$x, out$y)
  xgrid<- seq(  min( out$x), max( out$x),,100)
  fhat<- predict( out,xgrid)
  lines( xgrid, fhat, lwd=2)
  SE<- predictSE( out, xgrid)
  lines( xgrid,fhat + 1.96* SE, col="red", lty=2)
```

```
  lines(xgrid, fhat - 1.96*SE, col="red", lty=2)

## End(Not run)

# More involved example adding a covariate to the fixed part of model
## Not run:
data(RMprecip)
set.panel( 1,3)
# without elevation covariate
  out0<-Tps( RMprecip$x,RMprecip$y)
  surface( out0)
  US( add=TRUE, col="grey")

# with elevation covariate
  out<- Tps( RMprecip$x,RMprecip$y, XMat=RMprecip$elev)

# NOTE: out$d[4] is the estimated elevation coefficient
# it is easy to get the smooth surface separate from the elevation.
  out.p<-predictSurface( out, drop.XMat=TRUE)
  surface( out.p)
  US( add=TRUE, col="grey")

# and if the estimate is of high resolution and you get by with
# a simple discretizing -- does not work in this case!
  quilt.plot( out$x, out$fitted.values)

#
# the exact way to do this is evaluate the estimate
# on a grid where you also have elevations
# An elevation DEM from the PRISM climate data product (4km resolution)
  data(RMelevation)
  grid.list<- list( x=RMelevation$x, y= RMelevation$y)
  fit.full<- predictSurface( out, grid.list, XMatGrid= RMelevation)

# this is the linear fixed part of the second spatial model:
# lon,lat and elevation
  fit.fixed<- predictSurface( out, grid.list, just.fixed=TRUE,
                   XMatGrid= RMelevation)

# This is the smooth part but also with the linear lon lat terms.
  fit.smooth<-predictSurface( out, grid.list, drop.XMat=TRUE)

#
  set.panel( 3,1)

  fit0<- predictSurface( out0, grid.list)
  image.plot( fit0)
  title(" first spatial model (w/o elevation)")
  image.plot( fit.fixed)
  title(" fixed part of second model (lon,lat,elev linear model)")
  US( add=TRUE)
  image.plot( fit.full)
  title("full prediction second model")
```

```
  set.panel()

## End(Not run)
###
### fast Tps
# m=2   p= 2m-d= 2
#
# Note: aRange = 3 degrees is a very generous taper range.
# Use some trial aRange value with rdist.nearest to determine a
# a useful taper. Some empirical studies suggest that in the
# interpolation case in 2 d the taper should be large enough to
# about 20 non zero nearest neighbors for every location.

data(RMprecip)
out2<- fastTps( RMprecip$x,RMprecip$y,m=2, aRange=4,
                      profileLambda=FALSE)

# note that fastTps produces a object of classes  spatialProcess and mKrig
# so one can use all the
# the overloaded functions that are defined for these classes.
# predict, predictSE, plot, sim.spatialProcess
# summary of what happened note estimate of effective degrees of
# freedom
# profiling on lambda has been turned off to make this run quickly
# but it is suggested that one examines the the profile likelihood over lambda

  print( out2)



## Not run:
set.panel( 1,2)
surface( out2)

#
# now use great circle distance for this smooth
# Here "aRange" for the taper support is the great circle distance in degrees latitude.
# Typically for data analysis it more convenient to think in degrees. A degree of
# latitude is about 68 miles (111 km).
#
out3<- fastTps( RMprecip$x,RMprecip$y,m=2, lon.lat=TRUE, aRange= 4*68 )
print( out3)  # note the effective degrees of freedom is different.
surface(out3)

set.panel()

## End(Not run)

## Not run:
#
# simulation reusing Tps/Krig object
#
data(rat.diet)
```

```
fit<- Tps( rat.diet$t, rat.diet$trt)
true<- fit$fitted.values
N<-  length( fit$y)
temp<- matrix(  NA, ncol=50, nrow=N)
tau<- fit$tauHat.GCV
for (  k in 1:50){
ysim<- true + tau* rnorm(N)
temp[,k]<- predict(fit, y= ysim)
}
matplot( fit$x, temp, type="l")

## End(Not run)

#
#4-d example
data(BD)
fit<- Tps(BD[,1:4],BD$lnya,scale.type="range")

# plots fitted surface and contours
# default is to hold 3rd and 4th fixed at median values

surface(fit)
```

`transformx` *Linear transformation*

Description

Linear transformation of each column of a matrix. There are several choices of the type of centering and scaling.

Usage

```
transformx (x, scale.type = "unit.sd", x.center, x.scale)
```

Arguments

`x`	Matrix with columns to be transformed.
`scale.type`	Type of transformation; the default is "unit.sd": subtract the mean and divide by the standard deviation. Other choices are "unscaled" (do nothing), "range" (transform to [0,1]), "user" (subtract a supplied location and divide by a scale).
`x.center`	A vector of centering values to subtract from each column.
`x.scale`	A vector of scaling values to divide into each column.

Details

After deciding what the centering and scaling values should be for each column of x, this function just calls the standard utility scale. This function was created partly to attach the transformation information as attributes to the transformed matrix. It is used in Krig, cover.design, krig.image, etc. to transform the independent variables.

Value

A matrix whose columns have been transformed. This matrix also has the attributes: scale.type, x.center and x.scale with the transformation information.

See Also

scale

Examples

```
#
data(Chicago03)
newx<-transformx( Chicago03$x, scale.type="range")
```

US *Plot of the US with state boundaries*

Description

Quickly plots medium-resolution outlines of the US with the states and bodies of water. A simple wrapper for the map function from the maps package.

Usage

```
US( ...)
```

Arguments

...	These are the arguments that are passed to the map function from the maps package.

Details

The older version of this function (fields < 6.7.2) used the FIELDS dataset US.dat for the coordinates. Currently, this has been switched to use the maps package.

See Also

world

Examples

```
# Draw map in device color # 3
US( col=3)
```

US.dat *Outline of conterminous US and states.*

Description

This data set is used by the fields function US to draw a map. It is the medium-resolution outline that is produced by drawing the US from the maps package.

---

vgram *Traditional or robust variogram methods for spatial data*

Description

vgram computes pairwise squared differences as a function of distance. Returns an S3 object of class "vgram" with either raw values or statistics from binning. crossCoVGram is the same as vgram but differences are taken across different variables rather than the same variable.

plot.vgram and boxplotVGram create lineplots and boxplots of vgram objects output by the vgram function. boxplotVGram plots using the base R boxplot function, and plots estimates of the mean over the boxplot.

The getVGMean function returns the bin centers and means of the vgram object based on the bin breaks provided by the user.

Usage

```
vgram(loc, y, id = NULL, d = NULL, lon.lat = FALSE,
                    dmax = NULL, N = NULL, breaks = NULL, prettyBins = FALSE,
                    type=c("variogram", "covariogram", "correlogram"))

crossCoVGram(loc1, loc2, y1, y2, id = NULL, d = NULL, lon.lat = FALSE,
                        dmax = NULL, N = NULL, breaks = NULL,
                        type=c("cross-covariogram", "cross-correlogram"),
                        prettyBins = FALSE)

boxplotVGram(x, N=10, breaks = pretty(x$d, N, eps.correct = 1), plot=TRUE, plot.args, ...)

## S3 method for class 'vgram'
plot(x, N=10, breaks = pretty(x$d, N, eps.correct = 1), add=FALSE, ...)

getVGMean(x, N = 10, breaks = pretty(x$d, N, eps.correct = 1))
```

Arguments

loc	Matrix where each row is the coordinates of an observed point of the field
y	Value of the field at locations
loc1	Matrix where each row is the coordinates of an observed point of field 1
loc2	Matrix where each row is the coordinates of an observed point of field 2
y1	Value of field 1 at locations
y2	Value of field 2 at locations
id	A 2-column matrix that specifies which variogram differences to find. If omitted, all possible pairings are found. This can be used if the data has an additional covariate that determines proximity, for example, a time window.
d	Distances among pairs indexed by id. If not included, distances are found directly from loc.
lon.lat	If true, locations are assumed to be longitudes and latitudes and distances found are great circle distances (in miles; see rdist.earth). Default is FALSE.
dmax	Maximum distance to compute variogram.
N	Number of bins to use. The break points are found by the pretty function and so there may not be exactly N bins. Specify the breaks explicitly if you want exactly N bins.
breaks	Bin boundaries for binning variogram values. Need not be equally spaced but must be ordered.
x	An object of class "vgram" (an object returned by vgram)
add	If TRUE, adds empirical variogram lineplot to current plot. Otherwise, creates new plot with empirical variogram lineplot.
plot	If TRUE, creates a plot; otherwise, returns variogram statistics output by bplot.xy.
plot.args	Additional arguments to be passed to plot.vgram.
prettyBins	If FALSE, creates exactly N-1 bins. If TRUE, you are at the mercy of giving N to the pretty function!
type	One of "variogram", "covariogram", "correlogram", "cross-covariogram", and "cross-correlogram". vgram supports the first three of these and crossCoVGram supports the last two.
...	Additional arguments passed to plot for plot.vgram or to bplot.xy for boxplotVGram.

Value

vgram and crossCoVGram return a "vgram" object containing the following values:

vgram	Variogram or covariogram values
d	Pairwise distances
call	Calling string
stats	Matrix of statistics for values in each bin. Rows are the summaries returned by the stats function or describe. If neither breaks nor N arguments are supplied, then this component is not computed.

centers Bin centers.

If boxplotVGram is called with plot=FALSE, it returns a list with the same components as returned by bplot.xy

References

See any standard reference on spatial statistics. For example, Cressie, Spatial Statistics.

Author(s)

John Paige, Doug Nychka

See Also

vgram.matrix, bplot.xy, bplot

Examples

```
#
# compute variogram for the midwest ozone field  day 16
# (BTW this looks a bit strange!)
#
data( ozone2)
good<- !is.na(ozone2$y[16,])
x<- ozone2$lon.lat[good,]
y<- ozone2$y[16,good]

look<-vgram( x,y, N=15, lon.lat=TRUE) # locations are in lon/lat so use right
#distance
# take a look:
plot(look, pch=19)
#lines(look$centers, look$stats["mean",], col=4)

brk<- seq( 0, 250,, (25 + 1) ) # will give 25 bins.

## or some boxplot bin summaries using base R graphics
look<-vgram( x,y, N=45, lon.lat=TRUE)
bplot.xy( look$d, look$vgram,
          breaks=look$breaks, outline=FALSE,
          xlim=c( 0,250), xlab="miles",
          ylim=c( 0, 2000), ylab="Variogram")
lines( look$centers, look$stats[2,], type="b",
col="magenta", pch=16)

#
# compute equivalent covariogram, but leave out the boxplots
#
look<-vgram( x,y, N=15, lon.lat=TRUE, type="covariogram")
plot(look, breaks=brk, col=4)

#
# compute equivalent cross-covariogram of the data with itself
```

```
#(it should look almost exactly the same as the covariogram of
#the original data, except with a few more points in the
#smallest distance boxplot and points are double counted)
#
look = crossCoVGram(x, x, y, y, N=15, lon.lat=TRUE, type="cross-covariogram")
plot(look, breaks=brk, col=4)
```

---

vgram.matrix *Computes a variogram from an image*

Description

Computes a variogram for an image taking into account different directions and returning summary information about the differences in each of these directions.

Usage

```
vgram.matrix(dat, R = NULL, dx = NULL, dy = NULL)

## S3 method for class 'vgram.matrix'
plot(x,...)
```

Arguments

dat	Either a matrix, where the spacing of rows and columns is assumed to be the same distance, or a list in image format with components x, y and z.
R	Maximum radius for finding variogram differences assuming that the grid points are spaced one unit apart. Default is to go out to a radius of 5*max(c(dx, dy)).
dx	The spacing of grid points on the X axis. This is used to calculate the correct distance between grid points. If dx is not equal to dy, then the collapse argument must be FALSE. If an image object is passed and dx and dy are not specified, they will be calculated from the x and y components of the image list.
dy	The spacing of grid points on the Y axis. See additional notes for dx.
x	Returned object from vgram.matrix
...	Arguments for image.plot

Details

For the "full" case the statistics can summarize departures from isotropy by separating the variogram differences according to orientation. For small R this runs efficiently because the differences are found by subsetting the image matrix.

For example, suppose that a row of the ind matrix is (2,3). The variogram value associated with this row is the mean of the differences (1/2)*(X(i,j)- X(i+2,j+3))**2 for all i and j. (Here X(.,.) are the values for the spatial field.) In this example d= sqrt(13) and there will be another entry with the same distance but corresponding to the direction (3,2). plot.vgram.matrix attempts to organize all the different directions into a coherent image plot.

Value

An object of class vgram.matrix with the following components: d, a vector of distances for the differences, and vgram, the variogram values. This is the traditional variogram ignoring direction.

d.full, a vector of distances for all possible shifts up to distance R, ind, a two-column matrix giving the x and y increment used to compute the shifts, and vgram.full, the variogram at each of these separations. Also computed is vgram.robust, Cressie's version of a robust variogram statistic.

Also returned is the component N, the number of differences found for each separation case.

See Also

vgram

Examples

```
# variogram for Lennon image.
data(lennon)
out<-vgram.matrix( lennon)

plot( out$d, out$vgram, xlab="separation distance", ylab="variogram")
# image plot of vgram values by direction.

# look at different directions
out<-vgram.matrix( lennon, R=8)

plot( out$d, out$vgram)
# add in different orientations
points( out$d.full, out$vgram.full, col="red")

#image plot of variogram values for different directions.
set.panel(1,1)
plot.vgram.matrix( out)
# John Lennon appears remarkably isotropic!
```

Wendland *Wendland family of covariance functions and supporting numerical functions*

Description

Computes the compactly supported, stationary Wendland covariance function as a function of distance. This family is useful for creating sparse covariance matrices.

Usage

```
Wendland(d, aRange = 1, dimension, k,derivative=0, phi=NA, theta=NULL)

Wendland2.2(d, aRange=1, theta=NULL)
Wendland.beta(n,k)
wendland.eval(r, n, k, derivative = 0)
fields.pochup(q, k)
fields.pochdown(q, k)
fields.D(f,name,order = 1)
```

Arguments

d	Distances between locations. Or, for wendland.coef, the dimension of the locations.
aRange	Scale for distances. This is the same as the range parameter.
theta	Same as aRange.
dimension	Dimension of the locations
n	Dimension for computing Wendland polynomial coefficients
k	Order of covariance function.
derivative	Indicates derivative of covariance function
phi	Deprecated argument; will stop if not NA. (Formerly the scale factor to multiply the function. Equivalent to the marginal variance or sill if viewed as a covariance function.)
r	Real value in [0,1] to evaluate Wendland function.
q	Order of Pochhammer symbol
f	Numerical expression to differentiate.
name	Variable with which to take derivative.
order	Order of derivative.

Details

This is the basic function applied to distances and called by the `wendland.cov` function. It can also be used as the Covariance or Taper specifications in the more general stationary.cov and stationary.taper.cov functions. The proofs and construction of the Wendland family of positive definite functions can be found in the work of Wendland (1995). (H. Wendland. Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree. AICM 4(1995), pp 389-396.)

The Wendland covariance function is a positive polynomial on [0,aRange] and zero beyond aRange. It is further normalized in these fields functions to be 1 at 0. The parameter `k` determines the smoothness of the covariance at zero. The additional parameter `n` or `dimension` is needed because the property of positive definiteness for radial functions depends on the dimension being considered.

The polynomial terms of the Wendland function are computed recursively based on the values of k and dimension in the function wendland.eval. The matrix of coefficients found by Wendland.beta is used to weight each polynomial term and follows Wendland's original construction of these functions. The recursive definition of the Wendland coefficients depends on Pochhammer symbols akin to binomial coefficients:

fields.pochup(q, k) calculates the Pochhammer symbol for rising factorial q(q+1)(q+2)...(q+k-1)

and

fields.pochdown(q, k) calculates the Pochhammer symbol for falling factorial q(q-1)(q-2)...(q-k+1).

Derivatives are found symbolically using a recursive modification of the base function D (fields.D) and then evaluated numerically based on the polynomial form.

A specific example of the Wendland family is Wendland2.2 (k=2, dimension=2). This is included mainly for testing, but the explicit formula may also be enlightening.

Value

A vector of the covariances or its derivative.

Author(s)

Doug Nychka, Ling Shen

See Also

wendland.cov, stationary.taper.cov

Examples

```
dt<- seq( 0,1.5,, 200)

y<- Wendland( dt, k=2, dimension=2)

plot( dt, y, type="l")

# should agree with

y.test<- Wendland2.2( dt)
points( dt, y.test)

# second derivative
plot( dt, Wendland( dt, k=4, dimension=2, derivative=2), type="l")

# a radial basis function using the Wendland  the  "knot" is at (.25,.25)
gl<- list( x= seq( -1,1,,60), y = seq( -1,1,,60) )

bigD<- rdist( make.surface.grid( gl), matrix( c(.25,.25), nrow=1))
RBF<- matrix(Wendland( bigD, k=2, dimension=2), 60,60)
```

```
# perspective with some useful settings for shading.
persp( gl$x, gl$y, RBF, theta =30, phi=20, shade=.3, border=NA, col="grey90")
```

`world` *Plot of the world*

Description

Quickly plots medium-resolution outlines of large land masses. This is a simple wrapper for the map function from the maps package.

Usage

```
world(...)
world.land(  ...)
world.color( ... )
in.land.grid(...)
```

Arguments

`...`	Same arguments used by the `map` function from the maps package.

Details

See the longstanding `maps` package for documentation on this function. The functions world.land, world.color and in.land.grid have been deprecated but can be recovered from versions of fields 6.7.1 or older.

See Also

US, in.poly, in.poly.grid

Examples

```
## Not run:
world()
# add the US
US( add=TRUE,col="blue")

world( fill=TRUE) # land filled in black

## Western Europe
world( xlim=c(-10,18),ylim=c(36,60),fill=TRUE, col="darkgreen",
border="green1")

## End(Not run)
```

WorldBankCO2 *Carbon emissions and demographic covariables by country for 1999.*

Description

These data are a small subset of the demographic data compiled by the World Bank. The data has been restricted to 1999 and to countries with a population larger than 10 million. Also, only countries reporting all the covariables are included.

Usage

```
data(WorldBankCO2)
```

Format

This is a 75X5 matrix with the row names identifying countries and the columns being the covariables: "GDP.cap" "Pop.mid" "Pop.urb" "CO2.cap" "Pop"

- GDP.cap: Gross domestic product (in US dollars) per capita.
- Pop.mid: Percentage of the population within the ages of 15 through 65.
- Pop.urb: Percentage of the population living in an urban environment
- CO2.cap: Equivalent CO2 emissions per capita
- Pop: Population

Reference

Romero-Lankao, P., J. L. Tribbia and D. Nychka (2008) Development and greenhouse gas emissions deviate from the modernization theory and convergence hypothesis. *Climate Research* 38, 17-29.

Creating dataset

Listed below are scripts to create this data set from spreadsheet on the World Bank CDs:

```
## read in comma delimited spreadsheet
  read.csv("climatedemo.csv", stringsAsFactors=FALSE)->hold
## convert numbers to matrix of data
  Ddata<- as.matrix(  hold[,5:51] )
  Ddata[Ddata==".."] <- NA
## still in character form; parse as numeric
  Ddata<- matrix( as.numeric( Ddata), nrow=1248, ncol=ncol( Ddata),
  dimnames=list( NULL, format( 1960:2006) ))
## these are the factors indicating the different variables
### unique( Fac) gives the names of factors
  Fac<- as.character( hold[,1])
  years<- 1960:2006
# create separate tables of data for each factor
```

```
 temp<- unique( Fac)
## also subset Country id and name
  Country.id<- as.character( hold[Fac== temp[1],3])
  Country<- as.character( hold[Fac== temp[1],4])
  Pop<- Ddata[ Fac== temp[2],]
  CO2<- Ddata[ Fac== temp[1],]
  Pop.mid<- Ddata[ Fac== temp[3],]
  GDP.cap<- Ddata[ Fac== temp[4],]
  Pop.urb<- Ddata[ Fac== temp[5],]
  CO2.cap<- CO2/Pop
  dimnames( Pop)<- list( Country.id,format(years))
  dimnames( CO2)<- list( Country.id,format(years))
  dimnames( Pop.mid)<- list( Country.id,format(years))
  dimnames( Pop.urb)<- list( Country.id,format(years))
  dimnames( CO2.cap)<- list( Country.id,format(years))
# delete temp data sets
  rm( temp)
  rm( hold)
  rm( Fac)
# define year to do clustering.
  yr<- "1999"
# variables for clustering combined as columns in a matrix
  temp<-cbind( GDP.cap[,yr], Pop.mid[,yr], Pop.urb[,yr],CO2[,yr],Pop[,yr])
# add column names and figure how many good data rows there are.
        dimnames( temp)<-list( Country, c("GDP.cap","Pop.mid","Pop.urb",
                                          "CO2.cap", "Pop"))
        good<-complete.cases(temp)
        good<- good & Pop[,yr] > 10e6
# subset with only the complete data rows
        WorldBankCO2<- temp[good,]
  save(WorldBankCO2, file="WorldBankCO2.rda")
```

Examples

```
data(WorldBankCO2)
plot( WorldBankCO2[,"GDP.cap"], WorldBankCO2[,"CO2.cap"], log="xy")
```

`xline` *Draw a vertical line*

Description

Adds vertical lines in the plot region.

Usage

```
xline(x, ...)
```

Arguments

x	Values on x axis specifying location of vertical lines.
...	Any plotting options for abline.

See Also

yline, abline

Examples

```
plot( 1:10)
xline( 6.5, col=2)

world( col=3)
yline( seq( -80,80,10),col=4, lty=2)
xline( seq( -180,180,10),col=4,lty=2)
yline( 0, lwd=2, col=4)
```

---

yline *Draw horizontal lines*

Description

Adds horizontal lines in the plot region.

Usage

```
yline(y, ...)
```

Arguments

y	Values on y axis specifying location of horizontal lines.
...	Any plotting options for abline.

See Also

xline, abline

Examples

```
world( col=3)
yline( seq( -80,80,10),col=4, lty=2)
xline( seq( -180,180,10),col=4,lty=2)
yline( 0, lwd=2, col=4)
```

Index

∗ **IO**
summary.ncdf, 219
∗ **aplot**
arrow.plot, 5
tim.colors, 226
xline, 249
yline, 250
∗ **compact**
compactToMat, 23
∗ **covariance**
CovarianceUpper, 34
∗ **datasets**
AudiA4, 10
BD, 10
Chicago ozone test data, 14
CO2, 18
Colorado Monthly Meteorological Data, 20
fields, 47
flame, 58
glacier, 59
lennon, 111
LENSExample, 112
minitri, 113
NorthAmericanRainfall, 130
ozone2, 136
rat.diet, 175
RCMexample, 176
registeringCode, 181
RMprecip, 183
US.dat, 240
WorldBankCO2, 248
∗ **hplot**
add.image, 4
bplot, 11
bplot.xy, 13
colorbar.plot, 21
drape.plot, 41
fields.grid, 54
fields.hints, 55
image.plot, 69
image2lz, 81
imagePlot, 83
panelPlot, 137
plot.surface, 146
pushpin, 161
quantilePlot, 169
quilt.plot, 171
ribbon.plot, 182
set.panel, 186
US, 239
world, 247
∗ **manip**
as.image, 6
as.surface, 8
transformx, 238
∗ **matrix**
compactToMat, 23
∗ **misc**
fields testing scripts, 51
grid list, 62
∗ **smooth**
image.smooth, 78
naturalSplineBasis, 128
qsreg, 162
smooth.2d, 194
splint, 209
sreg, 211
Tps, 229
∗ **spatial**
circulantEmbedding, 15
Covariance functions, 25
cover.design, 35
Exponential, Matern, Radial Basis, 45
fields-stuff, 52
image.cov, 65
interp.surface, 95

Krig, 97
Krig.Amatrix, 105
Krig.null.function, 107
Krig.replicates, 108
KrigFindLambda, 109
mKrig, 113
mKrigMLE, 122
offGridWeights, 132
plot.Krig, 145
poly.image, 147
predict.Krig, 150
predictSE, 153
predictSurface, 155
print.Krig, 160
QTps, 164
rdist, 177
rdist.earth, 179
sim.spatialProcess, 187
spam2lz, 196
spatialProcess, 198
summary.Krig, 218
surface.Krig, 220
The Engines:, 222
vgram, 240
vgram.matrix, 243
Wendland, 244
∗ **univar**
stats, 215
stats.bin, 216
%d∗% *(*The Engines:*)*, 222

abline, *250*
add.image, 4, *49*, *74*
addColorBar, *141*
addColorBar *(*imagePlot*)*, 83
addColorBarTriangle *(*imagePlot*)*, 83
addLegend, *141*, *173*
addLegend *(*imagePlot*)*, 83
addTitleBox *(*panelPlot*)*, 137
addToDiagC *(*registeringCode*)*, 181
approximateCovariance2D, *157*, *188*
approximateCovariance2D *(*offGridWeights*)*, 132
arrow.plot, 5
as.image, 6, *49*, *68*
as.surface, 8
AudiA4, 10
augmentPredictionGrid *(*offGridWeights*)*, 132
average.image, *49*
average.image *(*image2lz*)*, 81

BD, 10
blasterBlue *(*tim.colors*)*, 226
boxplotVGram *(*vgram*)*, 240
bplot, 11, *242*
bplot.xy, 13, *242*
bubblePlot, *74*
bubblePlot *(*quilt.plot*)*, 171

Chicago ozone test data, 14
ChicagoO3, *136*
ChicagoO3 *(*Chicago ozone test data*)*, 14
circulantEmbedding, 15
circulantEmbeddingSetup, *68*
circulantEmbeddingSetup *(*circulantEmbedding*)*, 15
CO.elev *(*Colorado Monthly Meteorological Data*)*, 20
CO.elevGrid *(*Colorado Monthly Meteorological Data*)*, 20
CO.Grid *(*Colorado Monthly Meteorological Data*)*, 20
CO.id *(*Colorado Monthly Meteorological Data*)*, 20
CO.loc *(*Colorado Monthly Meteorological Data*)*, 20
CO.names *(*Colorado Monthly Meteorological Data*)*, 20
CO.ppt *(*Colorado Monthly Meteorological Data*)*, 20
CO.tmax *(*Colorado Monthly Meteorological Data*)*, 20
CO.tmean.MAM.climate *(*Colorado Monthly Meteorological Data*)*, 20
CO.tmin *(*Colorado Monthly Meteorological Data*)*, 20
CO.years *(*Colorado Monthly Meteorological Data*)*, 20
CO2, 18, *49*
coef.Krig *(*Krig*)*, 97
color.scale, *49*
color.scale *(*tim.colors*)*, 226
Colorado Monthly Meteorological Data, 20
coloradoRed *(*tim.colors*)*, 226
colorBar, *74*
colorBar *(*imagePlot*)*, 83

colorbar.plot, 21
COmonthlyMet, *49*
COmonthlyMet *(*Colorado Monthly Meteorological Data*)*, 20
compactToMat, 23
compactToMatC *(*registeringCode*)*, 181
confidenceIntervalMLE *(*spatialProcess*)*, 198
Covariance functions, 25
CovarianceUpper, 34
cover.design, 35, *49*
crop.image, *49*
crop.image *(*image2lz*)*, 81
crossCoVGram *(*vgram*)*, 240
cubic.cov *(*Covariance functions*)*, 25

darkBlue *(*tim.colors*)*, 226
darkGray *(*tim.colors*)*, 226
designer.colors, *49*, *74*
designer.colors *(*tim.colors*)*, 226
discretize.image *(*grid list*)*, 62
dist, *24*, *178*
distMatHaversin *(*registeringCode*)*, 181
distMatHaversin2 *(*registeringCode*)*, 181
divMap *(*imagePlot*)*, 83
drape.color *(*drape.plot*)*, 41
drape.plot, 41, *49*

earthBlue *(*tim.colors*)*, 226
energyYellow *(*tim.colors*)*, 226
ensembleEnvelope *(*sim.spatialProcess*)*, 187
envelopePlot, 44
envrGreen *(*tim.colors*)*, 226
Exp.cov, *178*, *220*
Exp.cov *(*Covariance functions*)*, 25
Exp.image.cov *(*image.cov*)*, 65
Exp.simple.cov *(*Covariance functions*)*, 25
Exponential, *35*
Exponential *(*Exponential, Matern, Radial Basis*)*, 45
Exponential, Matern, Radial Basis, 45
ExponentialUpper *(*CovarianceUpper*)*, 34
ExponentialUpperC *(*registeringCode*)*, 181

fastTps, *48*
fastTps *(*Tps*)*, 229
fields, 47
fields testing scripts, 51
fields-package *(*fields*)*, 47
fields-stuff, 52
fields.color.picker *(*fields.hints*)*, 55
fields.convert.grid *(*grid list*)*, 62
fields.D *(*Wendland*)*, 244
fields.derivative.poly *(*fields-stuff*)*, 52
fields.duplicated.matrix *(*fields-stuff*)*, 52
fields.evlpoly *(*fields-stuff*)*, 52
fields.evlpoly2 *(*fields-stuff*)*, 52
fields.grid, 54
fields.hints, 55
fields.mkpoly *(*fields-stuff*)*, 52
fields.pochdown *(*Wendland*)*, 244
fields.pochup *(*Wendland*)*, 244
fields.rdist.near, *180*
fields.rdist.near *(*rdist*)*, 177
fields.style *(*fields.hints*)*, 55
fields.tests *(*fields testing scripts*)*, 51
fields.x.to.grid *(*grid list*)*, 62
fieldsPlotColors *(*tim.colors*)*, 226
filled.contour, *74*
fillGrid *(*interp.surface*)*, 95
findGridBox *(*offGridWeights*)*, 132
fitted.Krig *(*Krig*)*, 97
flame, 58

gcv.sreg *(*KrigFindLambda*)*, 109
get.rectangle *(*image2lz*)*, 81
getVGMean *(*vgram*)*, 240
glacier, *49*, 59
goldenTech *(*tim.colors*)*, 226
grid list, 61
grid.list, *49*, *133*
grid.list *(*grid list*)*, 62

half.image, *49*
half.image *(*image2lz*)*, 81

image, *74*
image.cov, 65
image.plot, 69, *84*, *173*
image.smooth, 78
image2lz, 81
imagePlot, *49*, *69*, *74*, 83, *141*
in.land.grid *(*world*)*, 247

in.poly, *49*
in.poly *(*image2lz*)*, 81
interp.surface, 95, *135*

Krig, 97, *110*, *125*, *222*, *225*, *233*
Krig.Amatrix, 105
Krig.check.xY *(*The Engines:*)*, 222
Krig.coef *(*The Engines:*)*, 222
Krig.engine.default *(*The Engines:*)*, 222
Krig.engine.fixed *(*The Engines:*)*, 222
Krig.make.u *(*The Engines:*)*, 222
Krig.make.W *(*The Engines:*)*, 222
Krig.make.Wi *(*The Engines:*)*, 222
Krig.null.function, 107
Krig.replicates, 108, *116*
Krig.transform.xY *(*The Engines:*)*, 222
KrigFindLambda, 109

larry.colors *(*tim.colors*)*, 226
lennon, *49*, 111
LENSExample, 112
lightBlue *(*tim.colors*)*, 226
lightGray *(*tim.colors*)*, 226

make.surface.grid *(*grid list*)*, 62
makeMultiIndex *(*grid list*)*, 62
Matern *(*Exponential, Matern, Radial Basis*)*, 45
matern.image.cov *(*image.cov*)*, 65
Mines.colors *(*tim.colors*)*, 226
minitri, 113
mKrig, *48*, 113, *125*, *200*, *231*, *233*
mKrig.grid, *118*
mKrig.grid *(*fields.grid*)*, 54
mKrig.trace, *124*
mKrigCheckXY *(*mKrig*)*, 113
mKrigFastPredict, *68*, *135*
mKrigFastPredict *(*predictSurface*)*, 155
mKrigFastPredictSetup, *157*
mKrigFastPredictSetup *(*offGridWeights*)*, 132
mKrigJointTemp.fn *(*mKrigMLE*)*, 122
mKrigMLE, 122
mKrigMLEGrid, *48*, *204*
mKrigMLEGrid *(*mKrigMLE*)*, 122
mKrigMLEJoint, *48*, *204*
mKrigMLEJoint *(*mKrigMLE*)*, 122
mltdrb *(*registeringCode*)*, 181
mtext, *138*
multebC *(*registeringCode*)*, 181
multwendlandg *(*registeringCode*)*, 181
mutedBlue *(*tim.colors*)*, 226

NativeSymbolInfo, *181*
naturalSplineBasis, 128
NorthAmericanRainfall, *49*, 130
NorthAmericanRainfall2 *(*NorthAmericanRainfall*)*, 130

offGridWeights, 132
offGridWeights1D *(*offGridWeights*)*, 132
offGridWeights2D, *189*
offGridWeights2D *(*offGridWeights*)*, 132
optim, *123–125*
ozone *(*Chicago ozone test data*)*, 14
ozone2, *14*, *49*, 136

Paciorek.cov *(*Covariance functions*)*, 25
paleBlue *(*tim.colors*)*, 226
panelPlot, 137
panelPlotCommonAxes, *49*, *90*
panelPlotCommonAxes *(*panelPlot*)*, 137
parse.grid.list *(*grid list*)*, 62
plot.Krig, 145, *204*, *233*
plot.spatialProcess *(*spatialProcess*)*, 198
plot.sreg *(*plot.Krig*)*, 145
plot.surface, *74*, 146
plot.vgram *(*vgram*)*, 240
plot.vgram.matrix *(*vgram.matrix*)*, 243
plotMatrix *(*imagePlot*)*, 83
poly.image, *74*, 147
predict.fastTps *(*predict.Krig*)*, 150
predict.Krig, *110*, 150, *233*
predict.mKrig, *204*
predict.mKrig *(*mKrig*)*, 113
predict.sreg *(*sreg*)*, 211
predict.Tps *(*predict.Krig*)*, 150
predictDerivative.Krig *(*predict.Krig*)*, 150
predictSE, *49*, 153
predictSE.Krig, *233*
predictSE.mKrig, *204*
predictSEUsingKrigA *(*predictSE*)*, 153
predictSurface, *134*, 155, *233*
predictSurface.fastTps, *233*
predictSurface.mKrig, *187*
predictSurfaceSE *(*predictSurface*)*, 155

print.Krig, 160
print.mKrig (mKrig), 113
print.mKrigSummary (mKrig), 113
print.spatialProcess (spatialProcess), 198
print.spatialProcessSummary (spatialProcess), 198
PRISMelevation, *49*
PRISMelevation (RMprecip), 183
profileCI (mKrigMLE), 122
profileMLE (spatialProcess), 198
pushpin, 161

qqnorm, *171*
QSreg (QTps), 164
qsreg, 162
QTps, *48*, *163*, 164
quantilePlot, 169
quilt.plot, *49*, *74*, 171

Rad.cov (Covariance functions), 25
Rad.image.cov (image.cov), 65
Rad.simple.cov (Covariance functions), 25
RadialBasis (Exponential, Matern, Radial Basis), 45
rat.diet, *49*, 175
RCMexample, *49*, 176
rdist, *24*, 177, *180*
rdist.earth, *178*, 179, *241*
RdistC (registeringCode), 181
RdistEarth (rdist.earth), 179
redFlannel (tim.colors), 226
registeringCode, 181
resid.Krig (Krig), 97
ribbon.plot, 182
RMelevation, *49*
RMelevation (RMprecip), 183
RMprecip, 183

set.panel, 186
setup.image.smooth (image.smooth), 78
silver (tim.colors), 226
sim.Krig (sim.spatialProcess), 187
sim.rf, *45*, *49*
sim.rf (circulantEmbedding), 15
sim.spatialProcess, *48*, 187, *233*
simLocal.spatialProcess, *17*, *48*
simLocal.spatialProcess (sim.spatialProcess), 187
simSpatialData (sim.spatialProcess), 187
smooth.2d, 194
snow.colors, *89*
snow.colors (tim.colors), 226
spam2full (spam2lz), 196
spam2lz, 196
spam2spind (spam2lz), 196
spatialProcess, *48*, *49*, *100*, *116*, 198, *231*, *233*
spatialProcessSetDefaults (spatialProcess), 198
spind2full (spam2lz), 196
spind2spam (spam2lz), 196
splint, *48*, *129*, 209
sreg, *48*, *129*, *163*, 211, *233*
stationary.cov, *17*, *48*, *68*, *125*, *178*, *180*, *220*
stationary.cov (Covariance functions), 25
stationary.image.cov, *17*
stationary.image.cov (image.cov), 65
stationary.taper.cov (Covariance functions), 25
stationaryImageCov, *16*
stationaryImageCov (image.cov), 65
stats, 215
stats.bin, 216
summary.Krig, 218, *233*
summary.mKrig (mKrig), 113
summary.ncdf, 219
summary.spatialProcess (spatialProcess), 198
supportsArg, 219
surface, *49*
surface.Krig, 220, *233*
surface.mKrig (surface.Krig), 220

test.for.zero (fields testing scripts), 51
The Engines:, 222
tim.colors, *74*, 226
topo.colors, *89*
Tps, *48*, *49*, *110*, *129*, *204*, *225*, 229
Tps.cov, *129*, *233*
Tps.cov (Covariance functions), 25
transformx, 238
two.colors (tim.colors), 226

unrollXMatGrid *(grid list)*, 62
US, *49*, 239
US.dat, 240

vgram, *49*, 240, *244*
vgram.matrix, *49*, *242*, 243

Wendland, 244
wendland.cov *(Covariance functions)*, 25
wendland.eval *(Wendland)*, 244
wendland.image.cov *(image.cov)*, 65
Wendland2.2 *(Wendland)*, 244
which.max.image *(image2lz)*, 81
which.max.matrix *(image2lz)*, 81
world, *49*, 247
WorldBankCO2, *49*, 248

xline, 249

yline, 250, *250*