The fields Package

February 9, 2009

Version 5.02

Date February 6, 2009

Title Tools for spatial data

Author Reinhard Furrer, Douglas Nychka and Stephen Sain

Maintainer Doug Nychka <nychka@ucar.edu>

Description Fields is for curve, surface and function fitting with an emphasis on splines, spatial data and spatial statistics. The major methods include cubic, robust, and thin plate splines, multivariate Kriging and Kriging for large data sets. One main feature is any covariance function implemented in R can be used for spatial prediction. There are also useful functions for plotting and working with spatial data as images. This package also contains an implementation of a sparse matrix methods for large data sets and currently requires the sparse matrix (spam) package for testing (but not for the standard spatial functions.) Use help(fields) to get started and for an overview.

License GPL (>= 2)

URL http://www.image.ucar.edu/Software/Fields

Depends methods, spam

R topics documented:

`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
```

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 described in at http://www.image.ucar.edu/Data/US.monthly.met. Observed monthly precipitation, min and max temperatures for the conterminous US 1895-1997. See also http://www.image.ucar.edu/Data/US.monthly.met/CO.shtml for an on line document of this Colorado subset. 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.id alphanumeric station id codes

CO.loc locations in lon/lat

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

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")
```

Exponential, Matern, Radial Basis

Covariance functions

Description

Functional form of covariance function assuming the argument is a distance between locations.

Usage

```
Exponential(d, range = 1, alpha = 1/range, phi = 1)
Matern (d , scale = 1, range = 1,alpha=1/range,
     smoothness = 0.5, nu= smoothness, phi=scale)
RadialBasis(d,M,dimension)
```

Arguments

d	Vector of distances
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
scale	Same as phi
phi	Marginal variance.
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

Details

Exponential:

phi* exp(-d/range)

Matern:

phi*con*(dn̂u) * 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 phi=1.0 and d=0.

Radial basis functions:

C.m,d d(̂2m-dim) dim- odd

C.m,d d(̂2m-dim)ln(d) dim-even

where C.m.d is a constant based on spline theory. See radbas.constant.

Value

A vector of covariances.

Author(s)

Doug Nychka

References

Stein's book

See Also

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

`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.

`object`	A Krig object produced by the Krig (or Tps) function.
`x0`	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( ozone$x, ozone$y)
A<-Krig.Amatrix( tps.out, ozone$x)
test<- A%*%ozone$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` *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, predict.se, 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 = 80, scale.type = "user", x.center =
                  rep(0, ncol(x)), x.scale = rep(1, ncol(x)), rho = NA,
```

```
                    sigma2 = NA, method = "GCV", verbose = FALSE, mean.obj
                    = NA, sd.obj = NA, null.function =
                    "Krig.null.function", wght.function = NULL, offset =
                    0, outputcall = NULL, 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

x	Matrix of independent variables. These could the locations for spatial data or the indepedent variables in a regression.
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.
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.
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.
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).
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.
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.
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.)

`weights` Weights are proportional to the reciprocal variance of the measurement error. The default is equal weighting i.e. vector of unit weights.

`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.

`nstep.cv` Number of grid points for the coarse grid search to minimize the GCV RMLE and other related criterian for finding lambda.

`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.

`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.

`rho` Scale factor for covariance.

`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.

`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.

`verbose` If true will print out all kinds of intermediate stuff. Default is false, of course as this is used mainly for debugging.

`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.)

`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)

`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)

`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.

`outputcall` If NULL the output object will have a $call argument based on this call. If no NULL the output call will have whatever is passed. This is kludge for the Tps function so that it return a Krig object but have the right call argument. Sorry no one promised that fields would be pretty.

`cov.args` A list with the arguments to call the covariance function. (in addition to the locations)

`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.

`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.

`wght.args` Optional arguments to be passed to the weight function (wght.function) used to create the observation weight matrix.

`W` The explicit observatoin weight matrix.

`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.

`give.warnings`
If TRUE warnings are given in gcv grid search limits. If FALSE warnings are not given. Best to leave this TRUE!

`...` 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= 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 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.

If these 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.

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.

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.
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, predict.se.Krig, predict.surface.se, predict.surface, 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(ozone$x, ozone$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<- predict.surface( 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.
 predict.se( fit, xnew)

# surface of standard errors on a default grid
 predict.surface.se( fit)-> out.p # this takes some time!
 surface( out.p, type="C")
 points( fit$x)

# Using anohter stationary covariance.
# smoothness is the shape parameter for the Matern.

fit <- Krig(ozone$x, ozone$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)
#

# take a look ...
 surface(fit.flame, type="I")

#
# 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( ozone$x, ozone$y,cov.function="Rad.cov",
                      m=2,p=2,scale.type="range")

# See also the Fields function Tps
# out  should be identical to  Tps( ozone$x, ozone$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.
#

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

```
# 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
# Compare the function Tps to the call to Krig given above
# fit tps surfaces to the mean and sd  points.
# (a shortcut is being taken here just using the lon/lat coordinates)

 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
#NOTE: there are some missing values for day 16.

 fit<- Krig( ozone2$lon.lat, y16,
            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")

#
#
# explore some different values for the range and lambda using REML
 theta <- seq( 300,400,,10)
 PLL<- matrix( NA, 10,80)
# the loop
 for( k in 1:10){

# 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$lon.lat[good,], y16[good],
             cov.function="stationary.cov",
             theta=theta[k], mean.obj=mean.o3, sd.obj=sd.o3,
             Covariance="Matern",smoothness=.5,
             Distance="rdist.earth", nstep.cv=80,
             give.warnings=FALSE)$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

 }

# see the 2 column of $gcv.grid to get the effective degress of freedom.

 cat( "all done", fill=TRUE)
 contour( theta, 1:80, PLL)
```

`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( ozone$x, ozone$y)-> out

Krig.engine.default( out)-> stuff

# compare "stuff" to components in out$matrices

Krig.coef( out)$c
# compare to out$c

Krig.coef( out, yM = ozone$y)$c
# better be the same even though we pass as new data!
```

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

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

Description

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 but near equally in a more appropriate projection centered on the model domain. Precipitation is in a log 10 scale where values smaller than 4.39e-5 (the .87 quantile) have been been set to this value.

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

Units are degrees with longitude being 0-360 westward from the prime meridian. (See the shift argument in world for overlaying world map.) 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 `www.image.ucar.edu/Data` for more information about these data.

To facilatate an animation 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, shift=TRUE, lwd=2)
```

RMprecip *Monthly total precipitation (mm) for August 1963 in the Rocky Mountain Region and some gridded 4km elevation data sets.*

Description

RMprecip is a useful spatial data set of moderate size consisting of 865 locations. See www.image.ucar.edu/Data for the source of these data. 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.

elev Station elevation in meters.

y Monthly total precipitation in millimeters. for August 1963

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-latitude grid at approximately 4km resolution

elev 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 ftp://ftp.ncdc.noaa.gov/pub/data/prism100 archived at the National Climate Data Center. This work was primarily authored by Chris Daly 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.

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))
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)

par( save.par)
image.plot(RMelevation, col=topo.colors(256))
US( add=TRUE, col="grey", lwd=2)
title("PRISM elevations (m)")
```

`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 dimensional surface. This is a special case of the spatial process estimate. A "fast" version of this function uses a compactly supported Wendland covariance and computes the estimate for a fixed smoothing pararmeter.

Usage

```
Tps(x, Y, m = NULL, p = NULL, scale.type = "range", ...)

fastTps(x, Y, m = NULL, p = NULL,  theta,  ...)
```

Arguments

To be helpful, a more complete list of arguments are described that are the same as those for the Krig function.

`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`	Exponent for radial basis functions. Default is 2m-d.
`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 Wendlend compactly supported covariance. The covariace (i.e. the radial bais function) is zero beyond range theta.
`...`	Any argument that is valid for the Krig function. Some of the main ones are listed below.

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.

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.

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.

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

A thin plate spline is 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 currenty support the knots argument where one can use a reduced set of basis functions. This is mainly to simplify and a good alternative using knots would be to use a valid covariance from the Matern family and a large range parameter.

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

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 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 x<- scale(x) might be a good idea for putting the varaibles on a common sacel 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.

Value

A list of class Krig. This includes the predicted surface of fitted.values and the residuals. The results of the grid search minimizing the generalized cross validation function is 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, summary.Krig, predict.Krig, predict.se.Krig, plot.Krig, mKrig surface.Krig, sreg

Examples

```
#2-d example

fit<- Tps(ozone$x, ozone$y)  # fits a surface to ozone measurements.

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

summary(fit)

# 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 spline estimate is uses 7.5 df.
```

```
fit1<- Tps(ozone$x, ozone$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<-predict.surface( fit)
image( out.p)
surface(out.p) # perspective and contour plots of GCV spline fit
# predict at different effective
# number of parameters
out.p<-predict.surface( fit,df=10)


#1-d example
out<-Tps( rat.diet$t, rat.diet$trt) # lambda found by GCV
plot( out$x, out$y)
lines( out$x, out$fitted.values)

#
# compare to the ( much faster) one spline algorithm
#  sreg(rat.diet$t, rat.diet$trt)
#

# Adding a covariate to the fixed part of model
# Note: this is a fairly big problem numerically (850+ locations)

Tps( RMprecip$x,RMprecip$y, Z=RMprecip$elev)-> out

surface( out, drop.Z=TRUE)
US( add=TRUE, col="grey")

###
### fast Tps

# m=2   p= 2m-d= 2
#
# Note: theta =3 degrees is a very generous taper range.
# Use trials with rdist.nearest to sort an efficient tapre range
# for large spatial problems

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

```
#
# 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")

#
#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)
```

US *Plot of the US with state boundaries*

Description

Plots quickly, medium resolution outlines of the US with the states and bodies of water.

Usage

```
US( xlim=c(-124.7, -67.1), ylim = c(25.2, 49.4), add=FALSE,shift=FALSE, ...)
```

Arguments

ylim	range of latitudes
xlim	range of longitudes
add	If true will add the world map to current plot
shift	If TRUE shifts to be centered on the Dateline and longitude runs from 0 to 360. If FALSE centers on Prime Meridian and longitude runs from -180 to 180.
...	These are graphical arguments that are passed to the lines function that draws outline.

Details

This function uses the FIELDS dataset US.dat for the coordinates.

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.

`W.info` *Gives indexing imfomration for a wavelet decompostion*

Description

Functions for finding various indices and sizes of different parts of a 1-d multiresolution and converting to a sequential index.

Usage

```
W.info(m = 128, cut.min = 4)

W.i2s(ind, m, cut.min)

W.s2i(i, level, m, cut.min)
```

Arguments

`m`	Length of series
`cut.min`	Number of scale basis functions (or father wavelets) at the coarsest resolution)
`ind`	Vector of indices of the basis functions.
`i`	Position of the basis function within its level.
`level`	level of resolution (father wavelets have level==0).

Details

W.info gives summary information about the multiresolution. W.i2s converts a sequential index for the basis functions into a list with locations and levels. W.s2i is the inverse transformation from a list structure to an integer index.

Value

Return list for W.info

`m`	length of series
`cut.min`	cut.min (what else!)
`S`	first and last indices for father wavelets
`H`	a matrix where rows are levels of resolution and columns are the first and last indices for the mother basis functions.
`L`	number of basis funcion in each level of the mother wavelets.
`Lmax`	The number of levels of resolution
`offset`	A matrix where rows are levels of resolution and the columns is the offset index for the beginning of the indices for basis funtion at that resolution level.

Author(s)

Doug Nychka

See Also

`Wtransform`, `Wtransform.image`, `plot.Wimage`, `Wimage.info`

Examples

```
# series of length 64 where the coarsest level is 8 father basis functions.
W.info(64, 8)

#index for 4th basis location at the 2nd level

W.s2i( 4, 2, m=64, cut.min=8)
# location and level for the 48th basis function.
W.i2s( 48, m=64, cut.min=8)
```

`Wendland` *Wendland family of covariance 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=1)

Wendland2.2(d, theta=1)

wendland.coef(d,k)
```

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
`k`	Order of covariance function.
`derivative`	Indicates derivative of covariance function
`phi`	Scale factor to multiply the function. Equivalent to the marginal variance or sill if viewed as a covariance function.

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 Wendland covariance function is a polynomial on [0,theta] and zero beyond theta. The parameter `k` detemines the smoothness of the covariance at zero. The polynomial coefficients are computed recursively based on the values of `k` and `dimension` in the function `wendland.coef`. The polynomial is evaluated using `fields.evlpoly`.

A specific example of the Wendland family is `Wendland2.2` and this is included mainly for testing.

Value

A vector of the covariances or its derivative.

Author(s)

Doug Nychka

See Also

wendland.cov, stationary.taper.cov

Examples

```
DD<- seq( 0,1.5,, 200)

y<- Wendland( DD, k=2, dimension=2)

plot( DD, y, type="l")

# should agree with

y.test<- Wendland2.2( DD)
```

`Wimage.cov` *Functions for W-transform based covariance models*

Description

These functions are designed for 2-d Gaussian random fields on a large regular grid. They require a sparse or block diagonal model for the covariances among the wavelet coefficients. Wimage.cov multiplies a vector with the implied covariance matrix and Wtransform.image.cov has similar functionality to Wimage.cov but is the correct form for fitting surfaces using `kirg.image`. Wimage.sim generates a random field with the implied covariance.

Usage

```
Wimage.cov(Y = NULL, ind = NULL, H.obj, find.row = FALSE)
Wtransform.image.cov(ind1, ind2=ind1, Y, cov.obj)
Wimage.sim( H.obj)
```

Arguments

`Y`	The vector for the multiplication, if ind is missing this should a matrix with the same dimensions as the grid locations.
`ind`	If `Y` is not full grid `ind` gives the index location for the Y subset. See details below for indexing conventions.
`H.obj`	A list with components need to describe the grid size and components of H. See details below for a description and the example.
`find.row`	If true the row of the covariance given by ind is returned.
`ind2`	This is the same as`ind` in `Wimage.cov`.
`ind1`	Indices giving subset of the full grid for the result of the multipication. (See details below). Note that the default is that ind2 is set to ind1.
`cov.obj`	This is the same as H.obj above.

Details

Note that Wimage.cov only returns the product of a vector times the covariance matrix. This is usually all that is really needed and finesses the memory problems of dealing with very large covaraince matrices. But see the last example for a loop to extract a submatrix of the covariance.

In notation, if Sigma is the full covariance matrix among all grid points. and Y is a vector then the intended calculation in R syntax is: `Sigma[ind1, ind2]%*% Y`. This is accomplished by padding Y with zeroes to be the full vector/grid, doing the full multiplication of Sigma and then throwing away all elements but those in ind1. Note that for Wimage.cov, ind1 is assumed to be the full vector/grid and ind is equal to ind2 (when find.row is FALSE).

The way 2-d wavelet basis functions are indexed can be confusing. See the help files for Wimage.info and Wtransform.image for more background. For these functions, ind can either be a single vector and the index refers to the grid points in column stacked (or "vec") format.

If ind has a two columns these refer to the row/column of the grid. Currently only one sparse representation for H is implemented, a block/diagonal strategy. The covariance has the form Wi*H *H * Wi.t. Where Wi is mnXmn the matrix of wavelet basis functions with each column being a basis evaluated on the gird points and in stacked column format. Wi.t is its transpose. H is represented as partitioned matrix where H0 is a full matrix and the remaining rows only have a diagonal term. If ind0 denotes the indices for the elements of H0 then in R notation:

H0 = H[ind0, ind0]

To simplify the coding the diagonal elements are represented as an mXn matrix with those locations corresponding to H0 set to 1. Accordingly, in R notation

H1= diag(H), H1[ind0] <- 1

With this representation, multiplying H by a vector u becomes

u[ind0]<- H0%*% c(u[ind0])

u*H1

Note that u is assumed to be an mXn matrix based on the grid dimensions.

Under the assumptions the grid (or image) is mXn the components for H.obj are `m` , `n`, `cut.min`, specifying the coarsest level of wavelet basis functions, H0, ind, and H1.

By altering the multiplication of H within these and changing the H.obj or cov.obj list one can easily create other sparse representations for the covariance.

Value

For Wimage.cov and Wimage.sim an mXn matrix with the same extent as the grid defined by H.obj.

For Wtransform.image.cov a vector with the same length as `ind1` and at these grid locations.

Author(s)

Doug Nychka

See Also

Wtransform.image, Wimage.info, Exp.image.cov, matern.image.cov

Examples

```
# This example  creates a block/diagonal for H that is close to a
# Matern covariance
# The example while lengthy is a practical run through of the process
# of creating a reasonable sparse representation for H.

M<- 16 # a 16X16 grid of locations
MM<- M**2
x<- 1:M
y<- 1:M
grid<- list( x=x, y=y)
cut.min<- 4
# H0 is father and mothers at first level
# with cut.min=4

# get indices associated with H0
I<-rep( 1:cut.min, cut.min)
J<- sort( I)
Wimage.s2i(I, J, flavor=0, level=0, cut.min=cut.min, m=M, n=M, mat=FALSE)-> ind0
Wimage.s2i(I, J, flavor=1, level=1, cut.min=cut.min, m=M, n=M, mat=FALSE)-> ind1
Wimage.s2i(I, J, flavor=2, level=1, cut.min=cut.min, m=M, n=M, mat=FALSE)-> ind2
Wimage.s2i(I, J, flavor=3, level=1, cut.min=cut.min, m=M, n=M, mat=FALSE)-> ind3
IND<- c( ind0, ind1, ind2, ind3)
NH<- length( IND)   # H0 will be NH by NH in size.

# the Matern covariance function range=3, smoothness=1.
cov.obj<- stationary.image.cov( grid=grid, setup=TRUE, theta=3, smoothness=1)

# find elements of D0= H0**2  using the formula
#  D0 = W Sigma W.t  where W is the  2-d W transform ( the inverse of W.i above).
#  and Sigma is the Matern covariance
#
# the following looping algorithms are a way to avoiding explicitly creating
# gigantic covariance matrices and wavelet basis matrices associated with large
# grids. Of course in this example the grid is small enough (16X16) that the
# matrices could be formed explicitly
#

D0<- matrix( 0, NH,NH)
# fill in D0
 for ( k in 1:NH){
 tmp<- matrix( 0, M,M)
 tmp[IND[k]] <- 1
 hold<- Wtransform.image( tmp,  transpose=TRUE, cut.min=cut.min)
 hold<- stationary.image.cov(Y=hold, cov.obj=cov.obj)
 hold<- Wtransform.image(hold, cut.min=cut.min)
 D0[k,] <- hold[IND]
}

# sqrt of D0
temp<- eigen( D0, symmetric=TRUE)
```

```
# next line should be
#  H0<-temp$vectors

H0<- temp$vectors %*% diag(sqrt(temp$values))%*% t(temp$vectors)

# find H1
H1<- matrix(0,M,M)
for ( k in 1:MM){
 tmp<- matrix( 0, M,M)
 tmp[k] <- 1
 hold<- Wtransform.image( tmp,  transpose=TRUE, cut.min=cut.min)
 hold<- matern.image.cov(Y=hold, cov.obj=cov.obj)
 hold<- Wtransform.image(hold, cut.min=cut.min)
 H1[k] <- sqrt(hold[k]) }
# remember that H1 has the H0 diagonal values set to 1.
H1[IND] <- 1

#OK good to go.  Create the H.obj list
H.obj<- list( m=M, n=M, ind0=IND, cut.min=cut.min, H0=H0, H1=H1)
#
#
#

# mutliply the covariance matrix by a random vector
tmp<- matrix( rnorm( 256), 16,16)
Wimage.cov( tmp, H.obj=H.obj)-> look

# generate a random field
Wimage.sim(H.obj)-> look
image.plot( look)

#A check that this really works!
#find  the  135 =(9-1)*16 +7 == (7,9) row of covariance matrix
# we know what this should look like.

Wimage.cov( ind= 135, H.obj=H.obj, find.row=TRUE)-> look
image.plot( x,y,look) # the covariance of the grid point (7,9) with
                      # all other grid points -- a bump centered at (7,9)

#multiply a vector by just a subset of Sigma

ind<- sample( 1:256,50) # random set of 50 indices
Y<- rnorm( 50) # random vector of length 50
Wimage.cov(Y, ind= ind, H.obj=H.obj)[ind]-> look

# look is now of length 50 -- as expected.
# In R notation, this finding Sigma[ind, ind]

# OK suppose you really need Sigma[ind,ind]
# e.g. in order for   solve( Sigma[ind,ind], u)
# here is a loop to do it.
```

```
Sigma<- matrix( NA, 50,50)
for ( k in 1:50){
# for each member of subset find row and subset to just the columns identified by ind
Sigma[k,] <- c( Wimage.cov( ind= ind[k], H.obj=H.obj, find.row=TRUE)[ind])
}
```

`Wimage.info` *Finds key indices related to a 2-d multiresolution*

Description

Functions for finding the indices and other information for a 2-d basis in a multiresolution wavelet decomposition.

Usage

```
Wimage.info(m = 128, n = m, cut.min = 4)

Wimage.i2s(ind, m, n, cut.min)

Wimage.s2i(i, j, level, flavor, m, n, cut.min, mat = TRUE)
```

Arguments

`m`	Nmber of rows of image (x)
`n`	number of columns of image (y)
`cut.min`	The minimum number of father basis functions along one axis.
`ind`	indices for the basis functions can either be a vector or 2 column matrix.
`i`	Vector of row locations for basis functions. Or a list with components i, j, level and flavor.
`j`	Vector of column locations for basis functions.
`level`	Resolution level of basis functions.
`flavor`	Type of basis function (0=S, 1=H, 2=V, 3=Di). See details below.
`mat`	If TRUE returned index will be a matrix where the basis functions are indexed by a row and column.

Details

The wavelet coefficents are computed efficiently as a single large matrix/image but this format make it difficult to identify the indices ofspecific kinds of basis functions. The wavelet basis functions as found with a tensor product multiresolution. (e.g. `Wtransform.image`) are orgainzied in levels of resolution and type. At the coarsest level of resolution are father wavelets (dentoed by

S for "smooth") roughly centered on a rectanglar grid of locations (the size of the grid controlled by `cut.min`). At this resolution are three mother wavelets that capture horizontal (H), vertical (V) and diagonal(Di) structure at this scale. These basis functions are also centered on a lattice of grid locations. Subsequent levels only use the H,V, Di mother wavelet templates. For example given a 32X32 image with cut.min=4 There are three levels of resolution and 1024 basis functions total. The coarsest level is 16 S basis functions on a regular 4X4 grid, 16 H, 16 V and 16 Di basis function also on 4X4 grid. The next level has 64 H, V and Di basis functions on an 8X8 grid and the final level has 256 H, V and Di basis functions on a 16X16 grid, giving a grand total of 1024 basis functions.

Without some additional calculations it is possible to organize the results in different resolutions. This function provides the necessary indexing information to do this. See the function `plot.Wimag` as an exmaple of how this is used.

Indexing the image can happen in 3 ways. 1)as a structure where one specifies the location (i,j) , level, and flavor. 2) as the row and column indices of the image. 3) as a single index if the image is stacked as a long vector. The functions Wimage.s2i and Wimage.i2s convert indices between these forms.

Value

Wimage.info returns a list where all indices pertain to a location or subsets of the mXn matrix of wavelet coefficients.

`m`	Number of rows in image
`n`	Number of columns of image
`cut.min`	cut.min
`S`	A vector with 4 elements. S[1],S[2] give the min/max row indices for the father wavelets and S[3],S[4] give the min/max column indices for the father wavelets.
`H`	A matrix where each row is a resolution level and cloumns give s ranges of row and column indices. e.g. H[k,1:4] gives the ranges for the rows and columns for the horizontal basis functions at level k.
`V`	A matrix in teh same format as H that gives subsets for the vertical basis functions.
`Di`	A matrix in the same format as H that gives subsets for the diagonal basis functions.
`L`	A column matrix where rows index resolutaion and column give the grid size for each level of resolution.
`Lmax`	Total number of resolution levels.
`offset.m`	A 3 column matrix that has the row offsets that indicate the begining of a block of coeffiecients. Rows are levels of resolution and columns are H,V,Di.
`offset.n`	A 3 column matrix that has the column offsets that indicate the begining of a block of coeffiecients. Rows are levels of resolution and columns are H,V,Di.

Author(s)

Doug Nychka

See Also

plot.Wimage, Wtransform.image

Examples

```
#Find a basis function.
# For a basis on a 64X64 image with cut.min=8 find a
# horizontal basis function at second level of resolution in the (4,4)
# position. ( There are 16X16 horizontal basis functions at the 2 nd level
#
Wimage.s2i( i=3,j=2, level=2, flavor=1, m=64, n=64, cut.min=8)-> ind
tmp<- matrix( 0, 64,64)
tmp[ind] <- 1
Wtransform.image( tmp, cut.min=8, inv=TRUE)-> look
image.plot( look)

# A check of Wimage.i2s
Wimage.i2s( ind,m=64, n=64, cut.min=8)
# should get i=3,j=2, level=2, flavor=1

# complete check of functions
 Wimage.i2s( 1:512, cut.min=8, m=16, n=32)-> look
 Wimage.s2i( look, cut.min=8, m=16, n=32, mat=FALSE)-> look2
 sum( look2 - (1:512)) # sum should be zero

# W transform of John Lennon image

     data(lennon)
     m<- nrow(lennon)
     n<- ncol(lennon)

# add an grey strip down middle columns

     lennon[ , 128:132]<- 120

     look<- Wtransform.image( lennon, cut.min=8)

# get info
     info<- Wimage.info( n, m, cut.min=8)

# Zero out all V basis functions coefficients,

tmp<- look

Vind<- info$V
for (lev in 1: info$Lmax){
tmp[ Vind[lev,1]: Vind[lev,2], Vind[lev,3]: Vind[lev,4]]<- 0
}

look2<- Wtransform.image( tmp, cut.min=8, inv=TRUE)

# take a look at vertically  filtered image
```

```
set.panel( 2,1)
par( pty="s")
image( lennon, col=grey(seq(0,1,,256)))
image( look2, col=grey(seq(0,1,,256)))

# NOTE:
# What is vertical and horizontal is confusing here
# due to the convention of the R image plot of running rows
# of the image as the X coordinate
# The grey column in lennon image is plotted as
# a horizontal line
#
```

`Wimage.info.plot` *Plot to check 2-d multiresolution indexing*

Description

Plots in image format the differnt blocks of coefficents associated with a 2-d multiresolution. This function can be used to check the Wtransform indexing functions and is also an introduction to how the coefficients are organized.

Usage

```
Wimage.info.plot(m, n, cut.min)
```

Arguments

`m`	Number of rows in image
`n`	Number of columns of image
`cut.min`	Smallest number of smooth basis functions. Coarsest resolution will have atleast cut.min X cut.min basis functions with support on a regular grid.

Details

This function was used to check the (compicated) indexing functions work. But it also might a useful graphics to desribe how the different levels of wavelet coefficients are pack into the image format.

Author(s)

Doug Nychka

See Also

`Wtransform.image`, `W.info`, `Wimage.info`,

Examples

```
#
# 64X 128 image coarsest level has 8X16 smooth basis funcitons.
#
# NOTE as a matrix the image plot is upside down! Rows are along X-axis and
# columns on Y
# e.g. the (1,1) element is the lower left corner.
#

Wimage.info.plot( 64, 128, cut.min=8)
```

`Wtransform` *Quadratic W wavelet transform for 1-d vectors or rectangular or cylindrical images*

Description

Finds the forward or inverse discrete wavelet transform using the quadratic W basis.

Usage

```
Wtransform(x, inv = FALSE, transpose = FALSE, cut.min = 8)

Wtransform.image(x, inv=FALSE, transpose=FALSE, cut.min=8)

Wtransform.cylinder.image(x, inv=FALSE, transpose=FALSE,
byX=TRUE,cut.min=8)
```

Arguments

`x`	Matrix to be transformed. For Wtransform the number of rows must be dyadic or m=p*2L̂ where p is lees than or equal to cut.min. A 1-d transformation is effected on each column of x. For the 2-d transforms both the row and the column dimensions must satisfy this condition. A 2-d transformation is applied to the image.
`inv`	If true will compute the inverse transform default is false
`transpose`	If true will compute the transpose of transform default is false
`cut.min`	Minimum level of transformation. cut.min=8 means that the coarsest level will consist of 8 scale functions for the 1-d case and 64=8X8 scale functions centered on an 8X8 grid for the 2-d case.
`byX`	If TRUE enforces periodic boundary conditions on horizontal coordinate of the image and if FALSE applies condition to vertical coordinate. (See details.)

Details

The wavelet transform can be thought as matrix multiplication W %*% vec(x) where vec(x) is the matrix x stacked by columns. The inverse transform is inv(W) %*% vec(x) and transpose is t(W) %*% vec(x). One can interpret the columns of inv(W) as basis functions and they follow the usual pattern of translations and dilations of mother and father wavelets. (See example below.)

Another way of thinking of the transformation is by recursion: apply a smoother and a "rougher" to a vector and taking every other value. Now apply the same operation to the smooth results, now half the length of the previous vector. At each step one reduced the vector by a half and cut.min specifies the size when to stop. The function `WQS` performs one step of the recursion with smoother weights (-1, 3, 3, -1) and roughening weights (-1, 3, -3, 1) away from the edges. `WQS` actually works for a matrix where each column is transformed in this manner. Boundary adjustments are made to preserve the shape of the basis functions. (See example below.) By convention the returned matrix has the smoothed results in the first half of the columns and the rough results in the second half. The discrete wavelet transform performs this operation recursively on the smoothed results until the smoothed vector is less than a set size. Because each step reduces the size of the vector by 2 it only makes sense to apply this algorithm to vectors whose length is dyadic or the product of a small integer and a dyadic (e.g. 96=3*32). The precise tests are done by `dyadic.check` and `dyadic.2check` and if n is the dimension the constraint it that n=p*2L̂ where L is less or equal to than cut.min. The transform will result in p scale (father) basis functions if cut.min is equal to p.

At the end of the day this recursive algorithm defines a linear transformation of the original image to something that we call the wavelet decomposition. This is the full matrix W mentioned above. It is also possible to express multiplication of inv(W) and transposes by a similar recursive scheme and related sets of filter weights. (Try `WQSi( WQS(x))` as a test.) The reader is referred to `WQSi` `WQS.T` and `WQSi.T` for the filtering primitives. Finally it should be noted that the particular wavelets chosen here are not orthogonal by have nice smooth properties and the father wavelet resembles a Gaussian density while the mother looks like its derivative. (See example below for some plots of the implied basis functions.) Note that `Wtransform` is "vectorized" so that with little extra overhead one can do transforms for many separate 1-d series with one call. In particular, `Wtransform(diag(1,128),inv=TRUE)` will neatly generate the W matrix given above.

For two dimensions for the basic step one applies WQS to the columns of the matrix and then to the rows. e.g. `t(WQS(t(WQS(x))))`. This primitive step is implemented in WQS2d. The final algorithm calls WQS2d or its variants repeatedly on a matrix that decreases by a factor of two in size along each dimension at each iteration.

If the cylinder variant is specified the transform uses periodic boundaries on one of the coordinates. This is useful for data on zonal section of a sphere where a constant line of latitude should be periodic. For byX=T the wavelet basis functions wrap on the x-axis when an image plot is made. (See example below.) This convention may cause some confusion because R experts will know that the image plot rotates the the matrix so that the (1,1) element is at the lower left corner. Thus enforcing periodicity along the X-coordinate of the image pertains to the columns of the matrix used to represent the image in R. (Compare `matrix( 1:10,2,5)` to `image( matrix( 1:10,2,5))`.

Value

A matrix the same size as x.

References

Nychka,D. , C. Wikle and J.A. Royle, (2002) Multi-resolution models for nonstationary spatial covariance functions Statistical Modelling 2 315-332.

See also the original report on W matrices:

Man Kam Kwong and P. T. Peter Tang, "W-Matrices, Nonorthogonal Multiresolution Analysis, and Finite Signals of Arbitrary Length," Preprint MCS-P449-0794, September 1994

ftp://info.mcs.anl.gov/pub/tech_reports/reports/P449.ps.Z

See Also

Wtransform.image, WQS, WQSi

Examples

```
# W transform of a simple function
x<- seq( 0,1,256)
y<- x*(1-x)**3 + ifelse( x>.3, x,0)

Wtransform( y)-> out

# plot the implied wavelet basis functions
ID<- diag( 1, 256)
WQS.basis( 256)-> Wbasis
set.panel(2,2)
matplot( 1:256, Wbasis[,1:8], type="l", lty=c(1,2), col=2)
title("Father")
matplot( 1:256, Wbasis[,9:16], type="l", lty=c(1,2), col=2)
title("Mother")
matplot( 1:256, Wbasis[,17:32], type="l", lty=c(1,2), col=2)
title("Mother scaled by 2")
matplot( 1:256, Wbasis[,33:64], type="l", lty=c(1,2), col=2)
title("Mother scaled by 4")

set.panel( 1,1)

# test that the transform works

# Precise definition of what the transform is doing in terms of
# explicit matrix multiplication  all of
# these should be machine zero
# Note that the direct matrix multiplications will be substantially slower
# for large vectors.
# y<- rnorm( 256)
# y<- y /sqrt(mean( y**2))

#sqrt(mean( c( Wtransform(y, inv=TRUE) - Wbasis
#sqrt(mean( c(Wtransform(y, inv=TRUE, transpose=TRUE) - t(Wbasis)
#sqrt(mean( c(Wtransform(y) - solve(Wbasis)
#sqrt(mean( c(Wtransform(y, transpose=TRUE) - t(solve(Wbasis))
```

```
#
## 2-d examples
#

# Wtransform of John Lennon image
data(lennon)
look<- Wtransform.image( lennon)
#
### take a look:
 image.plot( look)


# a diagonal detail basis function
# we find this by just multipling W by a unit vector!

temp<- matrix(0, nrow=32, ncol=32)
temp[8,5]<- 1
look<- Wtransform.image( temp , inv=TRUE, cut.min=4)
image( look)
title("diagonal detail W-wavelet")

#just for fun: redo this example for all indices up to 8!
#
#set.panel( 8,8)
#par( mar=c(0,0,0,0))
#for (  k in (1:8)){
#for ( j in (1:8)){
#temp<- matrix( 0 , nx,ny)
#temp[k,j] <- 1
#Wtransform.image( temp, inv=T, cut.min=cut.min)-> look
#image( look, axes=FALSE, xlab="", ylab="")
#box()
#}
#}

# examine a basis function to see periodic condition enforces along
# X axis of image.
temp<- matrix(0, nrow=32, ncol=32)
temp[8,5]<- 1
image( Wtransform.cylinder.image( temp , inv=TRUE, cut.min=4))
# now along Y-axis
image( Wtransform.cylinder.image( temp , inv=TRUE, cut.min=4, byX=FALSE))

# reset panel
set.panel( 1,1)
```

`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 a2 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, nrow=64, ncol=64,weights=NULL,
 na.rm=FALSE, nx=NULL,ny=NULL, boundary.grid=FALSE)
```

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	Number of rows in image matrix (x-axis direction)
ncol	Number of columns in image matrix (y-axis direction)
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	Same as nrow
ny	Same as ncol
boundary.grid	If FALSE grid points are assumed to be the midpoints. If TRUE they are the grid box boundaries.

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, nrow=50, ncol=50)
image.plot( look)

# number of obs in each cell -- in this case equal to the
# aggregated weights because each obs had equal wieght 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, 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.
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
```

---

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,style = "tukey", outlier = TRUE, plot = TRUE, ...)
```

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.
style	Type of boxplot default is "tukey". The other choice is "quantile" where the whiskers are drawn to the 5 and 95 percentiles instead being based on the inner fences.
outlier	If true outliers (points beyond outer fences) will be added to the plots.
plot	If false just returns a list with the statistics used for plotting the box plots.
...	Other arguments controlling the boxplots (passed to bplot.obj) these are listed below. Other graphical plotting arguments not matched (e.g. yaxt) are used in the call to plot to set up the initial plot if add=T.

pos The boxplots will be plotted vertically and pos gives the x or y locations for their centers. If omitted the boxes are equally spaced at integer values.

width Width of boxplots (in user coordinates) if omitted then the width is a reasonable fraction of the distance between boxes and is set by the space argument.

labels Labels under each boxplot. If missing the columns names or components of x are used.

las Orient the axis labels for bplot groups. Default is to put them horizontal (las=1) if number of groups is less than 7 otherwise make them perpendicular (las=2) to keep the labels from running into each other. See help(par) for more about this option.

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.

space Space between boxplots.

sort.names If true plot the boxplot data set names are sorted in alphabetic order by their labels.

xlab Label for the x-axis

ylab Label for the y-axis

label.cex Boxplot label size where 1.0 is normal size characters. If zero labels will not be added.

xaxt Plotting parameter for x-axis generation. Default is not to produce an x-axis.

horizontal If true draw boxplots horizontally the default is false, produce vertical box plots.

lwd Width(s) of lines in box plots.

col Color(s) of bplots.

Details

This function was created as a complement to the usual S function for boxplots. The current function makes it possible to put the boxplots at unequal x or y positions. 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. If 5 or less observations are in a group the points themselves are plotted instead of a box.

The function is broken into two steps: a call to stats.bplot to find the statistics and a call to bplot.obj to plot the resulting object. The user is referred to describe.bplot to modify the statistics used and to draw.bplot.obj to modify how the bplot is drawn.

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

See Also

describe.bplot, draw.bplot.obj, stats.bplot, bplot.xy, bplot.obj

Examples

```
#
set.seed(123)
temp<- matrix( rnorm(12*8), ncol=12)
pos<- c(1:6,9:14)
bplot(temp)
#
bplot( temp, pos=pos, labels=paste( "D",1:12), horizontal=TRUE)
#
# boxplots in red
bplot( temp, pos=pos, label.cex=0, horizontal=TRUE, col="red")
# add an axis
axis( 2)
```

`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),
         style = "tukey", outlier = TRUE, plot = TRUE, xaxt =
         "s", ...)
```

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.
`style`	Type of boxplot default is "tukey". The other choice is "quantile" where the whiskers are drawn to the 5 and 95 percentiles instead being based on the inner fences.
`xaxt`	Plotting parameter for x-axis generation. Default is to produce an x-axis.
`outlier`	If true outliers (points beyond outer fences) will be added to the plots.
`plot`	If false just returns a list with the statistics used for plotting the box plots.
`...`	Any other optional arguments passed to the bplot.obj function there are quite a few that are useful, see the help file for bplot for details.

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= .6
set.seed( 123)
x<-rnorm( 2000)
y<- .6*x +  sqrt( 1- .6**2)*rnorm( 200)
#
#
bplot.xy( x,y, breaks=seq( -3, 3,,15) ,xlim =c(-4,4), ylim =c(-4,4))
points( x,y, pch=".", col=3)
```

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)))
```

`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 "spatial.design". 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)
```

`%d*%-methods` *Multiplying diagonal matrices in Package 'fields'*

Description

Recognizes a left vector as the elements of a diagonal matrix and does the right multiplication efficiently (this is not rocket science!). This method is used in the internal functions of Krig to make the code more readable. It avoids having a branch in the source code to handle the diagonal or nondiagonal cases. Note that this operator is not symmetric: a vector in the left argument is interpreted as a diagonal matrix and a vector in the right argument is kept as a column vector.

Methods

x = "matrix", y = "matrix" computes x%*%y

x = "matrix", y = "numeric" computes x%*% diag(y)

x = "numeric", y = "matrix" computes diag(x)%*% y

x = "numeric", y = "numeric" computes diag(x)%*% y, the diagonal matrix x multiplied by a vector.

---

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, ...)

drape.color(z, col = tim.colors(64), zlim = NULL,
   transparent.color = "white", midpoint)
```

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.
...	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 an (M-1)X(N-1) matrix where each element is a text string specifying the color.

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)

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

# 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)
  drape.color( dz, zlim =zlim)-> zcol

# 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)
```

`Covariance functions`

Exponential family, radial basis functions,cubic spline, compactly supported Wendland family and stationary covariances.

Description

Given two sets of locations computes 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.

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, theta = rep(1, ncol(x1)), p = 1, C = NA, marginal=FALSE)

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

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

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, Covariance="Exponential", Distance="rdist",
               Dist.args=NULL, theta=1.0,C=NA, marginal=FALSE,...)

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

wendland.cov(x1, x2, theta = rep(1, ncol(x1)), k = 2, C = NA,
             marginal =FALSE,Dist.args = NULL,
             spam.format = TRUE, derivative = 0)
```

Arguments

`x1`	Matrix of first set of locations where each row gives the coordinates of a particular point.
`x2`	Matrix of second set of locations where each row gives the coordinates of a particular point. If this is missing x1 is used.
`theta`	Range (or scale) parameter. This can be a scalar, vector or matrix. If a scalar or vector these are expanded to be the diagonal elements of a linear transformation of the coordinates. In R code the transformation applied before distances are found is: `x1 %*% t(solve(theta))` or if theta is a scalar: `x1/theta`. Default is theta=1. See Details below.
`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 form. 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 for the distance between locations.

`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 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 "C" option and is mainly called from within a predict function.

`...` 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 as a simple example and implements the R code discussed below. 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.

Functional Form: 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 covariance is found as exp(-(D.ij **p)) where D.ij is the Euclidean 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)**p)

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)**p) )`).

If C is passed then the returned value is

```
exp(-rdist(u, v)**p) %*% C
```

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",smooth
```

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))*Taper( rdist( x1, x2))
```

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 search may find more nonzero pairs than dimensioned by `max.points`. Given this error just pass a larger for `max.points` explicitly. For spam.format TRUE the multiplication with the `C` argument is done with the spam sparse multiplication routines through the "overloading" of the `%*%` operator. Currently this function only supports the Euclidean distance function.

About the FORTRAN: The actual function `Exp.cov` and `Rad.cov` calls 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( ozone$x, theta=100)

# out is a 20X20 matrix

out2<- Exp.cov( ozone$x[6:20,],ozone$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( ozone$x, ozone$y, Covariance="Matern", theta=100,smoothness=2)

data( ozone2)
x<- ozone2$lon.lat
y<- ozone2$y[16,]

# example of calling the taper version directly
# Note that default covariance is exponential and default taper is
# Wendland (k=2).
## Not run:

stationary.taper.cov( x,x, theta=1.5, Taper.args= list(k=2, theta=2.0),
                          mean.neighbor= 200 )-> temp
# temp is a tapered covariance matrix in sparse format.

 is.spam( temp)  # evaluates to TRUE

 temp<-  spam2full(temp) # should be identical to
 temp2<- Exp.cov( x,x, theta=1.5) * wendland.cov(x,x,
                      theta= 2.0*1.5,spam.format=FALSE)

 test.for.zero( temp, temp2)
## End(Not run)

# 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, theta=2.0) )
           ) -> out2
## End(Not run)

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

`fields internal` *Fields internal and secondary functions*

Description

Listed below are supporting fucntions for the major methods in fields.

Usage

```
COR(dat)

D4transform.image(x, inv = FALSE, transpose = FALSE, cut.min = 8)

Krig.df.to.lambda(df, D, guess = 1, tol = 1e-05)
```

```
Krig.fdf (llam, info)
Krig.fgcv (lam, obj)
Krig.fgcv.model (lam, obj)
Krig.fgcv.one (lam, obj)
Krig.find.gcvmin (info, lambda.grid, gcv, gcv.fun, tol, verbose =FALSE,
give.warnings = TRUE)
Krig.find.REML (info, lambda.grid, llike, llike.fun, tol, verbose = TRUE,
give.warnings = FALSE)
Krig.flplike (lam, obj)
Krig.fs2hat (lam, obj)
Krig.ftrace (lam, D)
Krig.parameters (obj, mle.calc=obj$mle.calc)
Krig.replicates (out, verbose = FALSE)
Krig.updateY (out, Y, verbose = FALSE, yM=NA)
Krig.which.lambda(out)
Krig.ynew (out, y=NULL, yM=NULL )


WD4 (x)
WD42d (x)
WD42di (x)
WD4i (x)

WQS (x)
WQS.T (x)
WQS.basis (N, cut.n = 8)
WQS2d (x, transpose = FALSE)
WQS2di (x, transpose = FALSE)
WQSdi (x, transpose = FALSE)
WQSi (x)
WQSi.T (x)
WQS.periodic(x)
WQS.periodic.T(x)
WQS.periodic.basis(N, cut.n = 8)
WQS.periodic.T(x)
WQS2d.cylinder(x, transpose = FALSE, byX=TRUE)
WQS2di.cylinder(x, transpose = FALSE, byX=TRUE)
WQSi.periodic(x)
WQSi.periodic.T(x)

bisection.search (x1, x2, f, tol = 1e-07, niter = 25, f.extra =
        NA, upcross.level = 0)

bplot.obj (data, pos = NA, width = NULL, labels = NULL, las=NULL
            ,add = FALSE, space = 0.25, sort.names = FALSE, xlab = "", ylab = "",
              label.cex = 1, xaxt = "n", outlier = TRUE, horizontal = FALSE,
              lwd=NA,col=NA,...)
```

```
cat.matrix (mat, digits = 8)

cat.to.list (x, a)

ceiling2 (m)

conjugate.gradient (b, multAx, start, tol = 1e-05, kmax =
             25, verbose = TRUE,    ...)

describe (x)
describe.bplot (temp, style = "tukey", outlier = TRUE)

double.exp(x)

dyadic.2check( m,n,cut.p=2)
dyadic.check( n,cut.p=2)

draw.bplot (temp, width, xpos, outlier = TRUE, style = "tukey")
draw.bplot.obj (obj, width, xpos, outlier = TRUE,
             horizontal = FALSE,lwd=NA,col=NA)

Exp.earth.cov (x1, x2, theta = 1)

fast.1way  (lev, y, w = rep(1, length(y)))

find.upcross (fun, fun.info, upcross.level = 0, guess = 1, tol =
1e-05)

gauss.cov (...)

golden.section.search (ax, bx, cx, f, niter = 25, f.extra = NA,
     tol = 1e-05)

grey.level (n=256)

image.plot.info (...)
image.plot.plt(x, add=FALSE,  legend.shrink = 0.9, legend.width = 1,
horizontal = FALSE, legend.mar=NULL, bigplot = NULL, smallplot = NULL,...)

in.poly (xd, xp, convex.hull = FALSE, inflation=1e-7)

krig.image.parameters (out)

lonlat2xy (lnlt, miles = FALSE)

make.surface.grid (grid.list)

minimax.crit (obj, des = TRUE, R)
```

```
periodic.cov.1d (x1, x2, a, b)
periodic.cov.cyl (x1, x2, a = 0, b = 365, theta = 1)
periodic.plane.3d (x1, x2, a = 0, b = 365, theta = 1)

## S3 method for class 'krig.image':
plot(x, main = NA, digits = 4, which = rep(TRUE, 4),
      graphics.reset = TRUE, ...)
## S3 method for class 'qsreg':
plot(x, pch = "*", main = NA,...)
## S3 method for class 'sim.krig.image':
plot(x,...)
## S3 method for class 'spatial.design':
plot(x,...)

## S3 method for class 'interp.surface':
predict(object, loc,...)
## S3 method for class 'krig.image':
predict(object, x, ...)
## S3 method for class 'qsreg':
predict(object, x, derivative = 0, model = object$ind.cv.ps,...)
## S3 method for class 'sreg':
predict(object, x, derivative = 0, model = 1,...)

## S3 method for class 'krig.image':
print(x,...)
## S3 method for class 'qsreg':
print (x, ...)
## S3 method for class 'spatial.design':
print (x,...)
## S3 method for class 'sreg':
print(x, ...)
## S3 method for class 'summary.Krig':
print (x, ...)
## S3 method for class 'summary.krig.image':
print (x, ...)
## S3 method for class 'summary.spatial.design':
print (x, digits = 4,...)
## S3 method for class 'summary.sreg':
print (x, ...)

qr.q2ty (qr, y)

qr.yq2 (qr, y)

qsreg.fit (x, y, lam, maxit = 50, maxit.cv = 10, tol = 1e-04,
    offset = 0, sc = sqrt(var(y)) * 1e-07, alpha = 0.5, wt = rep(1,
    length(x)), cost = 1)
```

```
qsreg.psi ( r,alpha=.5,C=1)
qsreg.rho ( r,alpha=.5,C=1)
qsreg.trace (x, y, lam, maxit = 50, maxit.cv = 10, tol = 1e-04,
    offset = 0, sc = sqrt(var(y)) * 1e-07, alpha = 0.5,
    wt = rep(1, length(x)), cost = 1)

radbas.constant (m, d)

replace.args.function (fun, ...)

sim.krig.image (out, nreps = 10)

sreg.df.to.lambda (df, x, wt, guess = 1, tol = 1e-05)
sreg.fdf (h, info)
sreg.fgcv (lam, obj)
sreg.fgcv.model (lam, obj)
sreg.fgcv.one (lam, obj)
sreg.fit (lam, obj, verbose=FALSE)
sreg.fs2hat (lam, obj)
sreg.trace (h, info)

stats.bplot (x, by, style = "tukey", outlier = TRUE)
stats.sim.krig.image (obj)

summary.gcv.Krig(object, lambda, cost = 1, verbose = FALSE,
                offset = 0, y = NULL, ...)
summary.gcv.sreg (object, lambda, cost = 1, nstep.cv = 20,
        offset = 0, verbose = TRUE,...)
## S3 method for class 'krig.image':
summary (object, digits = 4, ...)
## S3 method for class 'qsreg':
summary (object, ...)
## S3 method for class 'spatial.design':
summary (object, digits = 4, ...)
## S3 method for class 'sreg':
summary (object, digits = 4, ...)

surface(obj , ...)
## Default S3 method:
surface (obj, ...)
## S3 method for class 'krig.image':
surface(obj, grid.list = NA, extrap = TRUE,
      graphics.reset = FALSE, xlab = NULL, ylab = NULL, main = NULL,
      zlab = NULL,zlim = NULL,levels = NULL, ptype = "I", ...)

## S3 method for class 'surface':
surface (obj, ...)
```

```
unscale (x, x.center, x.scale)
```

`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.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.

Details

`fields.diagonalize` finds the matrix transformation G that will convert A to a identity matrix and B to a diagonal matrix:

$G\hat{T}$ A G= I $G\hat{T}$ B G= D.

`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 *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 can be used for spatial prediction.

fields stives to have readable and tutorial code. Take a look at the source code for `Krig` and `Krig.engine.default` to see how things work "under the hood".

Some major methods include:

- `Tps` Thin Plate spline regression (including GCV)
- `Krig` Spatial process estimation (Kriging) including support for conditional simulation.

The Krig function allow 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. Also check out `mKrig` (micro Krig) a fast Kriging routine, that can take advantage of sparse covariance functions and thus handle very large numbers of spatial locations.

Some other noteworthy functions are

- `cover.design` Gnerates space-filling designs where the distance function is expresed in R/S code
- `as.image`, `image.plot`, `drape.plot`, `quilt.plot` `add.image`, `crop.image`, `half.image`.
  convenient functions for working with image data and rationally (well, maybe reasonably) placing a color scale on an image plot.
- `sreg`, `qsreg` `splint` Fast 1-D smoothing splines and 1-D quantile/robust and interpolating cubic splines

There are also generic functions that support these methods such as

`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
`predict.se`- prediction standard errors at arbitrary points.
`sim.rf`- Simulate a random fields on a 2-d grid.

To get started, try some of the examples from help files for `Tps` or `Krig`. See also the manual/tutorial at http://www.image.ucar.edu/Software/Fields

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.

DISCLAIMER:

This is software for statistical research and not for commercial uses. The authors do not guarantee the correctness of any function or program in this package. Any changes to the software should not be made without the authors permission.

Examples

```
# some air quality data,daily surface ozone 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.

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")
```

---

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 . These 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.

Examples

```
##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.

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="")
}
```

---

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

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 the fields source code these are in a subdirectory "tests". Compare the output to the "XXX.Rout.save" text file. Keeping in mind that no test messages should print if all is well, this is really a formality. The main reason these comparisions are provided is for matching the conventions of the R package checking utilities.

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

"FAILED test value = ... "

If the object test.for.zero.flag exists (it can have any value), all the tests that pass print text beginning,

" PASSED test at tolerance ..."

This startegy means that if all tests succeed nothing and the `object test.for.zero.flag` does not exist then nothing is printed in the test scripts. This is option simplifies the output scripts for running through the tests – no news is good news.

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

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

Where `demon` 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.

`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 = 80, rmse = NA,
                  verbose = FALSE, tol = 1e-05, offset = 0,
                  y = NULL, give.warnings = TRUE, give.warnings.REML = FALSE)

gcv.sreg (out, lambda.grid = NA, cost = 1, nstep.cv = 80, rmse =
     NA, offset = 0, trmin = NA, trmax = NA, verbose = FALSE, tol = 1e-05)
```

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.
`give.warnings.REML`	If FALSE will suppress warnings about grid search being out of range when finding REML estimate of lambda.
`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( ozone$x, ozone$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

# 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<- matrix( NA, ncol=4, nrow=100)

for( k in 1:100){
# look at GCV estimates of lambda
# new data simulated
   y<- f + rnorm(150)*sigma
```

```
# save GCV estimates
hold[k,]<-  gcv.Krig(obj,  y=y, give.warnings=FALSE)$lambda.est[1,]
}
plot( hold[,2], hold[,4], xlab="estimated eff. df", ylab="sigma hat")
yline( sigma, col=2)
# note some occaisional flaky behaviour with GCV ( eff df > 20!)
```

`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, 1), boundary.grid=FALSE )
```

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.
`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.

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.

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 predict.surface. 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 predict.surface 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.

See Also

as.surface, predict.surface, 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.
#

discretize.image( RMprecip$x, m=15, n=15)-> look
Z<- matrix( 0, 15,15)

Z[look$ind]<- 1

image( look$grid$x, look$grid$y, Z) # indicator image of discretized locations.
points( RMprecip$x,col="magenta") # actual locations
# (there may be more than one location in the grid boxes)
```

`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. Exp.image.cov and matern.image.cov will be depreciated functions and are replaced by stationary.image.cov.

Usage

```
stationary.image.cov(ind1, ind2, Y, cov.obj = NULL, setup = FALSE,
grid, M=NULL,N=NULL, Covariance="Matern", Distance="rdist",...)

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,...)
```

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.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.
setup	If true do not do the multiplication but just return the covariance object required by this function.
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.
Covariance	Covariance function that is apllied to the distance between locations. (see stationary.cov) Default is the Matern model. (with smoothness=.5) the exponential.
Distance	Distance function applied to locations. Default is Euclidean distance. Another choice is "rdist.earth", great circle distnace for lon/lat coordinates.
...	Any arguments to pass to the covariance function in setting up the covariance object. This is only required if setup is TRUE. For the "Matern" the arguments are theta (the range default=1), and smoothness (default=.5 giving the exponential). theta can be 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 and matern.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, 100,10))
#  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)
# 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
 cov.obj<- stationary.image.cov(
             setup=TRUE, grid=grid, theta=1.25, smoothness=2)
 Y<- matrix(0,64,64)
 Y[16,16]<- 1

 result<- stationary.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 (16,16) for this case
```

`image.plot` *Draws 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. It also allows for plotting quadrilateral cells in the image format that often arise from regular grids transformed with a map projection.

Usage

```
image.plot(...,
                  add = FALSE, nlevel = 64, horizontal = FALSE,
                  legend.shrink = 0.9, legend.width = 1.2, legend.mar =
                  ifelse(horizontal, 3.1, 5.1), legend.lab = NULL,
                  graphics.reset = FALSE, bigplot = NULL, smallplot =
                  NULL, legend.only = FALSE, col = tim.colors(nlevel),
                  lab.breaks = NULL, axis.args = NULL, legend.args =
                  NULL, midpoint=FALSE)
```

Arguments

`...`	The usual arguments to the image function. This includes the use of the breaks argument for an unequal color scale. If a quadrilateral grid arguments must be explicitly x,y and z with x, and y being matrices of dimension equal or one more than z giving the grid locations.
`add`	If true add image and a legend strip to the existing plot.
`nlevel`	Number of color levels used in legend strip
`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.

`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.

`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.

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

`smallplot` Plot coordinates for legend. If not passed these will be determined within the function.

`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.

`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.

`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.args` Arguments for a complete specification of the legend label. This is in the form of list and is just passed to the mtext function. Usually this will not be needed. (See example below.)

`midpoint` If FALSE (default) for quadrilateral grids grid point will be extended to include z locations as midpoints. If true z values will be averaged to yield a midpoint value. (See help on poly.image for details). In most cases midpoint should be FALSE to preserve exact values for z.

Details

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 more details of this difference in formatting. What does one do if you don't 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 x and y are matrices then z[i,j] is rendered at 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`. 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 was added to help with plotting the gridded output of geophysical models where the regular grid is defined according to one map projection by the 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.

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.

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. Changing the plot margins directly replaces the `offset` argument in the older version of this function.

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 plot.image 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.

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. There may be problems with small plotting regions in fitting both of these elements in the plot region and one may have to change the default character sizes or margins to make things fit.

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 `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 and the code for `plot.Wimage` for more complicated arrangements of multiple image plots and summary legends.

Adding just the legend strip: Note that to add just the legend strip all the numerical information one needs is the `zlim` argument! We like `tim.colors` as a default color scale. The the topo-

graphic color scale (`topo.colors`) is also a close second showing our geophysical basis. See also `terrain.colors` for a subset and `designer.colors` to "roll your own". One nice option of this last one is to fix colors 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.

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.

See Also

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

# now add some points on diagonal with some clipping anticipated
   points( 5:12, 5:12, pch="X", cex=3)

image.plot(x,y,z, legend.lab="inches")

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

quantile(c(z), c( .05, .1,.5, .9,.95))-> zp

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.
```

```
#
#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 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))

#### example using a irregular quadrilateral grid
data( RCMexample)
image.plot( RCMexample$x, RCMexample$y, RCMexample$z[,,1])

#### 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){
image( matrix( rnorm(150), 10,15), zlim=c(-4,4), col=tim.colors())
}

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 and legend colorbar on the
# right to put a legend.

   split.screen( rbind(c(0, .8,0,1), c(.8,1,0,1)))

# now divide up the figure region
   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

A<- matrix( 1:48, ncol=6)
# Note that matrix(c(A), ncol=6) == A
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[6:1,]), 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

```
image.smooth(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, ...)
```

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
`...`	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, nrow=128, ncol=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)

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)

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 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)
title("free boundaries")
image.plot( look2) # 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 a image or reducing it size by averaging adjecent cells.

Usage

```
crop.image(obj, loc=NULL,...)
half.image(obj)
get.rectangle()
```

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.
...	Graphics arguments passed to image.plot. This is only relevant when loc is NULLand the locator function is called.

Details

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

Author(s)

Doug Nychka

See Also

drape.plot, image.plot

Examples

```
data(RMelevation)

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

# extract elevations for just CO frontrange.
FR<- crop.image(RMelevation, loc)
#  check:   image.plot( FR)

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

# extract the this averaged image
 FR2<- crop.image(temp, loc)
 zr<- range( FR$z)
 set.panel( 1,2)
 image.plot( FR, zlim =zr, horizontal=TRUE)
 image( FR2, zlim =zr, col=tim.colors())
```

`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

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.

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 the surface object
obj<- list( x= 1:20, y=1:20, z= lennon[ 201:220, 201:220])

# sample at 50 equally spaced points
temp<- seq( 1,20,,50)
make.surface.grid( list( temp,temp))-> loc
interp.surface( obj, loc)-> look
# take a look
image.plot( as.surface( loc, look))
```

`krig.image` *Spatial process estimate for large irregular 2-d dats sets.*

Description

Computes the spatial predictions for large numbers of irregularly spaced observations using the standard Kriging equations. The main approximation is that the locations are discretized to a regular grid, but the field need not be observed at all grid boxes.

In Bayesian terms this function computes the posterior mean for the field given the observations under the usual Gaussian assumptions for the fields and observations. The solution is found by the iterative solution of a large linear system using the conjugate gradient algorithm (CGA). Part of the calculations rely on discretizing the spatial locations to a regular grid to make use of the FFT for fast multiplication of a covariance matrix with a vector.

Usage

```
krig.image(x, Y, cov.function, m=NULL, n=NULL, lambda=0, start=NULL,
tol=1e-05, kmax=25,  cov.obj=NULL, grid=NULL,
weights=rep(1, length(Y)), verbose=FALSE, conv.verbose=FALSE, expand=1, ...)
```

Arguments

`x`	A 2 column matrix of observed locations
`Y`	Values of observed field. Missing values are omitted from computation.
`cov.function`	An S function that multiplies the covariance matrix by a vector. Two that are part of FIELDS are Exp.image.cov (Exponential and Gaussian) and W.image.cov (W transform covariance model)
`lambda`	The value of the smoothing parameter. Should be nonnegative. See the notes below for more information about this parameter

m	Number of grid points in the x axis. Default is to use the length of grid$x.
n	Number of grid points in the y axis. Default is to use the length of grid$y.
cov.obj	A covariance object that contains information to be used by the covariance function specified above. If this is not specified this object will be created within krig.image.
grid	A list with components x and y that specify the grid points in the x and y directions. The default is to use the number of point specified by m and n and use the ranges from the observed locations.
start	Starting values for omega2 in the iterative algorithm. Default is zero.
tol	Convergence tolerance for CGA.
kmax	Maximum number of iterations for CGA
weights	This vector is proportional to the reciprocal variance of the measurement errors. The default is a vector of ones.
verbose	If true all kinds of stuff is printed out! Default is false of course.
conv.verbose	If true the convergence criterion is printed out at each iteration of the CGA. The values are scaled as the criterion divided by the tolerance. So the algorithm terminates when the values are less than one.
expand	The amount the grid should be expanded beyond the range of the observed data. For example expand 1.1 will give a range that is 10 % larger on each end.
...	Any extra arguments are considered as information for the covariance function and are used to create the covariance object.

Details

From a functional point of view krig.image and supporting functions are similar to the class Krig. The main difference is that only 2-dimensional problems are considered and the solution is calculated for a fixed value of lambda. (The Krig function can estimate lambda.) For large data sets a practical way to estimate lambda is by out of sample cross-validation and the FIELDS manual gives a detailed example of this for the precip data set. Also see the manual for an explanation of the computational strategy (Conjugate Gradient) here.

Efficiency for large datasets comes with restrictions on the range of covariance functions and some other features. Currently FIELDS just has two covarince models: exponential/Gaussian and wavelet based. However, it is not difficult to modify these to other models. The default discretization is to a 64X64 grid however even 256X256 is manageable and quite likely to separate irregular locations in most cases. The user should also keep in mind that the estimate is the result of an iterative algorithm and so issues such as good starting values and whether the algorithm converged are present.

The spatial model includes a linear spatial drift and MLE estimates of the nugget variance and sill are found based on the values of lambda. If the weights are all equal to one and the covariance function is actually a correlation function, in the notation of this function, the "sill" is sigma2 + rho and the "nugget" is sigma2. Moreover sigma2 and rho are constrained so sigma2/rho =lambda. This is why lambda is the crucial parameter in this model.

Although the field is only estimated to the resolution of the grid, prediction off of the grid is supported by bilinear interpolation using the FIELDS function interp.surface.

Value

An list object of class krig.image. An explanation of some components:

call	The calling sequence
cov.function	A copy of the covariance S function
na.ind	logical indicating missing values in Y
xraw	Passed spatial locations having removed missing values
y	Observations having omitted missing values
N	Length of y
weights	passed weights having omitted missing cases.
lambda	
grid	list with components x an y indicating grid for discretization
cov.obj	List object to use with cov.function
m	Number of grid point in x axis
n	Number of grid point in y axis
index	A two column matrix indicating the indices of the closest grid point to each observed location.
x	Observed locations discretized to nearest grid point
yM	Observed values but with a weighted average replacing multiple values associated with the same grid point.
xM	Discretized locations associated with yM
weightsM	Weight vector associated with YM.
uniquerows	Logical indicating which rows of x are unique.
shat.rep	Pooled standard deviation among observations that fall within the same gird points
indexM	A two column matrix indicating the indices of the closest grid point to each observed location, yM.
qr.T	QR decomposition of the matrix of constant and linear terms at xM
multAx	The S function that is used for matrix multiplication in the CGA.
omega2	Parameter vector that describes the spatial process part of the conditional mean.
converge	CGA convergence information
beta	Constant, and the two linear parameters for the fixed linear part of the model
delta	Covariance matrix times delta give the spatial predictions.
rhohat, rho	Conditional on lambda the MLE for the parameter multiplying the covariance function.
sigma2, shat.MLE	Conditional on lambda the MLE for the parameter dividing the weight function.
surface	A list giving the predicted surface at the grid points.
fitted.values	Predicted values at true locations

References

Large spatial prediction problems and nonstationary fields (1998) Nychka, D., Wikle, C. and Royle, J.A.

FIELDS manual

See Also

plot.krig.image, predict.krig.image, Exp.image.cov, sim.krig.image

Examples

```
#
# fit a monthly precipitation  field over the Rocky Mountains
# grid is 64X64
out<- krig.image( x= RMprecip$x, Y = RMprecip$y, m=64,n=64,cov.function=
Exp.image.cov,
lambda=.5, theta=1, kmax=100)

#
# range parameter for exponential here is .5 degree in lon and lat.
#diagnostic plots.
plot( out)

# look at the surface
image.plot( out$surface)  #or just      surface( out)

#
#simulate 4 realizations from the conditional distribution
look<- sim.krig.image( out, nreps=4)
# take a look:  plot( look)

# check out another values of lambda reusing some of the objects from the
# first fit

out2<- krig.image( RMprecip$x, RMprecip$y, cov.function= Exp.image.cov,
lambda=4,
start= out$omega2,cov.obj=out$cov.obj)
#
# some of the obsare lumped together into a singel grid box
#
# find residuals among grid box means and predictions
res<- predict( out2, out2$xM) - out2$yM
#compare with sizes of out2$residuals (raw y data)

#starting values from first fit  in out$omega2
# covariance and grid information are
# bundled in the cov.obj
##

#
## fitting a thin plate spline. The default here is a linear null space
```

```
## and  second derivative type penalty term.
##  you will just have to try different values of lambda vary them on
##  log scale to

out<- krig.image( RMprecip$x, RMprecip$y, cov.function=Rad.image.cov,
lambda=1, m=64, n=64, p=2, kmax=300)

# take a look: image.plot( out$surface)

# check out  different values reuse some of the things to make it quicker
# note addition of kmax argument to increase teh number of iterations

out2<- krig.image( RMprecip$x, RMprecip$y,cov.function=Rad.image.cov,
lambda=.5, start= out$omega2, cov.obj=out$cov.obj, kmax=400)

# here is something rougher
out3<- krig.image( RMprecip$x, RMprecip$y,cov.function=Rad.image.cov,
lambda=1e-2, start= out2$omega2, cov.obj=out$cov.obj,kmax=400,
tol=1e-3)

# here is something close to an interpolation
out4<- krig.image( RMprecip$x, RMprecip$y,cov.function=Rad.image.cov,
lambda=1e-7, start= out3$omega2, cov.obj=out$cov.obj,kmax=500, tol=1e-3)

#compare the the four surfaces:
# but note the differences in scales ( fix zlim to make them the same)
#
# take a look
# set.panel( 2,2)
# image.plot( out$surface)
# points( out$x, pch=".")

# image.plot( out2$surface)
# image.plot( out3$surface)
# image.plot( out4$surface)

# some diagnostic plots)
set.panel( 4,4)
plot( out, graphics.reset=FALSE)
plot( out2, graphics.reset=FALSE)
plot( out3, graphics.reset=FALSE)
plot( out4, graphics.reset=FALSE)
set.panel(1,1)
```

`lennon` *Gray image of John Lennon.*

Description

A 256X256 image of John Lennon.

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 and a clear exposition of the computations. Lambda, the smoothing parameter must be fixed.

Usage

```
mKrig(x, y, weights = rep(1, nrow(x)),
  lambda = 0, cov.function = "stationary.cov",
    m = 2, chol.args=NULL,cov.args=NULL, ...)

## S3 method for class 'mKrig':
predict( object, xnew=NULL, derivative=0, ...)
## S3 method for class 'mKrig':
print( x, ... )
```

Arguments

x	Matrix of unique spatial locations (or in print or surface the returned mKrig object.)
y	Vector of observations at spatial locations, missing values are not allowed!
weights	Precision (1/variance) of each observation
lambda	Smoothing parameter or equivalently the ratio between the nugget and process varainces.
cov.function	The name, a text string of the covariance function.
m	The degree of the polynomial used in teh fixed part is (m-1)
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.
...	In mKrig and predict additional arguments that will be passed to the covariance function.
object	Object returned by mKrig. (Same as "x" in the print function.)
xnew	Locations for predictions.
derivative	If zero the surface will be evaluated. If not zero the matrix of partial derivatives will be computed.

Details

This function is an abridged version of Krig that 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 restriction simply the code for reading. Note that also little checking is done and the spatial locations are not transformed before the estimation.

`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.

`print.mKrig` is a simple summary function for the object.

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))
```

Value

`d`	Coefficients of the polynomial fixed part.
`c`	Coefficients of the nonparametric part.
`nt`	Dimension of fixed part.
`np`	Dimension of c.
`x`	Spatial locations used for fitting.
`cov.function.name`	Name of covariance function used.
`cov.args`	A list with all the covariance arguments that were specified in the call.
`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.

Author(s)

Doug Nychka, Reinhard Furrer

See Also

Krig, surface.mKrig, Tps, fastTps

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)
mKrig( x,y, theta = 2.0, lambda=.01)-> out
#
# NOTE this should be identical to
# Krig( x,y, theta=2.0, lambda=.01)

# interpolate using tapered version the taper scale is set to 1.5
# Default 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.

mKrig( x,y,cov.function="stationary.taper.cov",
       theta = 2.0, lambda=.01, Taper.args=list(theta = 1.5, k=2)
           ) -> out2

predict.surface( out2)-> out.p
surface( out.p)

# here is a series of examples with a bigger problem
# 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

mKrig( x,y, cov.function="wendland.cov",k=2, theta=.2,
            lambda=.1)-> look2

# take a look at fitted surface
predict.surface(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  spam.options() for the default values

# 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:
 mKrig( x,y, cov.function="wendland.cov",k=2, theta=.9, lambda=.1)-> look2
## End(Not run)

# The warning resets the memory allocation  for the covariance matirx according the
# values   'spam.options(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.

spam.options(nearestdistnnz=c(416052,400))
mKrig( x,y, cov.function="wendland.cov",k=2, theta=.9, lambda=1e-2)-> look2

# as a check notice that
#   print( look2)
# report 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

# 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
#
## Not run:

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

# success!

#################################################
# finding a good choice for theta
#################################################
# Suppose the target is a spatial prediction using roughly 50 nearest neighbors
# (tapering covariances is effective for rouhgly 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

set.seed(223)
```

```
 ind<- sample( 1:N,100)
 hold<- rdist( x[ind,], x)
 dd<- (apply( hold, 1, sort))[65,]
 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
# but omit the diagonal elements -- we know these are zero!

 hold<- nearest.dist( x, delta= dguess,upper=NULL, diag=FALSE)
# exploit spam format to get quick of number of nonzero elements in each row

 hold2<-  diff( hold@rowpointers)
 #  min( hold2) = 55   which we declare close enough

# now the following will use no less than 55 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)

#
#    Using mKrig for evaluating  a solution on a big grid.
#    (Thanks to Jan Klennin for motivating this example.)

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 cc

# 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

# theta sets range for the compact covariance function
# this will involve  less than 20 nearest neighbors tha have
# nonzero covariance

    theta<- c( 2.5*(glist$lon[2]-glist$lon[1]),
```

```
                2.5*(glist$lat[2]-glist$lat[1]))

# 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, theta=theta,
                lambda=0)->out2
# the big evaluation this takes about 45 seconds on a Mac G4 latop
    predict.surface( out2, nx=400, ny=400)-> look

# the nice surface
## Not run:

    surface( look)
    US( add=TRUE, col="white")
## End(Not run)
```

`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

`ozone` *Data set of ozone measurements at 20 Chicago monitoring stations.*

Description

The `ozone` 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.

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

---

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 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.

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.

See Also

Krig, summary.Krig, Tps, set.panel

Examples

```
fit<-Krig(ozone$x, ozone$y, theta=200)
# fitting a surface to ozone
# measurements

set.panel( 2,2)
plot(fit)

fit<-sreg(rat.diet$t,rat.diet$con)

# fit rat data
set.panel(2,2)
plot(fit)

set.panel(1,1) # reset graphics window.
```

---

plot.Wimage *Plots 2-d wavelet coefficents by level and type*

Description

Produces a panel of images using the `split.screen` tools that organize the wavelet coefficients from a multiresolution by location, resolution level and type.

Usage

```
## S3 method for class 'Wimage':
plot(x, cut.min, graphics.reset = TRUE, common.range = FALSE,
    color.table = tim.colors(128), Nlevel = NULL, with.lines = FALSE,
    omd.width = 0.2, ...)
```

Arguments

x	matrix of coefficients
cut.min	defines minimum level
graphics.reset	If TRUE will reset device to orignal settings including closing the split.screen mode.
common.range	If TRUE image plots will be on common color scale.
color.table	Color table to be used for image plots. Default is Tim Hoar's favorite.
Nlevel	Number of levels to plot. Default is to plot all.
with.lines	If TRUE will add white outlines of pixels. Default is FALSE to prevent from tiny pixels from turning white!
omd.width	Fration of device surface width devoted to the vertical color legend strips. Default usually accomodates most axis labels.
...	Other Graphical parameters to be passed to the par function.

Details

As with most complicated graphical figures you basically get what it draws although it should be easy to modify this function for customization. The split.screen set of graphical functions are used to divide up the plotting real estate into subplots of different sizes. The user can experiment with different outer margin space (omd.width) Use cex.axis argument in the call to change the size of the numerals in the color strip.

By setting grahics.reset to FALSE the function returns a matrix giving the split screen ids to reference each of the individual plots. Use the screen function to move to given plot and use a high level plotting function to overlay information. (See example below.) Use close.screen(all=TRUE) to turn off split.screen mode for subsequent and normal plotting.

Author(s)

Doug Nychka

Examples

```
old.par<- par(no.readonly=TRUE) # these functions may leave the device in
# with some funny defaults

#
#multiresolution of John Lennon
data(lennon)
Wtransform.image( lennon, cut.min=16)->look
plot.Wimage( look, cut.min=16)

# adding information
plot.Wimage( look, cut.min=16, Nlevel=3, graphics.reset=FALSE)-> plot.layout
# plot.layout here is a 4X3 matrix with the screen numbers
```

```
# move to the smooth coefficients plot

 screen( plot.layout[1,1])
 plot( c(.5,16.5), c( .5,16.5), type="n",axes=FALSE)
 points( 8,8, cex=2, pch="+")

# NOTE: just points( 8,8) will not work here. This has to do with split.screen not
# reseting the plotting pars correctly.

 box(col=6, lwd=2)# just for fun

close.screen( all=TRUE)
par( old.par) # reset  to old settings
```

`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 predict.surface.
`main`	Title for plot.
`type`	type="p" for a perspective/drape plot, type="I" for an image plot with a legend strip (see image.plot). type="C" is the "I" option 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, predict.surface, as.surface, drape.plot, image.plot

Examples

```
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<-predict.surface(fit, xy=c(2,3))
 plot.surface(out.p, type="C") # surface plot
```

---

poisson.cov *Poisson spherical covariance function*

Description

Given two sets of locations in lon/lat computes the cross covariance matrix for the Poisson covariance among all pairings.

Usage

```
poisson.cov(x1, x2, eta = .2)
```

Arguments

x1	Matrix of first set of locations where each row gives the coordinates of a particular point. First column is longitudes and the second column is latitudes.
x2	Matrix of second set of locations where each row gives the coordinates of a particular point. If this is missing x1 is used.
eta	Range (or scale) parameter. Should be in the interval [0,1]

Details

This covariance is one of the few closed form covariances for the sphere and also know as the Poisson kernel. If x1 and x2 are matrices where nrow(x1)=m and nrow(x2)=n then this function should return a mXn matrix where the (i,j) element is the covariance between the locations x1[i,] and x2[j,]. The covariance is found as

(1-eta**2)/(1 - 2 * eta * D.ij + eta**2)**(1.5)

where D.ij is the great circle distance between x1[i,] and x2[j,].

Value

If nrow(x1)=m and nrow(x2)=n then the returned cross covariance matrix, will be mXn.

See Also

Krig, rdist.earth

Examples

```
# plot of covariance

x<- make.surface.grid( list( x=seq( -180,180,,40), y= seq( -85,85,,40)))
x0<- matrix( c(0,0), ncol=2)
look<- poisson.cov( x,x0, eta=.5)
image.plot(as.surface(x,look))
```

---

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), transparent.color = "white",
 midpoint = FALSE, zlim = range(z, na.rm = TRUE),
 xlim = range(x), ylim = range(y), add = FALSE, border=NA,...)

poly.image.regrid(x)
```

Arguments

x	A matrix of the x locations of the grid.
y	A matrix of the y locations of the grid.
z	Values for each grid cell. Can either be the value at the grid points or interpreted as the midpoint of the grid cell.
col	Color scale for plotting.
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.
...	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)

# 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, ...)
```

Arguments

`object`	Fit object from the Krig or Tps function.
`x`	Matrix of x values on which to evaluate the kriging surface. If omitted, the data x values, i.e. out$x will be used.
`Z`	Vector/Matrix of additional covariates to be included in fixed part of spatial model
`drop.Z`	If TRUE only spatial fixed part of model is evaluated. i.e. Z covariates are not used.
`just.fixed`	Only fixed part of model is evaluated
`lambda`	Smoothing parameter. If omitted, out$lambda will be used. (See also df and gcv arguments)
`df`	Effective degrees of freedom for the predicted surface. This can be used in place of lambda (see the function Krig.df.to.lambda)
`model`	Generic argument that may be used to pass a different 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

`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.

`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)

`verbose`
: Print out all kinds of intermediate stuff for debugging

`...`
: Other arguments passed to predict.

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.

Value

Vector of predicted responses

See Also

Krig, predict.surface gcv.Krig

Examples

```
Krig(ozone$x,ozone$y, theta=50) ->fit
predict( fit) # gives predicted values at data points

# only the fixed part of the model

predict( fit, just.fixed=TRUE)

# in this case the default is a linear spatial drift  (m=2) and there
# are no additional covariates

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
out.p<- as.surface( grid, look) # reformat into $x $y $z image-type object
contour( out.p)
```

```
# 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))
```

`predict.se.Krig` *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. There are also provisions to use a different covariance for evaluation than the one used to define the BLUE.

Usage

```
## S3 method for class 'Krig':
predict.se(object, x = NULL, cov = FALSE, verbose = FALSE,...)
```

Arguments

`object`	A Krig object.
`x`	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 predict.se function.

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 `predict.se.KrigA` uses the A matrix to find the standard errors or covariances directly from the linear combination of the spatial predictor. Currently this is also the calculation in `predict.se.Krig` although a shortcut using the Kriging equations is planned for a later verion of fields.

Value

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

See Also

Krig, predict.Krig, predict.surface.se

Examples

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

  fit<- Krig(ozone$x,ozone$y,cov.function="Exp.cov", theta=10)    # Krig fit
  predict.se.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<- predict.se.Krig(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<- predict.surface.se( fit)
# image.plot( out)
```

`predict.se` *Standard errors of predictions*

Description

Calculates the standard error of predictions. This is usually the fitted object from a function estimate such as from Krig or Tps.

Usage

```
predict.se(object, ...)
```

Arguments

`object`	A fitted model object of a certain class
`...`	Additional arguments to be passed to a particular method. e.g. a grid.list or model specification.

Details

This function is generic and will call the appropriate function to calculate the standard errors for the object class. The prediction standard error is for the estimated function or parameters (a mean value) not for the prediction of a new observation.

Value

A vector of standard errors for the predicted values.

See Also

predict, predict.surface.se, predict.se.Krig

Examples

```
  fit<-Krig(ozone$x,ozone$y, Covariance="Matern",theta=50, smoothness=1)
  predict.se(fit)                              # std errors of predictions
#
# create a grid of points
  xg<- make.surface.grid(
        list(East.West=seq(-15,15,,20),North.South=seq(-20,20,,20) ) )
  out<- predict.se(fit,xg)
  image.plot( as.surface( xg, out))
```

`predict.surface` *Evaluates a fitted function or its standard errors as a surface object*

Description

Evaluates a a fitted model 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

```
predict.surface(object, grid.list = NA, extrap = FALSE, chull.mask =
                NA, nx = 80, ny = 80, xy = c(1, 2), order.variables="xy",...)

predict.surface.se(object, grid.list = NA, extrap = FALSE, chull.mask =
                NA, nx = 80, ny = 80, xy = c(1, 2),
                order.variables="xy",verbose=FALSE, ...)
```

Arguments

`object`	An object from fitting a function to data. In FIELDS this is usually a Krig 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.
`extrap`	Extrapolation beyond the range of the data. If false function will be restricted to the convex hull of the observed data or the hull defined from the points from the argument chull.mask.
`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 arguments is passed.

`order.variables`
: If "xy" the variables in grid.list are taken in order as "x" then "y". If "yx" the roles are reversed. Suppose a grid.list had components lat, lon, elevation and one wanted a lon/lat surface at a fixed elevation. Then one would set to "yx" to make "x" lon and "y" lat.

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

`...`
: Any other arguments to pass to the predict function associated with the fit object.

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.

Value

The usual list components for making contour and perspective plots (x,y,z) along with labels for the x and y variables.

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<- predict.surface(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<- predict.surface(fit, glist)
surface(out.p, type="C")

out.p<- predict.surface.se(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(ozone$x,ozone$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
#
matlines( xg, sm, col=c( 3,3,2,3), lty=1) # the spline
```

`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, nrow = 64, ncol = 64, grid = NULL,
                     add.legend=TRUE,add=FALSE,col=tim.colors(256),...)
```

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.
`nrow`	Number of grid boxes in x.
`ncol`	Number of grid boxes in y.

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.
...	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. Locations that fall into the same grid box will have their z values averaged.

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 legend strip "by hand"
par( mar=c( 5,5,5,10)) # save some room for the legend
quilt.plot( ozone2$lon.lat, ozone2$y[16,], add.legend=FALSE)
image.plot(ozone2$lon.lat, ozone2$y[16,],legend.only=TRUE)
```

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

rdist *Euclidean distance matrix*

Description

Given two sets of locations computes the full Euclidean distance matrix among all pairings or a sparse version for points within a fixed threshhold distance.

Usage

```
rdist(x1, x2)

fields.rdist.near(x1,x2, delta, max.points= NULL, mean.neighbor = 50)
```

Arguments

x1	Matrix of first set of locations where each row gives the coordinates of a particular point.
x2	Matrix of second set of locations where each row gives the coordinates of a particular point. If this is missing x1 is used.
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.

See Also

Exp.cov, rdist.earth, ind2spam, ind2full

Examples

```
out<- rdist( ozone$x)
# out is a 20X20 matrix.

out2<- rdist( ozone$x[1:5,], ozone$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
```

`rdist.earth` *Great circle distance matrix*

Description

Given two sets of longitude/latitude locations computes the Great circle (geographic) distance matrix among all pairings.

Usage

```
rdist.earth(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 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 this all done efficiently in R by dot products of the direction cosines. Thanks to Qing Yang for pointing this out a long time ago.

Value

The great circle distance matrix if nrow(x1)=m and nrow(x2)=n then the returned matrix will be mXn.

See Also

rdist, stationary.cov

Examples

```
out<- rdist.earth ( ozone$lon.lat)
#out is a 20X20 distance matrix
```

---

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)
```

---

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.Krig *Conditonal 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.Krig.standard(object, xp, M = 1, verbose = FALSE, sigma2 = NA, rho = NA)

sim.Krig.grid(object, grid.list = NA, M = 1, nx = 40, ny = 40, xy=c(1,2), verbose =
FALSE,
sigma2 = NA, rho = NA, extrap = FALSE)
```

Arguments

object	A Krig object
xp	Locations where to evaluate the conditional process.
M	Number of draws from conditional distribution.
verbose	If true prints out intermediate information.
sigma2	User specified value for nugget variance or measurement error. See Details below.
rho	User specified value for sill, or multiplier of spatial covariance function. See Detials below.
grid.list	Grid information for evaluating the conditional surface as a grid.list.
nx	Number of grid points in x.
ny	Number of grid points in y.
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 location matrix used to define the multidimensional surface from the Krig object. 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.
extrap	If FALSE conditional process is not evaluated outside the convex hull of observations.

Details

These functions generate samples from a conditional multivariate distribution that describes the uncertainty in the estimated spatial process under Gaussian assumptions. An important approximation throughout these functions is that all covariance parameters are fixed at their estimated or prescribed values.

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

Y.k = P(x.k) + h(x.k) + e.k

where P(x) is a low order, fixed polynomial and h(x) a Gaussian spatial process. With Y= Y.1, ..., Y.N, the goal is to sample the conditional distribution of the process.

[Z(x) | Y]

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 approxiamte 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 Z.hat(x).

1) Generate an unconditional spatial process and from this process simluate synthetic observations.

2) Use the spatial prediction model (using the true covariance) to estimate the spatial process at unobserved locations.

3) Find the difference between the simulated process and its prediction based on synthetic observations. Call this e(x).

4) Z.hat(x) + e(x) is a draw from [Z(x) | Y].

`sim.Krig.standard` follows this algorithm exactly.

`sim.Krig.grid` evaluates 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 correlation stationary fields.

Value

For `sim.Krig.standard` a matrix with columns indexed by the locations in `xp` and `M` rows.

For `sim.Krig.grid` a list with arguments `x` and `y` defining the grid locations in the usual manner and `z` contains the values of the simulated conditional field(s). `z` is a three dimesional array where the first two indices are "x" and "y" and the third index is between 1 and M and indexes the simulated fields.

Author(s)

Doug Nychka

See Also

sim.rf, Krig

Examples

```
data( ozone2)

set.seed( 399)

# fit to day 16 from Midwest ozone data set.
Krig( ozone2$lon.lat, ozone2$y[16,], Covariance="Matern",
theta=1.0,smoothness=1.0, na.rm=TRUE)-> out

# 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,]),]

# 50 draws from process at xp given the data
# this is an exact calculation
 sim.Krig.standard( out,xp, M=50)-> sim.out

# Compare: stats(sim.out)[3,] to  Exact: predict.se( out, xp)
```

```
# simulations on a grid
# NOTE this is approximate due to the bilinear interpolation
# for simulating the unconditional random field.

sim.Krig.grid(out,M=5)-> sim.out

# take a look at the ensemble members.

predict.surface( 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)
}
```

`sim.rf` *Simulates a random field*

Description

Simulates a random Gaussian field on a regular grid.

Usage

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

Arguments

`obj`	A covariance object that includes information about the covariance function and the grid for evaluation. Usually this created by a setup call to Exp.image.cov. (See details below.)
`...`	Additional arguments passed to a particular method.

Details

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 may not always work if the correlation scale is large (See the FIELDS manual for more details.) The simple fix is increase the size of the domain so that the correlation sale becomes smaller relative to the extent of th domain.

For a stationary model the covariance object has 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 the grid point values for x and y N and M is the size of the larger grid that is used for simulation (usually M= 2*m and N=2*n) to minimize periodic effects. 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.

Value

A matrix with the random field values

See Also

Exp.image.cov, matern.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)
# take a look
set.panel(2,1)
 image.plot( grid$x, grid$y, look)
 title("simulated gaussian field")
 image.plot( grid$x, grid$y, look2)
 title("another (independent) realization ...")
```

---

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)
```

```
spam2spind(obj)
```

Arguments

obj	Either a list with the sparse index components (spind) or an obj of class spam.

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

`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)
```

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.
`df`	Effective degrees of freedom. Default is to use lambda =0 or a df equal to the number of observations.

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 *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)
```

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.
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.)
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.

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 predict.se( fit.tps) differences are due to
# slightly different lambda values and using shat.MLE instad of shat.GCV
#

# 95
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 spatial process estimate*

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 print.summary.Krig.

Usage

```
## S3 method for class 'Krig':
summary(object, digits=4,...)
```

Arguments

object	A Krig 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

Examples

```
fit<- Krig(ozone$x, ozone$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

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(obj, grid.list = NA, 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(obj,
grid.list = NA, extrap = FALSE, graphics.reset = NULL, xlab = NULL,
        ylab = NULL, main = NULL, zlab = NULL, zlim = NULL, levels = NULL,
        type = "C", nx=80, ny=80, ...)
```

Arguments

obj	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 predict.surface 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 predict.surface, plot.surface, image.plot

Examples

```
fit<- Krig(ozone$x,ozone$y, theta=30)  # krig fit

#Image plot of surface with nice, smooth  contours and shading

surface(fit, type="C", nx=128, ny=128)
```

---

tim.colors *Some useful color tables for images.*

Description

Two color scales useful for image plots: a pleasing rainbow style color table patterned after that used in Matlab by Tim Hoar and also a simple colr interpolation between two colors passing through white.

Usage

```
tim.colors(n = 64)
two.colors(n=256, start="darkgreen", end="red", middle="white")
designer.colors( n=256, col= c("darkgreen", "white", "darkred"),
                              x= seq(0,1,, length(col)) )
```

Arguments

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.

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 matlab 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 http://www.image.ucar.edu/Data/MJProject.)

`designer.color` is the master function for two.colors and tim.colors. 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.

Value

A vector giving the colors in a hexadecimal format.

See Also

topo.colors, terrain.colors, image.plot, quilt.plot, grey.scale, fields.color.picker

Examples

```
tim.colors(10)
# returns an array of 10 strings in hex format
#e.g. (red, green,  blue) values of   (16,255, 239)
# translates to "#10FFEF" .

# veiw some color table choices
set.panel( 1,3)
par( pty="s")
z<- outer( 1:20,1:20, "+")

image( z, col=tim.colors( 200)) # 200 levels

image( z, col=two.colors() )
```

```
coltab<- designer.colors(col=c("blue", "grey", "green"),  x= c( 0,.3,1) )
image( z, 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(z), c(0,.25,.5,.75,1.0) )
# scale these to [0,1]
zr<- range( c(z))
x<- (x-zr[1])/ (zr[2] - zr[1])

coltab<- designer.colors(256,rainbow(5), x)
image( z, col= coltab ) # see image.plot for adding all kinds of legends

# colors now change at quantiles of data

set.panel()
```

`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( ozone$x, scale.type="range")
```

---

vgram *Finds a traditional or robust variogram for spatial data.*

Description

Computes pairwise squared differences as a function of distance. Returns either raw values or statistics from binning.

Usage

```
vgram(loc, y, id=NULL, d=NULL, lon.lat=FALSE, dmax=NULL, N=NULL, breaks=NULL)
```

Arguments

loc	Matrix where each row is the coordinates of an observed point of the field
y	Value of the field 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.
breaks	Bin boundaries for binning variogram values. Need not be equally spaced but must be ordered.

Value

A list with these components.

vgram	Variogram 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.

References

See any standard reference on spatial statistics. For example Cressie, Spatial Statistics

See Also

vgram.matrix bplot.xy, vgram.matrix

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$d, look$vgram)
#lines(look$centers, look$stats["mean",], col=4)

brk<- seq( 0, 250,,25)

## or some boxplot bin summaries

bplot.xy( look$d, sqrt(look$vgram), breaks=brk,ylab="sqrt(VG)")
lines(look$centers, look$stats["mean",], 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 )

plot.vgram.matrix(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

A list 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!
```

---

world *Plot of the world*

Description

Plots quickly, medium resolution outlines of large land masses and bodies of water.

Usage

```
world(ylim = c(-90, 90), xlim = NULL, add = FALSE, asp = 1,
                  xlab = "", ylab = "", xaxt = "n", yaxt = "n", eps =
                  0.1, col=1,shift = FALSE,
                  fill=FALSE, col.water="white", col.land="darkgrey", ...)

world.color(xlim= c(-180,180), ylim=c(-90,90),
              col.water="white",col.land="darkgrey", ... )
```

Arguments

ylim	range of latitudes
xlim	range of longitudes
add	logical; if true will add the world map to current plot.

asp	aspect ratio used if add is false, see plot.default.
xlab,ylab	labels for x- and y-axis; empty by default.
xaxt,yaxt	axis type for x- and y-axis; empty by default.
eps	Tolerance to decide when to insert line break about 0 if map is to be shifted. (leave this at .1)
shift	If TRUE shifts to be centered on the Dateline and longitude runs from 0 to 360. If FALSE centers on Prime Meridian and longitude runs from -180 to 180.
col	Color for map lines when fill is FALSE.
fill	If FALSE draws land outlines. If TRUE fills in land and water with different colors.
col.land	Color for land filling.
col.water	Color for water filling.
...	If the land is not filled these are graphical arguments that are passed to the lines (and plot if add is false) function that draws the outline.

If fill is TRUE then these arguments are passed to the polygon function that does the filling.

Details

Both functions use the FIELDS dataset world.dat for the coordinates. The main advantage of this function is that it is fast and easy to modify. The shift option to center over the dateline is useful because often plots of oceanic and atmospheric information center the map this way.

The function world.color can be used separately but is also called by world with fill being TRUE. When used alone it will just add the colored landmasses and water to an existing plot. It is easy to modify just to add the land masses and use the existing back ground color as water. Unfortunately world.color will not work when shift is TRUE. The current code could be modified if you need this option. Thanks to Steve McIntyre for suggesting and testin the fill option.

See Also

US

Examples

```
world()
# add the US
US( add=TRUE,col="blue")

world( fill=TRUE) # land filled in dark grey

## Western Europe (*which* big islands are missing?)
## with a coordinate grid:

world(xlim=c(-10,18),ylim=c(36,60),
    xaxt = "s", yaxt = "s", fill=TRUE, col.land="darkgreen")

box() # add back in the box that was obscured by the ocean fill.
```

```
# add back in outline of land.
world( add=TRUE,lwd=1.5, col="green")

grid()
```

`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