Package 'fields'

November 11, 2019

Version 10.0

Date 2019-11-08

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. 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 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 and the National Center for Atmospheric Research. See the Fields URL for a vignette on using this package and some background on spatial statistics.

License GPL (>= 2)

URL https://github.com/NCAR/Fields

Depends R (>= 3.0), methods, spam, maps

NeedsCompilation yes

Repository CRAN

Author Douglas Nychka [aut, cre],
Reinhard Furrer [aut],
John Paige [aut],
Stephan Sain [aut],
Florian Gerber [aut],
Matthew Iverson [aut],
University Corporation for Atmospheric Research [cph]

Date/Publication 2019-11-11 22:50:12 UTC

R topics documented:

`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 (right side of image at x) and 1 (left 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 horizontal 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 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 times 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 a 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 depreciated.
ncol	Same as ny this is depreciated.
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 NA's are removed from the Z vector.
nx	Number of grid point 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

An 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. NA's 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
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 an "surface" object from grid values.*

Description

Reformats the vector from evaluating a function on a grid of points into a list for use with 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 later 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 to 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 matrix of values then the input vector is c(zimage). To go from the stacked vector to the matrix one needs the the nrow ncol and explains why grid information must also be specified.

Note that the z input argument must be in the order values in order of stacking 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
```

BD — *Data frame of the effect of buffer compositions on DNA strand displacement amplification. A 4-d regression data set with with replication. This is a useful test data set for exercising function fitting methods.*

Description

The BD data frame has 89 rows and 5 columns. There are 89 runs with four buffer components (KCL, MgCl2, KP04, dnTP) systematically varied in a space-filliing design. The response is the DNA amplification rate.

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 OConnell.

Becton Dickinson Research Center Research Triangle Park, NC

See Also

Tps

Examples

```
# fitting a DNA strand
# displacement amplification  surface to various buffer compositions
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, 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 plots 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.
boxwex	A boxplot argument to control the width of the boxplot. It behaves a little different than as an argumetn passed directly to boxplot. To make this a general function it is useful to scale this according to size of 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. horizontalIf TRUE draw boxplots horizontally the default is false, produce vertical box plots. lwdWidth(s) of lines in box plots. colColor(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 distrubtion of [Y|X] in terms of boxplots.

Usage

```
bplot.xy(x, y, N = 10, breaks = pretty(x, N, eps.correct = 1), plot=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.
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!
...	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
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 be named ozone but was changed to avoid conflict with other packages. The Chicago03 data is a list of components, x and y. 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 lasting scientific value is probably minimal.

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 data base.

See Also

Tps, Krig

Examples

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

CO2 *Simulated global CO2 observations*

Description

This is an example of 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 with 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, theta=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, theta=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 take a while)
look<- predictSurface( out2, grid.list=list( x=CO2.true$x, y=CO2.true$y))
image.plot(look)


## End(Not run)
```

`Colorado Monthly Meteorological Data`

Monthly surface meterology for Colorado 1895-1997

Description

Source: These 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 tempertuare is the mean of the daily minimum (maximum) temperatures.

Data domain:

A rectagular 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 preicipitation or only report temperatures have been excluded. In addition stations that have mismatches between locations and elevations from the two meta data 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) means by station for the period 1960-1990. If less than 15 years are present over this period an NA is recorded. No detreding or other adjustments have been made for these mean estimates.

Creation of data subset

Here is the precise R script used to create this data subset from the larger US monthly data set. This parent, R binary file can be obtained by contacting Doug Nychka (nychka@mines.edu).

These technical details are not needed for casual use of the data – skip down to examples for some R code that summarizes these data.

```
attach("RData.USmonthlyMet.bin")

#To find a subset that covers Colorado (with a bit extra):


indt<- UStinfo$lon< -101 &  UStinfo$lon > -109.5
indt<- indt & UStinfo$lat<41.5 &  UStinfo$lat>36.5

# check  US(); points( UStinfo[indt,3:4])

#find common names restricting choices to the temperature names
tn<-  match(  UStinfo$station.id, USpinfo$station.id)
indt<- !is.na(tn) & indt

# compare  metadata locations and elevations.
# initial matches to precip stations
CO.id<- UStinfo[indt,1]
CO.names<- as.character(UStinfo[indt,5])
pn<-  match(  CO.id, USpinfo$station.id)

loc1<- cbind( UStinfo$lon[indt], UStinfo$lat[indt], UStinfo$elev[indt])
loc2<- cbind( USpinfo$lon[pn], USpinfo$lat[pn], USpinfo$elev[pn])

abs(loc1- loc2) ->  temp
indbad<- temp[,1] > .02 | temp[,2]> .02 |  temp[,3] > 100

# tolerance at 100 meters set mainly to include the CLIMAX station
# a high altitude station.

data.frame(CO.names[ indbad], loc1[indbad,], loc2[indbad,], temp[indbad,] )


#  CO.names.indbad.       X1    X2   X3    X1.1  X2.1 X3.1 X1.2 X2.2 X3.2
#1     ALTENBERN    -108.38 39.50 1734 -108.53 39.58 2074 0.15 0.08  340
#2     CAMPO 7 S    -102.57 37.02 1311 -102.68 37.08 1312 0.11 0.06    1
#3     FLAGLER 2 NW -103.08 39.32 1519 -103.07 39.28 1525 0.01 0.04    6
#4     GATEWAY 1 SE -108.98 38.68 1391 -108.93 38.70 1495 0.05 0.02  104
#5     IDALIA       -102.27 39.77 1211 -102.28 39.70 1208 0.01 0.07    3
#6     KARVAL       -103.53 38.73 1549 -103.52 38.80 1559 0.01 0.07   10
#7     NEW RAYMER   -103.85 40.60 1458 -103.83 40.58 1510 0.02 0.02   52

# modify the indt list to exclude these mismatches (there are 7 here)

badones<-  match(  CO.id[indbad], UStinfo$station.id)
indt[ badones] <- FALSE

###### now have working set of CO stations have both temp and precip
```

```
##### and are reasonably close to each other.

N<- sum( indt)
# put data in time series order instead of table of year by month.
CO.tmax<-  UStmax[,,indt]
CO.tmin<-  UStmin[,,indt]

CO.id<- as.character(UStinfo[indt,1])
CO.elev<- UStinfo[indt,2]
CO.loc <-  UStinfo[indt,3:4]
CO.names<- as.character(UStinfo[indt,5])

CO.years<- 1895:1997

# now find precip stations that match temp stations
pn<-  match(  CO.id, USpinfo$station.id)
# number of orphans
sum( is.na( pn))

pn<- pn[ !is.na( pn)]
CO.ppt<- USppt[,,pn]

# checks --- all should zero

ind<- match( CO.id[45], USpinfo$station.id)
mean( abs( c(USppt[,,ind])   - c(CO.ppt[,,45]) ) , na.rm=TRUE)

ind<- match( CO.id[45], UStinfo$station.id)
mean( abs(c((UStmax[,,ind])) - c(CO.tmax[,,45])), na.rm=TRUE)

mean( abs(c((UStmin[,,ind])) - c(CO.tmin[,,45])), na.rm=TRUE)


# check order
ind<- match( CO.id, USpinfo$station.id)
 sum( CO.id != USpinfo$station.id[ind])
ind<- match( CO.id, UStinfo$station.id)
 sum( CO.id != UStinfo$station.id[ind])


# (3 4 5) (6 7 8) (9 10 11)  (12 1 2)
N<- ncol( CO.tmax)

CO.tmax.MAM<- apply( CO.tmax[,3:5,],c(1,3), "mean")

CO.tmin.MAM<- apply( CO.tmin[,3:5,],c(1,3), "mean")

CO.ppt.MAM<- apply( CO.ppt[,3:5,],c(1,3), "sum")
```

```
# Now average over 1961-1990
ind<-  CO.years>=1960 & CO.years < 1990

 temp<- stats( CO.tmax.MAM[ind,])
 CO.tmax.MAM.climate<- ifelse( temp[1,] >= 15, temp[2,], NA)

 temp<- stats( CO.tmin.MAM[ind,])
 CO.tmin.MAM.climate<- ifelse( temp[1,] >= 15, temp[2,], NA)

 CO.tmean.MAM.climate<- (CO.tmin.MAM.climate + CO.tmin.MAM.climate)/2

 temp<- stats( CO.ppt.MAM[ind,])
 CO.ppt.MAM.climate<- ifelse( temp[1,] >= 15, temp[2,], NA)


save( list=c( "CO.tmax", "CO.tmin", "CO.ppt",
              "CO.id", "CO.loc","CO.years",
              "CO.names","CO.elev",
              "CO.tmin.MAM", "CO.tmax.MAM", "CO.ppt.MAM",
              "CO.tmin.MAM.climate", "CO.tmax.MAM.climate",
              "CO.ppt.MAM.climate", "CO.tmean.MAM.climate"),
               file="COmonthlyMet.rda")
```

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,  Z= CO.elev )
good<- !is.na(CO.tmax.MAM.climate )
out<- MLE.Matern(CO.loc[good,],CO.tmax.MAM.climate[good],
         smoothness=1.0, Z= CO.elev[good] )
#MLE search on range suggests Tps model
```

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

Description

Adds one or more color scales in a horizontal orientation, 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 function 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 of 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 distribution 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, link{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 and 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 spareness, implement more efficient multiplcation 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 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, theta = 1, p=1, distMat = NA,
  C = NA, marginal = FALSE, onlyUpper=FALSE)

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

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

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

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, theta = 1, V = NULL, C = NA, marginal = FALSE,
  derivative = 0, distMat = NA, onlyUpper = FALSE, ...)

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

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

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 coordinatesof a particular point. If this is missing x1 is used.
theta	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 theta before the covariance function is evaluated.
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 `theta`. See Details below. If one has a vector of "theta's" that are the scaling for each coordinate then just express this as `V = diag(theta)` in the call to this function.
`C`	A vector with the same length as the number of rows of x2. If specified the covariance matrix will be multiplied by this vector.
`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 will trivial for this function is a required argument and capability for all covariance functions used with Krig.
`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.
`m`	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.)
`with.constant`	If TRUE includes complicated constant for radial basis functions. See the function `radbad.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.
`Covariance`	Character string that is the name of the covariance shape function for the distance between locations. Choices in fields are `Exponential`, `Matern`
`Distance`	Character string that is the name of the distance function to use. Choices in fields are `rdist`, `rdist.earth`
`Taper`	Character string that is the name of the taper function to use. Choices in fields are listed in help(taper).
`Dist.args`	A list of optional arguments to pass to the Distance function.
`Taper.args`	A list of optional arguments to pass to the Taper function. `theta` should always be the name for the range (or scale) paremeter.
`spam.format`	If TRUE returns matrix in sparse matrix format implemented in the spam package. If FALSE just returns a full matrix.
`k`	The order of the Wendland covariance function. See help on Wendland.
`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`.
`verbose`	If TRUE prints out some useful information for debugging.
`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.
`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.

... 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 theta. Specifically if theta is a matrix to represent a linear transformation of the coordinates, then let u= x1%*% t(solve(theta)) and v= x2%*% t(solve(theta)). 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 theta is a scalar then this defines an isotropic covariance function and the functional form is essentially exp(-D/theta).

Implementation: The function r.dist 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:

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 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 linear subspace. See Rad.simple.cov for a coding of the radial basis functions in R code.

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,theta=MyTheta)`

and

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

are the same. This also the same as

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

Stationary tapered covariance `stationary.taper.cov`**:** The resulting cross covariance is the direct or Shure 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)/theta)*Taper( rdist( x1,x2)/Taper.args$theta)`

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.

About the FORTRAN: The actual function `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 a 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 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
out<- Exp.cov( ChicagoO3$x, theta=100)

# out is a 20X20 matrix

out2<- Exp.cov( ChicagoO3$x[6:20,],ChicagoO3$x[1:2,], theta=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", theta=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,] , theta=1.5, Taper.args= list(k=2,theta=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,], theta=1.5) * Wendland(rdist(x[1:3,],x[1:10,]),
                     theta= 2.0, k=2, dimension=2)
 test.for.zero(  as.matrix(temp), temp2)

# Testing that the "V" option works as advertized ...
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, theta=1.5 )

# do multiply explicitly

 temp2<- stationary.cov( x,x[1:10,],
        Covariance= "Wendland",
            k=2, dimension=2, theta=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", theta=1.5,
      cov.args= list(Taper.args= list(k=2, dimension=2,theta=2.0) )
          ) -> out2
# NOTE: Wendland is the default taper here.

## End(Not run)

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

## 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)
```

Arguments

distMat	The distance matrix to evaluate the covariance over.
range	Range parameter default is one. Note that the scale can also be specified through the "theta" scaling argument used in fields covariance functions)
alpha	1/range

Value

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

Author(s)

John Paige

See Also

Exponential

Examples

```
set.seed(123)

#make distance matrix using the random locations
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 great 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 ignore.
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 rows. 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 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 criteria 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 nrun is greate 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 neighbor 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 the 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)

#
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)
```

---

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
add.legend	If true a color strip is added as a legend.
horizontal	If true color strip is put at 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 color scale is formed for midpoints of z obtained by averaging 4 corners.
breaks	Numerical divisions for the color scale. If the default (NA) is N+1 equally spaced points in the range zlim where N is the number of colors in col. This is the argument has the same effect as used in the image and image.plot functions.
eps	Amount to inflate the range (1+/- eps) to inlude 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 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 it is the range of the z2 matrix is used to generate the ranges of the color bar. Note that this may be different than 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 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 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 matrx (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=terrain.colors(128),zlim=c(0,300),
                     zlim2=c( 120,165), border=NA)

# note tranparent color for facets outside the zlim2 range


#The projection has been saved in pm
# add a point marking the summit
max( volcano)-> zsummit
ix<- row( volcano)[volcano==zsummit]
iy<- col( volcano)[volcano==zsummit]
trans3d( x[ix], y[iy],zsummit,pm)-> uv
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 = tim.colors(64) )$color.index

# wireframe with these colors
  persp( volcano, col=zcol, theta=30, phi=20)

# add legend using image.plot function
  image.plot( zlim=zlim, legend.only =TRUE, horizontal =TRUE, col= tim.colors(64))
```

envelopePlot *Add a shaded the 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", ...)
```

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.
...	Additional arguments to the base R function polygon

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 rho argument in mKrig or Krig. To set the nugget variance, use te sigma2 argument in mKrig or Krig.

Usage

```
Exponential(d, range = 1, alpha = 1/range, phi=1.0)
Matern(d , range = 1,alpha=1/range, smoothness = 0.5,
       nu= smoothness, phi=1.0)
Matern.cor.to.range(d, nu, cor.target=.5, guess=NULL,...)
RadialBasis(d,M,dimension, derivative = 0)
```

Arguments

d	Vector of distances or for Matern.cor.to.range just a single distance.
range	Range parameter default is one. Note that the scale can also be specified through the "theta" scaling argument used in fields covariance functions)
alpha	1/range
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/theta) is the exponential covariance for points separated by distance and range theta. Throughout fields this parameter is equivalent to rho and it recommended that rho 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/range)

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 theta, this function determines numerically the value of theta so that

Matern(d,range=theta,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. NOTE: Earlier versions of fields used ln(r^2) instead of ln(r) and so differ by a factor of 2.

Value

For the covariance functions: a vector of covariances.

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 programs for curve and function fitting with an emphasis on spatial data and spatial statistics. The major methods implemented include cubic and thin plate splines, universal Kriging and Kriging for large data sets. One main feature is any covariance function implemented in R code can be used for spatial prediction. 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 and I recommend the excellent fields vignette created by Ashton and Mitch: Fields Vignette

fields stives to have readable and tutorial code. Take a look at the source code for `mKrig` to see how things work "under the hood" and how a statistical, linear algebra computation is overloaded to handle sparse matrices.

To load fields with the comments retained in the source use `keep.source = TRUE` in the `library` command. We also keep the source on-line: refer to the github directory `https://github.com/NCAR/Fields` for commented source.

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 by maximum likelihood. Default covariance model is a Matern covariance function.
- Tps Thin Plate spline regression including GCV and REML estimates for the smoothing parameter.
- Krig Spatial process estimation that is the core function of fields.

  The Krig function allows you to supply a covariance function that is written in native R code. See (stationary.cov) that includes several families of covariances and distance metrics including the Matern and great circle distance.
- mKrig (micro Krig) are fastTps fast efficient Universal Kriging and spline-like functions, that can take advantage of sparse covariance functions and thus handle very large numbers of spatial locations. QTps A easy to use extension of thin plate splines for quantile and robust surface fitting.
- mKrigMLEGrid for maximum likelihood estimates of covariance parameters. This function also handles replicate fields assumed to be independent realizations at the same locations.

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 expresed in R code.
- as.image, image.plot, drape.plot, quilt.plot add.image, crop.image, half.image, average.image, designer.colors, color.scale, in.poly Many convenient functions for working with image data and rationally (well, maybe reasonably) creating and placing a color scale on an image plot. See also grid.list for how fields works with grids and US and world for adding a map quickly.
- sreg splint Fast 1-D smoothing splines and interpolating cubic splines.

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 fit at arbitrary points
predictSE- prediction standard errors at arbitrary points.
sim.rf- Simulate a random fields on a 2-d grid.

Getting Started

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

Graphics tips

help(fields.hints) gives some R code tricks for setting up common legends and axes. And has little to do with this package!

Testing See help(fields.tests) for testing fields.

Some fields datasets

- CO2 Global satelite CO2 concentrations (simulated field)

- RCMexample Regional climate model output
- lennon Image of John Lennon
- COmonthlyMet Monthly mean temperatures and precip for Colorado
- RMelevation Digital elevations for the Rocky Mountain Empire
- 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
- NorthAmericanRainfall 50+ year average and trend for summer rainfall at 1700+ stations.
- rat.diet Small paired study on rat food intake over time.
- WorldBankCO2 Demographic and carbon emission data for 75 countries and for 1999.

DISCLAIMER: The authors can not 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)
x<-ozone2$lon.lat
y<- ozone2$y[16,] # June 18, 1987

# pixel plot of spatial data
quilt.plot( x,y)
US( add=TRUE) # add US map

fit<- Tps(x,y)
# fits a GCV thin plate smoothing spline surface to ozone measurements.
# Hey, it does not get any easier than this!

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

# quick plot of predicted surface
set.panel()
surface(fit) # 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")


fit2<- spatialProcess( x,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  fit2 !
```

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 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 meaninful and documented way.

Usage

```
test.for.zero( xtest, xtrue,  tol= 1.0e-8, relative=TRUE, 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.)
`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 insure 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 differnces:

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.diagonalize2(A,B, verbose=FALSE)
fields.diagonalize(A,B)
fields.duplicated.matrix(mat, digits = 8)

fields.mkpoly(x, m = 2)

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

```
fields.evlpoly( x, coef)

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

Arguments

A	A positive definite matrix
B	A positive definite matrix
mat	Arbitrary matrix for examining rows
digits	Number of significant digits to use for comparing elements to determine duplciate 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 polnomial terms.
verbose	If TRUE prints condition number of A+B

Details

Given two matrices A (positive definite) and B (nonnegative definite) `fields.diagonalize` and `fields.diagonalize2` finds the matrix transformation G that will convert A to a identity matrix and B to a diagonal matrix:

G\^T A G= I G\^T B G= D.

`fields.diagonalize2` is not as easy to follow as `fields.diagonalize` but may be more stable and is the version used in the Krig engine.

`fields.duplicated` finds duplicate rows in a matrix. The digits arguments 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` is are the componets of a vector. (The fields function mkpoly returns the number of these terms.) In 2 dimensions with m=3 there 6 polynomial terms up to quadratic (3-1 =2) order and 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.

`fields.deriviative.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 orgainzed to be a clean utility for the 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 ths 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 my the fields.mkpoly function.

Author(s)

Doug Nychka

See Also

Krig, Tps, as.image, predict.Krig, predict.mKrig, Krig.engine.default, Wendland

---

`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 forlarge data sets by using a compactly supported covariance. The example below shows how ot evaluate a solution on a big grid. (Thanks to Jan Klennin for this example.)

Examples

```
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 interplotation 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 an index to the defaultcolors 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 lithuim 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. <s-section name= "DATA DESCRIPTION"> This is list with the following components

Arguments

`x`	x is a 2 column matrix with the different Fuel and oxygen flow rates for the burner.
`y`	y is the response. The intensity of light at a particular wavelength indicative of Lithium ions.

`gcv.Krig` *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

```
gcv.Krig(
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 sigma**2
verbose	If true prints intermediate results.
tol	Tolerance in delcaring 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 mathcing 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 decompostions 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 decompostions for multiple data sets. (But keep in mind if the x values change then the object must be recalculated.) The example below show show 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 diagonostic 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, shat= Implied estimate of sigma , -Log Profile= negative log of profiel 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, shat.

Author(s)

Doug Nychka

See Also

Krig, Tps, predict.Krig

Examples

```
#
Tps( ChicagoO3$x, ChicagoO3$y)-> obj # default is to find lambda by GCV
summary( obj)

gcv.Krig( obj)-> out
print( out$lambda.est) # results agree with Tps summary

sreg( rat.diet$t, rat.diet$trt)-> out
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
sigma<- .1
y<- f + rnorm( 150)*sigma

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)*sigma
# save GCV estimates
lambdaTable<- gcv.Krig(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="sigma hat", pch=16, col=c("orange3", "green2"), type="p")
yline( sigma, col="grey", lwd=2)


## 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 gird.list. The function `discretize.image` is a useful tool for "registering" irregular 2-d points to a grid.

Usage

```
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)
unrollZGrid( grid.list, ZGrid)
```

Arguments

`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.

ZGrid — An array or list form of covariates to use for prediction. This must match the grid.list argument. e.g. ZGrid and grid.list describe the same grid. If ZGrid 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( ZGrid) == c(10,15,2)`. If ZGrid is a list then the components x and y shold match those of grid.list and the z component follows the shape described above for the no list case.

Details

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 an increasing sequence of points or a single vaules. Obviously there should be only be two variables with sequences to define a grid for a surface.

Most of 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 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` and 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 fucntion at these lcoations 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 dependin 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 explicited 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 half way between these midpoints. The first and last boundaries points are determined by extrapolating so that the first and last box has the midpoint in its center. (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.

unrollZGrid Checks that the ZGrid object is compatible with th e grid.list and concatenates the grid arrays into vectors. This version of the covariates are used 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
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 compuations (the setup) can also be used for fast simulation of the processes on a grid using the circulant embedding technique.

Usage

```
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,theta= 1.0, smoothness=.5)

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 ind2. If ind1 is missing ind2 is coerced to be all grid locations.
Y	Vector to multiply by the cross covariance matrix. Y must be 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.
cov.function	Name of the (stationary) covariance function.
setup	If true do not do the multiplication but 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 theta 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.
M	Size of 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= length(grid\$x). This will give an exact result but smaller values of M will yield an approximate, faster result.
N	Size of y-grid used to compute multiplication by the FFT.
theta	Scale parameter for Matern.
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 and supports the conjugate gradient solution in krig.image. In doing so the observations can be irregular 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 (i,j) element is the covariance between the grid locations indexed at ind1[i,] and ind2[j,]. The returned result is Sigma%*%Y. Note that one can always recover the coordinates themselves by evaluating the grid list at the indices. e.g. cbind(grid\$x[ind1[,1]], grid\$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 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 theta. Specifically,

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

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

Note that if theta 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.

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 computing steps:

temp<- matrix(0, 100,100)

temp[ind2] <- Y

temp2<- big.Sigma%*% temp

temp2[ind1]

Notice how much we pad with zeroes or at the end throw away! Here the matrix multiplication is effected through convolution/FFT tricks to avoid creating and multiplying big.Sigma explicitly. It is often faster to multiply the regular grid and throw away the parts we do not need then to deal directly with the irregular set of locations.

Note: In this entire discussion Y is treated as vector. However if one has complete data then Y can also be interpreted as a 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

smooth.2d, as.image, krig.image, stationary.cov

Examples

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

junk<- matrix(rnorm(100*100), 100,100)

cov.obj<- stationary.image.cov( setup=TRUE,
             grid=list(x=1:100,y=1:100),theta=8)
result<-  stationary.image.cov(Y=junk,cov.obj=cov.obj)

image( matrix( result, 100,100)) # NOTE that is also a smoother!

# to do it again, no setup is needed
#  e.g.
#  junk2<- matrix(rnorm(100**2, 100,100))
```

```
#  result2<-  stationary.image.cov(Y=junk2, cov.obj=cov.obj)

# generate a grid and set of indices based on discretizing the locations
# in the precip dataset

 out<-as.image( RMprecip$y, x= RMprecip$x)
 ind1<- out$ind
 grid<- list( x= out$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

 cov.obj<- stationary.image.cov( setup=TRUE, grid=grid, theta=1.25)

# multiply covariance matrix by an arbitrary vector
 junk<-  rnorm(nrow( ind1))
 result<- stationary.image.cov( ind1, ind1, Y= junk,cov.obj=cov.obj)

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


# evaluate the covariance between all grid points and the center grid point
 Y<- matrix(0,cov.obj$m, cov.obj$n)
 Y[32,32]<- 1
 result<- stationary.image.cov( Y= Y,cov.obj=cov.obj)
# covariance surface with respect to the grid point at (32,32)
#
# reshape "vector" as an image
 temp<-  matrix( result, cov.obj$m,cov.obj$n)
 image.plot(cov.obj$grid$x,cov.obj$grid$y, temp)
# or persp( cov.obj$grid$x,cov.obj$grid$y, temp)

# check out the Matern
 grid<- list( x= seq(-105,-99,,64), y= seq( 40,45,,64))
 cov.obj<- matern.image.cov(
             setup=TRUE, grid=grid, theta=.55, smoothness=1.0)
 Y<- matrix(0,64,64)
 Y[16,16]<- 1

 result<- matern.image.cov( Y= Y,cov.obj=cov.obj)
  temp<-  matrix( result, cov.obj$m,cov.obj$n)
 image.plot( cov.obj$grid$x,cov.obj$grid$y, temp)

# Note we have centered at the location (grid$x[16],grid$y[16]) for this case
#  using sim.rf to simulate an Matern field
  look<- sim.rf( cov.obj)
```

```
image.plot( grid$x, grid$y, look)
```

`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 that often arise from regular grids transformed with a map projection or a scaling and rotation of coordinates.

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 unequal 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 argruments passed here will only have impact on the image plot. To change the graphical defaults for the legend use the individual legend arguments and `legend.arg` listed below.
`add`	If true add 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 quadralaterals 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) 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 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 a 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 the rotate text or other details. This is in the form of 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 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 usual evident from the plot title.
legend.shrink	Amount to shrink the size of 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 that 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 for the case of unequally spaced grids with x and y being matrices of grid point locations. If FALSE (default) the quadralaterals 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 be used to define the quadralaterals. (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.

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 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 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 cloropleths too: It should be noted that this image function is slightly different than a cloropleth map because the legend is assuming that a continous scale has been discretized into a series of colors. To make the image.plot function as a cloropleth 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 than 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.Rd` and `as.image` to discretise irregular observations to a grid. If the values makes 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 to 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 polyimage (see the `border` argument above) option to draw in boundaries.

Grids with unequally spacing – quadrialteral pixels: If x and y are matrices that are a smooth transformation of a regular grid then z[i,j] can be interperted 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 it 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 a illustration of this application. One can add border colors in this case easily because these choices are just passed onto 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` here 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 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 matter.

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 and so if this what you use this can be omitted. Unfortunately this is not the default for the `image` function. 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 cloropleith 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) leaves 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 reset.graphics=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. levelplot that is part of the lattice package has a very similar function to image.plot and a formula syntax in the call.

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 legends.

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

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,"+")
```

```
  image.plot(x,y,z)

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

# add some points on diagonal using standard plot function
#(with some clipping beyond 10 anticipated)
  points( 5:12, 5:12, pch="X", cex=3)

# 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  at break 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.
# Two 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 a 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.
# The idea is leave some room in the margin and then over plot 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

#
# Here is a more learned strategy to add a common legend to a panel of
# plots  consult the split.screen help file for more explanations.
# For this example we draw two
# images top and bottom and add a single legend color bar on the right side

# first divide screen into the figure region (left) and legend region (right)
   split.screen( rbind(c(0, .8,0,1), c(.8,1,0,1)))

# now subdivide up the figure region into two parts
   split.screen(c(2,1), screen=1)-> ind
   zr<- range( 2,35)
# first image
   screen( ind[1])
   image( x,y,z, col=tim.colors(), zlim=zr)
```

```
# second image
   screen( ind[2])
   image( x,y,z+10, col=tim.colors(), zlim =zr)

# move to skinny region on right and draw the legend strip
   screen( 2)
   image.plot( zlim=zr,legend.only=TRUE, smallplot=c(.1,.2, .3,.7),
   col=tim.colors())

   close.screen( all=TRUE)


# you can always add a legend arbitrarily to any plot;
# note that here the plot is too big for the vertical strip but the
# horizontal fits nicely.
plot( 1:10, 1:10)
image.plot( zlim=c(0,25), legend.only=TRUE)
image.plot( zlim=c(0,25), legend.only=TRUE, horizontal =TRUE)

# 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)
#
#
# sorting out the difference in formatting between matrix storage
# and the image plot depiction
# this really has not much to do with image.plot but I hope it is useful

A<- matrix( 1:48, ncol=6, nrow=8)
# first column of A will be 1:8
#   ...  second  is  9:16

image.plot(1:8, 1:6, A)
# add labels to each box
text( c( row(A)), c( col(A)), A)
# and the indices ...
text( c( row(A)), c( col(A))-.25,
   paste( "(", c(row(A)), ",",c(col(A)),")", sep=""), col="grey")

# "columns" of A are horizontal and rows are ordered from bottom to top!
#
# matrix in its usual tabular form where the rows are y  and columns are x
image.plot( t( A[8:1,]), axes=FALSE)
```

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,
    theta = 1, grid = NULL, tol = 1e-08, xwidth = NULL, ywidth = NULL,
    weights = NULL,...)

setup.image.smooth(nrow = 64, ncol = 64, dx = 1, dy = 1,
                   kernel.function = double.exp,
                theta = 1, xwidth = nrow * dx, ywidth = ncol * dx, lambda=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 are equally spaced and define the rectangular. (see grid.list)
dx	Grid spacing in x direction
dy	Grid spacing in x 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
theta	the bandwidth or scale parameter.
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 doe snot have to be a desity 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)
#
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, theta= .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 ofthe image.

look<- image.smooth(out$z, dx=dx, dy=dy, theta= .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,
             theta=.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, theta=.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, theta=1.5)
look2<- image.smooth( test.image , xwidth=0,
                        ywidth=2.5, dx=dx,dy=dy, theta=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 function help in subsetting images or reducing its size by averaging adjecent 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 specifing the 2-d grid values. (See help(grid.list) for more details.)
inflation	A small expansion factor to insure 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 taking using the largest rectangle in the locations loc. This is useful if one needs to extract a image that is no bigger in extant than som edata location. If locations are omitted the parent image is plotted and the locations 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 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 maximun and value of the maximum in the image based on the grid values and also on the indicies of the image matrix.

average.image, half.image Takes 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 begs the question 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
```

`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)
```

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
`loc`	A matrix of (irregular) locations to interpolate. First column of loc isthe 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.

Details

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 an y. That is locx is between x1 and x2 and locy is between y1 and y2. Let ex= (l1-x1)/(x2-x1) ey= (l2-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 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 irrgeular locations. Many thanks to Jean-Olivier Irisson for making this code more efficient and concise.

Value

An vector of interpolated values. NA are returned for regions of the obj\$z that are NA and also for locations outside of the range of the parent grid.

See Also

image.smooth, as.surface, as.image, image.plot, krig.image,Tps

Examples

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

data( lennon)
# create an example in the right list format like image or contour
obj<- list( x= 1:20, y=1:20, z= lennon[ 201:220, 201:220])

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

# take a look
set.panel(2,2)
image.plot( obj)
image.plot( look)
```

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 processes. The assumed model is additive Y = P(x) + Z(X) + e, where P is a low order polynomial and Z 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, Z = NULL, cost = 1, knots = NA,
                 weights = NULL, m = 2, nstep.cv = 200, scale.type =
                 "user", x.center = rep(0, ncol(x)), x.scale = rep(1,
                 ncol(x)), rho = NA, sigma2 = NA, method = "REML",
                 verbose = FALSE, mean.obj = NA, sd.obj = NA,
                 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, ...)

## 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 choice 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 Kriging estimate is found under a fixed lambda model.
give.warnings	If TRUE warnings are given in gcv grid search limits. If FALSE warnings are not given. Best to leave this TRUE! This argument is set ot FALSE if warn is less than zero in the top level, R options function. See options()$warn
knots	A matrix of locations similar to x. These can define an alternative set of basis functions for representing the estimate. One choice may be a space-filling subset of the original x locations, thinning out the design where locations cluster. The default is to put a "knot" at all unique locations. (See details.)
lambda	Smoothing parameter that is the ratio of the error variance (sigma**2) to the scale parameter of the covariance function (rho). 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 used in 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 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 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.
rho	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 usuall 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" 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.
sigma2	Variance of the errors, often called the nugget variance. If weights are specified then the error variance is sigma2 divided by weights. Note that lambda is defined as the ratio sigma2/rho.
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. vector of unit weights.
wght.function	An R function that creates a weights matrix to the observations. This is only needed if the weight matirx 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 the locations for spatial data or the indepedent 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.
Z	A vector of matrix of covariates to be include in the fixed part of the model. If NULL (default) no addtional covariates are included.
...	Optional arguments that appear are assumed to be additional arguments to the covariance function. Or are included in methods functions (resid, fitted, coef) as a required argument.
object	A Krig object

Details

This function produces a 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.

The Kriging model is: Y.k= f(x.k) = P(x.k) + Z(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, Z 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) + Z(x) given the observed data. For this estimate K, is taken to be rho*cov.function and the errors have variance sigma**2. In more conventional geostatistical terms rho is the "sill" if the covariance function is actually a correlation function and sigma**2 is the nugget variance or measure error variance (the two are confounded in this model.) If the weights are given then the variance of e.k is sigma**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 sigma**2 Wi, where Wi is the inverse of W. It is straightforward to show that the estimate of f only depends on sigma and rho through the ratio lambda = sigma**2/ rho. 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 = Zb + F + E where Z 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 The Z argument in the function is the way to specify this additional component.

If the parameters rho and sigma2 are omitted in the call, then they are estimated in the following way. If lambda is given, then sigma2 is estimated from the residual sum of squares divided by the degrees of freedom associated with the residuals. Rho is found as the difference between the sums of squares of the predicted values having subtracted off the polynomial part and sigma2. These estimates are the MLE's under Gaussian assumptions on the process and errors. If lambda is also omitted is it estimated from the data using a variety of approaches and then the values for sigma and rho 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 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 what the old tps function used in FUNFITS.

rmse: Match the estimate of sigma**2 to a external value (called rmse)

pure error: Match the estimate of sigma**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 the 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 a nonlinear parameter(s) that often 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

A 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 sigma conditional on the vlaue of lambda.
`lambda.est`	A table of estimated smoothing parameters with corresponding degrees of freedom and estimates of sigma 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 sigma**2/rho. 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.
shat	Estimated standard deviation of the measurement error (nugget effect).
sigma2	Estimated variance of the measurement error (shat**2).
rho	Scale factor for covariance. COV(h(x),h(x)) = rho*cov.function(x,x) If the covariance is actually a correlation function then rho is also the "sill".
mean.var	Normalization of the covariance function used to find rho.
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)

fit <- Krig(ChicagoO3$x, ChicagoO3$y, theta=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", theta=10, smoothness=1.0)
summary( fit)

#
# Roll your own: creating very simple user defined Gaussian covariance
#

test.cov <- function(x1,x2,theta,marginal=FALSE,C=NA){
   # return marginal variance
     if( marginal) { return(rep( 1, nrow( x1)))}

    # find cross covariance matrix
      temp<- exp(-(rdist(x1,x2)/theta)**2)
      if( is.na(C[1])){
          return( temp)}
      else{
          return( temp%*%C)}
      }
#
# use this and put in quadratic polynomial fixed function
```

```
 fit.flame<- Krig(flame$x, flame$y, cov.function="test.cov", m=3, theta=.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)
#

# A Knot example

 data(ozone2)
 y16<- ozone2$y[16,]

# there are some missing values -- remove them
 good<- !is.na( y16)
 y<- y16[good]
 x<- ozone2$lon.lat[ good,]

#
# the knots can be arbitrary but just for fun find them with a space
# filling design. Here we select  50 from the full set of 147 points
#
 xknots<- cover.design( x, 50, num.nn= 75)$design  # select 50 knot points

 out<- Krig( x, y, knots=xknots,  cov.function="Exp.cov", theta=300)
 summary( out)
# note that that trA found by GCV is around 17 so 50>17  knots may be a
# reasonable approximation to the full estimator.
#
## Not run:
```

```
# the plot
 surface( out, type="C")
 US( add=TRUE)
 points( x, col=2)
 points( xknots, cex=2, pch="O")

## End(Not run)
## Not run:
## A quick way to deal with too much data if you intend to smooth it anyway
##  Discretize the locations to a grid, then apply Krig
##  to the discretized locations:
##
RM.approx<- as.image(RMprecip$y, x=RMprecip$x, nx=20, ny=20)

# take a look:
image.plot( RM.approx)
# discretized data (observations averaged if in the same grid box)
# 336 locations -- down form the  full 806

# convert the image format to locations, obs and weight vectors
yd<- RM.approx$z[RM.approx$ind]
weights<- RM.approx$weights[RM.approx$ind] # takes into account averaging
xd<- RM.approx$xd

obj<- Krig( xd, yd, weights=weights, theta=4)

# compare to the full fit:
# Krig( RMprecip$x, RMprecip$y, theta=4)

## End(Not run)

## Not run:
# A correlation model example
# fit krig surface using a mean and sd function to standardize
# first get stats from 1987 summer Midwest O3 data set
 data(ozone2)
 stats.o3<- stats( ozone2$y)
 mean.o3<- Tps( ozone2$lon.lat, c( stats.o3[2,]))
 sd.o3<- Tps(  ozone2$lon.lat, c( stats.o3[3,]))

#
# Now use these to fit particular day ( day 16)
# and use great circle distance


 fit<- Krig( ozone2$lon.lat, ozone2$y[16,],
            theta=350, mean.obj=mean.o3, sd.obj=sd.o3,
            Covariance="Matern", Distance="rdist.earth",
            smoothness=1.0,
            na.rm=TRUE) #


# the finale
```

```
 surface( fit, type="I")
 US( add=TRUE)
 points( fit$x)
 title("Estimated ozone surface")

## End(Not run)
## Not run:
#
#
# explore some different values for the range and lambda using REML
  theta <- seq( 100,500,,40)
  PLL<- 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!)
    PLL[k,]<- Krig( ozone2$lon.lat,ozone2$y[16,],
              cov.function="stationary.cov",
              theta=theta[k], mean.obj=mean.o3, sd.obj=sd.o3,
              Covariance="Matern",smoothness=.5,
              Distance="rdist.earth", nstep.cv=80,
              give.warnings=FALSE, na.rm=TRUE)$gcv.grid[,7]
#
# gcv.grid is the grid search output from
# the optimization for estimating different estimates for lambda including
# REML
# default grid is equally spaced in eff.df scale ( and should the same across theta)
#  here
  }
# get lambda grid  from looping
   k<- 1
   lam<-  Krig( ozone2$lon.lat,ozone2$y[16,],
              cov.function="stationary.cov",
              theta=theta[k], mean.obj=mean.o3, sd.obj=sd.o3,
              Covariance="Matern",smoothness=.5,
              Distance="rdist.earth", nstep.cv=80,
              give.warnings=FALSE, na.rm=TRUE)$gcv.grid[,1]
# see the 2 column of $gcv.grid to get the effective degress of freedom.
  contour( theta,log(lam) , PLL)


## 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 a 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.
	A Krig object produced by the Krig (or Tps) function.
x0ject	Locations for prediction 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 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 invovles loop 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
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 Z matrix)

Usage

```
Krig.null.function(x, Z = NULL, drop.Z = FALSE, m)
```

Arguments

x	Spatial locations
Z	Other covariates to be associated with each location.
drop.Z	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, Z and drop.Z 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 Z.

Author(s)

Doug Nychka

See Also

Krig

`Krig.replicates` *Collapse repeated spatial locations into unique locations*

Description

In case that several observations are available for a single spatial location find the group means and replicate variability

Usage

```
Krig.replicates(out, x, y, Z, weights=rep( 1, length(y)), verbose = FALSE)
```

Arguments

`out`	A list with components `x`, `y`, `weights`, and possibily `Z`.
`x`	Spatial locations.
`y`	Spatial observations
`Z`	Spatial covariates.
`weights`	Weights proportional to reciprocal varainces of observations.
`verbose`	If TRUE print out details for debugging.

Details

This function figures out which locations are the same and within the function fast.1way use `tapply` to find replicate group means and standard deviations. NOTE: it is assumed the Z covariates are unique at the locations. Currently these functions can not handle a model with common spatial locations but different values for the Z covariates.

Value

A list with components:

`yM`	Data at unique locations and where more than one observation is available this is the mean of the replicates.
`xM`	Unique spatial locations.
`weightsM`	Weights matching the unique lcoations proportional to reciprocal variances This is found as a combination of the original weights at each location.
`ZM`	Values of the covariates at the unique lcoations.

uniquerows	Index for unique rows of x.
shat.rep, shat.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])
```

---

lennon *Gray image of John Lennon.*

Description

A 256X256 image of John Lennon. Try:

```
image(lennon,col=grey(seq(0,1,,256)) )
```

---

minitri *Mini triathlon results*

Description

Results from a mini triathlon sponsored by Bud Lite, 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)

<s-section name= "DATA DESCRIPTION"> This is a dataframe. Row names are the place within this age group based on total time.

Arguments

swim	swim times
bike	bike times
run	run times

---

mKrig *"micro Krig" Spatial process estimate of a curve or surface, "kriging" with a known covariance function.*

Description

This is a simple version of the Krig function 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 higher level functions for maximum likelihood estimates of covariance paramters.

Usage

```
mKrig(x, y, weights = rep(1, nrow(x)), Z = NULL,
                  cov.function = "stationary.cov", cov.args = NULL,
                  lambda = 0, m = 2, chol.args = NULL, find.trA = TRUE,
                  NtrA = 20, iseed = 123, llambda = NULL, na.rm = FALSE,
                   collapseFixedEffect = TRUE,
                  ...)

## S3 method for class 'mKrig'
predict( object, xnew=NULL,ynew=NULL, grid.list = NULL,
derivative=0,
Z=NULL,drop.Z=FALSE,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, Z, 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.Z	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.
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.
just.fixed	If TRUE only the predictions for the fixed part of the model will be evaluted.
lambda	Smoothing parameter or equivalently the ratio between the nugget and process varainces.
llambda	If not NULL then lambda = exp(llambda)
m	The degree of the polynomial used in teh fixed part is (m-1)
na.rm	If TRUE NAs in y are omitted along with corresonding 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.)
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 measurment error model but independent from each other.

ynew	New observation vector. mKrig will reuse matrix decompositions and find the new fit to these data.
Z	Linear covariates to be included in fixed part of the model that are distinct from the default low order polynomial in x. (NOTE the order of the polynomial determined by m)
...	In mKrig and predict additional arguments that will be passed to the covariance function.

Details

This function is an abridged version of Krig. The m stand for micro and this function focuses on the computations in Krig.engine.fixed done for a fixed lambda parameter, for unique spatial locations and for data without missing values.

These restrictions simplify the code for reading. Note that also little checking is done and the spatial locations are not transformed before the estimation. 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. 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 thetrace of the smoothing matrix is (trace A(lambda)) is computationally expensive and a Monte Carlo approximation is supplied instead.

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 the combines the summary and print methods.

mKrig.coef finds the "d" and "c" coefficients represent the solution using the previous cholesky decomposition for a new data vector. This is used in computing the prediction standard error in predictSE.mKrig and can also be used to evalute the estimate efficiently at new vectors of observations provided the locations and covariance remain fixed.

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 postive definite linear system used to solve for the basis coefficients. The sparse of this system is essentially the same as the covariance matrix evalauted at the observed locations. As an example of resetting nzR to 450000 one would use the following argument for chol.args in mKrig:

```
chol.args=list(pivot=TRUE,memory=list(nnzR= 450000))
```

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 becomse 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

`d`	Coefficients of the polynomial fixed part and if present the covariates (Z).To determine which is which the logical vector ind.drift also part of this object is TRUE for the polynomial part.
`c`	Coefficients of the nonparametric part.
`nt`	Dimension of fixed part.
`np`	Dimension of c.
`nZ`	Number of columns of Z covariate matrix (can be zero).
`ind.drift`	Logical vector that indicates polynomial coefficients in the `d` coefficients vector. This is helpful to distguish between polynomial part and the extra covariates coefficients associated with Z.
`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.
`shat.MLE`	MLE of sigma.
`rho.MLE`	MLE or rho.
`rhohat`	Estimate for rho adjusted for fixed model degrees of freedom (ala REML).
`lnProfileLike`	log Profile likelihood for lambda
`lnDetCov`	Log determinant of the covariance matrix for the observations having factored out rho.
`Omega`	GLS covariance for the estimated parameters in the fixed part of the model (d coefficients0.

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 ids 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(lamdba).
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/NCAR/Fields

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 maxmimum likelihood
  out<- mKrig( x,y, theta = 2.0, lambda=.01)
  out.p<- predictSurface( out)
  surface( out.p)
#
# NOTE this should be identical to
# Krig( x,y, theta=2.0, lambda=.01)

##############################################################################
# an example using a "Z" 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,]
```

```
Z<- CO.elev[good]
out<- mKrig( xCO,yCO, Z=Z, cov.function="stationary.cov", Covariance="Matern",
                    theta=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()

#############################################################################
# Interpolate using tapered version of the exponential,
# the taper scale is set to 1.5 default taper covariance is the Wendland.
# Tapering will done at a scale of 1.5 relative to the scaling
# done through the theta  passed to the covariance function.
data( ozone2)
  y<- ozone2$y[16,]
  good<- !is.na( y)
  y<-y[good]
  x<- ozone2$lon.lat[good,]
  mKrig( x,y,cov.function="stationary.taper.cov",
       theta = 2.0, lambda=.01,
       Taper="Wendland",  Taper.args=list(theta = 1.5, k=2, dimension=2)
           ) -> out2

# Try out GCV on a grid of lambda's.
# For this small data set
# one should really just use Krig or Tps but this is an example of
# approximate GCV that will work for much larger data sets using sparse
# covariances and the Monte Carlo trace estimate
#
# a grid of lambdas:
  lgrid<- 10**seq(-1,1,,15)
  GCV<- matrix( NA, 15,20)
  trA<-  matrix( NA, 15,20)
  GCV.est<- rep( NA, 15)
  eff.df<- rep( NA, 15)
  logPL<- rep( NA, 15)
# loop over lambda's
  for(  k in 1:15){
      out<- mKrig( x,y,cov.function="stationary.taper.cov",
                    theta = 2.0, lambda=lgrid[k],
          Taper="Wendland",  Taper.args=list(theta = 1.5, k=2, dimension=2)  )
      GCV[k,]<- out$GCV.info
      trA[k,]<- out$trA.info
      eff.df[k]<- out$eff.df
      GCV.est[k]<- out$GCV
      logPL[k]<- out$lnProfileLike
  }
#
# plot the results different curves are for individual estimates
# the two lines are whether one averages first the traces or the GCV criterion.
#
```

```
  par( mar=c(5,4,4,6))
  matplot( trA, GCV, type="l", col=1, lty=2,
           xlab="effective degrees of freedom", ylab="GCV")
  lines( eff.df, GCV.est, lwd=2, col=2)
  lines( eff.df, rowMeans(GCV), lwd=2)
# add exact GCV computed by Krig
  out0<-  Krig( x,y,cov.function="stationary.taper.cov",
              theta = 2.0,
              Taper="Wendland", Taper.args=list(theta = 1.5, k=2, dimension=2),
              spam.format=FALSE)
  lines( out0$gcv.grid[,2:3], lwd=4, col="darkgreen")

# add profile likelihood
  utemp<- par()$usr
  utemp[3:4] <- range( -logPL)
  par( usr=utemp)
  lines( eff.df, -logPL, lwd=2, col="blue", lty=2)
  axis( 4)
  mtext( side=4,line=3, "-ln profile likelihood", col="blue")
  title( "GCV ( green = exact) and  -ln profile likelihood", cex=2)

##########################################################################
# 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, theta=.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 theta
# 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 theta=.9 because
# allocation for the  covariance matirx storage is too small.
# Here theta 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, theta=.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 becuase 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,
                   theta=.9, lambda=1e-2)
# as a check notice that
#   print( look2)
# reports the number of nonzero elements consistent with the specifc 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, theta=.1, lambda=1e2  )

## End(Not run)
# to avoid the warning
 look2<-mKrig( x,y,
            cov.function="wendland.cov", k=2, theta=.1,
            lambda=1e2, chol.args=list(pivot=TRUE, memory=list(nnzR= 450000)))

###############################################################################
# fiting 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, theta=.1,
            lambda=1e2  )
# note slight difference in summary because two data sets have been fit.
  print( look3)
#}

#################################################################
# finding a good choice for theta 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 trick to find the number of nonsero 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.
## Not run:
  mKrig( x,y, cov.function="wendland.cov",k=2, theta=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", theta=.1,
            lambda=1e2, distMat=distMat )
  #precompute distance matrix in standard form
  distMat = rdist(x)
  look3<- mKrig( x,Y,cov.function="stationary.cov", theta=.1,
            lambda=1e2, distMat=distMat )

## End(Not run)
```

mKrig.MLE *Maximizes likelihood for the process marginal variance (rho) and nugget standard deviation (sigma) parameters (e.g. lambda) over a 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 sigma and rho. They are depreciated and my be omitted in later versions of fields with their roles being replaced by other functions. See details below.

Usage

```
mKrig.MLE(x, y, weights = rep(1, nrow(x)), cov.fun="stationary.cov",
cov.args = NULL,
          Z = NULL, par.grid = NULL, lambda = NULL, lambda.profile = TRUE,
          verbose = FALSE, relative.tolerance = 1e-04, ...)

mKrig.MLE.joint(x, y, weights = rep(1, nrow(x)),
                lambda.guess = 1, cov.params.guess=NULL,
                cov.fun="stationary.cov", cov.args=NULL,
                Z = NULL, optim.args=NULL, find.trA.MLE = FALSE,
                ..., verbose = FALSE)

fastTps.MLE(x, y, weights = rep(1, nrow(x)), Z = NULL, ...,
                 par.grid=NULL, theta, lambda = NULL, lambda.profile = TRUE,
                 verbose = FALSE, relative.tolerance = 1e-04)
```

Arguments

cov.args	Additional arguments that would also be included in calls to the covariance function to specify the fixed part of the covariance model.
cov.fun	The name, a text string, of the covariance function.
cov.params.guess	A list of initial guesses for covariance parameters over which the user wishes to perform likelihood maximization. The list contains the names of the parameters as well as the values.
find.trA.MLE	If TRUE will estimate the effective degrees of freedom using a simple Monte Carlo method throughout joint likelihood maximization. Either way, the trace of the mKrig object with the best log-likelihood is calculated depending on find.trA. Computing the trace will add to the computational burden by approximately NtrA solutions of the linear system but the cholesky decomposition is reused.
lambda	If lambda.profile=FALSE the values of lambda to evaluate the likelihood if "TRUE" the starting values for the optimization. If lambda is NA then the optimum value from previous search is used as the starting value. If lambda is NA and it is the first value the starting value defaults to 1.0.

lambda.guess	The initial guess for lambda in the joint log-likelihood maximization process.
lambda.profile	If TRUE maximize likelihood over lambda.
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(Note that the first parameter is lambda and the others are the covariance parameters in the order they are given in cov.params.guess. Also note that the optimization is performed on a log-scale, and this should be taken into consideration when passing arguments to optim.
par.grid	A list or data frame with components being parameters for different covariance models. A typical component is "theta" 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 ....
relative.tolerance	Relative tolerance used to declare convergence when maximizing likelihood over lambda.
theta	Range parameter for compact Wendland covariance. (seefastTps)
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 observations vectors generated from the same covariance and measurment error model.
Z	Linear covariates to be included in fixed part of the model that are distinct from the default low order polynomial in x
...	Additional arguments that would also be included in a call to mKrig to specify the covariance model and fixed model covariables.

Details

The "mKrig" prefixed functions are depreciated and are replaced in functionality by mKrigMLEJoint and mKrigMLEGrid.

The observational model follows 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 (sigma) and the marginal variance of the process (rho). It is useful to reparametrize as rho and\ lambda= sigma^2/rho. The likelihood can be maximized analytically over rho and the parameters in the fixed part of the model the estimate of rho can be substituted back into the likelihood to give a expression that is just a function of lambda and the remaining covariance parameters. It is this expression that is then maximized numerically over lambda when lambda.profile = TRUE.

Note that fastTps.MLE is a convenient variant of this more general version to use directly with fastTps, and mKrig.MLE.joint is similar to mKrig.MLE, except it uses the optim function to optimize over the specified covariance parameters and lambda jointly rather than optimizing on a grid. Unlike mKrig.MLE, it returns an mKrig object.

Value

mKrig.MLE returns a list with the components:

summary	A matrix giving the results for evaluating the likelihood for each covariance model.
par.grid	The par.grid argument used.
cov.args.MLE	The list of covariance arguments (except for lambda) that have the largest likelihood over the list covariance models. To fit the surface at the largest likelihood among those tried `do.call( "mKrig",c(obj$mKrig.args,obj$cov.args.MLE,list(lambda=obj$lambda.opt)) )` where obj is the list returned by this function.
call	The calling arguments to this function.

mKrig.MLE.joint returns an mKrig object with the best computed log-likelihood computed in the maximization process with the addition of the summary table for the mKrig object log-likelihood and:

lnLike.eval	A table containing information on all likelihood evaluations performed in the maximization process.

Author(s)

Douglas W. Nychka, John Paige

References

https://github.com/NCAR/Fields

See Also

mKrig Krig stationary.cov optim

Examples

```
# some synthetic data
  N<- 100
  set.seed(123)
  x<- matrix(runif(2*N), N,2)
  theta<- .2
  Sigma<-  Matern( rdist(x,x)/theta , smoothness=1.0)
  Sigma.5<- chol( Sigma)
  sigma<- .1
  M<-5 #  Five (5) independent spatial data sets
  F.true<- t( Sigma.5)%*% matrix( rnorm(N*M), N,M)
  Y<-  F.true +  sigma* matrix( rnorm(N*M), N,M)
# find MLE for lambda with range and smoothness fixed in Matern for first
# data set
  obj<- mKrig.MLE( x,Y[,1], Covariance="Matern", theta=.2, smoothness=1.0)
  obj$summary # take a look
  fit<- mKrig( x,Y[,1], Covariance="Matern", theta=.2,
```

```
                                    smoothness=1.0, lambda= obj$lambda.best)
#
# search over the range parameter and use all 5 replications for combined
# likelihood
## Not run:
  par.grid<- list( theta= seq(.1,.25,,6))
# default starting value for lambda is .02 subsequent ones use previous optimum.
  obj<- mKrig.MLE( x,Y, Covariance="Matern",lambda=c(.02,rep(NA,4)),
                                   smoothness=1.0, par.grid=par.grid)

## End(Not run)

#perform joint likelihood maximization over lambda and theta.
#optim can get a bad answer with poor initial guesses.
set.seed(123)
obj<- mKrig.MLE.joint(x,Y[,1],
                      cov.args=list(Covariance="Matern", smoothness=1.0),
                      cov.params.guess=list(theta=.2), lambda.guess=.1)

#look at lnLik evaluations
obj$lnLik.eval

## Not run:
#perform joint likelihood maximization over lambda, theta, and smoothness.
#optim can get a bad answer with poor initial guesses.
set.seed(123)
obj<- mKrig.MLE.joint(x,Y[,1],
                      cov.args=list(Covariance="Matern"),
                      cov.params.guess=list(theta=.2, smoothness=1), lambda.guess=.1)

#look at lnLik evaluations
obj$lnLik.eval

#generate surface plot of results of joint likelihood maximization
#NOTE: mKrig.MLE.joint returns mKrig object while mKrig.MLE doesn't,
#so this won't work for mKrig.MLE.
surface(obj)

## End(Not run)
```

`mKrigMLE` *Maximizes likelihood for the process marginal variance (rho) and nugget standard deviation (sigma) parameters (e.g. lambda) over a many covariance models or covariance parameter values.*

Description

These function are designed to explore the likelihood surface for different covariance parameters with the option of maximizing over sigma and rho. They used the `mKrig` base are designed for computational efficiency.

Usage

```
mKrigMLEGrid(x, y, weights = rep(1, nrow(x)), Z = NULL, mKrig.args = NULL,
    cov.fun = "stationary.cov", cov.args = NULL, na.rm = TRUE,
    par.grid = NULL, lambda = NULL, lambda.profile = TRUE,
    relative.tolerance = 1e-04,
    REML = FALSE, verbose = FALSE)

mKrigMLEJoint(x, y, weights = rep(1, nrow(x)), Z = NULL, mKrig.args
                = NULL, na.rm = TRUE, cov.fun = "stationary.cov",
                cov.args = NULL, lambda.start = 0.5, cov.params.start
                = NULL, optim.args = NULL, abstol = 1e-04,
                parTransform = NULL, REML = FALSE, verbose = FALSE)


fastTpsMLE(x, y, weights = rep(1, nrow(x)), Z = NULL, ...,
                par.grid=NULL, theta, lambda = NULL, lambda.profile = TRUE,
                verbose = FALSE, relative.tolerance = 1e-04)

mKrigJointTemp.fn(parameters, mKrig.args, cov.args, parTransform,
           parNames, REML = FALSE, capture.env)
```

Arguments

abstol	Absolute convergence tolerance used in optim.
capture.env	For the ML obective function the frame to save the results of the evaluation. This should be the environment of the function calling optim.
cov.fun	The name, a text string, of the covariance function.
cov.args	Additional arguments that would also be included in calls to the covariance function to specify the fixed part of the covariance model.
cov.params.start	A list of initial starts for covariance parameters over which the user wishes to perform likelihood maximization. The list contains the names of the parameters as well as the values.
lambda	If lambda.profile=FALSE the values of lambda to evaluate the likelihood if "TRUE" the starting values for the optimization. If lambda is NA then the optimum value from previous search is used as the starting value. If lambda is NA and it is the first value the starting value defaults to 1.0.
lambda.start	The initial guess for lambda in the joint log-likelihood maximization process
lambda.profile	If TRUE maximize likelihood over lambda.
mKrig.args	A list of additional parameters to supply to the base 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).
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(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 for evaluate the likelihood.

par.grid — A list or data frame with components being parameters for different covariance models. A typical component is "theta" 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 ....

parNames — Names of the parameters to optimize over.

parTransform — A function that maps the parameters to a scale for optimization or effects the inverse map from the transformed scale into the original values. See below for more details.

relative.tolerance — Tolerance used to declare convergence when maximizing likelihood over lambda.

REML — Currently using REML is not implemented.

theta — Range parameter for compact Wendland covariance. (see fastTps)

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 observations vectors generated from the same covariance and measurment error model.

Z — Linear covariates to be included in fixed part of the model that are distinct from the default low order polynomial in x

... — Other arguments to pass to the mKrig function.

Details

The observational model follows 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 (sigma) and the marginal variance of the process (rho). It is useful to reparametrize as rho and\ lambda= sigma^2/rho. The likelihood can be maximized analytically over rho and the parameters in the fixed part of the model the estimate of rho can be substituted back into the likelihood to give a expression that is just a function of lambda and the remaining covariance parameters. It is this expression that is then maximized numerically over lambda when lambda.profile = TRUE.

Note that fastTpsMLE is a convenient variant of this more general version to use directly with fastTps, and mKrigMLEJoint is similar to mKrigMLEGrid, except it uses the optim function to

optimize over the specified covariance parameters and lambda jointly rather than optimizing on a grid. Unlike mKrigMLEJoint, it returns an mKrig object.

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 giving the results for evaluating the likelihood for each covariance model.
par.grid	The par.grid argument used.
cov.args.MLE	The list of covariance arguments (except for lambda) that have the largest likelihood over the list covariance models. NOTE: To fit the surface at the largest likelihood among those tried do.call("mKrig",c(obj$mKrig.args,obj$cov.args.MLE,list(lambda=o) where obj is the list returned by this function.
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.
par.MLE	The maximum likelihood estimates.
parTransform	The transformation of the parameters used in the optimziation.

Author(s)

Douglas W. Nychka, John Paige

References

https://github.com/NCAR/Fields

See Also

mKrig Krig stationary.cov optim

Examples

```
#perform joint likelihood maximization over lambda and theta.
#optim can get a bad answer with poor initial starts.
data(ozone2)
x<- ozone2$lon.lat
y<- ozone2$y[16,]
obj<- mKrigMLEJoint(x,y,
                     cov.args=list(Covariance="Matern", smoothness=1.0),
                     cov.params.start=list(theta=.2), lambda.start=.1)
#
# check  lnLikeihood evaluations that were culled from optim
# these are in obj$lnLike.eval
# funny ranges are set to avoid  very low likelihood values

quilt.plot( log10(cbind(obj$lnLike.eval[,1:2])), obj$lnLike.eval[,5],
            xlim=c(-1.2,-.40), ylim=c( -1,1), zlim=c( -625, -610))
            points( log10(obj$pars.MLE[1]), log10(obj$pars.MLE[2]),
     pch=16, col="grey" )

# some synthetic data with replicates
  N<- 50
  set.seed(123)
  x<- matrix(runif(2*N), N,2)
  theta<- .2
  Sigma<-  Matern( rdist(x,x)/theta , smoothness=1.0)
  Sigma.5<- chol( Sigma)
  sigma<- .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)
  F.true<- t( Sigma.5)%*% matrix( rnorm(N*M), N,M)
  Y<-  F.true +  sigma* matrix( rnorm(N*M), N,M)

# find MLE for lambda with grid of ranges
# and smoothness fixed in Matern
 par.grid<- list( theta= seq( .1,.35,,8))
  obj1b<- mKrigMLEGrid( x,Y,
     cov.args = list(Covariance="Matern", smoothness=1.0),
        par.grid = par.grid
                   )
  obj$summary # take a look
# profile over theta
  plot( par.grid$theta, obj1b$summary[,"lnProfileLike.FULL"],
    type="b", log="x")

  ## Not run:
# m=0 is a simple switch to indicate _no_ fixed spatial drift
# (the default and highly recommended  is linear drift, m=2).
# this results in MLEs that are less biased -- in fact it nails it !
  obj1a<- mKrigMLEJoint(x,Y,
```

```
                    cov.args=list(Covariance="Matern", smoothness=1.0),
                    cov.params.start=list(theta=.5), lambda.start=.5,
                     mKrig.args= list( m=0))

 test.for.zero( obj1a$summary["sigmaMLE"], sigma, tol=.0075)
 test.for.zero( obj1a$summary["theta"], theta, tol=.05)

## End(Not run)

## Not run:
#perform joint likelihood maximization over lambda, theta, and smoothness.
#note: finding smoothness is not robust optim
#      can get a bad answer with poor initial guesses.
obj2<- mKrigMLEJoint(x,Y,
                     cov.args=list(Covariance="Matern"),
                     cov.params.start=list(theta=.18, smoothness=1.1),
                      lambda.start=.08)

#look at lnLikelihood  evaluations
obj2$summary
#compare to REML
obj3<- mKrigMLEJoint(x,Y,
                     cov.args=list(Covariance="Matern"),
                     cov.params.start=list(theta=.18, smoothness=1.1),
                       lambda.start=.08
                      , REML=TRUE)

## End(Not run)
## Not run:
#look at lnLikelihood  evaluations
obj3$summary
# check convergence of MLE to true fit with no fixed part
#
obj4<- mKrigMLEJoint(x,Y,
                     mKrig.args= list( m=0),
                     cov.args=list(Covariance="Matern", smoothness=1),
                     cov.params.start=list(theta=.2),
                      lambda.start=.1, REML=TRUE)
#look at lnLikelihood  evaluations
obj4$summary
# nails it!

## End(Not run)
```

MLESpatialProcess *Estimates key covariance parameters for a spatial process.*

Description

Maximizes the likelihood to determine the nugget variance (sigma^2), the sill (rho) and the range (theta) for a spatial process.

Usage

```
MLESpatialProcess(x, y, weights = rep(1, nrow(x)), Z = NULL, mKrig.args
                  = NULL, cov.function = "stationary.cov", cov.args =
                  list(Covariance = "Matern", smoothness = 1),
                  lambda.start = 0.5, theta.start = NULL, theta.range =
                  NULL, gridN = 20, optim.args = NULL, na.rm = TRUE,
                  verbose = FALSE, abstol = 1e-04, REML = FALSE, ...)
```

Arguments

x	A matrix of spatial locations with rows indexing location and columns the dimension (e.g. longitude/latitude)
y	Spatial observations
weights	Precision (1/variance) of each observation
Z	Linear covariates to be included in fixed part of the model that are distinct from the default low order polynomial in x
mKrig.args	A list containing other objects to pass to mKrig.
lambda.start	The initial guess for lambda, the nugget to sill ratio.
theta.start	The initial guess for theta, the correlation range parameter.
theta.range	Range of range parameters (aka theta) to search over. Default is the range from the 2 and 97 percent quantiles of the pairwise distances among locations.
gridN	Number of points to use in grid search over theta.
cov.function	The name of the covariance function (See help on Krig for details.)
cov.args	A list with arguments for the covariance functions. These are usually parameters and other options such as the type of distance function.
optim.args	Additional arguments passed to the optim function for likelihood maximization. The default value is: `optim.args = list(method = "BFGS",control = list(fnscale = -1,parscale = c(0.5,0.5),ndeps = c(0.05,0.05)))`
na.rm	If TRUE remove missing values in y and corresponding locations in x.
verbose	If TRUE print out intermediate information for debugging.
abstol	Absolute tolerance used to judeg convergence in optim.
REML	If TRUE use maximize the restricted Likelihood instead of the concentrated likelihood.(Preliminary experience suggests this does not make much difference.)
...	Additional arguments to pass to the mKrig function.

Details

MLESpatialProcess is designed to be a simple and easy to use function for maximizing the likelihood for a Gaussian spatial process. For other fixed, covariance parameters, the likelihood is maximized over the nugget and sill parameters using the `mKrig` function. `lambda` and `theta` are optimized using the `mKrigMLEJoint` function on a log scale.

MLESpatialProcess.fast is an older fields function also using the optim function to maximize the likelihood computed from the mKrig function. It will eventually be removed from later versions of fields but is still useful as a cross check on newer functions

Note the likelihood can be maximized analytically over the parameters of the fixed part of the spatial model and with the nugget (sigma) and sill (rho) reduced to the single parameter lambda= sigma^2/rho. The likelihood is maximized numerically over lambda and theta if there are additional covariance parameters (such as smoothness for the Matern) these need to be fixed and so the MLE is found for the covariance conditional on these additional parameter values. From a practical point of view it is often difficult to estimate just these three from a moderate spatial data set and the user is encourage to try different combinations of fixing covariance parameters with ML for the remaining ones.

Value

MLESpatialProcess: A list that includes components: theta.MLE,rho.MLE,sigma.MLE,lambda.MLE being the maximum likelihood estimates of these parameters. The component REML.grid is a two column matrix with the first column being the theta grid and the second column being the profiled and restricted likelihood for that value of theta. Here profile means that the likelihood has already been evaluated at the maximum over sigma and rho for this value of theta. eval.grid is a more complete "capture" of the evaluations being a 6 column matrix with the parameters theta, lambda, sigma, rho, profile likelihood and the effective degrees of freedom.

MLESpatialProcess.fast has been depreciated and is included for backward compatibility.

Author(s)

Doug Nychka, John Paige

See Also

Krig, mKrigMLEGrid, mKrigMLEJoint, optim, fastTps.MLE, spatialProcess

Examples

```
#
#
#generate observation locations (100 is small just to make this run quickly)
n=100
set.seed(124)
x = matrix(runif(2*n), nrow=n)
#generate observations at the locations
trueTheta = .1
trueSigma = .01
Sigma = exp( -rdist(x,x) /trueTheta )
# y = t(chol(Sigma))%*% (rnorm(n))  +  trueSigma * rnorm( n)
y = t(chol(Sigma))%*% (rnorm(n))  +  trueSigma * rnorm( n)
# Use exponential covariance estimate constant function for mean
out = MLESpatialProcess(x, y,
                          smoothness=.5,
                        mKrig.args = list( m = 1)
                        )
```

```
# Use exponential covariance, use a range to determine MLE of range parameter
## Not run:
#Use Matern covariance, compute joint MLE of range, smoothness, and lambda.
#This may take a few seconds
testSmoothness = c(.5, 1, 2)
for( nu in testSmoothness){
  out = MLESpatialProcess(x, y, cov.args=list(Covariance="Matern"), smoothness=nu)
  print( out$MLEJoint$summary)
}


## End(Not run)

# example with a covariate
## Not run:
data(COmonthlyMet)
ind<- !is.na( CO.tmean.MAM.climate)
x<- CO.loc[ind,]
y<- CO.tmean.MAM.climate[ind]
elev<- CO.elev[ind]
obj2<-  MLESpatialProcess( x,y)
obj3<-  MLESpatialProcess( x,y, Z=elev)

# elevation makes a difference
obj2$MLEJoint$summary
obj3$MLEJoint$summary


## End(Not run)
 ## Not run:
# fits for first 10 days from ozone data
data( ozone2)
NDays<- 10
O3MLE<- matrix( NA, nrow= NDays, ncol=7)
for( day in 1: NDays){
cat( day, " ")
ind<- !is.na(ozone2$y[day,] )
x<- ozone2$lon.lat[ind,]
y<- ozone2$y[day,ind]
print( length( y))
O3MLE[day,]<- MLESpatialProcess( x,y,
            Distance="rdist.earth")$MLEJoint$summary
}
# NOTE: names of summary:
#[1] "lnProfileLike.FULL" "lambda"
#[3] "theta"              "sigmaMLE"
#[5] "rhoMLE"             "funEval"
#[7] "gradEval"
plot( log(O3MLE[,2]), log(O3MLE[,3]))

## End(Not run)
```

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 for the period 1950-2010. Data is based on 1720 stations located in North America.

Format

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 precipitaion in 1/10 mm (precip/254 converts to inches of rainfall)

`precip` is the intercept for 1980.5 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 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 North 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$y)
 NorthAmericanRainfall$projection<- .Last.projection
```

Use `NorthAmericanRainfall$orientation` to access the stereographic projection orientation.

Source

The monthly data used to construct this summary was generously provided by Xuebin Zhang, however, the orignal source is freely available as the Global Historical Climate Network Version 2 Precipitation quality controlled, curated and served by the US National Climatic Data Center (NCDC). The adjusted data from this archive has been modified from its raw form to make the record more homogenous. Heterogenities can come from a variety of sources such as a moving the station a short distance or changes in instruments. See https://www.ncdc.noaa.gov/data-accessand goto GHCN.

Examples

```
data(NorthAmericanRainfall)
x<- cbind(NorthAmericanRainfall$longitude,  NorthAmericanRainfall$latitude)
y<- NorthAmericanRainfall$precip
quilt.plot( x,y)
world( add=TRUE)
```

```
Zstat<- NorthAmericanRainfall$trend / NorthAmericanRainfall$trendSE
quilt.plot( x, Zstat)
```

---

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: <s-args> <s-arg name="y"> a 89X153 matrix of ozone values. Rows are days and columns are the sites. </s-arg> </s-arg name="lon.lat"> Site locations in longitude and latitude as a 153X2 table </s-arg> <s-arg name="chicago.subset"> Logical vector indicating stations that form teh smaller Chicagoland subset. (see FIELDS ozone data set) </s-arg> </s-args> <s-section name="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)
```

`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 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 sigma 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 are 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. GCV function is with respect to the standardized data if a correlation model is specified. A vertical line indicates the minimium found.

For `Krig` and `sreg` objects the fourth plot is a histogram of the standardized residuals. For sreg if multiple lambdas are given plotted are boxplots of the residuals for each fit.

For `spatialProcess` object the fourth plot is the profile likelihood for the theta 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, theta=200)
# fitting a surface to ozone
# measurements
set.panel( 2,2)
plot(fit1)

fit2<-spatialProcess(x,y)
# fitting a spatial process model to ozone
# measurements
# Although an example does not make too much sense for only 20 observations!
set.panel( 2,2)
plot(fit2)

# fit rat data
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 plots, perspective plots.

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 the 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 contours lines added. type="b" gives both "p" and "C" as a 2X1 panel
zlab	z-axes label
xlab	x-axes label
ylab	y-axes labels
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.persp or image.plot

See Also

surface, predictSurface, as.surface, drape.plot, image.plot

Examples

```
x<- seq( -2,2,,80)
y<- seq( -2,2,,80)
# a lazy way to create some test image
z<- outer( x,y, "+")

# create basic image/surface object
obj<- list(x=x, y=y,z=z)

# basic 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.
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 given 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,...)

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 cells 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 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.
...	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 ambigious 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 accomplisehd by finding the midpoints of x and y grid points and adding two outside rows and cols to complete the grid. The new result is a new grid that is 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 nieghboring 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, Z = NULL, drop.Z = FALSE, 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, Z = NULL, drop.Z = FALSE, just.fixed =
                 FALSE, xy = c(1, 2), ...)
```

Arguments

derivative	The degree of the derivative to be evauated. 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 marginal sd's and add the mean function. This 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 MLE's of rho and sigma2

grid.list	A grid.list specfiying a grid of locations to evaluate the fitted surface.
just.fixed	Only fixed part of model is evaluated
lambda	Smoothing parameter. If omitted, out\$lambda 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, i.e. out\$x 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)
Z	Vector/Matrix of additional covariates to be included in fixed part of spatial model
drop.Z	If TRUE only spatial fixed part of model is evaluated. i.e. Z covariates are not used.
...	Other arguments passed to 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 on the calculations depend on the observed locations not on lambda or the observed data. Note 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 can not handle the correlation model form.

Value

Vector of predicted responses or a matrix of the partial derivatives.

See Also

Krig, predictSurface gcv.Krig

Examples

```
  Krig(ChicagoO3$x,ChicagoO3$y, theta=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, Z = NULL, verbose = FALSE, drop.Z
                 = FALSE, ...)
```

Arguments

drop.Z	If FALSE find standard error without including the additional spatial covariates described by Z. 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 predict standard error or the prediction cross covariance matrix.
x	Same as xnew – points to compute the predict 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. 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.
...	These additional arguments passed to the predictSE function.
Z	Additional matrix of spatial covariates used for prediction. These are used to determine the additional covariance contributed in teh 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) + Z(X) - LY)**2. where P(x)+Z(x) is the value of the surface at x and LY is the linear combination

that estimates this point. Finding this expected value is straight forward given the unbiasedness of LY for P(x) and the covariance for Z 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 `predictSE.mKrig` for mKrig objects.

Value

A vector of standard errors for the predicted values of the Kriging fit.

See Also

Krig, predict.Krig, predictSurfaceSE

Examples

```
#
# Note: in these examples predictSE will default to predictSE.Krig using
# a Krig object

  fit<- Krig(ChicagoO3$x,ChicagoO3$y,cov.function="Exp.cov", theta=10)    # Krig fit
  predictSE.Krig(fit)      # std errors of predictions at obs.

# make a  grid of X's
  xg<-make.surface.grid(
  list(East.West=seq(-27,34,,20),North.South=seq(-20,35,,20)))
  out<- predictSE(fit,xg)   # std errors of predictions

#at the grid points out is a vector of length 400
#reshape the grid points into a 20X20 matrix etc.

   out.p<-as.surface( xg, out)
   surface( out.p, type="C")

# this is equivalent to  the single step function
# (but default is not to extrapolation beyond data
# out<- predictSurfaceSE( fit)
# image.plot( out)
```

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 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.

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 'fastTps'
predictSurface(object, grid.list = NULL,
       extrap = FALSE, chull.mask = NA, nx = 80, ny = 80,
       xy = c(1,2),  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, ZGrid = NULL,
    drop.Z = FALSE, just.fixed=FALSE,  ...)

## S3 method for class 'mKrig'
predictSurface(object,  ...)

## Default S3 method:
predictSurfaceSE( object, grid.list = NULL, extrap =
FALSE, chull.mask = NA, nx = 80, ny = 80, xy = c(1,2), verbose =
FALSE, ...)

## S3 method for class 'surface'
predict(object,...)
```

Arguments

object	An object from fitting a function to data. In fields this is usually a Krig, mKrig, or fastTps 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. 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`.)

`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, NA's 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.Z` If TRUE the fixed part of model depending on covariates is omitted.

`just.fixed` If TRUE the nonparametric surface is omitted.

`...` 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.

Z A matrix of covariates for the fixed part of model.

`ZGrid` An array or list form of covariates to use for prediction. This must match the `grid.list` argument. e.g. ZGrid and grid.list describe the same grid. If ZGrid 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( ZGrid) == c(10,15,2)`. If ZGrid is a list then the components x and y shold match those of grid.list and the z component follows the shape described above for the no list case.

`verbose` If TRUE prints out some imtermediate results for debugging.

Details

This function creates the right grid using the grid.list information or the attribute in xg, calls the predict function for the object with these points and also adding any extra arguments passed in the ... section, and then reforms the results as a surface object (as.surface). To determine the 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 point on 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 evaluting only the convex subset compared to the full grid.

predictSurface.fastTps is a specific version (m=2, and k=2) that can be much more efficient because it takes advantage of a low level FORTRAN call to evaluate the Wendland covariance function. Use predictSurface or predict for other choices of m and k.

predictSurface.Krig is designed to also include covariates for the fixed in terms of grids. Due to similarity in output and the model. predictSurface.mKrig just uses the Krig method.

NOTE: predict.surface has been depreciated and just prints out a warning when called.

Value

The usual list components for making 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 nx, ny, 2.

See Also

Tps, Krig, predict, grid.list, make.surface.grid, as.surface, surface, in.poly

Examples

```
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")
```

---

print.Krig *Print kriging fit results.*

Description

Prints the results from a 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

```
fit<- Krig(ChicagoO3$x,ChicagoO3$y, theta=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 the 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 is found by GCV based on the 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 is 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 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 psuedo 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 psuedo 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 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

Examples

```
# fit a CV  quantile spline
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 good value for
#lambda
# for example
lam<- fit50$cv.grid[,1]
tr<- fit50$cv.grid[,2]
# 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% 50% and 90%.
#
matlines( xg, sm, col=c( 3,3,2,3), lty=1) # the spline
```

QTps *Robust and Quantile smoothing using a thin-plate spline*

Description

This function uses the standard thin plate spline function Tps and a algorithm based on psuedo 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 is not clear how efficient the psuedo-data algorithm is acheiving 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 found by an approximate corss-validation criterion.
...	Any other arguments to pass to the Tps function.
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 interations of the psuedo 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 re stopped.
verbose	If TRUE intermediate results are printed out.

Details

These are experimental functions that use the psuedo-value algorithm to compute a class of robust and quantile problems. `QTps` use 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 the (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) + Z(x.k) + e.k

with the goal of estimating the smooth function: f(x)= P(x) + Z(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.psuedo<-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 psuedo values. Convergence is achieved when the the subsequent estimates of f.hat do not change between interations. The advantage of this algorithm is at every step a standard "least squares" thin plate spline is fit to the psuedo 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 psuedo data can be fit using the least squares spline and the standard smoothing techinques are valid. For example one can consider looking at the cross-validation function for the psuedo-data as a robust version to select a smoothing parameter. This approach is different from the weighted least squared algorithm used in the `qsreg` function. Also `qsreg` is only designed to work with 1-d cubic smoothing splines.

The "rho" function indicating the departure from a pure quadratic loss function has the definition

```
qsreg.rho<-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 ceratin range the residual is truncated to a constant value. This is similar to the Windsorizing operation in classical robust statistics.

Another advantage of the psuedo data algotrithm is that at convergence one can just apply all the usual generic functions from Tps to the psuedo 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)

## Not run:
# 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)
```

```
# least squares spline.
    look.outlier2<- QTps( x,y.outlier, psi.scale=100 , df= 50)
#
    set.panel(2,2)
    surface( look1)
    title("robust spline")
    surface( look2)
    title("least squares spline")
    surface( look.outlier1,  zlim=c(0,250))
    title("robust spline w/outlier")
    points( rbind(x[58,]), pch="+")
    surface( look.outlier2, zlim=c(0,250))
    title("least squares spline w/outlier")
    points( rbind(x[58,]), pch="+")
    set.panel()

## End(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.
## Not run:
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 point 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
  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])
     cat(k, " ")
     CV[k] <- temp.obj$Qinfo$CV.psuedo
     fhat.old<- temp.obj$fitted.values
  }
  plot( df, CV, type="l", lwd=2)
# psuedo data estimate
  points( qsreg.fit50$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)
```

---

quilt.plot *Image plot for 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, ...)
```

Arguments

x	A vector of the x coordinates of the locations -or- a 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.
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	Depreciated, same as nx.
ncol	Depreciated 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 FALSE add to existing plot.
col	Color scale for the image, the default is tim.colors – a pleasing spectrum.
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 zand so a grid box even one of these values may be an NA. (See details below.)
...	arguments to be passed to the image.plot function

Details

This function combines the discretization to an image by the function `as.image` and 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 good 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.

As always, 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.

Author(s)

D.Nychka

See Also

as.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)

#
# and ... if you are fussy
# do it again
# quilt.plot( ozone2$lon.lat, ozone2$y[16,],add=TRUE)
# to draw over the state boundaries.
#

### adding a common legend strip "by hand"
## and a custom color table

coltab<- two.colors( 256, middle="grey50" )

par( oma=c( 0,0,0,5)) # save some room for the legend
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)
}
par( oma=c(0,0,0,1))
image.plot(zlim=zr,legend.only=TRUE, col=coltab)
# may have to adjust number of spaces in oma to make this work.
```

---

rat.diet *Experiment studying an appetite supressant in rats.*

Description

The 'rat.diet' data frame has 39 rows and 3 columns. These are data from a study of an appetite supressant 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

RCMexample *3-hour precipitation fields from a regional climate model*

Description

These are 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 Amreica forced by observation data. The fields are 3 hour precipitation for 8 time periods in January 1, 1979. The grid is unequally spaced in longitude and latitude 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 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 extension series of numerical experiments the North American Regional Climate Change and Assessment Program (NARCCAP). NARCCAP has used 4 global climate models and observational data to supply the atmospheric boundery 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 Janurary 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 facilatate 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` computes the full Euclidean distance matrix among all pairings or a sparse version for points within a fixed threshhold 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`	Threshhold distance. All pairs of points that 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. In case that there are more points that are 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 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 the 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

```
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 justuse

  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 first column is the longitudes and second is the latitudes.
x2	Matrix of second set of lon/lat coordinates 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 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 values 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 reduce memory usage. The speed seems simillar 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 compiledfunctions 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 are have their definitions from the compliation 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 codemKrig.

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 Mulitplies 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 sline 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 5.4 of Writing R Extensions. See http://r.789695.n4.nabble.com/Registration-of-native-routines-td4728874.html for additional dicsussion of code registration.

Examples

```
print(addToDiagC)
```

REML.test *Maximum Likelihood estimates for some Matern covariance parameters.*

Description

For a fixed smoothness (shape) parameter these functions provide different ways of estimating and testing restricted and profile likehiloods for the Martern covariance parameters. MLE.Matern is a simple function that finds the restricted maximum likelihood (REML) estimates of the sill, nugget and range parameters (rho, sigma2 and theta) of the Matern covariance functions. The remaining functions are primarily for testing.

Usage

```
MLE.Matern(x, y, smoothness, theta.grid = NULL, ngrid = 20,
                 verbose = FALSE, niter = 25, tol = 1e-05,
                 Distance = "rdist", m = 2, Dmax = NULL, ...)

MLE.Matern.fast(x, y, smoothness, theta.grid = NULL, ngrid=20, verbose=FALSE,
                                          m=2, ...)
MLE.objective.fn( ltheta,info, value=TRUE)

MaternGLSProfile.test(x, y, smoothness = 1.5, init = log(c(0.05,1)))
MaternGLS.test(x, y, smoothness = 1.5, init = log(c(1, 0.2, 0.1)))
MaternQR.test (x, y, smoothness = 1.5, init = log(c(1, 0.2, 0.1)))
MaternQRProfile.test (x, y, smoothness = 1.5, init = log(c(1)))

REML.test(x, y, rho, sigma2, theta, nu = 1.5)
```

Arguments

Dmax	Maximum distance for grid used to evaluate the fitted covariance function.
Distance	Distance function used in finding covariance.
x	A matrix of spatial locations with rows indexing location and columns the dimension (e.g. longitude/latitude)
y	Spatial observations
smoothness	Value of the Matern shape parameter.
theta.grid	Grid of theta parameter values to use for grid search in maximizing the Likelilood. The defualt is do an initial grid search on ngrid points with the range at the 3 an d 97 quantiles of the pairwise distances.If only two points are passed then this is used as the range for a sequence of ngrid points.
ngrid	Number of points in grid search.
init	Initial values of the parameters for optimization. For the first three functions these are in the order rho, theta sigma2 and in a log scale. For MaternQRProfile.test initial value is just log(theta).
verbose	If TRUE prints more information.
rho	Marginal variance of Matern process (the "sill")
sigma2	Variance of measurement error (the "nugget")
theta	Scale parameter (the "range")
nu	Smoothness parameter
ltheta	log of range parameter
info	A list with components x,y,smoothness,ngrid that pass the information to the optimizer. See details below.
value	If TRUE only reports minus log Profile likelihood with profile on the range parameter. If FALSE returns a list of information.
m	Polynomial of degree (m-1) will be included in model as a fixed part.
niter	Maximum number of interations in golden section search.
tol	Tolerance for convergence in golden section search.
...	Additional arguments that are passed to the Krig function in evaluating the profile likelihood.

Details

MLE.Matern is a simple function to find the maximum likelihood estimates of using the restricted and profiled likeilihood that is intrinsic to the ccomputations in Krig. The idea is that the likelihood is concentrated to the parameters lambda and theta. (where lambda = sigma2/rho). For fixed theta then this is maximized over lambda using Krig and thus concetrates the likelihood on theta. The final maximization over theta is implemented as a golden section search and assumes a convex function. All that is needed is for three theta grid points where the middle point has a larger likelihood than the endpoints. In practice the theta grid defualts to a 20 points equally spaced between the .03 and .97 quantiles of the distribution of the pairwise distances. The likelihood is evaluated at these points and a possible triple is identified. If no exists from the grid search the

function returns with NAs for the parameter estimates. Note that due to the setup of the golden section search the computation actually minimizes minus the log likelihood. `MLE.Matern.fast` is a similar function but replaces the optimaiztion step computed by Krig to a tighter set of code in the function `MLE.objective.fn`. See also `mKrigMLEGrid` for an alternative and streamlined function using `mKrig` rather than `Krig`.

Value

For MLE.Matern (and MLE.Matern.fast)

`smoothness`	Value of the smoothness function
`pars`	MLE for rho, theta and sigma
`REML`	Value of minus the log restricted Profile likelihood at the maxmimum
`trA`	Effective degrees of freedom in the predicted surface based on the MLE parameters.
`REML.grid`	Matrix with values of theta and the log likelihood from the initial grid search.

Note

See the script REMLest.test.R and Likelihood.test.R in the tests directory to see how these functions are used to check the likelihood expressions.

Author(s)

Doug Nychka

Examples

```
# Just look at one day from the ozone2
data(ozone2)

out<- MLE.Matern( ozone2$lon.lat,ozone2$y[16,],1.5, ngrid=8)
plot( out$REML.grid)
points( out$pars[2], out$REML, cex=2)
xline( out$pars[2], col="blue", lwd=2)
## Not run:
# to get a finer grid on initial search:
out<- MLE.Matern( ozone2$lon.lat,ozone2$y[16,],1.5,
                      theta.grid=c(.3,2), ngrid=40)

# simulated data  200 points uniformly distributed
set.seed( 123)
x<- matrix( runif( 2*200), ncol=2)
n<- nrow(x)
rho= 2.0
sigma= .05
theta=.5

Cov.mat<-  rho* Matern( rdist(x,x), smoothness=1.0, range=theta)
A<- chol( Cov.mat)
```

```
gtrue<- t(A) %*% rnorm(n)
gtrue<- c( gtrue)
err<-  rnorm(n)*sigma
y<- gtrue + err
out0<- MLE.Matern( x,y,smoothness=1.0) # the bullet proof version
# the MLEs and -log likelihood at maximum
print( out0$pars)
print( out0$REML)

out<- MLE.Matern.fast( x,y, smoothness=1.0) # for the impatient
# the MLEs:
print( out$pars)
print( out$REML)


# MLE for fixed theta (actually the MLE from out0)
# that uses MLE.objective.fn directly
info<- list( x=x,y=y,smoothness=1.0, ngrid=80)
# the MLEs:
out2<- MLE.objective.fn(log(out0$pars[2]), info, value=FALSE)
print( out2$pars)

## End(Not run)

## Not run:
# Now back to Midwest ozone pollution ...
# Find the MLEs for ozone data and evaluate the Kriging surface.
  data(ozone2)
  out<- MLE.Matern.fast( ozone2$lon.lat,ozone2$y[16,],1.5)
#use these parameters to fit surface ....
  lambda.MLE<- out$pars[3]/out$pars[1]
  out2<- Krig( ozone2$lon.lat,ozone2$y[16,] , Covariance="Matern",
              theta=out$pars[2], smoothness=1.5, lambda= lambda.MLE)
  surface( out2) # uses default lambda -- which is the right one.

# here is another way to do this where the new lambda is given in
# the predict step
  out2<- Krig( ozone2$lon.lat,ozone2$y[16,] , Covariance="Matern",
               theta=out$pars[2], smoothness=1.5)
# The default lambda is that found by GCV
# predict on a grid but use the MLE value for lambda:
  out.p<- predictSurface(out2, lambda= lambda.MLE)
  surface(out.p) # same surface!

## End(Not run)

# One could also use mKrig with a fixed lambda to compute the surface.

## Not run:
# looping  through all the days of the ozone data set.
  data( ozone2)
  x<- ozone2$lon.lat
  y<- ozone2$y
```

```
  out.pars<- matrix( NA, ncol=3, nrow=89)

  for ( k in 1:89){
    hold<- MLE.Matern.fast( x,c(y[k,]), 1.5)$pars
    cat( "day", k," :", hold, fill=TRUE)
    out.pars[k,]<- hold }

## End(Not run)
```

`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 `segment` 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 acolor 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 Rocky Mountain region at 4km resolution. Note that the gridded elevations from the PRISM data product are different than the exact station elevations. (See example below.)

Format

The data set `RMprecip` is a list containing the following components:

x Longitude-latitude position of monitoring stations. Rows 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

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 http://www.prism.oregonstate.edu/documents/PRISM_downloads_FTP.pdf and also archived at the National Climate Data Center. This work is authored by Chris Daly http://www.prism.oregonstate.edu and his PRISM group but had 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.

```
# explicit 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

```
# this data set was created  the
# historical data  taken from
# Observed monthly precipitation, min and max temperatures for the coterminous US
# 1895-1997
```

```
# NCAR_pinfill
# see the Geophysical Statistics Project datasets page for the supporting functions
# and details.

# plot
quilt.plot(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 a 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.rf *Simulates a Stationary Gaussian random field*

Description

Simulates a stationary Gaussian random field on a regular grid with unit marginal variance.

Usage

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

Arguments

obj	A 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. (See details below.)
...	Additional arguments passed to a particular method.

Details

The simulated field has the marginal variance that is determined by the covariance function for zero distance. Within fields the exponential and matern 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 `cov.function(0)` 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 variance one can simply scale the result of this function. See the third example below.

This function takes an object that includes some preliminary calculations and so is more efficient for simulating more than one field from the same covariance. However, the algorithm using a 2-d FFT (known as circulant embedding) may not always work if the correlation range is large. The 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 and exact simulation of the random field.

For a stationary model the covariance object should have the components:

names(obj) "m" "n" "grid" "N" "M" "wght",

where m and n are the number of grid points in x and y, grid is a list with components x and y giving the grid points in each coordinate. N and M is the size of the larger grid that is used for simulation. Usually M = 2*m and N =2*n and results in an exact simulation of the stationary Gaussian field. wght is a matrix from the FFT of the covariance function. The easiest way to create this object is to use for example Exp.image.cov with setup=T (see below).

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. Micheal Stein and Tilman Gneiting have also made some additional contributions to the algortihms and theory.

Value

A matrix with the random field values

See Also

Exp.image.cov, matern.image.cov, stationary.image.cov

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<-Exp.image.cov( grid=grid, theta=.5, setup=TRUE)
look<- sim.rf( obj)
# Now simulate another ...
look2<- sim.rf( obj)

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

# take a look at first two
set.panel(2,1)
 image.plot( grid$x, grid$y, look)
 title("simulated gaussian fields")
 image.plot( grid$x, grid$y, look2)
 title("another realization ...")
```

`sim.spatialProcess` *Conditional simulation of a spatial process*

Description

Generates exact (or approximate) random draws from the conditional distribution of a spatial process given specific observations. This is a useful way to characterize the uncertainty in the predicted process from data. This is known as conditional simulation in geostatistics or generating an ensemble prediction in the geosciences. sim.Krig.grid can generate a conditional sample for a large regular grid but is restricted to stationary correlation functions.

Usage

```
sim.spatialProcess(object, xp,  M = 1, verbose = FALSE, ...)

sim.Krig(object, xp, M = 1, verbose = FALSE, ...)

sim.Krig.approx(object, grid.list = NULL, M = 1, nx = 40, ny = 40,
                  verbose = FALSE, extrap = FALSE,...)

sim.mKrig.approx(mKrigObject, predictionPoints = NULL,
                  predictionPointsList = NULL, simulationGridList =
                  NULL, gridRefinement = 5, gridExpansion = 1 + 1e-07, M
                  = 1, nx = 40, ny = 40, nxSimulation = NULL,
                  nySimulation = NULL, delta = NULL, verbose = FALSE,...)

sim.fastTps.approx(fastTpsObject,
                  predictionPointsList, simulationGridList =
                  NULL, gridRefinement = 5, gridExpansion = 1 + 1e-07, M
                  = 1,  delta = NULL, verbose=FALSE,...)
```

Arguments

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 theta for the Wendland but a multiple of this maybe needed to obtain a positive definite circulant covariance function.
extrap	If FALSE conditional process is not evaluated outside the convex hull of observations.
fastTpsObject	The output object returned by fastTps
grid.list	Grid information for evaluating the conditional surface as a grid.list.
gridRefinement	Amount to increase the number of grid points for the simulation grid.
gridExpansion	Amount to increase the size of teh simulation grid. This is used to increase the simulation domain so that the circulant embedding algorithm works.
mKrigObject	An mKrig Object
M	Number of draws from conditional distribution.
nx	Number of grid points in prediction locations for x coordinate.
ny	Number of grid points in prediction locations for x coordinate.
nxSimulation	Number of grid points in the circulant embedding simulation x coordinate.
nySimulation	Number of grid points in the circulant embedding simulation x coordinate.
object	A Krig object.
predictionPoints	A matrix of locations defining the points for evaluating the predictions.
predictionPointsList	A grid.list defining the rectangular grid for evaluating the predictions.
simulationGridList	A gridlist describing grid for simulation. If missing this is created from the range of the locations, nx, ny, gridRefinement, and gridExpansion or from the range and and nxSimulation, nySimulation.
xp	Same as predictionPoints above.
...	Any other arguments to be passed to the predict function. Usually this is the Z or drop.Z argument when there are additional covariates in the fixed part of the model. (See example below.)
verbose	If true prints out intermediate information.

Details

These functions generate samples from a conditional multivariate distribution, or an approximate one, that describes the uncertainty in 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.

Given a spatial process h(x)= P(x) + g(x) observed at

Y.k = Z(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 Z(x.k) is a vector of covariates that are also indexed by space (such as elevation). Z(x.k)d is a linear combination of the 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 Z(x)d + h(x)

For fixed a covariance this is just a multivariate normal sampling problem. `sim.Krig.standard` samples this conditional process at the points `xp` and is exact for fixed covariance parameters. `sim.Krig.grid` also assumes fixed covariance parameters and does approximate sampling on a grid.

The outline of the algorithm is

0) Find the spatial prediction at the unobserved locations based on the actual data. Call this h.hat(x) and this is the conditional mean.

1) Generate an unconditional spatial process and from this process simluate synthetic observations. At this point the approximation is introduced where the field at the observation locations is approximated using interpolation from the nearest grid points.

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].

`sim.spatialProcess` Follows this algorithm exactly. For the case of an addtional covariate this of course needs to be included. For a model with covariates use `drop.Z=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 are the same. The naming is through convenience.

`sim.Krig` Also follows this algorithm exactly but for the older `Krig` object. Note the inclusion of drop.Z=TRUE or FALSE will determine whether the conditional simulation includes the covariates Z or not. (See example below.)

`sim.Krig.approx` and `sim.mKrig.approx` evaluate the conditional surface on grid and simulates the values of h(x) off the grid using bilinear interpolation of the four nearest grid points. Because of this approximation it is important to choose the grid to be fine relative to the spacing of the observations. The advantage of this approximation is that one can consider conditional simulation for large grids – beyond the size possible with exact methods. Here the method for simulation is circulant embedding and so is restricted to stationary fields. The circulant embedding method is known to fail if the domain is small relative to the correlation range. The argument `gridExpansion` can be used to increase the size of the domain to make the algorithm work.

`sim.fastTps.approx` Is optimized for the approximate thin plate spline estimator in two dimensions and `k=2`. For efficiency the ensemble prediction locations must be on a grid.

Value

sim.Krig and sim.spatialProcess a matrix with rows indexed by the locations in xp and columns being the M independent draws.

sim.Krig.approx a list with components x, y and z. x and y define the grid for the simulated field and z is a three dimensional array with dimensions c(nx,ny,M) where the first two dimensions index the field and the last dimension indexes the draws.

sim.mKrig.approx a list with predictionPoints being the locations where the field has been simulated.If these have been created from a grid list that information is stored in the attributes of predictionPoints. Ensemble is a matrix where rows index the simulated values of the field and columns are the different draws, call is the calling sequence. Not that if predictionPoints has been omitted in the call or is created beforehand using make.surface.grid it is easy to reformat the results into an image format for ploting using as.surface. e.g. if simOut is the output object then to plot the 3rd draw:

```
imageObject<- as.surface(simOut$PredictionGrid, simOut$Ensemble[,3] )
image.plot( imageObject)
```

sim.fastTps.approx is a wrapper function that calls sim.mKrig.approx.

Author(s)

Doug Nychka

See Also

sim.rf, Krig, spatialProcess

Examples

```
## Not run:
# conditional simulation with covariates
# colorado climate example
  data(COmonthlyMet)
  fit1E<- spatialProcess(CO.loc,CO.tmin.MAM.climate, Z=CO.elev   )
# conditional simulation at missing data
  good<- !is.na(CO.tmin.MAM.climate )
  infill<- sim.spatialProcess( fit1E, xp=CO.loc[!good,],
                Z= CO.elev[!good], M= 10)
# get an elevation grid  ... NGRID<- 50 gives a nicer image but takes longer
 NGRID <- 25
 # get elevations on a grid
   COGrid<- list( x=seq( -109.5, -101, ,NGRID), y= seq(39, 41.5,,NGRID) )
   COGridPoints<- make.surface.grid( COGrid)
 # elevations are a bilinear interpolation from the 4km
 # Rocky Mountain elevation fields data set.
   data( RMelevation)
   COElevGrid<- interp.surface( RMelevation, COGridPoints )
# NOTE call to sim.Krig 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,  Z= COElevGrid, M= 30)
# for just the smooth surface in lon/lat
#  SEout<- sim.spatialProcess( fit1E, xp=COGridPoints,  drop.Z=TRUE, M= 30)
# in practice M should be larger to reduce Monte Carlo error.
   surSE<- apply( SEout, 2, sd )
   image.plot( as.surface( COGridPoints, surSE))
   points( fit1E$x, col="magenta", pch=16)


## End(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",
            theta=1.0,smoothness=1.0, na.rm=TRUE)

# NOTE theta =1.0 is not the best choice but
# allows the sim.rf circulant embedding algorithm to
# work without increasing the domain.

#six missing data locations
 xp<-  ozone2$lon.lat[ is.na(ozone2$y[16,]),]

# 5 draws from process at xp given the data
# this is an exact calculation
 sim.Krig( out,xp, M=5)-> sim.out

# Compare: stats(sim.out)[3,] to  Exact: predictSE( out, xp)
# simulations on a grid
# NOTE this is approximate due to the bilinear interpolation
# for simulating the unconditional random field.
# also more  grids points ( nx and  ny) should be used

sim.Krig.approx(out,M=5, nx=20,ny=20)-> sim.out

# take a look at the ensemble members.

predictSurface( out, grid= list( x=sim.out$x, y=sim.out$y))-> look

zr<- c( 40, 200)

set.panel( 3,2)
image.plot( look, zlim=zr)
title("mean surface")
for ( k in 1:5){
image( sim.out$x, sim.out$y, sim.out$z[,,k], col=tim.colors(), zlim =zr)
}



## Not run:
```

```
data( ozone2)
y<- ozone2$y[16,]
good<- !is.na( y)
y<-y[good]
x<- ozone2$lon.lat[good,]
O3.fit<- mKrig( x,y, Covariance="Matern", theta=.5,smoothness=1.0, lambda= .01 )
set.seed(122)
O3.sim<- sim.mKrig.approx( O3.fit, nx=100, ny=100, gridRefinement=3, M=5 )
set.panel(3,2)
surface( O3.fit)
for ( k in 1:5){
image.plot( as.surface( O3.sim$predictionPoints, O3.sim$Ensemble[,k]) )
}
# conditional simulation at missing data
xMissing<- ozone2$lon.lat[!good,]
O3.sim2<- sim.mKrig.approx( O3.fit, xMissing, nx=80, ny=80,
                            gridRefinement=3, M=4 )

## End(Not run)
## Not run:
#An example for fastTps:
  data(ozone2)
  y<- ozone2$y[16,]
  good<- !is.na( y)
  y<-y[good]
  x<- ozone2$lon.lat[good,]
  O3FitMLE<- fastTpsMLE( x,y, theta=1.5 )
  O3Obj<- fastTps( x,y, theta=1.5, lambda=O3FitMLE$lambda.MLE)
# creating a quick grid list based on ranges of locations
  grid.list<- fields.x.to.grid( O3Obj$x, nx=100, ny=100)
  O3Sim<- sim.fastTps.approx( O3Obj,predictionPointsList=grid.list,M=5)
# controlling the grids
  xR<- range( x[,1], na.rm=TRUE)
  yR<- range( x[,2], na.rm=TRUE)
  simulationGridList<- list( x= seq(xR[1],xR[2],,400), y= seq( yR[1],yR[2], ,400))
# very fine localized prediction grid
    O3GridList<- list( x= seq( -90.5,-88.5,,200), y= seq( 38,40,,200))
    O3Sim<- sim.fastTps.approx( O3Obj, M=5, predictionPointsList=O3GridList,
                 simulationGridList = simulationGridList)
# check
 plot( O3Obj$x)
 US( add=TRUE)
 image.plot( as.surface( O3GridList,O3Sim$Ensemble[,1] ), add=TRUE)
 points( O3Obj$x, pch=16, col="magenta")

## End(Not run)
```

---

smooth.2d *Kernel 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.

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 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 are 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 function 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 need only be of size L+1.
Mwidth	See Nwidth.
...	Parameters that are passed to the smoothing kernel. (e.g. the scale parameter theta for the exponential or gaussian)

Details

The irregular locations are first discretized to a regular grid (using as.image) then a 2d- 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 by zeroes where data is not obsevred one can sum the kernel over irregular sets of locations. A second convolutions to find the normalization of the kernel weights.

The kernel function is specified by an 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 theta is the bandwidth. It is easy to write other 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)
#
look<- smooth.2d( RMprecip$y,  x=RMprecip$x, theta=.25)

# finer resolution used in computing the smooth
look3<-smooth.2d( RMprecip$y, x=RMprecip$x, theta=.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,theta=.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 more easily checked format based directly on the row and column indices of non zero 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 obj of class spam.
add.zero.rows	If TRUE 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 differencee in formats is best illustarted 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 position 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 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 nugget (sigma^2) and process variance (rho) and the range parameter (theta) by restricted maximum likelihood and then computes the spatial model with these estimated parameters. Other parameters of the covariance are kept fixed and need to be specified.

Usage

```
spatialProcess(x, y, weights = rep(1, nrow(x)), Z = NULL,
mKrig.args = list(m = 2),
cov.function = "stationary.cov", cov.args = list(Covariance = "Matern",
        smoothness = 1), theta = NULL, theta.start = NULL, lambda.start = 0.5,
         theta.range = NULL,
    abstol = 1e-04, na.rm = TRUE, verbose = FALSE, REML = 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, ...)
```

Arguments

x	Observation locations
y	Observation values
weights	Weights for the error term (nugget) in units of reciprocal variance.
Z	A matrix of extra covariates for the fixed part of spatial model. E.g. elevation for fitting climate data over space.
mKrig.args	Arguments passed to the mKrig function.
cov.function	A character string giving the name of the covariance function for the spatial component.
cov.args	A list specifying parameters and other components of the covariance function.
theta	If not NULL the range parameter for the covariance is fixed at this value.
theta.start	Starting value for MLE fitting of the scale (aka range) parameter. If omitted the starting value is taken from a grid search ove theta.

lambda.start	Starting value for MLE fitting of the lambda parameter. Note lambda is the ratio of the nugget variance to the process variance. In code variables this is sigma^2 divided by rho.
theta.range	A range for the ML search to estimate theta. Default is based on quantiles of the location pairwise distances.
na.rm	If TRUE NAs are removed from the data.
object	A spatialProcess object returned from the spatialProcess function.
REML	If TRUE the parameters are found by restricted maximum likelihood.
verbose	If TRUE print out intermediate information for debugging.
...	Any other arguments that will be passed to the mKrig function and interpreted as additional arguments to the covariance function. E.g. smoothness for the Matern covariance.
abstol	The absolute tolerance bound used to judge convergence. This is applied to the difference in log likelihood values.
digits	Number of significant digits in printed summary
which	The vector 1:4 or any subset of 1:4, giving the plots to draw. See the description ofthese plots below.

Details

This function makes many choices for the user in terms of defaults and it is important to be aware of these. The spatial model is

Y.k= P(x.k) + Z(x.k)%*%d2 + 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) and Z is a matrix of covariates (optional) that enter as a linear model the fixed part. g is a mean zero, Gaussian stochastic process with a marginal variance of rho and a scale (or range) parameter, theta. The measurement errors, e.k, are assumed to be uncorrelated, normally distributed with mean zero and standard deviation sigma. If weights are supplied then the variance of e is assumed to be sigma^2/ weights.

Perhaps the most important aspect of this function is that the range (theta), nugget (sigma**2) and process variance (rho) parameters for the covariance are estimated by restricted maximum likelihood and this is the model that is then used for spatial prediction. Geostatistics usaually refers to sigma**2 + rho 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 model we reparametrize as lambda= sigma**2/ rho and rho. Thinking about h as the spatial signal and e as the noise lambda can be interpreted as the noise to signal variance ratio in this spatial context.(See the comparision with fitting the geoR model in the examples section.)

The likelihood and the cross valdiation function can be concentrated to only depend on lambda and theta and so in reported the optimiztation of these two criterion we focus on this form of the parameters. Once lambda and theta are found, the MLE for rho has a closed form and of course then sigma is then determined from lambda and rho.

Often the lambda parameter is difficult to interpret when covariates and a linear function of the coordinates is 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 trA in R code and the output summaries) associated with the predicted surface or curve. This measure has a one to one relationship with lamdba and is easier to interpret. For example an eff degrees of freedom that is very small suggests that the surface is rwell represented by a low ordoer polynomial. Degrees of freedom close to the number of locations indicates a surface that is close to interpolating the observations and suggests 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 implementd using the `stationary.cov` covariance that can take a argument for the form of the covariance, a sill and range parameters and possibily additional parameter might comtrol the shape.

See the example below 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 transfromed to remove the fixed linear part of the model. If the user just wants to fix the range parameter theta 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),
                  abstol = abstol,
                   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 an quick plot of a spatial field to discern obvious spatial paterns.

summary method forms a list with class `spatialProcessSummary` that hasa 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 replacated 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 a panel of 4 diagnositic plots of the fit. Use `set.panel(2,2)` to see all 4 at once. The third plot gives the likelihood and GCV functions as a function of lambda evaluated at the global MLE for theta. This is based on the gird evaluations in the component MLEInfo$MLEProfileLambda. The fourth plot is a profile likelihood trace for theta having maximized over lambda and is based on the component MLEInfo$MLEGrid.

print method prints the `spatialProcessSummary` object of the fit, adding some details and explanations.

Value

An object of classes `mKrig` and `SpatialProcess`. The main difference from mKrig is an extra component, `MLEInfo` that has the results of the profile likelihood grid evaluation over theta (having maximizing lamdba), joint maximization over theta and lambda, and a grid evaluation over lambda with theta fixed at its MLE.

Author(s)

Doug Nychka

See Also

Tps, MLESpatialProcess, 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$lon.lat
# y is a vector of ozone measurements at day 16 a the locations.
  y<- ozone2$y[16,]
  obj<- spatialProcess( x, y)
# 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 = sigma^2 / rho
#        the x axis has transformed lambda
#        in terms of effective degrees of freedom
#        to make it easier to interpret
#        Note that here the GCV function is minimized
#        while the REML is maximzed.
# plot 4 the log profile likelihood used to
#        determine theta.
#
# predictions on a grid
  surface( obj)
#(see also predictSurface for more control on evaluation grid
# and plotting)
#

## 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, Z=CO.elev,
                              theta.range= c(.25, 2.0) )
  set.panel( 2,2)
  plot( fit1E)

# conditional simulation at missing data
```

```
  notThere<- is.na(CO.tmin.MAM.climate )
  xp <- CO.loc[notThere,]
  Zp <- CO.elev[notThere]
  infill<- sim.spatialProcess( fit1E, xp=xp,
                     Z= Zp, M= 10)
#
# interpretation is that these infilled values are all equally plausible
# given the observations and also given the estimated covariance model
#
# for extra credit one could now standardized the infilled values to have
# conditional mean and variance from the exact computations
#  e.g. predict( fit1E, xp=CO.loc[!good,],  Z= CO.elev[!good])
#  and  predictSE(fit1E, xp=CO.loc[!good,],  Z= 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,
                               theta.range= c(.25, 2.0) )
# note larger range parameter
# create 2500 grids using handy function
gridList <- fields.x.to.grid( fit2$x, nx=50,ny=50)
xGrid<- make.surface.grid( gridList)
ensemble<- sim.spatialProcess( fit2, xp=xGrid, M= 5)
# this is an "n^3" computation so increasing the grid size
# can slow things down for computation
image.plot( as.surface( xGrid, ensemble[1,]))
set.panel()

## End(Not run)

## Not run:
data( ozone2)
# x is a two column matrix where each row is a location in lon/lat
# coordinates
  x<- ozone2$lon.lat
# y is a vector of ozone measurements at day 16 a the locations.
  y<- ozone2$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.
obj$pars
obj2$pars
obj3$pars
# since the exponential is Matern with smoothness == .5 the first two
# fits can be compared in terms of their likelihoods
# the REML value is slightly higher for obj verses obj2 (598.4  > 596.7)
# these are the _negative_ log  likelihoods so suggests a preference for the
# exponential 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")

## End(Not run)
set.panel(1,1)

## Not run:
### comparison with GeoR
  data(ozone2)
  x<- ozone2$lon.lat
  y<- ozone2$y[16,]
  good<-!is.na(y)
  x1<- x[good,]
  y1<- y[good]

  obj<- spatialProcess( x, y, mKrig.args= list(m=1), smoothness = .5 )
```

```
  library( geoR)
  ml.n <- likfit(coords= x1, data=y1, ini = c(570, 3), nug = 50)
  # compare to
  stuffFields<- obj$MLEInfo$MLEJoint$summary[c(1,3,4,5)]
  stuffGeoR<- c( ml.n$loglik, ml.n$phi, sqrt(ml.n$nugget),ml.n$sigmasq)
  test.for.zero(  max(stuffFields/stuffGeoR), 1, tol=.004)

## 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)
```

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 a first or second derivative should be evaluated.
wt	Weights for different obsrevations 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

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 many options in sreg or other more statistical implementations. To make the solution well posed the 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 Tibshriani.

See Also

sreg, Tps

Examples

```
x<- seq( 0, 120,,200)

# 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. <!–brief description–>

Usage

```
sreg(x, y, lambda = NA, df = NA, offset = 0,
     weights = rep(1, length(x)), cost = 1,
     nstep.cv = 80, tol=1e-5,find.diagA = TRUE, trmin = 2.01,
     trmax = NA, lammin = NA,
     lammax = NA, verbose = FALSE,
     do.cv = TRUE, method = "GCV", rmse = NA,
      na.rm =  TRUE)

## S3 method for class 'sreg'
predict(object, x, derivative = 0, model = 1,...)
```

Arguments

x	Vector of x value
y	Vector of y values
lambda	Single smoothing parameter or a vector of values . If omitted smoothing parameter 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 deriviatve to evaluate. Must be 0,1, or 2.
df	Amount of smoothing in term 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 stores 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 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 falseof 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 NA's are removed from y before analysis.
...	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 sigma**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 the 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 sigma**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.
`shat.pure.error`	Estimate of sigma from replicate groups.
`shat.GCV`	Estimate of sigma using estimated lambda from GCV minimization
`trace`	Effective degrees of freedom for the spline estimate(s)
`gcv.grid`	Values of trace, GCV, shat. 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 spline having 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 the smoothing the group means at the unique x locations.

See Also

Krig, Tps, splint

Examples

```
# fit a GCV spline to
# control group of rats.
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 sacel 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$shat.GCV*sqrt(fit$diagA)

# compare to predictSE( fit.tps) differences are due to
# slightly different lambda values and using shat.MLE instad of shat.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)
lines( fit$predicted, lwd=2)
matlines( fit$x,
cbind( lower, upper, lowerB, upperB), type="l", col=c( 2,2,4,4), lty=1)
title( "95 pct pointwise  and simultaneous intervals")
# or try the more visually  honest:
plot( fit$x, fit$y)
lines( fit$predicted, lwd=2)
segments(  fit$x, lowerB, fit$x, upperB, col=4)
segments(  fit$x, lower, fit$x, upper, col=2, lwd=2)
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 relevent groups when a numeric vector and a by vector are given. The default set of statistics are 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)
```

Arguments

x	Values to use to decide bin membership
y	A vector of data
N	Number of bins. If the 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.

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 NA's. (If there is no data for a given bin, NA's 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 (sigma) 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

```
fit<- Krig(ChicagoO3$x, ChicagoO3$y, theta=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 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 often large geophysical data sets are 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 it use. The function is also a a useful device to see how the ncdf object is structured.

Author(s)

D. Nychka

See Also

ncdf

supportsArg *Tests if 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 tested for support 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, ...)

## 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, ...)
```

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 an 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).
`...`	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: that any Z covariates will b edropped and only the spatial part of the model will be evaluated.

See Also

`Krig` predictSurface, plot.surface, image.plot

Examples

```
fit<- Krig(ChicagoO3$x,ChicagoO3$y, theta=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.knots(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,Z, weights, na.rm, verbose = FALSE)
Krig.cor.Y(obj, 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 digaonal matrix.
weights	Raw weights vector passed to Krig
Z	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 computed the estimated curve or surface coefficients. All the engine 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 are components 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 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 measure 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 finds the decompositions in the case that the covariance is evaluated at arbitrary locations possibly different than the data locations (called knots). The intent of these decompositions is to facilitate the evaluation at different values for lambda. There will be computational savings when the number of knots is less than the number of unique locations. (But the knots

are as densely distributed as the structure in the underlying spatial process.) This function call fields.diagonalize, a function that computes the matrix and eigenvalues that simultaneous diagonalize a nonnegative definite and a positive definite matrix. These decompositions also facilitate multiple evaluations of the likelihood and GCV functions in estimating a smoothing parameter and also multiple solutions for different y vectors.

`Krig.engine.fixed` are specific decomposition 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 between 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 eigen decomposition. If the decompostion is the Cholesky based then this function returns NA for the u component in the list.

`Krig.check.xY` Checks for removes missing values (NAs).

`Krig.cor.Y` Standardizes the data vector Y based on a correlation model.

`Krig.transform.xY` Finds all replicates and collapse 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.knots`:

u A transformed vector that is based on the data vector.

D Eigenvalues of decomposition

G Matrix from diagonalization

qr.T QR decomposition of the matrix for the fixed component. i.e. sqrt(Wm)%*%T

pure.ss pure error sums of squares including both the variance from replicates and also the sums of squared residuals from fitting the full knot model with lambda=0 to the replicate means.

`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

shat.rep Sample standard deviation of replicates

shat.pure.error Same as shat.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

```
Krig( ChicagoO3$x, ChicagoO3$y, theta=100)-> out

Krig.engine.default( out)-> stuff

# 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)

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(256), zlim =NULL,
transparent.color="white", eps= 1e-8)

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 is covered.
n	Number of color levels. The setting n=64 is the orignal 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
x	Positions of colors on a [0,1] scale. Default is to assume that the x values are equally spacesd 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 NA's or values outside zlim
eps	A small inflation of the range to avoid boundary values of z being coded as NAs
...	Additional plotting arguments to codeimage.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 intermediate to the specified colors.

Ask Tim Hoar about tim.colors! He is a Mattlab black belt and this is his favorite scale in that system. 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 how this was formed from an orignal raw 256 level scale. Note the that first color in this table is grey and is desigend to represent the minimum value of the range (e.g. 0). If the image in 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.

`designer.color` is the master function for the otther 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 three colors to provide a transtion 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.color( 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 kept the assigment of colors consistent across several vectors by specifiying a common `zlim` range.

`plotColorScale` 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.

See Also

topo.colors, terrain.colors, image.plot, 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 tranparency 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


set.panel()
```

Tps *Thin plate spline regression*

Description

Fits a thin plate spline surface to irregularly spaced data. The smoothing parameter is chosen by generalized cross-validation. The assumed model is additive Y = f(X) +e where f(X) is a d dimen-

sional surface. This function also works for just a single dimension and is a special case of a spatial process estimate (Kriging). A "fast" version of this function uses a compactly supported Wendland covariance and computes the estimate for a fixed smoothing parameter.

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,  theta, lon.lat=FALSE,
          find.trA = TRUE, lambda=0, ...)
```

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.
theta	The tapering range that is passed to the Wendland compactly supported covariance. The covariance (i.e. the radial basis function) is zero beyond range theta. The larger theta the closer this model will approximate the standard thin plate spline.
lon.lat	If TRUE locations are interpreted as lognitude and latitude and great circle distance is used to find distances among locations. The theta 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
lambda	Smoothing parameter the ratio of error variance to process variance, default is zero which corresponds to interpolation. See fastTpsMLE to estimate this paramter from the data.

`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.

`...` 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.

Arguments for Tps:

lambda Smoothing parameter that is the ratio of the error variance (sigma**2) to the scale parameter of the covariance function. If omitted this is estimated by GCV.

Z Linear covariates to be included in 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. 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 divided into each column after centering. Must have scale.type="user".

rho Scale factor for covariance.

sigma2 Variance of errors or if weights are not equal to 1 the variance is sigma**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

Both of these functions are special cases of using the `Krig` and `mKrig` functions. See the help on each of these for more information on the calling arguments and what is returned. `Tps` makes use of the stable computations via eigen decompositions in `Krig`. `fastTps` follows the more standard computations for spatial statistics centered around the Cholesky decomposition in `mKrig`.

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))**22 + 2(Dxy(f))**2 + (Dyy(f))**22

(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.

CAUTION about `lon.lat=TRUE`: 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 be large enough that the basis function 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 the locations being the direction cosines. This will give approximate great circle distances for locations that are close and also the numerical methods will always have a positive definite matrices.

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
  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 `mKrig` with the Wendland covariance function. This is experimental and some care needs to exercised in specifying the taper

range and power (p) which describes the polynomial behavior of the Wendland at the origin. 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. 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 for more background. Also, the function predictSurface.fastTps has been made more efficient for the case of k=2 and m=2. Also see the handy function fastTpsMLE to estimate lambda by maximum likelihood.

See also the mKrig function for handling larger data sets and also for an example of combining Tps and mKrig for evaluation on a huge grid.

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 Also

Krig, mKrig, spatialProcess, summary.Krig, predict.Krig, predictSE.Krig, predictSurface, predictSurface.fastTps, plot.Krig, surface.Krig, sreg, fastTpsMLE

Examples

```
#2-d example

fit<- Tps(ChicagoO3$x, ChicagoO3$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 estiamtes 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 correpsonds to 7.5 df.

fit1<- Tps(ChicagoO3$x, ChicagoO3$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)
image( 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, Z= CO.elev)
  out.p<-predictSurface( obj,
            grid.list=CO.Grid, ZGrid= CO.elevGrid)
  image.plot( 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
  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:
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, Z=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.Z=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, ZGrid= 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, ZGrid= RMelevation)
# This is the smooth part but also with the linear lon lat terms.
  fit.smooth<-predictSurface( out, grid.list, drop.Z=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: theta =3 degrees is a very generous taper range.
# Use some trial theta 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.

  fastTps( RMprecip$x,RMprecip$y,m=2,lambda= 1e-2, theta=3.0) -> out2

# note that fastTps produces an mKrig object so one can use all the
# the overloaded functions that are defined for the mKrig class.
# summary of what happened note estimate of effective degrees of
# freedom
  print( out2)

## Not run:
set.panel( 1,2)
surface( out2)

#
# now use great circle distance for this smooth
# Here "theta" 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).
#
fastTps( RMprecip$x,RMprecip$y,m=2,lambda= 1e-2,lon.lat=TRUE, theta= 3.0) -> out3
print( out3)  # note the effective degrees of freedom is different.
surface(out3)

set.panel()
```

```
## End(Not run)

## Not run:
#
# simulation reusing Tps/Krig object
#
fit<- Tps( rat.diet$t, rat.diet$trt)
true<- fit$fitted.values
N<-  length( fit$y)
temp<- matrix(  NA, ncol=50, nrow=N)
sigma<- fit$shat.GCV
for (  k in 1:50){
ysim<- true + sigma* rnorm(N)
temp[,k]<- predict(fit, y= ysim)
}
matplot( fit$x, temp, type="l")


## End(Not run)
#
#4-d example
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 subtract from 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 between transformed. This matrix also has the attributes: scale.type, x.center and y.center with the transformation information.

See Also

scale

Examples

```
#
newx<-transformx( ChicagoO3$x, scale.type="range")
```

US *Plot of the US with state boundaries*

Description

Plots quickly, 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. Currenty 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 coterminous 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 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,
                    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"))

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 differnces to find. If omitted all possible pairing are found. This can 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 from from 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 ther may not be exactly N bins. Specify the breaks explicity if you want excalty 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.
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 argument 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 not either breaks or N arguments are not 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

boxplotVGram(look, breaks=brk, plot.args=list(type="o"))
plot(look, add=TRUE, breaks=brk, col=4)

#
# 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=5, dx = 1,dy = 1 )

## S3 method for class 'vgram.matrix'
plot(x,...)
```

Arguments

dat	A matrix spacing of rows and columns are assumed to have the same distance.
R	Maximum radius for finding variogram differences assuming that the grid points are spaced one unit a part. Default is go out to a radius of 5.
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.
dy	The spacing of grid points on the Y axis. See additional notes for dx.
x	Returned list 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 sub-setting 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 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 csae.

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, stationatry Wendland covariance function as a function ofdistance. This family is useful for creating sparse covariance matrices.

Usage

```
Wendland(d, theta = 1, dimension, k,derivative=0, phi=NA)

Wendland2.2(d, theta=1)
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.
theta	Scale for distances. This is the same as the range parameter.
dimension	Dimension of the locations
n	Dimension for computing Wendland polynomial coefficients
k	Order of covariance function.
derivative	Indicates derivative of covariance function
phi	Depreciated argument will give stop if not an 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 station.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,theta] and zero beyond theta. It is further normalized in these fields functions to be 1 at 0. The parameter k detemines the smoothness of the covariance at zero. The additional parameter n or dimension is needed because the property of positive definitness for radial functions depends on the dimension being considered.

The polynomial terms of the Wenland 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

Plots quickly, 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 depreciated 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 1 million. Also, only countries reporting all the covariables are included.

Usage

```
data(WorldBankCO2)
```

Format

This a 75X5 matrix with the row names identifying countries and columns 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: Precentage of the population living in an urban environment
- CO2.cap: Equivalent CO2 emmissions 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. Cli- mate Research 38, 17-29.

Creating dataset

Listed below are scripts to create this data set from spread sheet on the World Bank CDs:

```
## read in comma delimited spread sheet
  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 ploting 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 vertical lines.
...	Any ploting 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