Package 'LatticeKrig'

October 9, 2024

Version 9.3.0

Date 2024-10-07

Title Multi-Resolution Kriging Based on Markov Random Fields

Maintainer Douglas Nychka <nychka@mines.edu>

Description Methods for the interpolation of large spatial datasets. This package uses a basis function approach that provides a surface fitting method that can approximate standard spatial data models. Using a large number of basis functions allows for estimates that can come close to interpolating the observations (a spatial model with a small nugget variance.) Moreover, the covariance model for this method can approximate the Matern covariance family but also allows for a multi-resolution model and supports efficient computation of the profile likelihood for estimating covariance parameters. This is accomplished through compactly supported basis functions and a Markov random field model for the basis coefficients. These features lead to sparse matrices for the computations and this package makes of the R spam package for sparse linear algebra.
An extension of this version over previous ones (< 5.4) is the support for different geometries besides a rectangular domain. The Markov random field approach combined with a basis function representation makes the implementation of different geometries simple where only a few specific R functions need to be added with most of the computation and evaluation done by generic routines that have been tuned to be efficient. One benefit of this package's model/approach is the facility to do unconditional and conditional simulation of the field for large numbers of arbitrary points. There is also the flexibility for estimating non-stationary covariances and also the case when the observations are a linear combination (e.g. an integral) of the spatial process. Included are generic methods for prediction, standard errors for prediction, plotting of the estimated surface and conditional and unconditional simulation. See the 'LatticeKrigRPackage' GitHub repository for a vignette of this package.

Development of this package was supported in part by the National Science Foundation Grant 1417857 and the National Center for Atmospheric Research.

License GPL (>= 2)

URL https://www.r-project.org

Depends R (>= 4.0.0), methods, spam, spam64, fftwtools, fields (>= 9.9)

NeedsCompilation yes

Author Douglas Nychka [aut, cre],
Dorit Hammerling [aut],
Stephan Sain [aut],
Nathan Lenssen [aut],
Colette Smirniotis [aut],
Matthew Iverson [aut],
Antony Sikorski [aut]

Repository CRAN

Date/Publication 2024-10-09 21:40:02 UTC

Contents

`directionCosines` *Utility functions for spherical coordinate and projections.*

Description

Convert back and forth between lon/lat and direction cosines and also project spherical coordinates on a local tangent plane.

Usage

```
directionCosines(x)
toSphere( Grid)
projectionSphere(x1, x2)
```

Arguments

`x`	A two column matrix of lon/lat coordinates in degrees.
`x1`	A vector of direction cosines defining the intersection of the tangent plane with the unit sphere
`x2`	A matrix of direction cosines that will be projected onto the plane defined by x2.
`Grid`	A three column matrix of direction cosines

Details

The conversion functions are based on straight forward definitions of spherical coordinates. The projection function is done by two rotations, first around the Z axis and then around the Y axis.

Value

directionCosines A three column matrix of direction cosines.

toSphere A two column matrix of longitudes and latitudes.

projectionSphere A two column matrix of Euclidean coordinates on the tangent plane to `x1`. The convention is that the origin (0,0) is mapped to `x1` and the X- axis are points along the meridian through `x1`. The Y axis are points on the great circle passing through `x1` and perpendicular to the meridian.

Author(s)

Doug Nychka

Examples

```
#
# icosahedron:
 phi = (1 + sqrt(5))/2
  V = matrix(c(0, 1, phi, 0, 1, -phi, 0, -1, phi, 0, -1, -phi,
        1, phi, 0, -1, phi, 0, 1, -phi, 0, -1, -phi, 0, phi,
        0, 1, -phi, 0, 1, phi, 0, -1, -phi, 0, -1), 12, 3, byrow = TRUE)

# check : library( rgl); plot3d( V, size=10, col="red4" )
# as lon lat:
V2<- toSphere( V)
plot( V2)

# lon lat grid
 lGrid<- make.surface.grid(  list(x= seq( -10,10,, 10), y= seq( -20,20,,10)) )

 dGrid<- directionCosines( lGrid)
 pairs( dGrid)
 # also try:   library( rgl); plot3d(  dGrid)
```

`gridList-class` *Class* "`gridList`". *A description of a regular and multidimensional grid.*

Description

This object is mainly designed to work with methods that take a set of locations organized on a grid. The object is a list where there are as many components as dimensions and each list component is a vector of values being the grid points in that dimension. It is consistent with the older use of the older `grid.list` format used in the fields package. This form is somewhat redundant because for an equally spaced grid all one needs is the beginning value, spacing and number of points but it makes it simpler to pass the grid information to functions such as image and contour.

Usage

```
gridListInfo(gridList)
```

Arguments

`gridList`	A gridList object.

Objects from the Class

This object is a list where each component is a vector of grid points in a particular dimension. For example

```
grid<- structure(list( x= seq( -1,1,,20), y= seq( 0,1,,15)), class= "gridList"
```

would create this object for a 2d grid with 20 and 15 points over the ranges [-1,1] and [0,1]. The component names ("x" and "y" in this case) are optional.

The function `gridListInfo` extracts some summary information that is used to support the summary function for this class.

Methods

LKrigDistance `signature(x1 = "matrix", x2 = "gridList", delta = "numeric")`: ...

Author(s)

Doug Nychka

See Also

LKrigDistance and LKrigDistGrid LKrigLatticeCenters

Examples

```
showClass("gridList")
# a 3-d grid
grid<- structure(
 list( x= seq( -1,1,,20), y= seq( 0,1,,15) ,oneMore = 1:10) ,
 class= "gridList" )
```

IcosahedronGrid *Icosahedral multi-resolution grids*

Description

Creates a multi-resolution grid based on subdividing triangles starting with faces of an icosahedron.

Usage

```
IcosahedronGrid(K)

IcosahedronFaces(K)
```

Arguments

K	Number of levels.

Details

Creates a nearly regular grid by taking the first level as the 12 points from a regular icosahedron. The subsequent levels generate a finer set of points by subdividing each triangular face into 4 new triangles. The three new mid points from the subdivision are added to the previous nodes to give the new level of resolution. The triangles tend to be roughly equilateral and so the nodes will tend to be roughly equally space but there is some variation in distances among nearest neighbors.

To depict the faces and nodes in a snazzy way use the `rgl` package and the following code

```
library( rgl)
# show level 3
Level<- 3
    SGrid <- IcosahedronFaces(4)
    Tri <- SGrid$Face[[Level]]
    L <- dim(Tri)[3]
    plot3d(rbind(c(0, 0, 0)), xlim = c(-1, 1), ylim = c(-1, 1),
        zlim = c(-1, 1), axes = FALSE, xlab = "", ylab = "",
        zlab = "", type = "p", box = TRUE)
    for (k in 1:L) {
        U <- Tri[, , k]
        rgl.triangles(U, col = "grey80")
    }
    plot3d(SGrid$nodes[[Level]], col = "green4", type = "s",
        radius = 0.03, add = TRUE)
```

Value

IcosahedronGrid A list with `K` components each component is a three column matrix giving the direction cosines for each grid point.

IcosahedronFaces A list with components:

MultiGrid The same list returned by IcosahedronGrid.

Faces A list with `K-1` components. Each components are the faces at a given level represented as a three dimensional array (3X3XN with N the number of faces and a given level). The array indices are vertices of triangle, coordinates and faces within a resolution level. e.g. to extract the 10th face (out of 80) for the 2nd level:

```
look<- IcosahedronFaces(3)$Faces
triangle <- (look[[2]])[,,10]
print (triangle)
           [,1]       [,2]      [,3]
[1,] -0.5257311 -0.8506508  0.000000
[2,] -0.8090170 -0.5000000  0.309017
[3,] -0.8090170 -0.5000000 -0.309017
rowSums( triangle^2)
[1] 1 1 1
```

`triangle` will be the 10th face for the second level where the columns are the 3d coordinates of the direction cosines and the rows index the three vertices.

Author(s)

Doug Nychka and Zachary Thomas

See Also

toSphere, directionCosines, projectionSphere

Examples

```
# second level in lon lat coordinates
look<- IcosahedronGrid(3)
lonlat<- toSphere( look[[3]])
plot( lonlat, xlab="lon", ylab="lat")
```

KrigingExampleData *Synthetic data for kriging examples*

Description

This is a sample of 50 (x,y) ordered pairs: the x values are in sorted order, drawn from the interval [0,1]. The smallest x value is exactly 0, and the largest is exactly 1. The y values are the function y=9*x*(1-x)^3 evaluated at each x value, with a small measurement error (normal with mean 0, standard deviation 0.01) added on.

Usage

```
data(KrigingExampleData)
```

Format

The data in `KrigingExampleData` is listed in two vectors, `x` and `y`.

Details

The following code was used to generate the data:

```
set.seed(123)
x <- c(0,sort(runif(48)),1 )
y <-  9*x*(1-x)^3  + rnorm(50, sd=0.01)
KrigingExampleData<- list( x=x, y=y)
save( KrigingExampleData, file="KrigingExampleData.rda")
```

Examples

```
## Not run:
data(KrigingExampleData)
x<- KrigingExampleData$x
y<- KrigingExampleData$y
plot(x,y)
kFit <- LatticeKrig(x, y)
xGrid <- seq(0,1,0.001)
lines(xGrid, predict(kFit, xGrid), col='red')

## End(Not run)
```

`LatticeKrig` *User-friendly spatial prediction and inference using a compactly supported multi-resolution basis and a lattice model for the basis coefficients.*

Description

This is a simple and high level function to fit the LatticeKrig spatial model to a data set. In is simplest form for 2-d spatial data: `obj<-LatticeKrig(x,y)` will fit a LatticeKrig type model to 2d locations `x` and observation vector `y`. Several (actually many!) default choices are made for the multi-resolution spatial covariance in this top level function. It uses the defaults that are based on a "thin-plate spline" like model for the spatial estimator and also uses `LKrigFindLambda` to estimate some covariance parameters (sill and nugget variances) through likelihood maximization (i.e. estimates the measurement and process variances.) For the simplest "black box" use, only the observations and their 2-d locations need to be specified. But please see the caveats below in the Details section and also see the vignette https://github.com/NCAR/LatticeKrig/tree/master/Vignette for a gentle introduction.

Despite the simple syntax, the LatticeKrig function still takes advantage of the multi-resolution features of the basic `LKrig` function and any `LKrig` parameter can be passed through the function call. See the example below for varying the range parameter. Also, see `LKinfo` and`LKrigSetup` for documentation on the complete object that describes the LatticeKrig model and the function to create it easily. See `LKGeometry` for documentation on extending or adding other spatial models to this package.

The returned value from this function can be used subsequently for prediction, conditional simulation, and other parts of the spatial analysis. See `predict.LKrig` and `LKrig.sim.conditional`

Usage

```
LatticeKrig(x, y, Z = NULL,weights = NULL, nlevel = 3, findAwght = FALSE, LKinfo = NULL,
 X=NULL, U=NULL, na.rm =
                 TRUE, tol = 0.005, verbose = FALSE, ...)
## S3 method for class 'LatticeKrig'
print( x, digits=4, ...)
```

Arguments

x	Spatial locations of observations. For the LatticeKrig function this should be a matrix where the columns index the spatial dimensions and rows index the observations. For example for 100 2-d locations, x would be a 100X2 matrix. Or for the function print.LatticeKrig x is the returned object from the LatticeKrig function.
y	Spatial observations. No missing values are allowed.
Z	Linear covariates to be included in fixed part of the model that are distinct from the default first order polynomial in x (i.e. the spatial drift).
X	For linear inverse problems the matrix that maps coefficients of the basis to the predicted values of observations. X must be in spam format. To convert from spind or dense format to spam format see help(spam) as an alternative help(spind2spam). See an example for this extension in the LKrig help file.
U	For linear inverse problems the matrix that maps coefficients of the fixed part of the model to the predicted values of observations. This needs to specified along with X
nlevel	Number of levels for the multi-resolution basis. Each level increases the number of basis functions by roughly a factor of 4.
findAwght	If FALSE the default a.wght parameter (related to correlation range) is set to mimic a thin plate spline. If TRUE this parameter and hence the range is estimated my maximum likelihood.
LKinfo	An optional list giving the full specification of the covariance. If this is missing it will be created internally and returned. If passed this parameterization will be used except lambda will be re-estimated by maximum likelihood.
na.rm	If TRUE NA's are removed from y and x is subsetted.
tol	Tolerance for the log likelihood used to judge convergence.
verbose	If TRUE print out intermediate results.
weights	A weight vector to be used for the measurement error variance This is in the same role as the weight vector in ordinary least squares fitting. I.e. the errors are assumed to have variances tau^2/weights where tau^2 is estimated by maximum likelihood if it is not specified.
...	Additional arguments to pass to LKrig. The easiest way to pass a full specification is to create an LKinfo object beforehand and then just pass that (see example below.) This gives more control and the setup function will do some error checking on arguments. Also see help(LKrig) for a complete list of arguments to pass. For convenience we note that if you get some pesky memory warnings from spam you can set the storage higher by adding the argument choleskyMemory. For example to bump up to 2E6 include: choleskyMemory=list(nnzR= 2E6).
digits	Number of significant digits in printed summary.

Details

Keep in mind that overall LatticeKrig is just a specific type of spatial estimate that is designed to handle larger size data sets. It focuses on a specific form of covariance function, but the estimator is still the Kriging/Multivariate Gaussian Conditional Expectation/BLUE that is standard in this field.

The simplest model fit is:

$$Y_k = p(x_k) + h(x_k) + e_k$$

Y_k is the k^{th} observation at location x_k with measurement error e_k. Here $p(x)$ is a low order polynomial of degree m-1 with the default `m==2`. $h(x)$ is a mean zero Gaussian process with the representation:

$$h(x) = \sum_{l=1}^{L} \sum_{j=1}^{m(j)} \phi_{m,l}(x) c_{m,l}$$

where ϕ are multi-resolution basis functions and the coefficients have mean zero and spatial dependence specified by a Markov random field. Keep in mind that this unusual form still implies a specific covariance function for $h(x)$. In fact one can use the Krig or mKrig function from fields to reproduce the LatticeKrig estimate for smaller data sets and check computations. (See `LKrig.cov` with examples). Details on the basis functions and the Markov random field are given in the `LKrig` help function. Throughout this package we assume the standard deviation of e_k is `tau` and the marginal variance of $h(x)$ is `sigma2`. An important derived parameter of the spatial estimate is `lambda = tau^2/ sigma2` the noise to signal ratio. `tau` and `sigma2` are estimated my restricted maximum likelihood in `LatticeKrig`.

This top level function is built on the more basic function `LKrig` supports a very flexible covariance. `LKrig` depends on the parameters `nlevel`, `a.wght` and `alpha` specifying all these relevant parameters may be discouraging to a new (or impatient!) user. Thus LatticeKrig is a "wrapper" that generates some simplified, default model choices to call the more general function `LKrig`. It is useful for users not fully familiar with the LatticeKrig methodology or those that wish to try a default approach to get a quick look at a spatial analysis. You always go back and add some specific non default choices to the LatticeKrig call (e.g. changing `a.wght`). For the 2-dimensional case the default values are set to give about 4 times as many basis functions as observations, use 5 extra lattice points on the edge to minimize boundary effects, and to use four levels of multi-resolution. An important default is that a linear spatial drift is included in the model so the model will relax to a linear prediction based on the x values in the absence of a spatial component. In other words, the model includes by default a fixed part that is linear in x. The spatial correlation range is nearly stationary and set large to mimic a thin-plate spline. The smoothness mimics the Whittle covariance function (smoothness = 1 for the Matern). (See LKrig.cov.plot to get a plot of the implied covariance function.) LatticeKrig also provides maximum likelihood estimates of the measurement error standard deviation ("tau") and process variance parameter ("sigma2") that are perhaps the parameters that most effect the shape of the estimated spatial field. The ubiquitous parameter lambda throughout LatticeKrig is just the reparameterization lambda == tau^2 / sigma2.

This top level function is pitched with all the caveats that statistical model assumptions should always be checked and applying generic methods to a specific problems without checking the appropriateness can lead to misleading results. So plot your data and try several models. Details on the full computations can be found in the `LKrig` man page. The `lambda = tau^2/ sigma2` parameter in `LKrig` is essential to the Lattice Krig computation and an inappropriate value will result in over or under fitting and incorrect interpolated values. The function `LKrigFindLambda` is used within `LatticeKrig` to estimate a lambda value from the data using maximum likelihood.

One interesting feature of this package is the ability to handle spatial processes on different geometries and the form is specified by the `LKGeometry` argument. The current choices are:

`LKRectangle` A 2 dimensional Euclidean spatial domain. The default

`LKInterval` A 1 dimensional Euclidean spatial domain.

LKBox A 3 dimensional Euclidean spatial domain.

LKRing A 2 dimensional spatial domain where the first coordinate is periodic (on [0,360]) and the second is Euclidean. E.g. a slice around the equator and not having a large latitudinal range.

LKCylinder A 3 dimension model where an additional coordinate is added to the LKRing geometry. This is useful for representing a small section of the sphere where one also has a height component.

LKSphere A full 2-d spherical geometry. Coordinates are given in longitude, latitude but the distances and any structures are on the sphere.

One important feature of this package is that the different geometries all use the same computation engine LKrig, following the same computational algorithm. The differences in setting up the problem and in evaluating the function are implemented as S3 methods. The details of this strategy are described in LKGeometry and allow the user to add new geometries.

This function also supports a model where the observations are simply expressed as linear combinations of the basis function coefficients. Equivalently this is a model where the observed data can be expressed as linear functionals applied to the polynomial term and the spatial process. Typically these linear maps represent observing integrals or weighted combinations of the fields and are important for data that is aggregated over by space. See help(LKrig) for an example of how this model is set up at the end of the Examples section.

Value

The main call inside LatticeKrig is to LKrig and so a LKrig object is returned. Thus all of the functionality that comes with LKrig objects such as predict, summary, predictSurface, etc. remain the same as described in LKrig. Also, see the components residuals and fitted.values in the returned object for these parts of the fitted spatial model. The component LKinfo has all the details that specify the basis functions and co variance model. The component MLE gives details of the likelihood evaluations to estimate the tau and sigma2 parameters.

Author(s)

Doug Nychka

See Also

LKrig, LKrig.setup, LKrigFindLambda, LKinfo, LKrig.sim.conditional

Examples

```
# Load ozone data set
  data(ozone2)
  x<-ozone2$lon.lat
  y<- ozone2$y[16,]

# thin plate spline-like model with the lambda parameter estimated by
# maximum likelihood. Default choices are made for a.wght, nlevel, NC
# and alpha.

  obj<- LatticeKrig( x, y)
```

```
## Not run:
# summary of fit and a plot of fitted surface
  print( obj)
  surface( obj )
  US(add=TRUE)
  points(x)
# prediction standard errors
  out.se<- predictSE( obj, xnew= x)

# predict at observations:
# here x can be any two column matrix of coordinates this
# function returns a vector of predictions
  out.fhat<- predict( obj, xnew= x)

# conveniently predict on a 100X100 grid for plotting
# use the grid.list arugment to give more control over the grid choice.
# output object is the standard list with components x, y, and z
# suitable for contour, persp, image, etc.
 out.surf<- predictSurface( obj, nx=100, ny=100)
# image.plot( out.surf)

## End(Not run)

# running an example by first setting up the model object
# this is the main way to specify  the spatial model components
## Not run:
# this is just a small model to run quickly
# compare the LKinfo object here  to one created implicitly:  obj$LKinfo
LKinfo1<- LKrigSetup( x, NC=5, nlevel=3, a.wght=4.1, nu=1.0)
obj1<- LatticeKrig( x,y, LKinfo= LKinfo1)

## End(Not run)
#
# In this example lon/lat are treated as just Euclidean coordinates
# a quick adjustment for small regions is to account for the difference
# in physical distance in N-S verses E_W
# is to just scale the longitude degrees to be comparable to degrees in latitude
# at least in the middle of the domain. The assumption is that for small spatial
# domains this approximation will not be bad for the coordinates at the edges too.
# You accomplish this by adding a scaling, V matrix:
# Here the V argument is rolled into the LKinfo object created within the function
#
## Not run:
  meanLat<- mean( x[,2])*pi/180
  Vlonlat <- diag(  c( 1/cos(meanLat), 1) )
  obj1<- LatticeKrig( x, y, V = Vlonlat )

## End(Not run)

## Not run:
# Refit using with just one level of  basis functions
# on a 20X20 grid within the spatial domain ( so about 400)
# actually number is 720 ( see obj1b$LKinfo) due adding edge nodes
```

```
# Add an aspect ratio of spatial domain
# and find the a.wght parameter along with nugget and process variances.
# this takes a while partly because LatticeKrig model is not optimized for small data sets!
  obj1b<- LatticeKrig( x, y, nlevel=1, NC=20, findAwght=TRUE)
# rudimentary look at how likelihood was optimized
#log lambda and omega =  log(a.wght-4)/2 are useful parameterization ...
  quilt.plot( obj1b$MLE$lnLike.eval[,c("logLambda","omega")],
       obj1b$MLE$lnLike.eval[,"lnProfileLike.FULL"],
       xlab="loglamda", ylab="omega",
       zlim =c(-640,-612))
  points( obj1b$MLE$lnLike.eval[,c("logLambda","omega")],cex=.25)


## End(Not run)
# fitting replicate spatial data sets
# here we use the common observations over days for the ozone
# data set. Whether these are true replicated fields is in question
# but the analysis is still useful

## Not run:
Y<-  na.omit( t( ozone2$y) )
ind<- attr( Y,"na.action")
X<- ozone2$lon.lat[-ind, ]

out1<- LatticeKrig( X, Y, nlevel=1, NC=20, findAwght=TRUE)
out2<- LatticeKrig( X, Y, nlevel=1, NC=20, findAwght=TRUE,
                        collapseFixedEffect=TRUE)
# compare the two models
# Note second a.wght reflects more spatial correlation when individual
# fixed effect is not removed ( 4.4 verses 4.07)
# nugget variance is nearly the same!
out1$MLE$summary[1:7]
out2$MLE$summary[1:7]




## End(Not run)
## Not run:
# Refit using the tensor product type of basis functions
# (default is "Radial"). An example how an additional argument that is
# passed to the LKrigSetup function to create the LKinfo object.
  obj2<- LatticeKrig( x, y, BasisType="Tensor")

## End(Not run)

#
# A 1-d example with 3 levels of basis functions
# See LKrig for an explanation if nlevel, NC,  alpha and a.wght
# covariance parameters.


## Not run:
```

```
 x<- matrix(rat.diet$t)
 y<- rat.diet$trt
 fitObj<- LatticeKrig( x, y)
# NOTE lots of defaults are set for the model! See print( fitObj)
 plot( x,y)
 xg<- matrix(seq( 0,105,,100))
 lines( xg, predict(fitObj, xg) )

## End(Not run)

## Not run:
#  a 3D example
set.seed( 123)
N<- 1000
x<-  matrix( runif(3* N,-1,1), ncol=3, nrow=N)
y<-   10*exp( -rdist( x, rbind( c(.5,.5,.6) ) )/.5)

# NOTE setting of memory size for Cholesky. This avoids some warnings and
# extra computation by the spam package
LKinfo<- LKrigSetup( x,  nlevel=1,  a.wght= 6.01, NC=6, NC.buffer=2,
                     LKGeometry="LKBox", normalize=FALSE, mean.neighbor=200,
                     choleskyMemory=list(nnzR= 2E6) )
out1<- LatticeKrig( x,y, LKinfo=LKinfo)

glist<- list( x1=seq( -1,1,,30), x2=seq( -1,1,,30), x3 = 0)
xgrid<- make.surface.grid( glist)

yhat<- predict( out1, xgrid)
# compare yhat to true function created above
image.plot( as.surface( glist, yhat))


## End(Not run)
#
#############################################################################
# Including a covariate (linear fixed part in spatial model)
#############################################################################
## Not run:
  data(COmonthlyMet)

  obj  <- LatticeKrig(CO.loc,  CO.tmin.MAM.climate, Z=CO.elev)
  obj2 <- LatticeKrig(CO.loc, CO.tmin.MAM.climate)

# compare with and without linear covariates
  set.panel(1,2)
  surface(obj)
  US(add=TRUE)
  title("With Elevation Covariate")

  surface(obj2)
  US(add=TRUE)
  title("Without Elevation Covariate")
```

```
## End(Not run)
## Not run:
 data(COmonthlyMet)
# Examining a few different "range" parameters
a.wghtGrid<-  4  +  c(.05, .1, .5, 1, 2, 4)^2

#NOTE smallest is "spline like" the largest is essentially independent
# coefficients at each level.  In this case the "independent" end is
# favored but the eff df. of the surface is very similar across models
# indicating about the same separate of the estimates into spatial
# signal and noise
#
for( k in 1:5 ){
obj  <- LatticeKrig(CO.loc,  CO.tmin.MAM.climate, Z=CO.elev,
                      a.wght=a.wghtGrid[k])
cat( "a.wght:", a.wghtGrid[k], "ln Profile Like:",
            obj$lnProfileLike, "Eff df:", obj$trA.est, fill=TRUE)
}

# MLE
obj0  <- LatticeKrig(CO.loc,  CO.tmin.MAM.climate, Z=CO.elev,
                     findAwght=TRUE)
print(obj0$MLE$summary)

## End(Not run)

#############################################################################
# Reproducing some of the analysis for the example in the
# JCGS LatticeKrig paper.
#############################################################################

#### Here is an example of dealing with approximate spherical geometry.
## Not run:
data(NorthAmericanRainfall)
library(mapproj)
x<- cbind(NorthAmericanRainfall$longitude, NorthAmericanRainfall$latitude)
y<- NorthAmericanRainfall$precip
log.y<- log(y)
elev<- NorthAmericanRainfall$elevation
# this is a simple projection as part of this and handled by the mapproj package
x.s<- mapproject( x[,1], x[,2], projection="stereographic")
x.s<- cbind( x.s$x, x.s$y)

# an alternative is to transform coordinates using another projection,
# e.g. a Lambert conformal projection
# with the project function from the rgdal package
# library( rgdal)
# x.s<- project(x,"+proj=lcc +lat_1=22 +lat_2=58 +lon_0=-93 +ellps=WGS84")
# this package has the advantage that the inverse projection is also
# included ( inv=TRUE) so it is easy to evaluate the surface back on a Mercator grid.

obj0<- LatticeKrig(x.s, log.y, Z=elev )
```

```
fitSurface<- predictSurface( obj0, drop.Z=TRUE)
fitSurface$z<-  exp(fitSurface$z)/100
colorTable<- designer.colors( 256, c("red4", "orange", "yellow","green1", "green4", "blue"))
image.plot( fitSurface, col=colorTable)
map( "world", add=TRUE, col="grey30", lwd=3, proj="")


## End(Not run)
```

LKDiag *Create a matrix with given entries on the given diagonals*

Description

This function builds a matrix in which each entry on a given diagonal has the same value. The matrix can be any size, not necessarily square, and the user can specify arbitrary diagonals to fill. If `ncol` is not set, the output will be a square matrix. If `diags` is not set, the entries will be placed as close as possible to the main diagonal; e.g. passing in one value for `entries` will produce a diagonal matrix, and passing in a vector of 3 values will produce a tridiagonal matrix. If an even number of entries are provided, they will be placed symmetrically around the main diagonal, leaving the main diagonal as zeroes.

Usage

```
LKDiag(entries, nrow, diags = NULL, ncol = nrow,
full = FALSE)
```

Arguments

`entries`	The values to put on each diagonal.
`nrow`	The number of rows in the matrix; if `ncol` isn't set, the matrix will be square with `nrow` rows and columns.
`diags`	The diagonals to put the entries on. 0 corresponds to the main diagonal, positive values go above the main diagonal, and negatives go below.
`ncol`	The number of columns in the matrix, if it's different from the number of rows.
`full`	If TRUE return a dense matrix in the usual R matrix format. If FALSE return a sparse matrix in spam format.

Value

LKDiag returns a (dense) matrix with the given diagonals.

Author(s)

Matt Iverson

Examples

```
#produces a 5x5 identity matrix
LKDiag(1, 5)

#produces a 6x6 tridiagonal matrix
LKDiag(c(1, -2, 1), 6)
LKDiag(c(1, -2, 1), 6, diags=(-1:1))

#produces a 5x5 matrix with 3 on the main diagonal and the corners
n <- 5
LKDiag(3, n, diags=c(0, n-1, -(n-1)))

# each of the following produces a 4x6 matrix with a 2 in the diagonal
# below the main diagonal and 5 in the diagonal  above it
LKDiag(c(2, 5), nrow = 4, ncol = 6)
LKDiag(c(2, 5), nrow = 4, ncol = 6, diags = c(-1, 1))
LKDiag(c(5, 2), nrow = 4, ncol = 6, diags = c(1, -1))
```

LKDist *Find all pairwise distances within a maximum distance.*

Description

These are the lower level functions to compute the distances among two sets of locations but being limited to distances less than a maximum threshold (see delta below). These functions are useful for generating a sparse matrix on distances and evaluating a compactly supported function (such as the Wendland). The location - location method supports the distance metrics: Euclidean, spherical, component-wise and Manhattan. `LKDistComponent` and `LKDistComponentGrid` return the coordinate-wise distances and are useful for evaluating a tensor product basis functions.

Usage

```
LKDist(x1, x2, delta, max.points = NULL, mean.neighbor = 50,
             distance.type = "Euclidean")
LKDistComponents(x1, x2, delta, max.points = NULL, mean.neighbor = 50,
    distance.type = "Euclidean")

LKDistGrid(x1, gridList, delta, max.points = NULL, mean.neighbor = NULL,
distance.type = "Euclidean", periodic)

LKDistGridComponents(
x1, gridList, delta,
  max.points = NULL, mean.neighbor = NULL,
  distance.type = "Euclidean")

LKGridCheck(distance.type, x1, gridList )

LKGridFindNmax(n1, max.points, mean.neighbor, delta, gridList)
```

Arguments

gridList	A list with each component vector that specifies the grid points for an equally spaced grid. Can have class gridList. (See also help on gridlist).
n1	Number of rows of x1.
x1	A matrix with rows indexing locations and columns indexing coordinates.
x2	A matrix with rows indexing locations and columns indexing coordinates.
delta	The maximum distance to find pairwise distances.
max.points	Used for dynamically assigning matrix size this should be larger than the total number of pairwise distances less than delta.
mean.neighbor	Used for dynamically assigning matrix size this is the average number of points that are less that delta in distance to the x1 locations.
periodic	A logical vector with length ncol(x1). If a component is TRUE then that dimension is treated as periodic.
distance.type	A text string either "Euclidean", "GreatCircle", "Chordal", "Manhattan".

Value

LKDist and **LKDistGrid** a list representing a sparse matrix in spind format.

LKDistComponent and**LKDistGridComponent** a list representing a sparse matrix using the spind format except the ra component is now a matrix. The columns of ra being the individual distances for each coordinate.

directionCosine a matrix where rows index the different points and columns index x,y,z.

LKGridFindNmax returns the maximum number of nonzero elements expected in a pairwise distance matrix.

LKGridFindNmax checks that the calling arguments are compatible with the pairwise distance computation.

Author(s)

Doug Nychka

See Also

spind2spam, spind2full

Examples

```
set.seed( 123)
x<- matrix( runif(100*2), 100,2)

DMatrix<- LKDist( x,x, delta=.1)
# coerce to spam matrix format
DMatrix2<- spind2spam( DMatrix)

# a grid
gridL<- list( x1= seq(0,1,.2), x2= seq( 0,2,.2) , x3= seq( -1,1,.2))
```

```
class(gridL)<- "gridList"
x1<- cbind( runif( 100), runif(100)*2, 2*(runif( 100) -.5) )
look<- LKDistGrid( x1, gridL, delta=.45)
# check against rdist.
# look2<- rdist( x1, make.surface.grid(gridL))
# look2[ look2 >= .45] <- 0
# max( abs(look- look2)[look>0] )

# test of periodic option
 gridL<- structure(
            list( x1= seq(0,1,.02),
                  x2= seq( 0,1,.02)),
            class="gridList")
 look1<- LKDistGrid( rbind(c(0,0)), gridL, delta=.35,
                     periodic=c(TRUE,FALSE))
 look2<- spind2full(look1)
 image.plot( as.surface( gridL, look2) )

 look1<- LKDistGrid( rbind(c(0,0)), gridL, delta=.35,
                      periodic=c(TRUE,TRUE))
 look2<- spind2full(look1)
 image.plot( as.surface( gridL, look2) )
```

LKGeometry *Tailoring the LatticeKrig model to a specific geometry.*

Description

To implement a spatial model for specific geometry, e.g. a rectangular spatial domain, one needs to specify a few functions that set reasonable default choices, create the lattice, define the Markov random field and define the basis functions. One could also include some specific functions that are particularly efficient for some of the computations for that geometry. This help section gives an overview of what is needed to introduce a new geometry.

A quick start

At the outset if you are an example person (and an impatient one) take a look at the functions created for the 1-d LatticeKrig model (`LKInterval`). (In the source for the package this is the file ModelLKInterval.R in the R subdirectory.) The specific functions are:

LKrigSetupLattice.LKInterval Creates a 1-d lattice.

LKrigSAR.LKInterval Creates a sparse 1-d spatial autoregressive matrix.

LKrigLatticeCenters.LKInterval Returns the spatial locations of the lattice points.

Nothing else is needed to coax the LatticeKrig computation to fit a 1-d spatial model. There are more details and flexibility in specifying the model, but the ones listed above are generic in that they make sense for any LatticeKrig model and are not specific to the 1-d case. The `.LKInterval` tag on each of these functions indicates that these are S3 versions of a "method". The actual source code in LatticeKrig just relies on calling the generic method, e.g. LKrigSetupLattice, and the geometry,

added as a class indicates, which function is actually used. Checking what each of these functions does and then seeing how they are called in LKrigSetup and LKrig.basis will give a simple introduction to how the code is structured. See LKRectangle for details on the most common 2-d geometry.

Use of the LKInfo object

The basic idea behind this package is to consolidate all the details of the spatial model including the geometry in the LKinfo object. The way to create this object is by the function SetupLKinfo and we believe that this function has enough flexibility to avoid modifying this object outside of this function call. Typically for deliberate spatial modeling the first step will be creating the LKinfo object; however, the quick and more "black box" function LatticeKrig is designed to take a minimal amount of information and then calls SetupLKinfo internally to fill out the LKinfo object. In fact, LatticeKrig(x,y) with just x as locations and y as observations will fit a spatial model using reasonable defaults and the returned object from this fit will include the full LKinfo object that it used.

A subtle and perhaps a potential disadvantage with the coding is that the LKinfo object is built in steps within the LKrigSetup function. In particular the method that setups the lattice is called before information is added about the alpha and a.wght parameters. The source code LKrigSetup has been kept concise and it should be clear how the LKinfo is formed. Conceptually, the sequence of steps are:

The calling arguments Initialize the LKinfo object with the all the passed arguments.

Call: setDefaultsLKinfo If necessary add some components that fill in specific default parameters for a given method.

Create basis information The component LKinfo\$basisInfo holds the specific components for the basis functions. Currently there is not much flexibility here are as it is just derived from the calling arguments.

Call: LKrigSetupLattice Setup the lattice information and put in the component LKinfo\$latticeInfo

Call: LKrigSetupAlpha Setup the format for the alpha parameters based on what was passed in and out this in the component LKinfo\$alpha

Call: LKrigSetupAwght Setup the format for the alpha parameters based on what was passed in and out this in the component LKinfo\$a.wght

Check using: LKinfoCheck Check for the required components in the object.

The LKinfo object is a list with class LKinfo and comes with a print method to make it easier to read and also has the class of the geometry, e.g LKInterval for the 1-d model. The functions that are required for a geometry all take the LKinfo object as their first argument and the S3 object dispatching then calls the correct version. Put more simply, if LKinfoExample has class LKInterval then the code LKrigLatticeCenters(LKinfoExample, Level= 1) will result in computing the lattice centers at the first level and using the 1-d geometry. In this way much of this package can use "generic" functions for the computational steps without revealing the extra complications and exceptions for particular models and geometries. The structure of the LKinfo object is not completely rigid. Certain components of the list are required and these are checked by LKinfoCheck. Other components of this list can be geometry and problem dependent and this is a useful way to pass information to the computations. In general when faced with what should be included in LKinfo follow the style in LKInterval but also include additional components that are needed to define the model. Some examples are mentioned below.

Required methods

For any model there are three functions required for a new geometry. By a new geometry we mean any variation from the regular grids and beyond a second order SAR. As a first step choose a name for the new geometry, e.g. MyNewIdea, and this should also be the name passed as the argument LKGeometry in the `LKrigSetup` function. Three new functions need to be supplied and that end in .MyNewIdea and the user will recognize these as S3 methods being added to LatticeKrig.

The specific functions you will have to add to LatticeKrig package are:

LKrigSetupLattice.MyNewIdea(LKinfo) This function will take the information in the LKinfo list and create any indexing and do other computations to create the specific lattice of spatial points. The return value is a list with several basic components that are required. For example `m` is the total number of basis functions in the model. See help(LKrigSetupLattice) for a description of these required arguments. In addition one can add other arguments that help in coding the following two functions. It possible that this function could be avoided by doing all the setup whenever the SAR and the centers functions are used but we suspect this would result in more complicated code and be less modular. The 1-d implementation in the source file ModelInterval.R is a good example to get an idea of the whole process.

LKrigSAR.MyNewIdea(LKinfo, Level) Returns a sparse spatial autoregressive matrix at a specific level of the multi-resolution and in `spind` sparse matrix format. See `help( LKrigSAR)`. Note that the lattice information from the previous function is in the component `latticeInfo` and so this function needs to access any lattice information in this way from the LKinfo object. e.g. `LKinfo\$latticeInfo\$m` are the total number of basis functions.

LKrigLatticeCenters.MyNewIdea(LKinfo, Level) Returns the spatial locations of the lattice points at a specific level of the multi-resolution. This can either be as a matrix with columns indexing dimension and rows indexing points or another class of object. These locations (or object) are subsequently passed to the second argument of the distance function `LKrigDistance` to create the basis functions.

For initially creating a new geometry one can just rely on this function returning the lattice locations. Once the whole geometry works one can consider returning a more specialized object and pairing these with more efficient distance finding algorithms. For example if the lattice points are on a regular, equally spaced grid the nearest neighbor locations can be found quickly just by simple arithmetic and this is much faster than a search. This is implemented for the LKInterval, LKBox, and LKRectangle geometries by creating the class `gridList` for the grid of lattice centers and passing this object directly into the second argument of the distance function LKrigDistance. See also `help(LKrigLatticeCenters)` and `help(LKrigDistance)`

Optional Methods

To make the modeling more flexible and take advantage of faster computation some additional methods can be added but are not required. Any new version should add your geometry name to the end. e.g. `setDefaultsLKinfo` should have the new name: `setDefaultsLKinfo.MyNewIdea` and have the default arguments listed in the generic function (`setDefaultsLKinfo`).

setDefaultsLKinfo This function takes the skeleton LKinfo list based on the arguments passed to LKrigSetup and can modify and add to them. The returned value is now taken as the initial LKinfo object. The default method is that no changes are made. But adding versions to this method can be very useful for specific cases (e.g. see `setDefaultsLKinfo`) without complicating the main function with lots of conditional statements.

LKrigSetupAlpha These methods convert the alpha information passed into LKrigSetup to a list format. A default is provided if you do not supply a specific version. See help(LKrigSetupAlpha).

LKrigSetupAwght These methods convert the a.wght information passed into LKrigSetup to a list format.

LKrigNormalizeBasisFast This is a method to implement any fast methods for normalizing the basis functions. There is no default method because it is assumed this is geometry dependent. See LKrigNormalizeBasisFast.LKRectangle as an example of how this is done. Note that there is already a "slow" method, LKrigNormalizeBasis involving a Cholesky solve of the precision matrix for each point where the basis function is evaluated.

LKinfoCheck This function checks the LKinfo object for necessary components and does some rudimentary comparisons to make sure arrays are the right size. It is always good to add more checks!

Author(s)

Doug Nychka

See Also

LKrigSetup, LKrigSAR, LKrigLatticeCenters

LKinfoCheck *Check the LKinfo object*

Description

This method performs some simple checks on the LKinfo object before it is returned by LKrigSetup. One benefit is that it checks for the essential components in each of the parts of the LKinfo object. Currently there is only a default method supplied.

Here is an example of how the checks work

```
LKinfo<- LKrigSetup( cbind(c(0,1)), LKGeometry="LKInterval",
    a.wght=5, nlevel=2, nu=1.5, NC=4)
    LKinfo$alpha
# corrupt it
    LKinfo$alpha <- list( 1,.5,.25)
try( LKinfoCheck( LKinfo) )
```

Usage

```
LKinfoCheck(object, ...)
## Default S3 method:
LKinfoCheck(object, ...)
## S3 method for class 'LKRectangle'
LKinfoCheck(object, ...)
```

Arguments

object	An LKinfo object to be checked.
...	Additional arguments to method.

Value

There is no returned value. The benefit is the side effect of an error message and a "stop if the are problems in the LKinfo object.

Author(s)

Doug Nychka

See Also

LKrigSetup

LKRectangle *Summary of the LKRectangle geometry for a standard two dimensional spatial domain.*

Description

The basic LatticeKrig model is for a 2-d spatial domain and is identified by the geometry class LKRectangle. The lattices used in this case are equally spaced regular grids that are nested and the default distance function is standard Euclidean distance. The number of lattice points in the first level is NC inside the spatial domain and the subsequent levels decrease the lattice spacing by factors of 2. The spatial autoregressive coefficients, referred to as the a.wght parameter(s) have several levels of detail depending on the stationarity and isotropy. In the simplest case a.wght is the central value, and should be greater than 4. Also the four nearest neighbors are take to be -1. In the most complicated `a.wght` can include all 8 nearest neighbors can be specified differently at every lattice node. To be stable, the sum of the `a.wght` parameters for a node should be greater than zero.

Details

Here is a simple and small three level example that sets up the LatticeKrig model for spatial estimation and prediction. It assumes a spatial domain with extent [-1,1] in the horizontal and [-1,1] in vertical and beginning with 4 grid points in lowest level and with a default of 5 extra points outside the domain for boundary corrections. The assumed SAR model defaults to a central value of 4.1 and the 4 nearest weights are -1. Defining the extent of the spatial domain explicitly was done to simplify the example. Typically one just defines the domain to the minimum and maximum x and y locations of the spatial data (and the locations can passed as the first argument in the example below instead of finding ranges.) Note that does not mean that the spatial locations fill out a rectangle and they do not need to regularly spaced. The `nu` parameter is a handy way to specify the relative weights given each level. For a larger numbers of levels this parameter is equivalent to the Matern smoothness parameter.

Setting up the LKinfo object.

```
 sDomain<- cbind( c(-1,1), c( -1, 1))
LKinfo<- LKrigSetup(sDomain, nlevel=3, NC=4, a.wght=4.1, nu=1.0)
```

with the result

```
print(LKinfo)
Classes for this object are:  LKinfo LKRectangle
The second class usually will indicate the geometry
     e.g.  2-d rectangle is  LKRectangle

Ranges of locations in raw scale:
     [,1] [,2]
[1,]   -1   -1
[2,]    1    1

Number of levels: 3
delta scalings: 0.6666667 0.3333333 0.1666667
with an overlap parameter of  2.5
alpha:  0.7619048 0.1904762 0.04761905
based on smoothness nu =  1
a.wght:  4.1 4.1 4.1

Basis  type: Radial using  WendlandFunction  and
Euclidean  distance.
Basis functions will be normalized

Total number of basis functions  1014
 Level Basis size
     1        196 14 14
     2        289 17 17
     3        529 23 23

Lambda value:  NA
```

About the lattice The number of nodes define the number of basis functions and at first may seem a bit mysterious. However at the first level we get 4 lattice points with 5 extra boundary points added in the x direction amounting to 14 total and similarly 14 in the y direction because it is exactly of the same size. The default is the the parameter NC determines the number of lattice points in the larger dimension and the smaller dimension is divided according to the spacing from the longer dimension. For example if the y extent was [-1,.5] the number of lattice points would be spaced according to

```
delta scalings: 0.6666667 0.3333333 0.1666667
```

Thus `seq( -1, .5, 0.6666667 )` are the (three) generated points within the spatial domain, and of course 5 would also added beyond each endpoint.

To query the LKinfo object this information is in the `latticeInfo` component. E.g. the first set of lattice locations.

```
LKinfo$latticeInfo$grid[[1]]
$x
 [1] -4.3333 -3.6667 -3.0000 -2.3333 -1.6667
 [6] -1.0000 -0.3333  0.3333  1.0000  1.6667
[11]  2.3333  3.0000  3.6667  4.3333

$y
 [1] -4.3333 -3.6667 -3.0000 -2.3333 -1.6667
 [6] -1.0000 -0.3333  0.3333  1.0000  1.6667
[11]  2.3333  3.0000  3.6667  4.3333

attr(,"class")
[1] "gridList"
```

By default equal spacing of the lattice is assumed in the x and y directions. To change this use the optional `V` argument to scale the coordinates. For example, to set 4 lattice points in both dimension for the above example:

```
sDomain2<- cbind( c(-1,1), c( -1, .5))
LKinfo<- LKrigSetup(sDomain2, nlevel=3, NC=4,
                a.wght=4.1, nu=1.0,
                V= diag(c(2, 1.5)) )

print(LKinfo$latticeInfo$mx)

     [,1] [,2]
[1,]   14   14
[2,]   17   17
[3,]   23   23
```

About the basis functions With the lattice points defined the default basis functions are radial Wendland functions centered at each point. The scaling of the basis functions is determined by the `overlap` and the `delta` values. In R code, if the lattice point is x0 at level l then the basis function evaluated at location x1 is

```
basisFunctionValue <- Wendland( rdist( x1,x0)/( delta[l]*overlap))
```

If V is included then x1 is transformed as `x1%*%solve(V)` before evaluating in the basis function.

About `a.wght`

For this geometry the basic form of awght is as a list with as many components as levels. However, the LKrigSetup function will reshape a scalar or vector argument in this this format. `a.wght` can take the following forms:

Scalar value: In this case the value is used at all lattice points and at all levels in the SAR. Four nearest neighbors are set to -1.

List/vector of length `nlevel` In this case separate values for the a.wght will be used for the central SAR value at each level.

List of vectors With a list of length `nlevel` and each component is a 9 element vector, the values in the vector correspond to the central lattice point and 8 nearest neighbors with the indexing:

```
1 4 7
2 5 8
3 6 9
```

E.g. the 5 element is the center, 3 is the lower left hand corner etc.

Non-stationary models This is the nuclear option to handle a completely non-stationary correlation structure! In this case one specifies the models by passing prediction objects for one or more of `sigma2.object`, `alphaObject` or `a.wghtObject`.

Author(s)

Doug Nychka

See Also

`LatticeKrig` `LKGeometry` `LKrig`

Examples

```
# the grid with only 2 extra boundary points
  sDomain<- cbind( c(-1,1), c( -1, 1))
  LKinfo<- LKrigSetup(sDomain, nlevel=3, NC=4, a.wght=4.1,
           NC.buffer=2, alpha=c(1,.5,.125) )
  LKgrid<- LKinfo$latticeInfo$grid
  plot(   make.surface.grid(LKgrid[[1]]),
                          pch=16, cex=1.5)
  points( make.surface.grid(LKgrid[[2]]),
                          pch=15, cex=.8, col="red" )
  points( make.surface.grid(LKgrid[[3]]),
                          pch="+", col="green" )
  rect(sDomain[1,1],sDomain[1,2],
     sDomain[2,1],sDomain[2,2], lwd=3 )

# basis functions on a grid
# this function actually evaluates all of them on the grid.
  xg<- make.surface.grid(
        list(x=seq( -2,2,,80), y=seq( -2,2,,80)) )
  out<- LKrig.basis( xg, LKinfo)
# basis functions 20, 26, 100  and 200
  plot(   make.surface.grid( LKgrid[[1]] ) ,
                          pch=16, cex=.5)
  rect(sDomain[1,1],sDomain[1,2],
     sDomain[2,1],sDomain[2,2], lwd=3,border="grey" )
  contour( as.surface(xg, out[,20]), col="red1",
                                     add=TRUE)
  contour( as.surface(xg, out[,36]), col="red4",
                                     add=TRUE)
  contour( as.surface(xg, out[,100]), col="blue1",
                                     add=TRUE)
  contour( as.surface(xg, out[,200]), col="blue4",
                                      add=TRUE)
  title( "basis functions 20, 26, 100, 200")
```

LKrig *Spatial prediction and inference using a compactly supported multi-resolution basis and a lattice model for the basis coefficients.*

Description

A variation of Kriging with fixed basis functions that uses a compactly supported covariance to create a regular set of basis functions on a grid. The coefficients of these basis functions are modeled as a Gaussian Markov random field (GMRF). Although the precision matrix of the GMRF will be sparse the model can still exhibit longer ranges of spatial dependence. The multi-resolution feature of this model allows for the approximation of a wide variety of standard covariance functions. There are also some simple extensions where this function can be used to solve a linear inverse type problem (see X and U below).

Usage

```
LKrig( x, y, weights = NULL, Z = NULL, LKinfo = NULL, iseed =
                  NA, NtrA = 20, use.cholesky = NULL, return.cholesky =
                  TRUE, X = NULL, U = NULL, wX = NULL, wU = NULL,
                  return.wXandwU = TRUE,
                  ..., getVarNames = TRUE, verbose = FALSE)

## S3 method for class 'LKrig'
predict(object, xnew = object$x, Znew = NULL, drop.Z = FALSE,
                  just.fixed = FALSE, return.levels = FALSE,
                  collapseFixedEffect=object$collapseFixedEffect, ...)

## S3 method for class 'LKrig'
predictSE(object, xnew = NULL, Znew = object$Z, verbose = FALSE,
                  ...)

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

## S3 method for class 'LKrig'
predictSurface(object, grid.list = NULL, extrap = FALSE,
 chull.mask = NA, nx = 80, ny = 80, xy = c(1, 2), verbose = FALSE,
                  ZGrid = NULL, drop.Z = FALSE, ...)

## S3 method for class 'LKrig'
print( x, digits=4, ...)

## S3 method for class 'LKrig'
summary( object, digits=4, stripAwght = TRUE,...)

createLKrigObject(x, y, weights = NULL, Z, X, U, LKinfo,
                  xName = "xVar", ZName = "ZVar", UName = "UVar",
```

```
                verbose = FALSE)
```

Arguments

x	Spatial locations of observations. Or the LKrig object for printing.
y	Spatial observations.
weights	A vector that is proportional to the reciprocal variances of the errors. I.e. errors are assumed to be uncorrelated with variances tau^2/weights.
Z	Linear covariates to be included in fixed part of the model that are distinct from the default first order polynomial in x (i.e. the spatial drift).
chull.mask	An additional constraint for evaluating predictions see help(in.poly). Usually this is not needed.
collapseFixedEffect	If FALSE the fixed part of the model is found separately for each replicated data set. If TRUE the estimate is polled across replicates.This is largely a modeling decision whether variation among the replicate fields is due to the spatial component or also include variation in the fixed effects across replicates – guess they are not really fixed then!
digits	Number of digits in printed output.
drop.Z	If true the fixed part will only be evaluated at the spatial drift polynomial part of the fixed model. The contribution from the other, Z, covariates in the fixed component will be omitted.
extrap	If TRUE will extrapolate predictions beyond convex hull of locations.
getVarNames	If TRUE the name of the pasted object for x, Z or U will be used in the column names for the fixed model coefficients. This should be set to FALSE when LKrig is called from the do.call function because the name of the original matrices are not preserved.
grid.list	A list with components x and y specifying the grid (see help(grid.list)).
iseed	Random seed used in the Monte Carlo technique for approximating the effective degrees of freedom (trace of the smoothing matrix). If NA, no seed is set.
just.fixed	If TRUE just the fixed part of the model is evaluated.
LKinfo	An object whose components specify the LatticeKrig spatial model. This is usually created by the function LKrigSetup. If NULL, this object is created and returned as a component of the LKrig object.
nx	Number of grids in x for prediction grid.
ny	Number of grids in y for prediction grid.
NtrA	Number of random samples used in Monte Carlo method for determining effective degrees of freedom.
object	An LKrig object.
return.cholesky	If TRUE the Cholesky decomposition is included in the output list (with the name Mc). This is needed for some of the subsequent computations such as

finding prediction standard errors. Set this argument to FALSE to avoid much larger objects when the decomposition is not needed. This option is often paired with a subsequent call to `LKrig` with `use.cholesky`. See the MLE.LKrig source code for details.

`return.levels` If TRUE the predicted values for each level are returned as columns in a matrix.

`return.wXandwU` If TRUE the matrices wX and xU are included in the LKrig object.

`stripAwght` If TRUE the attributes for the a.wght returned by the summary are wiped out. Why do this? In some cases the a.wght list has some attributes attached to it that are large. For example if fastNormalize is TRUE for LKRectangle there are some precomputed matrices that are included to speed the normalization of the basis functions. These attributes make the a.wght itself hard to print out and if the summaries for many fits are accumulated these attributes greatly inflate the size of the results.

`U` For linear inverse problems the matrix that maps coefficients of the fixed part of model to the predicted values of observations.

`UName` Name to use for the U variable.

`verbose` If `TRUE` print out intermediate results and messages.

`use.cholesky` Use the symbolic part of the Cholesky decomposition passed in this argument.

`X` For linear inverse problems the matrix that maps coefficients of the basis to the predicted values of observations. X must be in spam format.

`xName` Name to use for the x variable.This is needed so it can be passed from the top level `LKrig` function.

`wU` The matrix U multiplied by the weights.

`wX` The matrix X multiplied by the weights.

`xnew` Matrix of locations for prediction.

`xy` Order of evaluating surface coordinates. This would be used if for example a lon-lat surface needed to be transposed as a "lat-lon" object. Usually not needed.

`ZName` Name to use for the Z variable.

`Znew` Values of covariates, distinct from the spatial drift for predictions of data locations.

`ZGrid` An array or list form of covariates to use for prediction. This must match the `grid.list` argument. e.g. ZGrid and grid.list describe same grid. If ZGrid is an array then the first two indices are the x and y locations in the grid. The third index, if present, indexes the covariates. e.g. For evaluation on a 10X15 grid and with 2 covariates. `dim( ZGrid) == c(10,15, 2)`. If ZGrid is a list then the components x and y should match those of grid.list and the z component follows the shape described above for the no list case.

`...` For the methods additional arguments to pass to generic methods.

For LKrig these extra arguments are interpreted as appropriate for the LKrigSetup function and in fact they are just passed through to this function to create an LKinfo object. Of particular use is resetting the size of the memory allocation for the sparse decomposition in spam. This is done by the argument `choleskyMemory`. This is a list in the format of the spam `memory` argument for the sparse cholesky

function. For example if a warning is printed that nnzR needs to be at least 2e5 in size, passing `choleskyMemory= list( nnzR= 2e5)` will avoid this warning.

See the help on `LKrigSetup` for explanation of the options. Typically one would setup the LKinfo object outside of this call and just pass a few arguments that are being varied. In particular to use LKrig with different lambda values `lambda = 1e-3` in the call would reset the lambda value to 1e-3. The minimal set of arguments for a 2-d rectangular domain, however, are `alpha`, `nlevel`, `NC`, and `a.wght`.

alpha The sequence of positive weights for each level of the multi-resolution process. Usually these sum to one and are interpreted as the variance of the process at each level.

a.wght The weight given to the central lattice point in the spatial autoregression (see details below).

nlevel Number of levels for the multi-resolution basis. Each level increases the number of basis functions by roughly a factor of 4.

NC Number of lattice points in first level and along the largest dimension. e.g. if NC=5 and the domain is square the domain will contain 25 lattice points at the first level. Note that there may be additional lattice points added as buffers outside the spatial domain (default is 5 on each side).

Some additional arguments are `V` that defines a linear scaling of the coordinates before the LatticeKrig model is applied. See details below.

Details

This method combines compactly supported basis functions and a Markov random field covariance model to provide spatial analysis for large data sets. The model for the spatial field (or spatial process) is

f(x) = N(x) + Z d + sum Phi.j(x) c.j.

x is a location in two dimensions, N(x) is a low order (linear) polynomial, Z is a matrix of spatial covariates and d a coefficient vector. Phi.j for 1 <= j <= m is a set of fixed basis functions and c.j the coefficients. The variance of f(x) is given by the parameter `sigma2` throughout this package. As explained below the process f is a sum of `nlevel` independent processes that have different scales of spatial dependence. The `alpha` gives the relative weighting between these processes. Thus, the minimum set of parameters needed to describe the covariance of f are the integer `NC`, two scalars `sigma2` and `a.wght`, and a vector `alpha`. `alpha` has a length of the number of multi-resolution levels but we recommend that it be constrained sum to one. The parameter sigma2 is the marginal variance of the process and this multiples alpha. Thus in total there are a 1 + 2 + (nlevel - 1) parameters for a minimal specification of the covariance. Note that this parsimonious specification results in a covariance that is close to being stationary and isotropic when `normalize` is `TRUE`. An additional constraint on `alpha` is to make the weights `alpha[j]` proportional to `exp( - 2*j*nu)` where `nu` controls decay of the alpha weights. There is some theory to suggest that `nu` is analogous to the smoothness parameter from the Matern family (e.g. nu=.5 approximates the exponential). In this case the covariance model requires just four parameters, `NC, sigma2, a.wght,nu`.

The data model is

Y.k = f(x.k) + e.k

Y.k are (scalar) observations made at spatial locations x.k with e.k uncorrelated normal errors with variance tau^2/weights. Thus there is a minimum of one new parameter from the data model:

tau. Note that prediction only depends on the ratio `lambda = tau^2/ sigma2` and not surprisingly lambda plays a key role in specifying and fitting a spatial model. Also taken with the model for f the minimum parameters needed for a spatial prediction are still four `NC,a.wght, nu` and `lambda`. For fixed lambda there are closed form expressions for the MLEs for tau and sigma2. LKrig exploits this feature by depending on lambda and then computing the MLEs for tau and sigma2.

The data model can also be written in vector form as

Y = T d + PHI c + e

Where T is a matrix that combines the low order polynomial with other possible covariates and PHI is the matrix basis functions evaluated at the observations. A simple extension is considering a linear inverse problem form. This is an experimental of LatticeKrig that is intended for solving large linear inverse problems. In this case one supplies to the `LKrig` function two matrices (in spam sparse format) such that

Y = U d + X c + e.

This features allows for observations that are linear functionals of f and not necessarily the function evaluated at the observation locations. This model is useful for working with demographic or tomographic problems where the observations are expressed as particular integrals of f over different regions or different projections. See the last example on how to use this option. If U and X are not supplied the default is to consider a model with observations made at evaluating f at the observation locations.

About dense matrix computations: The value of this package is to handle larger spatial data sets by exploiting sparse matrix methods using the spam package. However, for small problems, checking, or timing it is useful to do the computations using the usual dense matrix decomposition. If the component `dense` in the LKinfo object is `TRUE` then dense (i.e. standard) methods will be used. Set this switch from the LKrigSetup function.

Spatial prediction: The basis functions are assumed to be fixed and the coefficients follow a multivariate Gaussian distribution. Given this spatial model for f, it is possible to compute the conditional expectation of f given Y and also maximize the likelihood for the model parameters, lambda, alpha, and a.wght. This setting is known as fixed rank Kriging and is a common strategy for formulating a spatial model. Typically fixed rank Kriging is used to reduce the dimension of the problem by limiting the number of basis functions. We take a different approach in allowing for models that might even have more basis functions than observations. This provides a spatial model that can come close to interpolating the observations and the spatial process is represented with many degrees of freedom. The key is to make sure the model ingredients result in sparse matrices where linear algebra is required for the computations. By doing so in this package it is possible to compute the estimates and likelihood for large numbers of spatial locations. This model has an advantage over covariance tapering or compactly supported covariance functions (e.g. `fastTps` from `fields`), because the implied covariance functions can have longer range correlations.

Radial basis functions (Phi.j) : The basis functions are two-dimensional radial basis functions (RBFs) that are derived from scaling and translations of a single covariance function. The default in LatticeKrig is to use the Wendland compactly supported stationary covariance (order 2 for 2 dimensions) that is scaled to be zero beyond a distance of 1. For d the distance between spatial locations, this Wendland function has the standard form:

`(1 - d)^6 * (35 * d^2 + 18 * d + 3))/3` for d in [0,1]

0 otherwise.

For a single level the RBFs are centered at a regular grid of points and with radial support `delta*overlap` where `delta` is the spacing between grid points. We will also refer to this grid of centers as a lattice

and the centers are also referred to as "nodes" in the RBF literature. The overlap for the Wendland has the default value of 2.5 and this represents a compromise between the number of nonzero matrix elements for computation and the shape of the covariance functions.

To create a multi-resolution basis, each subsequent level is based on a grid with delta divided by 2. See the example below and `help(LKrig.basis)` for more details. For multiple levels the basis functions can be grouped according to the resolution levels and the coefficients can be grouped in a similar manner. There is one important difference in the basis construction – a normalization – and this aspect makes it different from a simple radial basis function specification and is described below.

Markov random field (GMRF) for the coefficients (c.j) : Because the coefficients are identified with locations on a lattice it is easy to formulate a Markov random field for their distribution based on the relationship to neighboring lattice points. The distribution on the basis function coefficients is a multivariate normal, with a mean of zero and the the precision matrix, Q, (inverse of Q is the covariance matrix). Q is partitioned in a block diagonal format corresponding to the organization of the basis functions and coefficients into levels of resolution. Moreover, coefficients at different levels are assumed to be independent and so Q will be block diagonal. If `nlevels` are specified, the ith block has a precision matrix based on a spatial autoregression with `a.wght[i]` being related to the spatial autoregressive parameter(s). Schematically in the simplest case the weighting for an interior lattice point with its four neighbors is

```
.    .      .      .    .
.    .     -1      .    .
.   -1   a.wght   -1    .
.    .     -1      .    .
.    .      .      .    .
```

The fundamental idea is that these weights applied to each point in the lattice will result in a lattice of random variables that are independent. The specific precision matrix for each block (level), Q.i, is implemented by `LKrig.MRF.precision`. In the case when alpha is a scalar, let C.i be the vector of basis coefficients at the ith level then we assume that `B %*% C.i` will be independent N(0,1) random variables. By elementary properties of the covariance it follows that the precision matrix for C.i is Q.i= t(B)%*%B. Thus, given B one can determine the precision matrix and hence the covariance matrix. Each row of B, corresponding to a point in the lattice in the interior, is "a" (`a.wght[i]`) on the diagonal and -1 at each of the four nearest neighbors of the lattice points. Points on the edges and corners just have less neighbors but get the same weights.

This description is a spatial autoregressive model (SAR). The matrix Q will of course have more nonzero values than B and the entries in Q can be identified as the weights for a conditional autoregressive model (CAR). Moreover, the CAR specification defines the neighborhood such that the Markov property holds. Values for `a.wght[i]` that are greater than 4 give reasonable covariance models. Moreover setting a.wght[i] to 4 and `normalize` to FALSE in the call to LKrig will give a thin-plate spline type model that does not have a range parameter. An approximate strategy, however, is to set a.wght close to 4 and get some benefit from the normalization to reduce edge effects.

Multi-resolution process Given basis functions and coefficients at each level we have defined a spatial process g.i that can be evaluated at any location in the domain. These processes are weighted by the parameter vector alpha and then added together to give the full process. It is also assumed

that the coefficients at different resolution levels are independent and so the processes at each level are also independent. The block diagonal structure for Q does not appear to limit how well this model can approximate standard spatial models and simplifies the computations. If the each g.i is normalized to have a marginal variance of one then g will have a variance that is the sum of the alpha parameters. Usually it is useful to constrain the alpha parameters to sum to one and then include an additional variance parameter, sigma2, to be the marginal variance for g. So the full model for the spatial process used in LatticeKrig is

g(x) = sqrt(sigma2) * sum.i sqrt(alpha[i]) * g.i(x)

The specification of the basis and GMRF is through the components of the object LKinfo, a required component for many LatticeKrig functions. High level functions such as `LKrig` only require a minimal amount of information and combined with default choices create the LKinfo list. One direct way to create the complete list is to use `LKrigSetup` as in the example below. For `nlevel==1` one needs to specify `a.wght`, `NC`, and also `lambda` related to the measurement error variance. For a multi-resolution setup, besides these parameters, one should consider different values of `alpha` keeping in mind that if this vector is not constrained in some way (e.g. `sum(alpha)==1`) it will not be identifiable from `lambda`.

The covariance derived from the GMRF and basis functions: An important part of this method is that the GMRF describes the coefficients of the basis functions rather than the field itself. Thus in order to determine the covariance for the observed data one needs to take into account both the GMRF covariance and the expansion using the basis functions. The reader is referred to the function `LKrig.cov` for an explicit code that computes the implied covariance function for the process f. Formally, if P is the covariance matrix (the inverse of Q) for the coefficients then the covariance between the field at two locations x1 and x2, will be

sum_ij P_ij Phi.i(x1) Phi.j(x2)

Moreover, under the assumption that coefficients at different levels are independent this sum can be decomposed into sums over the separate levels. The function `LKrig.cov` evaluates this formula based on the LKrig object (`LKinfo`) at arbitrary groups of locations returning a cross covariance matrix. `LKrig.cov.plot` is a handy function for evaluating the covariance in the x and y directions to examine its shape. The function `LKrig.cov` is also in the form to be used with conventional Kriging codes in the fields package (loaded by LatticeKrig) `mKrig` or `Krig` and can be used for checking and examining the implied covariance function.

Normalizing the basis functions The unnormalized basis functions result in a covariance that has some non-stationary artifacts (see example below). For a covariance matrix P and for any location x one can evaluate the marginal variance of the process using unnormalized basis functions for each multi-resolution level. Based this computation there is a weighting function, say w.i(x), so that when the normalized basis w.i(x) Phi.i(x) is used the marginal variance for the multi-resolution process at each level will be one. This makes the basis functions dependent on the choice of Q and results in some extra overhead for computation. But we believe it is useful to avoid obvious artifacts resulting from the use of a finite spatial domain (edge effects) and the discretization used in the basis function model. This is an explicit way to make the model stationary in the marginal variance with the result that the covariance also tends to be closer to a stationary model. In this way the discretization and edges effects associated with the GMRF can be significantly diminished.

The default in `LKrig` is `normalize = TRUE`. It is an open question as to whether all levels of the multi-resolution need this kind of explicit normalization. There is the opportunity within the `LKrig.basis` function to only normalize specific levels with the `normalize` being extended from a single logical to a vector of logicals for the different levels. To examine what these edge effect

artifacts are like the marginal variances for a 6 by 6 basis is included at the end of the Examples Section.

Non-stationary and anisotropic modifications to the covariance The simplest way to build in departures from isotropy is to scale the coordinates. The device to specify this transformation is including the `V` argument in setting up the LKinfo object or passed in the ... for this function. `V` should be a matrix that represents the transposed, inverse transformation applied to the raw coordinates. For example if `x1` are locations d- dimensions (i.e. x1 has d columns) and `V` is a dXd matrix then the transformed coordinates are

x1Transformed <- x1 %*% t(solve(V))

and all subsequent computations will use the transformed coordinates for evaluating the basis functions. This also means that the lattice centers are locations in the transformed scale. The following example might be helpful:

```
set.seed(123)
x<- matrix( runif( 200),100,2)
V<- cbind( c( 2,1), c( -1,1) )
y<-  x[,1] + x[,2]
LKinfo<- LKrigSetup( x, V=V, nlevel=1, a.wght=5, NC=20)

gridCenters<- LKrigLatticeCenters( LKinfo, Level=1)
# grid in transformed space:
plot( make.surface.grid( gridCenters))
# transformed data:
points( x%*%solve(t(V)), col="red", pch=16)
# the model is fit to these locations.
```

Keep in mind that a practical use of this argument is just to scale a variable(s) that is `V`, a diagonal matrix with diagonal elements being the values to scale the individual coordinates.

Given that the process at each level has been normalized to have marginal variance of one there are several other points where the variance can be modified. The variance at level i is scaled by the parameters `alpha[i]` and the marginal variance of the process is scaled by `sigma2`. Each of these can be extended to have some spatial variation and thus provide a model for non-stationarity.

An option in specifying the marginal variance is to prescribe a spatially varying multiplier. This component is specified by the object `sigma2.object`. By default this is not included (or assumed to be identically one) but, if used, the full specification for the marginal variance of the spatial process at location `x` is formally: `sigma2 * predict(sigma2.object, x) * sum( alpha)` There is then a problem of identifiability between these and a good choice is to constrain `sum(alpha) ==1` so that `sigma2* predict(sigma2.object, x)` is associated with the marginal variance of the full spatial process.

A second option is to allow the alpha variance component parameters to vary across space and at each level. The model in this case could be

g(x) = sqrt(sigma2(x)) * sum.i sqrt(alpha(x).i) * g.i(x)

To specify this case `alpha` should be a list with `nlevel` components and each component being a "predict" object that will evaluate the alpha variance at a given level at arbitrary spatial locations. Explicitly alpha should be set up so that `predict(alpha[[l]], x1)` will return a vector of values for alpha at level `l` and at locations `x1`. This happens in `LKrig.basis` and the user is referred to

this function to see how it is handled and possibly to make modifications to the model. Similar to the scalar alpha case we recommend that the alpha's across levels sum to one and it still may make sense to have the sigma2 parameter vary across space as well. Here alpha would control a varying correlation range and sigma2 the marginal variance.

LKrig also has the flexibility to handle more general weights in the GMRF. This is accomplished by `a.wght` being a list with as many components as levels. If each component is a vector of length nine then these are interpreted as the weights to be applied for the lattice point and its 8 nearest neighbors see the commented source code in `LKrigSAR.LKRectangle` for details. If each component is a matrix then these are interpreted as the (non-stationary) center weights for each lattice point. Finally if the component is an array with three dimensions this specifies the center and 8 nearest neighbors for every point in the lattice. At this point the choice of these weights beyond a stationary model is experimental and we will defer further documentation of these features to a future version of this package.

The smoothing parameter lambda and effective degrees of freedom Consistent with other fields package functions, the two main parameters in the model, tau^2 and sigma2 are parameterized as lambda = tau^2/sigma2 and sigma2. The MLEs for sigma2 and tau can be written in closed form as a function of lambda and these estimates can be substituted into the full likelihood to give a concentrated version that can numerically be maximized over lambda. The smoothing parameter lambda is best varied on a log scale and is sometimes difficult to interpret independent of the particular set of locations, sample size, and covariance. A more useful interpretation of lambda is through the effective degrees of freedom and an approximate value is found by default using a Monte Carlo technique. The effective degrees of freedom will vary with the dimension of the fixed regression part of the spatial model (typical 3 = constant + linear terms) and the total number of observations. It can be interpreted as the approximate number of "parameters" needed to represent the spatial prediction surface. For a fixed covariance model the spatial estimate at the observation locations can be represented as f hat = A(lambda) y where A is a matrix that does not depend on observations. The effective number of degrees of freedom is the trace of A(lambda) in analogy to the least squares regression "hat" matrix and has an inverse, monotonic relationship to lambda. The Monte Carlo method uses the fact that if e are iid N(0,1) E(t(e) A(lambda) e) = trace(A(lambda)).

Descriptions of specific functions and objects:

`LKrig`**:** Find spatial process estimate for fixed covariance specified by `nlevel`, `alpha`, `a.wght`, `NC`, and `lambda` or this information in an `LKinfo` list.

`predict.LKrig`, `predictSE.LKrig`**:** These functions evaluate the model at the the data locations or at `xnew` if it is included. Note the use of the `drop.Z` argument to either include the covariates or just restrict the computation to the spatial drift and the smooth component. If `drop.Z` is FALSE then then `Znew` needs to be included for predictions off of the observation locations. The standard errors are computed under the assumption that the covariance is known, that it is the TRUE covariance for the process, and both the process and measurement errors are multivariate normal. The formula that is used involves some recondite shortcuts for efficiency but has been checked against the standard errors found from an alternative formula in the fields Krig function. (See the script Lkrig.se.tests.R in the tests sub-directory for details.)

Value

LKrig: A LKrig class object with components for evaluating the estimate at arbitrary locations, describing the fit and as an option (with `Mc.return=TRUE`) the Cholesky decomposition to allow for fast updating with new values of lambda, alpha, and a.wght. The "symbolic" first step in the

sparse Cholesky decomposition can also be used to compute the sparse Cholesky decomposition for a different positive definite matrix that has the same pattern of zeroes. This option is useful in computing the likelihood under different covariance parameters. For the LKrig covariance the sparsity pattern will be the same if NC, level, overlap and the data locations x are kept the same. The returned component LKinfo has class LKinfo and is a list with the information that describes the layout of the multi-resolution basis functions and the covariance parameters for the GMRF. (See help(LKinfo) and also LK.basis as an example.)

predict.LKrig, predictSE.LKrig: A vector of predictions or standard errors.

predictSurface.LKrig: A list in image format (i.e. having components x,y,z) of the surface evaluated on a regular grid. This surface can then be plotted using several different R base package and fields functions e.g. image, image.plotcontour, persp, drape.plot. The surface method just calls this function and then a combination of the image and contour plotting functions.

Author(s)

Doug Nychka

See Also

LatticeKrig, LKrig.sim.conditional, LKrig.MLE, mKrig, Krig, fastTps, Wendland, LKrig.coef, Lkrig.lnPlike, LKrig.MRF.precision, LKrig.precision

Examples

```
# NOTE: CRAN requires that the "run" examples  execute within  5 seconds.
# so to meet this constraint many of these have been
# severely limited to be quick, typically making NC and nlevel
# very small.

# Load ozone data set
  data(ozone2)
  x<-ozone2$lon.lat
  y<- ozone2$y[16,]
# Find location that are not 'NA'.
# (LKrig is not set up to handle missing observations.)
  good <-  !is.na( y)
  x<- x[good,]
  y<- y[good]

# fairly arbitrary choices for covariance parameters and lambda
# just to show a basic level call
  obj1<- LKrig( x,y, a.wght=5, nlevel=3, nu=1.0, NC=10, lambda=.1)
  print(obj1)

# estimates of fixed model parameters with their SEs
  summary( obj1)$coefficients
#
# this is the same as:
#  LKinfoEX<- LKrigSetup( x, a.wght=5, nlevel=3, nu=1.0, NC=4)
#  obj1<- LKrig( x,y, LKinfo= LKinfoEX, lambda=.1)
```

```
# thin plate spline-like model with the lambda parameter estimated by
# maximum likelihood. Default choices are made for a.wght, nlevel, NC
# and alpha.
## Not run:
  obj<- LatticeKrig( x, y)
# summary of fit and a plot of fitted surface
  print( obj)
  surface( obj )
  US(add=TRUE)
  points(x)
# prediction standard errors
  out.se<- predictSE( obj, xnew= x)

## End(Not run)

# breaking down the LatticeKrig function into several steps.
# also use a different covariance model that has fewer basis functions
# (to make the example run more quickly)

## Not run:
  LKinfo<- LKrigSetup( x, nlevel=3, nu=1, NC=5, a.wght=5,
                       lambda=.01)
# maximize likelihood over lambda see help( LKrig.MLE) for details
# this assumes the value of 5 for a.wght.  In many cases the fit is not
# very sensitive to the range parameter such as a.wght in this case --
# but very sensitive to lambda when varied on a log scale.

  MLE.fit<- LKrig.MLE(x,y, LKinfo=LKinfo)
  MLE.fit$summary # summary of optimization over lambda.
# fit using MLE for lambda MLE function has added MLE value of lambda to
# the LKinfo object.
  obj<- LKrig( x,y, LKinfo=MLE.fit$LKinfo.MLE)
  print( obj)
# find prediction standard errors at locations based on fixing covariance
# at MLE's
  out.se<- predictSE( obj, xnew= x)
# one could evaluate the SE on a grid to get the surface of predicted SE's
# for large grids it is better to use LKrig.sim.conditional to estimate
#  the variances by Monte Carlo

## End(Not run)

#############################################################################
# Use multi-resolution model that approximates an exponential covariance
# Note that a.wght, related to a range/scale parameter
# is specified at a (seemingly) arbitrary value.
#############################################################################

  LKinfo<- LKrigSetup( x, NC=4, nu=1, nlevel=2, a.wght= 5)
# take a look at the implied covariance function solid=along x
#  and  dashed=along y
  check.cov<- LKrig.cov.plot( LKinfo)
  matplot( check.cov$d, check.cov$cov, type="l", lty=c(1,2))
```

```
##############################################################################
# Search over lambda to find MLE for ozone data with approximate exponential
# covariance
##############################################################################
## Not run:
  LKinfo.temp<-  LKrigSetup( x, NC=6, nu=1,  nlevel=3, a.wght= 5, lambda=.5)
# starting value for lambda optimization
  MLE.search<- LKrig.MLE(x,y, LKinfo=LKinfo.temp)
# this function returns an LKinfo object with the MLE for lambda included.
  MLE.ozone.fit<- LKrig( x,y,  LKinfo= MLE.search$LKinfo.MLE)

## End(Not run)
##############################################################################
# Including a covariate (linear fixed part in spatial model)
##############################################################################

  data(COmonthlyMet)
  y.CO<- CO.tmin.MAM.climate
  good<- !is.na( y.CO)
  y.CO<-y.CO[good]
  x.CO<- as.matrix(CO.loc[good,])
  Z.CO<- CO.elev[good]
# single level with large range parameter -- similar to a thin plate spline
#  lambda specified

# fit with elevation
# note how for convenience the LKrig parameters are
# just included here and not passed as a separate LKinfo object.
  obj.CO.elev<- LKrig( x.CO,y.CO,Z=Z.CO, nlevel=1, NC=50, alpha=1, lambda=.005,
                         a.wght=4.1)
# BTW the coefficient for the linear term for elevation  is obj.CO$d.coef[4]
# fitted surface without the elevation term
## Not run:
   LKinfo<- LKrigSetup( x.CO, nlevel=1, NC=20,alpha=1, a.wght=4.1, lambda=1.0)
# lambda given here is just the starting value for MLE optimization
  CO.MLE<- LKrig.MLE( x.CO,y.CO,Z=Z.CO, LKinfo=LKinfo)
  obj.CO.elev<- LKrig( x.CO,y.CO,Z=Z.CO, LKinfo= CO.MLE$LKinfo.MLE)
  CO.surface2<- predictSurface( obj.CO.elev, drop.Z=TRUE, nx=50, ny=50)
# pull off CO elevations at same locations on grid as the surface
  data( RMelevation)
# a superset of elevations at 4km resolution
  elev.surface<- interp.surface.grid( RMelevation, CO.surface2)
   CO.full<- predictSurface( obj.CO.elev, ZGrid= elev.surface, nx=50, ny=50)

# for comparison fit without elevation as a linear covariate:
  CO.MLE2<- LKrig.MLE( x.CO,y.CO, LKinfo=LKinfo)
  obj.CO<- LKrig( x.CO,y.CO, LKinfo= CO.MLE2$LKinfo.MLE)
# surface estimate
  CO.surface<- predictSurface( obj.CO, nx=50, ny=50)
  set.panel( 2,1)
  coltab<- topo.colors(256)
  zr<- range( c( CO.full$z), na.rm=TRUE)
```

```
  image.plot( CO.surface,  col=coltab, zlim =zr)
    US( add=TRUE,lwd=2)
    title( "MAM min temperatures without elevation")
  image.plot( CO.full, col=coltab, zlim=zr)
    title( "Including elevation")
    US( add=TRUE,lwd=2)

## End(Not run)
## Not run:
#### a 3-d example using alternative geometry
set.seed( 123)
N<- 500
x<-  matrix( runif(3* N,-1,1), ncol=3, nrow=N)
y<-   10*exp( -rdist( x, rbind( c(.5,.5,.6) ) )/.5)
LKinfo<- LKrigSetup( x,
                 LKGeometry = "LKBox",
                     nlevel = 1,
                     a.wght = 6.01,
                         NC = 5,
                  NC.buffer = 2,
                  normalize = TRUE,
              choleskyMemory = list(nnzR= 2e6),
                     lambda = .1,
                     mean.neighbor=200
                    )
# NOTE: memory for the spam routines has been increased to
# avoid warnings
  out1<- LKrig( x,y, LKinfo=LKinfo)

## End(Not run)

## Not run:
# MLE search over lambda and  a bigger problem
# but no normalization
N<- 5e4
x<-  matrix( runif(3* N,-1,1), ncol=3, nrow=N)
y<-   10*exp( -rdist( x, rbind( c(.5,.5,.6) ) )/.5)
LKinfo<- LKrigSetup( x,
                 LKGeometry = "LKBox",
                     nlevel = 1,
                     a.wght = 6.01,
                         NC = 20,
                  NC.buffer = 2,
                  normalize = FALSE,
              choleskyMemory = list(nnzR= 25e6),
                     lambda = .1,
                     mean.neighbor=200
                    )
# add in timing
 system.time(  out1<- LKrig( x,y, LKinfo=LKinfo) ) # about 25 seconds
# LKinfo$normalize<- TRUE
# system.time(  out2<- LatticeKrig( x,y, LKinfo=LKinfo) )# ~250 seconds
```

```
## End(Not run)
#############################################################################
# for a more elaborate search over  a.wght, alpha and lambda to find
# joint MLEs see help(LKrig.MLE)
#############################################################################

#############################################################################
# A bigger problem: 26K observations and 4.6K basis functions
# fitting takes about 15 seconds on a laptop for a fixed covariance
#  LKrig.MLE to find the MLE (not included) for lambda takes about
#  8 minutes
#############################################################################
## Not run:
  data(CO2)
  # the Ring geometry will be periodic in the first dimension and rectangular on
  # second. A useful approximation for spherical data omitting the polar caps.

  LKinfo.CO2<- LKrigSetup(CO2$lon.lat, NC=100,nlevel=1, lambda=.2,
                       a.wght=5, alpha=1,
                       LKGeometry="LKRing", choleskyMemory = list(nnzR=2e6) )
  print(LKinfo.CO2)
  obj1<- LKrig( CO2$lon.lat,CO2$y,LKinfo=LKinfo.CO2)
# 5700+ basis functions 101X57 lattice  (number of basis functions
# reduced in y direction because of a rectangular domain
  obj1$trA.est # about 2900+ effective degrees of freedom
#
  glist<- list( x= seq( -180,180,,200),y=seq( -80,80,,100) )
  fhat<- predictSurface( obj1,grid.list=glist)
#Plot data and gap-filled estimate
  set.panel(2,1)
  quilt.plot(CO2$lon.lat,CO2$y,zlim=c(373,381))
  title("Simulated CO2 satellite observations")
  world(add=TRUE,col="magenta")
  image.plot( fhat,zlim=c(373,381))
  world( add=TRUE, col="magenta")
  title("Gap-filled global predictions")

## End(Not run)
 set.panel()
#############################################################################
#  Comparing LKrig to ordinary Kriging
#############################################################################

# Here is an illustration of using the fields function mKrig with the
# LKrig covariance to reproduce the computations of LKrig. The
# difference is that mKrig can not take advantage of any sparsity in
# the precision matrix because its inverse, the covariance matrix, is
# not sparse.  This example reinforces the concept that LKrig finds the
# the standard geostatistical estimate but just uses a particular
# covariance function defined via basis functions and the precision
# matrix.
# Load ozone data set (AGAIN)
## Not run:
```

```
  data(ozone2)
  x<-ozone2$lon.lat
  y<- ozone2$y[16,]
# Find location that are not 'NA'.
# (LKrig is not set up to handle missing observations.)
  good <-  !is.na( y)
  x<- x[good,]
  y<- y[good]
  a.wght<- 5
  lambda <-  1.5
  obj1<- LKrig( x,y,NC=16,nlevel=1, alpha=1,  lambda=lambda, a.wght=5,
                NtrA=20,iseed=122)

# in both calls iseed is set the same so we can compare
# Monte Carlo estimates of effective degrees of freedom
  obj1$trA.est
# The covariance "parameters" are all in the list LKinfo
# to create this special list outside of a call to LKrig use
  LKinfo.test <- LKrigSetup( x, NC=16, nlevel=1, alpha=1.0,  a.wght=5)

# this call to mKrig should be identical to the LKrig results
# because it uses the LKrig.cov covariance with all the right parameters.
  obj2<- mKrig( x,y, lambda=lambda, m=2, cov.function="LKrig.cov",
                      cov.args=list( LKinfo=LKinfo.test), NtrA=20, iseed=122)
# compare the two results this is also an
# example of how tests are automated in fields
# set flag to have tests print results
  test.for.zero.flag<- TRUE
  test.for.zero( obj1$fitted.values, obj2$fitted.values,
                 tag="comparing predicted values LKrig and mKrig")
# compare standard errors.
  se1<- predictSE.LKrig( obj1)
  se2<- predictSE.mKrig(obj2)
  test.for.zero( se1, se2,
                 tag="comparing standard errors for LKrig and mKrig")

## End(Not run)

## Not run:
#########################################################################
#  a linear inverse problem
#########################################################################
# NOTE the settings in the model are small to make for a fast example
#
data(ozone2)
x<- ozone2$lon.lat
y<- ozone2$y[16,]
good<- !is.na(y)
x<- x[good,]
y<- y[good]

LKinfo<- LKrigSetup(x, a.wght=4.5, nlevel=3, nu=1, NC=4, lambda=.01)
```

```
# now observe a linear combination
  NNDist<- LKDist(x,x, delta=1.5)
  A<- NNDist
  A$ra<- exp(-NNDist$ra)
# A is a weight matrix based on neighbors close by and
# in spind sparse matrix format
# now convert to spam format
  A<- spind2spam(A)

TMatrix <- get(LKinfo$fixedFunction)(x = x)
# Tmatrix is a 3 column matrix of constant and the two spatial coordinates
#  i.e. a linear function of the spatial variables
PHI<- LKrig.basis(x, LKinfo)
X<-  A%*% PHI
U<-  A%*%TMatrix
yIndirect<- A%*%y
#
# X<-  A

out0<- LatticeKrig(x,y, LKinfo=LKinfo)
out1<- LatticeKrig(x,yIndirect, U=U, X=X, LKinfo=LKinfo)

# the predict function evaluates f in this case -- not the fitted values that
# are related to the
# observations
# partial agreement but not exact -- this is because the observational models
# assume different errors
#
plot( predict(out0,x), predict( out1,x))

out<- LKrig.sim.conditional( out1,M=5, nx=10, ny=10 )

## End(Not run)
```

`LKrig Internal` *Internal functions for LatticeKrig package.*

Description

Some internal functions for LKrig that estimate the coefficients of the basis functions and compute the likelihood.

Usage

```
LKrigMakewU(object, verbose = FALSE)
LKrigMakewX(object, verbose = FALSE)
LKrig.coef( GCholesky, wX, wU, wy, lambda,
collapseFixedEffect = FALSE,  verbose = FALSE)
LKrig.lnPlike( GCholesky, Q, quad.form, nObs, nReps, weights, LKinfo)
```

```
LKrig.traceA(GCholesky, wX, wU, lambda, weights, NtrA, iseed = NA)
LKrigUnrollZGrid( grid.list, ZGrid=NULL)
LKDefaultVarNames(A, tag)
```

Arguments

A	A matrix to add column names if not present.
collapseFixedEffect	If FALSE estimate fixed effects separately for each replicated data set.
grid.list	The grid for evaluating surface
GCholesky	SPAM cholesky decomposition of the "G" matrix.
iseed	Random seed used to generate the Monte Carlo samples. Keep the same to compare results with mKrig and also for multiple values of lambda.
lambda	The ratio of the nugget variance (tau squared) to the parameter controlling the marginal variance of the process (called sigma2 in fields).
LKinfo	The LKinfo object. See help(LKinfo)
NtrA	Number of Monte Carlo samples to estimate trace. Default is 20 in LKrig.
nObs	Number of observations.
nReps	Number of replicate fields.
object	The LKrig object.
Q	Precision matrix for coefficients.
quad.form	The part of the log likelihood that is a quadratic form. (This is typically found in LKrig.coef.)
tag	Text string to use as the column name. This will be followed 1, 2, ... for the names. E.g. X1 X2 X3. Default value will be the name of A.
verbose	If TRUE intermediate debugging information is printed.
weights	A vector that is proportional to the reciprocal variances of the errors. I.e. errors are assumed to be uncorrelated with variances tau^2/weights.
wU	Weighted U matrix the fixed part of the model.
wX	Weighted X matrix (in spam format) related to nonparametric (stochastic) part of model. Here weights refer to the sqrt(weights). NOTE: predicted values are U%*%d.coef + X%*%c.coef
wy	Weighted observations.
ZGrid	A list or array with the covariates on the same grid as that specified by the grid.list argument.

Details

The LatticeKrig article can be used as a reference for the matrix computations and the G matrix from those formulas figures prominently. The GCholesky object in these functions is the cholesky decomposition of this matrix. For compatibility with older version of this package this object may also be named as Mc (Cholesky of the M matrix) but the user should not identify this M with that in the article. Ideally all coding using Mc should be changed to GCholesky.

createLKrigObject Based on the arguments passed into LKrig forms the prototype LKrig object. This object is added to as one computes additional steps in the LKrig function. The Names argument in the call is awkward device to pass the names of the original x, Z and U objects when substitued these names are used as the default prefixs for column names of these matrices.

LKrigMakewU and LKrigMakewX construct the weighted U and X matrices from what is passed. In the case of observations that are point locations wU is found the weights and using the fixedFunction and wX is found from the weights and the multi-resolution basis functions. Note that X and wX are assumed to be in spam sparse matrix format.

LKrig.coef and LKrig.lnPlike are two low level functions to find the basis function coefficients and to evaluate the likelihood. The coefficients (c.mKrig) are also found because they provide for shortcut formulas for the standard errors and MLE estimates. These coefficients are identical to the basis coefficients (c.coef) found for usual Kriging in the mKrig function. LKrig.lnPlike also finds the profile MLE of tau and sigma2 given a fixed value for lambda (and alpha and a.wght). See the source for LKrig and also MLE.LKrig to see how these functions are used.

LKrig.traceA finds an estimate of the effective degrees of freedom of the smoothing matrix based a simple Monte Carlo scheme. The smoothing matrix A is the matrix for fixed covariance parameters so that y.hat = A y, where y.hat are the predicted values at the data locations. trace(A) is the effective degrees of freedom. If e are iid N(0,1) then the expected value of t(e)% * % A % * % e is equal to the trace of A. This is the basis for estimating the trace and the standard error for this estimate is based on NtrA independent samples.

dfind2d is a fast FORTRAN subroutine to find nearest neighbors within a fixed distance and is called by Wendland.basis. The function dfind3d is currently not used but is intended for future use to determine chordal distance between points on a sphere or cylinder.

LKrigDefaultFixedFunction Is called to construct the fixed part of the spatial model. The default is a polynomial of degree (m-1).

LKDefaultVarNames A handyfunction to create some simple column names to a matrix if they are missing. This is used so that the tables of parameter estimates will have labels.

Value

LKrig.coef a list with components d.coef the coefficients of the spatial drift and for covariates (Z) and c.coef the basis function coefficients. The logical vector ind.drift from the LKrig object indicates with components of d.coef are associated with the polynomial spatial drift and which are other fixed spatial covariates.

LKrig.lnPlike has the components:

lnProfileLike the log likelihood profiled for lambda, alpha and a.wght

sigma2.MLE the MLE of sigma2 given lambda, alpha and a.wght

shat.MLE the MLE of tau given lambda, alpha and a.wght

quad.form the quadratic form in the exponent of the multivariate normal likelihood

lnDetCov the log determinant of the covariance matrix in the likelihood

LKrigDefaultFixedFunction A matrix with dimension nrow(x) and columns of the number of polynomial terms and the number of columns of Z if given.

LKDefaultVarNames The original column names (e.g. the result of colnames(A)) or some simple choices filled in.

Author(s)

Doug Nychka

References

Nychka, D., Bandyopadhyay, S., Hammerling, D., Lindgren, F., & Sain, S. (2015). A multi-resolution Gaussian process model for the analysis of large spatial datasets.Journal of Computational and Graphical Statistics, 24(2), 579-599.

See Also

LKrig, LKrig.basis

LKrig Miscellaneous Matrix Functions

Miscellaneous internal functions for LatticeKrig package.

Description

Some utility functions used internally by higher level `LKrig` functions. Currently these are simple functions that perform shifts of a matrix and operations on indices for multidimensional arrays.

Usage

```
LKrig.shift.matrix( A, shift.row=0, shift.col=0, periodic=c(FALSE, FALSE))
LKrig.rowshift.periodic( A, shift.row)
LKrig.rowshift( A, shift.row, shift.col)
LKArrayShift(A, shift, periodic = FALSE)


expandMatrix0( A, B)
expandMatrix( ...)
expandMList( Mlist, byrow=TRUE)


convertIndexPeriodic(I, nGrid, nPad = NULL)
convertIndexArray(I, nGrid)
grid2Index(I, grid)
```

Arguments

`A`	A matrix.
`byrow`	If TRUE matrices will be repeated row by row. If FALSE this will be done column by column.
`B`	Another matrix.
`grid`	A vector giving the size of each dimension of array.

I	A matrix of multidimensional indices where each row identifies an element of the array. e.g. If I[1,] == c(3,4,2) this refers to A[3,4,2]
Mlist	A list where each component is a matrix.
nGrid	A array giving number of elements in each dimension.
nPad	An array with the number of padding indices in each dimension.
periodic	A vector of logicals columns.TRUE indicates an index where the shift will be periodic – entries shifted beyond the dimensions will be wrapped to the other side. e.g. for the matrix version c(FALSE, TRUE) indicates the columns of the matrix will be shifted in a periodic way but not the rows.
shift	A integer vector that gives the shifts for each dimension of the array.
shift.row	An integer that specifies the number of positions that the rows of the matrix are shifted.
shift.col	An integer that specifies the number of positions that the columns of the matrix are shifted.
...	Matrices to be expanded.

Details

Shift related: These functions are used to create the nearest neighbor indices for the precision matrices.

Expand related: These functions are useful for creating a sets of covariance parameters that follow a factorial pattern. For example repeating the rows of the "alpha" parameters as the "a.wght" parameters are varied. `expandMList` is particularly useful for creating a factorial design of parameters to pass to LKrig.MLE for searching the likelihood.

Index related: The function `convertIndexPeriodic` converts a single index for a multidimensional array, into an index that reflects wrapping into a smaller grid to reflect the padding. This is used in the `LKDistGrid` function to handle distances when the grid has periodic dimensions. The other two functions are used for checking and debugging of going back and forth between the multidimensional and single indexes.

Value

Shift: A matrix of shifted values. Entries that are not defined due to the shift are set to NA. A column shift is done by a combination of transpose operations and a row shift.

```
A<- matrix( 1:12,3,4)
A
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12

#shift of 2 for rows:
 LKrig.rowshift( A, 2)
    [,1] [,2] [,3] [,4]
[1,]   NA   NA   NA   NA
```

```
[2,]   NA   NA   NA   NA
[3,]    1    4    7   10

#periodic case
LKrig.rowshift.periodic( A, 2)
     [,1] [,2] [,3] [,4]
[1,]    2    5    8   11
[2,]    3    6    9   12
[3,]    1    4    7   10
```

Expand: `ExpandMList` Returns a list of matrices where the original matrices are repeated so that combinations of rows are represented. The example below illustrates. `byrow=FALSE` does the repetition by columns instead of rows.

```
> A
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> B
     [,1]
[1,]   11
[2,]   12
[3,]   13
> C
[1,] 100
> expandMList( list( A=A, B=B, C=C))
$A
     [,1] [,2]
[1,]    1    3
[2,]    2    4
[3,]    1    3
[4,]    2    4
[5,]    1    3
[6,]    2    4

$B
     [,1]
[1,]   11
[2,]   11
[3,]   12
[4,]   12
[5,]   13
[6,]   13

$C
     [,1]
[1,]  100
[2,]  100
[3,]  100
```

```
[4,]  100
[5,]  100
[6,]  100
```

Author(s)

Doug Nychka

Examples

```
A<- array( 1:90, c( 4,5,3))
LKArrayShift( A, c( -1,-1,0))

# welcome to the world of unrolling multiarray indices
A<- array( 1:60, c( 4,3,5))
I<- rbind( c(1,2,1), c( 3,2,5))
look<- grid2Index( I, c( 4,3,5) )
# A has been filled with the right unrolled index
print( look)
print(A[look])
```

LKrig.basis *High level functions for generating and simulating a basis function,Gaussian Process.*

Description

These functions support the LKrig function. Their main function is to create and evaluate radial basis functions of varying support on a nested set of regular grids. This series of grids forms a multi-resolution basis. The Gaussian process model is an expansion in these basis functions where the basis coefficients follow a Markov random field model for each resolution level. This family of functions generate the basis using sparse matrices, evaluate the covariance function of the process, and also simulate realizations of the process. LKrig.cov.plot is a useful function to get a quick plot of the covariance function implied by a LatticeKrig specification.

Usage

```
#
LKrig.cov(x1, x2 = NULL, LKinfo, C = NA, marginal = FALSE,
                  aRange = NA)
LKrig.cov.plot( LKinfo, NP=200, center = NULL, xlim = NULL, ylim = NULL)
LKrigCovWeightedObs(x1, wX, LKinfo)

LKrig.basis(x1, LKinfo, Level = NULL, raw = FALSE, verbose = FALSE)
LKrig.precision(LKinfo, return.B = FALSE, verbose=FALSE)
LKrig.quadraticform( Q, PHI, choleskyMemory = NULL)
LKrig.spind2spam(obj, add.zero.rows=TRUE)
LKrigMarginalVariance(x1, LKinfo, verbose = FALSE)
LKFindSigma2VarianceWeights(x1, LKinfo)
```

Arguments

add.zero.rows	If TRUE the conversion of the sparse matrix to spam format will have at least one element in each row. If there are no elements explicitly given in obj then an element with value zero is added. This technical detail is needed to accommodate the spam format for sparse matrices.
C	If passed the covariance matrix will be multiplied by this vector or matrix.
center	The point in the spatial domain that is used to evaluate the covariance function. The evaluation is done on x and y transects through the spatial domain intersecting at center and finding the covariance with respect to this point. If NULL defaults to the center of the spatial domain.
choleskyMemory	A list giving the memory requirements for a sparse cholesky decomposition. See chol.spam for details and also LKrig.
LKinfo	A list with components that give the information describing a multi-resolution basis with a Markov random field used for the covariance of the basis coefficients. This list is created in LKrig or by LKrigSetup and returned in the output object. (See section on returned Value below for this list's description.)
Level	If NULL the entire basis is evaluated. If an integer just that level is evaluated.
marginal	If TRUE returns the marginal variance. Currently not implemented!
NP	Number of points to evaluate the covariance function along each transect of the spatial domain.
obj	An object returned by LKrig or a sparse matrix in row/column format passed to LKrig.spind2spam.
PHI	A sparse matrix of basis functions (rows index points for evaluation, columns index basis functions).
raw	Do not normalize the basis functions even if it is indicated in the LKinfo object.
return.B	If TRUE B is returned instead of the precision matrix t(B)%*%B.
Q	A sparse (spam format) precision matrix.
aRange	Currently aRange is a place holder argument for future development where the correlation range can be specified.
wX	The wX matrix constructed in the LKrig function.
x1	A two column matrix of 2-dimension locations to evaluate basis functions or the first set of locations to evaluate the covariance function or the locations for the simulated process. Rows index the different locations: to be precise x1[i,1:2] are the "x" and "y" coordinates for the i th location.
x2	Second set of locations to evaluate covariance function.
xlim	Limits in x coordinate for evaluating the covariance model. Default is the spatial domain.
ylim	Limits in y coordinate for evaluating the covariance model.Default is the spatial domain.
verbose	If TRUE intermediate steps and other debugging information are printed.

Details

The basis functions are two-dimensional radial basis functions based on the compactly supported stationary covariance function (Wendland covariance) and centered on regular grid points with the scaling tied to the grid spacing.

For a basis at the coarsest level, the grid centers are generated by expanding the two sequences

```
seq(grid.info$xmin,grid.info$xmax,grid.info$delta)
seq(grid.info$ymin,grid.info$ymax,grid.info$delta)
```

into a regular grid of center points. The same spacing `delta` is used in both directions. The unnormalized basis functions are evaluated at locations `x1` by finding the pairwise, radial distances among `centers` and `x1`, scaling by `grid.info$delta * overlap` and then evaluating with the function name passed as `BasisFunction`. By default this is the 2-d Wendland covariance of order 2. Perhaps the most important point about the `LKrig.basis` is that it is designed to return a matrix of all basis functions as a sequence of points. There is no need to have a function that evaluates basis functions individually. In R code for a set of locations `x1` and a rectangular spatial domain with ranges `xmin, xmax, ymin ,ymax`:

```
centers<- expand.grid(seq(xmin,xmax,delta),
                      seq(ymin,ymax,delta) )
bigD<- rdist( x1, centers)/(delta*2.5)
PHI<- Wendland.function( bigD)
```

Note that there will be `nrow(centers)` basis functions generated where the precise number depends on the range of the domain and the choice of delta. The basis functions are indexed by the columns in `PHI` and this is a convention throughout this package. There will be `nrow(x1)` rows in `PHI` as each basis function will be evaluated at each 2-d location.

The basis functions are then normalized by scaling the basis functions at each location so that resulting marginal variance of the process is 1. This is done to coax the covariance model closer to a stationary representation. It is easiest to express this normalization by pseudo R code:

If `Q` is the precision matrix of the basis coefficients then in R/LatticeKrig code:

```
Omega<-  solve(Q)
process.variance <- diag(PHI%*% Omega %*%t(PHI) )
PHI.normalized <-  diag(1/sqrt(process.variance)) %*% PHI
```

where `Omega` is the unnormalized covariance matrix of the basis function coefficients.

Although correct, the code above is not an efficient algorithm to compute the unnormalized process variance. First the normalization can be done level by level rather than dealing with the entire multiresolution process at once. Also it is important to work with the precision matrix rather than the covariance. The function `LKrigNormalizeBasis` takes advantage of the sparsity of the precision matrix for the coefficients. `LKrigNormalizeBasisFast` is a more efficient version for an isotropic type model when a.wght is constant for a given level and takes advantage of the Kronecker structure on the precision matrix at each level. `LKrigNormalizeBasisFFTInterpolate` is the fastest method, which provides an accurate but approximate method for the normalization, which takes advantage of situations where the number of locations significantly exceeds the number of basis functions.

The precision matrix for the basis coefficients at each resolution has the form `t(B)%*% B`. These matrices for the individual levels are assembled by `LKrig.precision` as the block diagonals of a larger precision matrix for the entire vector of coefficients. Note these matrices are created in a sparse format. The specific entries in B, the object created by `LKrig.MRF.precision`, are a first order Markov random field: without edge adjustments the diagonal elements have the value `a.wght` and the first order neighbors have the value -1.

Below we give more details on how the weights are determined. Following the notation in Lindgren and Rue a.wght= 4 + k2 with k2 greater than or equal to 0. Some schematics for filling in the B matrix are given below (values are weights for the SAR on the lattice with a period indicating zero weights).

```
                                                  __________
   .   -1     .          |  -e      .            |  a.wght  -e
                         |                       |
  -1  a.wght  -1         | a.wght   -e           |   -e      .
                         |                       |
   .  -1      .          |  -e      .            |    .      .

Interior point         Left edge               Upper left corner
```

To adjust for edges and corner lattice points we take two strategies. The first is add extra lattice points around the edges so the actual spatial domain has lattice points with the complete number of neighbors. The default for the LKRectangle geometry is to add a buffer of 5 points (`NC.buffer = 5`) around the edges. In addition the neighbor weights are inflated sum to a fixed value. For example in the case of LKRectangle and following the schematic above we want the neighbor weights to sum to -4. Thus "e" for the middle figure will be set to -4/3 and for the right figure -2. Empirically these adjustments were found to improve how a.wght parameters chosen close to 4 could generate long range spatial correlations in the lattice coefficients. Note that these simple boundary adjustments happen to the buffer points and so are less likely to introduce artifacts into the spatial domain.

Value

LKrig.basis: A matrix with number of rows equal to the rows of `x1` and columns equal to the number of basis functions (LKinfo$m). Attached to the matrix is an `info` attribute that contains the list passed as `LKinfo`. Usually this value is in spam sparse matrix format.

LKrig.precision: For `return.B ==FALSE` a sparse, square matrix with dimensions of the number of basis functions. For `return.B == TRUE` the "B" SAR matrix is returned. This is useful for checking this function.

LKrig.cov: If `C=NA` a cross covariance matrix with dimensions `nrow(x1)` and `nrow(x2)` is used. If `C` is passed the result of multiplying the cross covariance matrix times `C` is used.

LKrigMarginalVariance: Gives the marginal variance of the LatticeKrig process at each level and at the locations in `x1`. Returned value is a matrix with `nlevel` columns indexing the levels and the number of rows equal to `nrow(x1)`. If `varLevels` is a row of this matrix then sum(varLevels* LKinfo$alpha) is the marginal variance for the full process when the different levels are weighted by alpha. This is weighted sum is the marginal variance returned by `LKrig.cov` and `marginal=TRUE` (Also assuming that `LKinfo$sigma2.object` is NULL, which it usually is.) .

LKFindSigma2VarianceWeights Return either the marginal variance of the process as a scalar or the variance at arbitrary locations x1. This second role is part of a nonstationary specification of the model where the process marginal variance can vary over space.

LKrig.sim: A matrix with dimensions of nrow(x1) by M. Each column are vectors of simulated values at the locations x1.

LKrig.cov.plot: Evaluates the covariance specified in the list LKinfo with respect to the point center along a transects in the x and y directions intersecting this point. Note the rectangular extent of the spatial domain is part of the grid information in LKinfo. Returns components u, d and cov. Each of these are two column matrices with the first column being results in the x direction and second column in the y direction. d are the distances of the points from the center and u are the actual x or y spatial coordinates. cov are the values of the covariance function. If normalize is TRUE these will in fact be the correlation functions. To plot the returned list use

```
out<- LKrig.cov.plot(LKinfo)
matplot( out$d, out$cov, type="l")
```

LKrig.quadraticform: Returns a vector that is diag(t(PHI)%*% solve(Q) %*% PHI)) closely related to the marginal variance of the process.

LKrigNormalizeBasis, **LKrigNormalizeBasisFast**, **LKrigNormalizeBasisFFTInterpolate**: A vector of variances corresponding to the unnormalized process at the locations.

LKrig.spind2spam: This converts a matrix in spind sparse format to spam sparse format. Although useful for checking, LatticeKrig now uses the the spam function directly to do the conversion. If obj is a list in spind format then the

```
obj2<- LKrig.spind2spam(obj)
```

and

```
obj2<- spam(obj[c("ind","ra")], nrow=obj$da[1], ncol=obj$da[2])
```

give the same result.

Author(s)

Doug Nychka

See Also

LKrig, mKrig, Krig, fastTps, Wendland, LKrigSAR, Radial.basis

Examples

```
# Load ozone data set
  data(ozone2)
  x<-ozone2$lon.lat
  y<- ozone2$y[16,]
# Find location that are not 'NA'.
# (LKrig is not set up to handle missing observations.)
  good <-  !is.na( y)
  x<- x[good,]
  y<- y[good]
```

```
  LKinfo<- LKrigSetup( x,NC=20,nlevel=1, alpha=1, lambda= .3 , a.wght=5)
# BTW lambda is close to MLE

# What does the  LatticeKrig covariance function look like?
# set up LKinfo object
# NC=10 sets the grid for the first level of basis functions
# NC^2 = 100 grid points in first level if square domain.
# given four levels the number of basis functions
# = 10^2 + 19^2 +37^2 + 73^2 = 5329
# effective range scales as roughly kappa where a.wght =  4 + kappa^2
# or exponential decreasing marginal variances for the components.
    NC<- 10
    nlevel <- 4
    a.wght <-  4 + 1/(.5)^2
    alpha<-  1/2^(0:(nlevel-1))
    LKinfo2<- LKrigSetup( cbind( c( -1,1), c(-1,1)), NC=NC,
                  nlevel=nlevel, a.wght=a.wght,alpha=alpha)
# evaluate covariance  along the  horizontal line through
# midpoint of region -- (0,0) in this case.
    look<- LKrig.cov.plot( LKinfo2)
# a plot of the covariance function in x and y with respect to (0,0)
    set.panel(2,1)
    plot(look$u[,1], look$cov[,1], type="l")
    title("X transect")
    plot(look$u[,2], look$cov[,2], type="l")
    title("Y transect")
    set.panel(1,1)
#
#
## Not run:
# full 2-d view of the covariance (this example follows the code
# in LKrig.cov.plot)
 x2<- cbind( 0,0)
 x1<- make.surface.grid( list(x=seq( -1,1,,40),  y=seq( -1,1,,40)))
 look<- LKrig.cov( x1,x2, LKinfo2)
 contour( as.surface( x1, look))
# Note nearly circular contours.
# of  course  plot(look[,80/2]) should look like plot above.
#

## End(Not run)

## Not run:
#Some correlation functions from different models
set.panel(2,1)
# a selection of ranges:
  hold<- matrix( NA, nrow=150, ncol=4)
  kappa<- seq( .25,1,,4)
  x2<- cbind( 0,0)
  x1<-  cbind( seq(-1,1,,150), rep( 0,150))
  for( k in 1:4){
    LKtemp<-  LKrigSetup( cbind( c( -1,1), c(-1,1)), NC=NC,
                  nlevel=nlevel,
```

```
                  a.wght= 4  + 1/(kappa[k]^2),
                  alpha=alpha)
    hold[,k]<-  LKrig.cov( x1,x2, LKinfo=LKtemp)
  }
  matplot( x1[,1], hold, type="l", lty=1, col=rainbow(5), pch=16 )
# a selection of smoothness parameters
  ktemp<- .5 # fix range
  alpha.power<- seq( 1,4,,4)
  LKtemp<- LKinfo2
  for( k in 1:4){
   LKtemp<-  LKrigSetup( cbind( c( -1,1), c(-1,1)), NC=NC,
                  nlevel=nlevel,
                  a.wght= 4  + 1/(ktemp^2),
                  alpha=alpha^alpha.power[k])
    hold[,k]<-  LKrig.cov( x1,x2, LKinfo=LKtemp)
  }
  matplot( x1[,1], hold, type="l", lty=1, col=rainbow(5) )
 set.panel()

## End(Not run)

## Not run:
# generating a basis on the domain [-1,1] by [-1,1] with 1 level
# Default number of buffer points are added to each side.
  LKinfo<- LKrigSetup(cbind( c(-1,1), c(-1,1)), NC=6,
                                nlevel=1, a.wght=4.5,alpha=1, NC.buffer=0 )
# evaluate the basis functions on a grid to look at them
  xg<- make.surface.grid( list(x=seq(-1,1,,50), y= seq(-1,1,,50)))
  PHI<- LKrig.basis( xg,LKinfo)
  dim(PHI) # should be  2500=50^2  by  36=6^2
# plot the 9th basis function  as.surface is a handy function to
# reformat the vector as an image object
# using the grid information in an attribute of the grid points
  set.panel(1,3)
  image.plot(as.surface(xg, PHI[,9]))
  points(  make.surface.grid( LKrigLatticeCenters(LKinfo, 1)) , col="grey", cex=.5)
  title("A radial basis function")
# compare to the tensor product basis type
  LKinfo2<- LKrigSetup(cbind( c(-1,1), c(-1,1)), NC=6,
                                nlevel=1, a.wght=4.5,alpha=1, NC.buffer=0,
                                BasisType="Tensor" )
  PHI2<- LKrig.basis( xg,LKinfo2)
  image.plot(as.surface(xg, PHI2[,9]))
  points(  make.surface.grid( LKrigLatticeCenters(LKinfo, 1)), col="grey", cex=.5)
  title("Tensor product basis function")

  image.plot(as.surface(xg, PHI[,9] - PHI2[,9]))
  points(  make.surface.grid( LKrigLatticeCenters(LKinfo, 1)), col="grey", cex=.5)
  title(" Radial - Tensor for 9th basis function")
set.panel()

## End(Not run)
#
```

```
# example of basis function indexing
#
## Not run:
# generating a basis on the domain [-1,1]X[-1,1] with 3 levels
# note that there are no buffering grid points.
  set.panel(3,2)
  LKinfo<-LKrigSetup(cbind( c(-1,1), c(-1,1)), NC=6,
                   a.wght=5, alpha=c(1,.5,.25), nlevel=3,
                   NC.buffer=0)
# evaluate the basis functions on a grid to look at them
  xtemp<- seq(-1,1,,40)
  xg<- make.surface.grid( list(x=xtemp, y= xtemp) )
  PHI<- LKrig.basis( xg,LKinfo)
# coerce to dense matrix format to make plotting easier.
  PHI<- spam2full(PHI)
# first tenth, and last basis function in each resolution level
# basis functions centers are added
 set.panel(3,3)
    for(  j in 1:3){
      id1<- LKinfo$latticeInfo$offset[j]+ 1
      id2<-  LKinfo$latticeInfo$offset[j]+ 10
      idlast<- LKinfo$latticeInfo$offset[j] +
                  LKinfo$latticeInfo$mx[j,1]*LKinfo$latticeInfo$mx[j,2]

      centers<-  make.surface.grid(LKrigLatticeCenters(LKinfo, j) )
      image.plot( as.surface(xg, PHI[,id1]))
      points( centers, cex=.2, col="grey")
      image.plot(as.surface(xg, PHI[,id2]))
      points( centers, cex=.2, col="grey")
      image.plot( as.surface(xg, PHI[,idlast]))
      points( centers, cex=.2, col="grey")
}
  set.panel()

## End(Not run)
## Not run:
# examining the stationarity of covariance model
  temp.fun<-
     function( NC.buffer=0, NC=4,  a.wght=4.01){
        LKinfo<- LKrigSetup(cbind( c(-1,1), c(-1,1)),nlevel=1, alpha=1,
                                 a.wght=a.wght, NC=NC,
                                 NC.buffer=NC.buffer,
                                  choleskyMemory=list(nnzR=2e6))
        cov1y<- cov1x<- cov0x<- cov0y<-  matrix( NA, nrow=200, ncol=20)
        cov1dx<- cov1dy<- cov0dx<- cov0dy<- matrix( NA, nrow=200, ncol=20)
        cgrid<- seq( 0,1,,20)
        for( k in 1:20){
            hold<- LKrig.cov.plot( LKinfo,
                           center=rbind( c(cgrid[k], cgrid[k])), NP=200)
            cov1x[,k] <- hold$cov[,1]
            cov1y[,k] <- hold$cov[,2]
            cov1dx[,k] <- hold$d[,1]
            cov1dy[,k] <- hold$d[,2]
```

```
            hold<- LKrig.cov.plot( LKinfo,
                           center=rbind( c(cgrid[k],0) ), NP=200)
            cov0x[,k] <- hold$cov[,1]
            cov0y[,k] <- hold$cov[,2]
            cov0dx[,k] <- hold$d[,1]
            cov0dy[,k] <- hold$d[,2]
                }
          matplot( cov1dx, cov1x, type="l", col= rainbow(20),
                         xlab="", ylab="correlation")
          mtext( side=1, line=-1, text="diagonal X")
          title( paste(  " buffer=",NC.buffer), cex=.5)
          matplot( cov1dy, cov1y, type="l", col= rainbow(20),
                        xlab="", ylab="")
          mtext( side=1, line=-1, text="diagonal Y")
          matplot(cov0dx, cov0x, type="l", col= rainbow(20),
                        xlab="",       ylab="")
          mtext( side=1, line=-1, text="middle X")
          matplot( cov0dy, cov0y, type="l", col= rainbow(20),
                         xlab="",   ylab="")
          mtext( side=1, line=-1, text="middle Y")
          title( paste( NC, a.wght), cex=.5)
}


 set.panel(3,4)
par(mar=c(3,4,1,0), oma=c(1,1,1,1))
temp.fun(  NC.buffer=5, NC=4, a.wght=4.05)
temp.fun(  NC.buffer=5, NC=16, a.wght=4.05)
temp.fun(  NC.buffer=5, NC=64, a.wght=4.05)

set.panel(4,4)
par(mar=c(3,4,1,0), oma=c(1,1,1,1))
temp.fun( NC.buffer=0, NC=8)
temp.fun( NC.buffer=2, NC=8)
temp.fun( NC.buffer=4, NC=8)
# this next one takes a while
temp.fun( NC.buffer=8,  NC=8)
# stationary == curves should all line up!


## End(Not run)
```

`LKrig.MLE` *Functions to search over covariance parameters in the LatticeKrig model.*

Description

Given a list of different covariance parameters for the Lattice Krig covariance model this function computes the likelihood or a profiled version (over lambda) and approximates a generalized cross-

validation function at each of the parameter settings. This is an experimental function that has been productively used with a Latin hypercube design package to efficiently search through the LatticeKrig covariance parameter space.

Usage

```
LKrigFindLambda(x, y, Z = NULL, U = NULL, X = NULL, ..., LKinfo,
                 use.cholesky = NULL, lambda.profile = TRUE,
                 lowerBoundLogLambda = -16, tol = 0.005, verbose =
                 FALSE)

LKrigFindLambdaAwght(x, y, ..., LKinfo, use.cholesky = NULL,
                 lowerBoundLogLambda = -16, upperBoundLogLambda = 4,
                 lowerBoundOmega = -3, upperBoundOmega = 0.75, factr =
                 1e+07, pgtol=1e-1, maxit = 15, verbose = FALSE)


LambdaAwghtObjectiveFunction(PARS, LKrigArgs, capture.env, verbose=FALSE )

LKrig.MLE( x,y,..., LKinfo, use.cholesky = NULL,
                    par.grid=NULL,
                    lambda.profile=TRUE,
                    verbose=FALSE,
                    lowerBoundLogLambda = -16,
                    nTasks = 1, taskID = 1,
                    tol = 0.005)
LKrig.make.par.grid(par.grid=NULL, LKinfo = NULL)
omega2Awght (omega, LKinfo)
Awght2Omega (Awght, LKinfo)
```

Arguments

`Awght`	Value of Awght parameter to convert to omega form
`capture.env`	The environment to save to the likelihood evaluation to the object `capture.evaluations`.
`lambda.profile`	A logical value controlling whether the likelihood is maximized over lambda. For LKrigFindLambda if TRUE the likelihood is maximized over lambda at the covariance values in LKinfo and if FALSE the likelihood is just evaluated at LKinfo including the lambda value in this list. For LKrig.MLE if TRUE for each set of parameters in par.grid the value of lambda is found that maximizes the likelihood. In this case the llambda value is the starting value for the optimizer. If llammbda[k] is NA then the lambda value found from the k-1 maximization is used as a starting value for the k step. (In the source code this is `llambda.opt`.) Of course this only makes sense if the other parameters are ordered so that the results for k-1 make sense as a lambda starting value for k. If FALSE the likelihood is evaluated for the covariance parameters at the kth positions in the par.grid list including lambda.
`LKinfo`	An LKinfo object that specifies the LatticeKrig covariance. Usually this is obtained by a call to `LKrigSetup` or as the component LKinfo from the LKrig

object. The search sequentially replaces the alpha and a.wght arguments in this list by the values in par.grid but leaves everything else the same. If par.grid is not passed the parameter values in LKinfo are used to evaluate the likelihood. This option is most useful if one has fixed values of alpha and a.wght and the goal is to maximize the likelihood over lambda.

`LKrigArgs` Argument list to call LKrig.

`lowerBoundLogLambda`
Lower limit for lambda in searching for MLE.

`lowerBoundOmega`
Lower limit for omega in searching for MLE.

`maxit` Maximum number of iterations (passed to optim function)

`nTasks` If using Rmpi the number of slaves available.

`omega` Value of the omega parameter to convert to the a.wght format.

`PARS` PAR[1]= log(lambda) and PAR[2]= .5*log(a.wght-4) (also referred to as `omega`). For the LKRectangle geometry.

`par.grid` A list with components llambda, alpha, a.wght giving the different sets of parameters to evaluate. If M is the number of parameter setting to evaluate llambda is a vector length M and alpha and a.wght are matrices with M rows and nlevel columns. Thus, the kth trial has parameters

```
par.grid$llambda[k]
par.grid$alpha[k,]
par.grid$a.wght[k,]
```

Currently this function does not support passing a non-stationary spatial parameterization for alpha. The LKinfo object details the other parts of the covariance specification (e.g. number of levels, grid sizes) that do not change. Note that par.grid assumes *ln* lambda not lambda. See details below for some other features of the par.grid arguments.

`factr` Controls convergence for the BFGS-L method. (passed to optim).

`pgtol` Mysterious tolerance for gradient convergence in L-BFGS. This seems to influence the number of iterations the most.

`tol` Tolerance on log likelihood use to determine convergence.

`taskID` If using Rmpi the slave id.

`verbose` If TRUE prints out intermediate results.

`upperBoundOmega`
Upper limit for omega in searching for MLE.

`upperBoundLogLambda`
Upper limit for log lambda in searching for MLE.

`use.cholesky` If not NULL then this object is used as the symbolic cholesky decomposition of the covariance matrix for computing the likelihood.

`U` U matrix if an inverse problem model.

`x` The spatial locations.

`X` X matrix if an inverse problem model.

y	The observations.
Z	Matrix of additional covariates for fixe part of the model.
...	Any arguments to be passed to LKrig. E.g. x, y, Z a covariate or weights.

Details

LKrigFindLambda: Uses a simple one dimensional optimizer optimize. To maximize the log likelihood for log lambda over the range: llambda.start + [-8,5]. This function is used to determine lambda in LatticeKrig.

LKrigFindLambdaAwght: Uses a simple optimizer optim. To maximize the log likelihood for lambda and a.wght over the range.

LKrig.MLE: This is a simple wrapper function to accomplish repeated calls to the LKrig function to evaluate the profile likelihood and/or to optimize the likelihood over the lambda parameters. The main point is that maximization over the lambda parameter (or equivalently for tau and sigma2) is the most important and should be done before considering variation of other parameters. If lambda is specified then one has closed form expressions for tau, sigma2 that can then be substituted back into the log full likelihood. This operation that is the default throughout LatticeKrig (and fields) can concentrate the likelihood on a reduced set of parameters. The further refinement when lambda.profile==TRUE is to maximize the concentrated likelihood over lambda and report this result. This will be a profile likelihood over the remaining parameters of the covariance.

The covariance/model parameters are alpha, a.wght, and log lambda and are separate matrix or vector components of the par.grid list. The cleanest version of this function would just require the par.grid list, however, to be easier to use there are several options to give partial information and let the function itself create the master parameter list. For example, just a search over lambda should be easy and not require creating par.grid outside the function. To follow this option one can just give an LKinfo object. The value for the lambda component in this object will be the starting value with the default starting value being lambda =1.0.

In the second example below most of the coding is getting the grid of parameters to search in the right form. It is useful to normalize the alpha parameters to sum to one so that the marginal variance of the process is only parameterized by sigma2. To make this easy to implement there is the option to specify the alpha parameters in the form of a mixture model so that the components are positive and add to one (the gamma variable below). If a component gamma is passed as a component of par.grid then this is assumed to be in the mixture model form and the alpha weights are computed from this. Note that gamma will be a matrix with (nlevel - 1) columns while alpha has nlevel columns.

For those readers that use which.max these functions are natural extensions and are handy for looking at interpolated surfaces of the likelihood function.

which.max.matrix: Finds the maximum value in a matrix and returns the row/column index.

which.max.image Finds the maximum value in an image matrix and returns the index and the corresponding grid values.

LKrig.make.par.grid: This is usually used as an internal function that converts the list of parameters in par.grid and the LKinfo object into an more complex data structure used by LKrig.MLE. Its returned value is a "list of lists" to make the search over different parameters combinations simple.

omega2Awght, Awght2omega Converts between the Awght parameter (the diagonal elements of the SAR matrix) and the omega parameter that provides an unconstrained range for optimization. The link is

```
Awght <- LKinfo$floorAwght + exp(omega) * (xDimension)
```

.

Note that LKinfo supplied the lower bound on the Awght because this is geometry/problem specific. For the 2-d rectangle this is 4.

Value

LKrigFindLambda

summary	Giving information on the optimization over lambda.
LKinfo	Covariance information object.
llambda.start, lambda.MLE	Initial and final values for lambda.
lnLike.eval	Matrix with all values of log likelihood that were evaluated
call	Calling arguments.
Mc	Cholesky decomposition.

LKrig.MLE

summary	A matrix with columns: effective degrees of freedom, ln Profile likelihood, Generalized cross-validation function, MLE tau, MLE sigma2, full likelihood and number of parameter evaluations. The rows correspond to the different parameters in the rows of the par.grid components.
par.grid	List of parameters used in search. Some parameters might be filled in from the initial par.grid list passed and also from LKinfo.
LKinfo	LKinfo list that was either passed or created.
index.MLE	Index for case that has largest Likelihood value.
index.GCV	Index for case that has largest GCV value.
LKinfo.MLE	LKinfo list at the parameters with largest profile likelihood.
lambda.MLE	Value of lambda from grid with largest profile likelihood.
call	Calling sequence for this function.

which.max.matrix Returns a 2 column matrix with row and column index of maximum.

which.max.image For an object in image format returns components x,y,z giving the location and the maximum value for the image. Also included is the component ind that is the row and column indices for the maximum in the image matrix.

LKrig.make.par.grid Returns a list with components, alpha, a.wght. Each component is a list where each component of the list is a separate set of parameters. This more general format is useful for the non-stationary case when the parameters alpha might be a list of nlevel matrices.

Author(s)

Douglas Nychka

See Also

LKrig LatticeKrig

Examples

```
#
# fitting summer precip for  sub region of North America (Florida)
# (tiny subregion is just to make this run under 5 seconds).
# total precip in 1/10 mm for JJA
  data(NorthAmericanRainfall)
# rename for less typing
  x<- cbind( NorthAmericanRainfall$longitude, NorthAmericanRainfall$latitude)
  y<- log10(NorthAmericanRainfall$precip)
# cut down the size of this data set so examples run quickly
  ind<- x[,1] > -90 & x[,2] < 35 #
  x<- x[ind,]
  y<- y[ind]

# This is a single level smoother

  LKinfo<- LKrigSetup(x,NC=4, nlevel=1, a.wght=5, alpha=1.0)
  lambdaFit<- LKrigFindLambda( x,y,LKinfo=LKinfo)
  lambdaFit$summary

## Not run:
# grid search over parameters
  NG<-15
  par.grid<- list( a.wght= rep( 4.05,NG),alpha= rep(1, NG),
                      llambda=  seq(-8,-2,,NG))
  lambda.search.results<-LKrig.MLE( x,y,LKinfo=LKinfo,
                                    par.grid=par.grid,
                                    lambda.profile=FALSE)
  lambda.search.results$summary
# profile likelihood
  plot( lambda.search.results$summary[,1:2],
         xlab="effective degrees freedom",
         ylab="ln profile likelihood")
# fit at largest likelihood value:
  lambda.MLE.fit<- LKrig( x,y,
                   LKinfo=lambda.search.results$LKinfo.MLE)

## End(Not run)

## Not run:
# optimizing  Profile likelihood over lambda using optim
# consider 3 values for a.wght (range parameter)
# in this case the log lambdas passed are the starting values for optim.
  NG<-3
  par.grid<- list( a.wght= c( 4.05,4.1,5) ,alpha= rep(1, NG),
                      llambda= c(-5,NA,NA))
# NOTE: NAs in llambda mean use the previous MLE for llambda as the
# current starting value.
```

```
  LKinfo<- LKrigSetup(x,NC=12,nlevel=1, a.wght=5, alpha=1.0)
  lambda.search.results<-LKrig.MLE(
                             x,y,LKinfo=LKinfo, par.grid=par.grid,
                             lambda.profile=TRUE)
  print(lambda.search.results$summary)
# note first result a.wght = 4.05 is the optimized result for the grid
# search given above.

## End(Not run)
#############################################################################
# search over two multi-resolution levels varying the  levels of alpha's
#############################################################################
## Not run:
# NOTE: search ranges found largely by trial and error to make this
# example work also the grid is quite coarse ( and NC is small) to
# be quick as a help file example
  data(NorthAmericanRainfall)
# rename for less typing
  x<- cbind( NorthAmericanRainfall$longitude, NorthAmericanRainfall$latitude)
# total precip in 1/10 mm for JJA
 y<- log10(NorthAmericanRainfall$precip)
# cut down the size of this data set so examples run quickly
# examples also work with  the full data set. Also try NC= 100 for a
# nontrivial model.
  ind<- x[,1] > -90 & x[,2] < 35 #
  x<- x[ind,]
  y<- y[ind]

  Ndes<- 10
# NOTE: this is set to be very small just to make this
#       example run fast
  set.seed(124)
  par.grid<- list()
# create grid of alphas to sum to 1 use a mixture model parameterization
#  alpha1 = (1/(1 + exp(gamma1)) ,
# alpha2 = exp( gamma1) / ( 1 + exp( gamma1))
#
  par.grid$gamma<- cbind(runif( Ndes, -3,2), runif( Ndes, -3,2))
  par.grid$a.wght<- rep( 4.5, Ndes)
# log lambda grid search values
  par.grid$llambda<- runif( Ndes,-5,-3)
  LKinfo1<- LKrigSetup( x, NC=5, nlevel=3, a.wght=5, alpha=c(1.0,.5,.25))
# NOTE: a.wght in call is not used. Also a better search is to profile over
#  llambda

 alpha.search.results<- LKrig.MLE( x,y,LKinfo=LKinfo1, par.grid=par.grid,
                                   lambda.profile=FALSE)

#############################################################################
# Viewing the search results
#############################################################################

# this scatterplot is good for a quick look because  effective degrees
```

```
# of freedom is a useful summary of fit.
  plot( alpha.search.results$summary[,1:2],
         xlab="effective degrees freedom",
         ylab="ln profile likelihood")
#

## End(Not run)

## Not run:
# a two level model search
# with profiling over lambda.
data(NorthAmericanRainfall)
# rename for less typing
  x<- cbind( NorthAmericanRainfall$longitude,
             NorthAmericanRainfall$latitude)
# mean total precip in 1/10 mm for JJA
  y<- log10(NorthAmericanRainfall$precip)

#  This takes a few minutes
  Ndes<- 40
  nlevel<-2
  par.grid<- list()
## create grid of alphas to sum to 1 use a mixture model parameterization:
#    alpha1 = (1/(1 + exp(gamma1)) ,
#   alpha2 = exp( gamma1) / ( 1 + exp( gamma1))
  set.seed(123)
  par.grid$gamma<- runif( Ndes,-3,4)
## values for range (a.wght)
  par.grid$a.wght<- 4 + 1/exp(seq( 0,4,,Ndes))
# log lambda grid search values (these are the starting values)
  par.grid$llambda<- rep(-4, Ndes)

  LKinfo1<- LKrigSetup( x, NC=15, nlevel=nlevel,
                          a.wght=5, alpha=rep( NA,2) )
##
## the search over the parameter list in par.grid  maximizing over lambda
  search.results<- LKrig.MLE( x,y,LKinfo=LKinfo1, par.grid=par.grid,
                                lambda.profile=TRUE)
# plotting results of likelihood search
set.panel(1,2)
 plot( search.results$summary[,1:2],
         xlab="effective degrees freedom",
         ylab="ln profile likelihood")
 xtemp<- matrix(NA, ncol=2, nrow=Ndes)
 for( k in 1:Ndes){
   xtemp[k,] <- c( (search.results$par.grid$alpha[[k]])[1],
                  (search.results$par.grid$a.wght[[k]])[1] )
}
 quilt.plot( xtemp,search.results$summary[,2])
#  fit using Tps
 tps.out<- Tps(  xtemp,search.results$summary[,2], lambda=0)
 contour( predictSurface(tps.out), lwd=3,add=TRUE)
 set.panel()
```

```
## End(Not run)
## Not run:
# searching over nu
data(ozone2)
x<- ozone2$lon.lat
y<- ozone2$y[16,]
good<- !is.na(y)
y<- y[good]
x<- x[good,]
par.grid<- expand.grid( nu = c(.5,1.0, 1.5), a.wght= list(4.1,4.5,5) )
par.grid$llambda<- rep( NA, length(par.grid$nu))
LKinfo<- LKrigSetup(x,  nlevel=3, nu=.5, NC=5, a.wght=4.5)
out<- LKrig.MLE( x,y, LKinfo=LKinfo, par.grid=par.grid)
# take a look
cbind( par.grid, out$summary[,1:5])

## End(Not run)
## Not run:
# an MLE fit taking advantage of replicated fields
# check based on simulated data

N<- 200
M<-50 # number of independent replicated fields
tau<- sqrt(.01)
set.seed(123)
x<- matrix( runif(N*2), N,2)

LKinfo<- LKrigSetup( x, NC=16, nlevel=1,
                 a.wght=4.5, lambda=.01,
                fixed.Function=NULL,
                normalize=TRUE)

# the replicate fields
truef<-  LKrig.sim(x,LKinfo=LKinfo, M=M )
set.seed(222)
error<- tau*matrix( rnorm(N*M), N,M)
y<- truef + error
# with correct lambda
obj<- LKrig( x,y, LKinfo=LKinfo, lambda=.01, )
print( obj$tau.MLE.FULL)
print( obj$sigma2.MLE.FULL)

fitMLE1<- LKrigFindLambda( x,y, LKinfo=LKinfo)
fitMLE1$summary
aWghtGrid<-  c( 4.01, 4.05, 4.1, 4.2, 4.5, 4.6, 4.7, 5, 8, 16)
par.grid<- list( a.wght = aWghtGrid)

fitMLE2<- LKrig.MLE( x,y, LKinfo=LKinfo,
                      par.grid= par.grid )
fitMLE2$summary

LKinfo1<- LKinfoUpdate( LKinfo, lambda=.1, a.wght= 4.2)
```

```
fitMLE4<- LKrigFindLambdaAwght( x,y, LKinfo=LKinfo1)
fitMLE4$summary

plot(  log( aWghtGrid -4)/2, fitMLE2$summary[,2], type="b",
  xlab="log( a.wght - 4)/2",
  ylab= "log Profile likelihood" )


points( log(fitMLE4$summary["a.wght.MLE"] -4)/2,
     fitMLE4$summary["lnProfLike"], pch="+", col="red"  )
xline( log(fitMLE4$summary["a.wght.MLE"] -4)/2, col="red", lty=2)

## End(Not run)
```

LKrig.sim *Functions for simulating a multi resolution process following the Lattice Krig covariance model.*

Description

The fields are Gaussian and can be either simulated unconditionally or conditional on the field values and a set of irregular locations.

Usage

```
#
LKrig.sim(x1, LKinfo, M=1,just.coefficients = FALSE)
LKrig.sim.conditional( LKrigObj, M=1, x.grid= NULL, grid.list=NULL,
                           nx=80, ny=80,...,Z.grid=NULL, seed=42, verbose=FALSE)

simConditionalDraw(index = 1, LKrigObj, ghat, x.grid, Z.grid, PHIGrid,
                 seeds = 123, verbose = FALSE)
```

Arguments

grid.list	Specifies a grid of spatial locations using the grid.list format (help(grid.list)). These are the locations used to evaluate the fields generated from conditional simulation. The default is to generate an 80X80 grid based on range of the observations.
just.coefficients	If TRUE just simulates the coefficients from the Markov Random field.
LKinfo	A list with components that give the information describing a multi-resolution basis with a Markov random field used for the covariance of the basis coefficients. This list is created in LKrig and is returned as part of the output object or in a more hands on manner directly using LKrigSetup (See section on returned value below for this list's description.)

M	Number of independent simulated fields.
nx	Number of grid points in x coordinate for output grid.
ny	Number of grid points in y coordinate for output grid.
LKrigObj	An LKrig object, i.e. the output list returned by LKrig.
seed	Seed to set random number generator.
x1	A two column matrix of 2-dimension locations to evaluate basis functions or the first set of locations to evaluate the covariance function or the locations for the simulated process. Rows index the different locations: to be precise x1[i,1:2] are the "x" and "y" coordinates for the i th location.
x.grid	Locations to evaluate conditional fields. This is in the form of a two column matrix where each row is a spatial location.
Z.grid	The covariates that are associated with the x.grid values. This is useful for conditional simulation where the fields are evaluated at x.grid locations and using covariate values Z.grid. Z.grid is matrix with columns indexing the different covariates and rows indexed by the x.grid locations.
...	Arguments to be passed to the LKrig function to specify the spatial estimate. These are components in addition to what is in the LKinfo list returned by LKrig.
verbose	If TRUE prints out debugging information.
ghat	The predicted surface at the grid.
index	The index for the random seed to use in the vector seeds.
PHIGrid	Basis function matrix at grid points.
seeds	A vector of random seeds.

Details

The simulation of the unconditional random field is done by generating a draw from the multi-resolution coefficients using a Cholesky decomposition and then multiplying by the basis functions to evaluate the field at arbitrary points. Currently, there is no provision to exploit the case when one wants to simulate the field on a regular grid. The conditional distribution is a draw from the multivariate normal for the random fields conditioned on the observations and also conditioned on covariance model and covariance parameters. If the nugget/measurement error variance is zero then any draw from the conditional distribution will be equal to the observations at the observation locations. In the past conditional simulation was known to be notoriously compute intensive, but the major numerical problems are finessed here by exploiting sparsity of the coefficient precision matrix.

The conditional field is found using a simple trick based on the linear statistics for the multivariate normal. One generates an unconditional field that includes the field values at the observations. From this realization one forms a synthetic data set and uses LKrig to predict the remaining field based on the synthetic observations. The difference between the predicted field and the realization (i.e. the true field) is a draw from the conditional distribution with the right covariance matrix. Adding the conditional mean to this result one obtains a draw from the full conditional distribution. This algorithm can also be interpreted as a variant on the bootstrap to determine the estimator uncertainty. The fixed part of the model is also handled correctly in this algorithm. See the commented source for LKrig.sim.conditional for the details of this algorithm.

simConditionalDraw is low level function that is called to generate each ensemble member i.e. each draw from the conditional distribution. The large number of arguments is to avoid recomputing many common elements during the loop in generating these draws. In particular passing the basis function matrices avoid having to recompute the normalization at each step, often an intensive computation for a large grid.

Value

LKrig.sim: A matrix with dimensions of `nrow(x1)` by `M` of simulated values at the locations `x1`.

LKrig.sim.conditional: A list with the components.

xgrid The locations where the simulated field(s) are evaluated.

ghat The conditional mean at the xgrid locations.

g.draw A matrix with dimensions of `nrow(x.grid)` by `M` with each column being an independent draw from the conditional distribution.

Author(s)

Doug Nychka

See Also

LKrig, mKrig, Krig, fastTps, Wendland

Examples

```
# Load ozone data set
  data(ozone2)
  x<-ozone2$lon.lat
  y<- ozone2$y[16,]
# Find location that are not 'NA'.
# (LKrig is not set up to handle missing observations.)
  good <-  !is.na( y)
  x<- x[good,]
  y<- y[good]
  LKinfo<- LKrigSetup( x,NC=20,nlevel=1, alpha=1, lambda= .3 , a.wght=5)
# BTW lambda is close to MLE
# Simulating this  LKrig process
# simulate 4 realizations of process and plot them
# (these have unit marginal variance)
  xg<- make.surface.grid(list( x=seq( -87,-83,,40), y=seq(36.5, 44.5,,40)))
  out<- LKrig.sim(xg, LKinfo,M=4)
## Not run:
  set.panel(2,2)
  for( k in 1:4){
    image.plot( as.surface( xg, out[,k]), axes=FALSE) }

## End(Not run)
  obj<- LKrig(x,y,LKinfo=LKinfo)
  O3.cond.sim<- LKrig.sim.conditional( obj, M=3,nx=40,ny=40)
## Not run:
```

```
  set.panel( 2,2)
  zr<- range( c(  O3.cond.sim$draw,  O3.cond.sim$ghat), na.rm=TRUE)
  coltab<- tim.colors()
  image.plot( as.surface( O3.cond.sim$x.grid, O3.cond.sim$ghat), zlim=zr)
  title("Conditional mean")
  US( add=TRUE)
  for( k in 1:3){
    image( as.surface( O3.cond.sim$x.grid, O3.cond.sim$g.draw[,k]),
              zlim=zr, col=coltab)
    points( obj$x, cex=.5)
    US( add=TRUE)
  }
  set.panel()

## End(Not run)
```

LKrigDefaultFixedFunction

Creates fixed part of spatial model.

Description

Creates matrix of low order polynomial in the spatial coordinates and adds any other spatial covariates that are part of the linear model.

Usage

```
 LKrigDefaultFixedFunction(x, Z = NULL, m=2,distance.type="Euclidean")
 LKrigPeriodicFixedFunction(x, Z = NULL, m = 2, distance.type = "Euclidean")

predictLKrigFixedFunction(object, xnew, Znew = NULL, drop.Z = FALSE,
collapseFixedEffect = FALSE)
```

Arguments

collapseFixedEffect	If FALSE the fixed part of the model is found separately for each replicated data set. If TRUE the estimate is polled across replicates.
drop.Z	If TRUE only spatial drift is evaluated the contribution for covariates is omitted.
distance.type	The distance metric. See the entry in LKrig for details.
m	The order of the polynomial. Following the convention for splines the polynomial will have maximum order (m-1). Throughout LKrig m==2 is the default giving a linear polynomial.
object	A LKrig object.
x	A 2 column matrix of 2-d locations to evaluate the polynomial.
xnew	Locations for predictions.

Z	A matrix specifying additional spatial covariates.
Znew	Same as Z.

Details

LKrigDefaultFixedFunction This function creates the regression matrix for the fixed part of the spatial model. The default is a low order polynomial regression matrix of degree m-1. To this matrix are bound as columns any covariates passed as Z. Typically one would not need to modify this function. For more exotic fixed part models one can specify create and then specify a different function. See LKrig.setup and LKrig. NOTE: If the argument for this function is passed as NULL then the subsequent computations do not include a fixed part in the model.

LKrigDefaultFixedFunction This is same as LKrigDefaultFixedFunction except the first coordinate is ignored. i.e. it is assumed to be periodic so adding a polynomial does not make sense.

predictLKrigFixedFunction This function is simple, but is introduced to make the code modular and to handle the case for cylindrical geometry where only latitude should have a spatial term (to preserve periodicity in longitude).

Value

A matrix where rows index the locations and columns are the different spatial polynomial and covariates.

Author(s)

Doug Nychka

See Also

LKrig.basis, LKrig

Examples

```
x<- matrix( runif(100), nrow=50)
# linear polynomial
T.matrix<- LKrigDefaultFixedFunction(x, m=2)
# quadratic polynomial
T.matrix<- LKrigDefaultFixedFunction(x, m=3)
```

LKrigDistance-methods *Distance function methods for* LKrigDistance *in Package* **LatticeKrig**

Description

Distance function for distances within a limited radius and creating a sparse matrix. Supports either coordinates or locations specified by a grid. This method allows for finding pairwise distances among locations where one set can Methods for function LKrigDistance in package **LatticeKrig**

Methods

`signature(x1 = "matrix", x2 = "gridList", delta = "numeric")` Finds pairwise distances within the radius delta between all locations `x1` and all grid points. Will call either `LKDistGrid` or `LKDistGridComponents`. The function is particularly efficient when at least one set of locations is a regular grid and this is the main reason for this "overloading".

`signature(x1 = "matrix", x2 = "matrix", delta = "numeric")` Finds pairwise distances within the radius delta between all locations `x1` and `x2`. Will call either `LKdist` or `LKDistComponents`. These methods also take addtional arguments that specify the distance type. See help on `LKdist` and Radial.basis for details.

LKrigLatticeCenters *Methods to report the locations or scales associated with the lattice points.*

Description

These method takes the lattice information for a particular geometry from an LKinfo object and finds the locations or scales at each lattice points. These locations are the "nodes" or centers of the basis functions. The "scales" scales that distance function when the basis functions are evaluated and combine the spacing of lattice and the specified overlap.

Usage

```
LKrigLatticeCenters(object, ...)
## Default S3 method:
LKrigLatticeCenters(object, ...)
## S3 method for class 'LKInterval'
LKrigLatticeCenters(object, Level, ...)
## S3 method for class 'LKRectangle'
LKrigLatticeCenters(object, Level, ...)
## S3 method for class 'LKBox'
LKrigLatticeCenters(object, Level, ...)
## S3 method for class 'LKCylinder'
LKrigLatticeCenters(object, Level = 1, physicalCoordinates = FALSE, ...)
## S3 method for class 'LKRing'
LKrigLatticeCenters(object, Level = 1,
                    physicalCoordinates = FALSE, ...)
## S3 method for class 'LKSphere'
LKrigLatticeCenters(object, Level, ...)
## Default S3 method:
LKrigLatticeScales(object, ...)
LKrigLatticeScales(object, ...)
```

Arguments

object	An LKinfo object.
Level	The multi-resolution level.
physicalCoordinates	If TRUE the centers are returned in the untransformed scale. See the explanation of the V matrix in LKrigSetup. This is useful to relate the lattice points to other physical components of the problem. For example with the LKRing geometry representing the equatorial slice of the solar atmosphere one observes a line of sight integral through the domain. This integral is obvious found with respect to the physical coordinates and not the lattice points.
...	Any additional arguments for this method.

Details

This method is of course geometry dependent and the default version just gives an error warning that a version based on the geometry is required. Typically generating these lattice points from the information in LKinfo should be easy as the grid points are already determined.

The scales reported are in the simplest form delta*overlap where delta is a vector of the lattice spacings and overlap (default is 2.5) is the amount of overlap between basis functions.

See the source for the function LKrig.basis for how each of these is used to evaluate the basis functions.

Value

Centers A matrix where the rows index the points and columns index dimension. In the case of the LKRectangle geometry attribute is added to indicate the grid points used to generate the lattice. For LKSphere the centers are in lon/lat degrees. (Use directionCosines to convert to 3-d coordinates from lon/lat.)

Scales The default method returns the vector delta*offset with length being the number of multi-resolution levels.

Author(s)

Doug Nychka

See Also

LKrig.basis LKrigSetup, LKrigSetupAwght, LKrigSAR, LKrig

Examples

```
  x<- cbind( c(-1,2), c(-1,2))
  LKinfo<- LKrigSetup( x, alpha=c( 1,.2,.01),
                  nlevel=3, a.wght=4.5, NC= 10)
# lattice centers for the second level
# not points added for buffer outside of spatial domain
   look<- LKrigLatticeCenters(LKinfo, Level=2)
```

```
# convert grid format (gridList)  to just locations
   look<- make.surface.grid( look)
   plot( look,  cex=.5)
   rect( -1,-1,2,2, border="red4")

    x<- cbind( c(0, 360), c( 1,3))
    LKinfo<- LKrigSetup( x, LKGeometry="LKRing",
                  nlevel=1, a.wght=4.5, NC= 10, V= diag(c( 1,.01) ) )

    polar2xy<- function(x){
x[,2]*cbind( cos(pi*x[,1]/180), sin(pi*x[,1]/180))}

    look1<- LKrigLatticeCenters( LKinfo, Level=1)
    look2<- LKrigLatticeCenters( LKinfo, Level=1, physicalCoordinates=TRUE )
    look3<- polar2xy( look2$Locations )
# Basis scales:
      LKrigLatticeScales( LKinfo)

    set.panel(3,1)
    plot( make.surface.grid( look1))
    plot( look2$Locations)
    plot( look3)
```

`LKrigNormalizeBasis` *Methods and functions to support normalizing to marginal variance of one.*

Description

The basis functions can be normalized so that the marginal variance of the process at each level and at all locations is one. A generic function `LKrigNormalizeBasis` computes this for any LatticeKrig model. However, in special cases the normalization can be accelerated. `LKrigNormalizeBasisFast` takes advantage of rectangular geometry and a constant `a.wght`, and further decomposes the SAR matrix using Kronecker products. This method calculates the exact marginal variance while providing an increase in computational efficiency. The `LKrigNormalizeBasisFFTInterpolate` method is the fastest, yet it provides an approximate marginal variance, and can only be used when there are significantly less basis functions than locations.

Usage

```
LKrigNormalizeBasis(
LKinfo, Level, PHI = NULL, x1 = NULL, gridList = NULL, ...)
LKrigNormalizeBasisFast(LKinfo, ...)
LKrigNormalizeBasisFFTInterpolate(LKinfo, Level, x1)
## Default S3 method:
LKrigNormalizeBasisFast(LKinfo, ...)
## S3 method for class 'LKRectangle'
```

```
LKrigNormalizeBasisFast(LKinfo, Level, x1,  ...)
LKRectangleSetupNormalization(mx, a.wght)
```

Arguments

a.wght	A.wght parameters.
gridList	If not NULL this is a gridList object that will be expanded to give a grid of locations and used for x1.
LKinfo	An LKinfo object. NOTE: Here choleskyMemory, a Spam memory argument, can be a component of LKinfo and is subsequently passed through to the (spam) sparse cholesky decomposition
Level	The multi-resolution level.
mx	Matrix of lattice sizes.
PHI	Unnormalized basis functions evaluated at the locations to find the normalization weights.
x1	Locations to find normalization weights. If NULL then locations obtained from gridList argument.
...	Additional arguments for method.

Details

Normalization to unit variance is useful for reducing the artifacts of the lattice points and creates a model that is closer to being stationary. The computation must be done for every point evaluated with the basis functions The basic calculation is

```
tempB <- LKrigSAR(LKinfo, Level = Level)
    tempB <- LKrig.spind2spam(tempB)
    wght <- LKrig.quadraticform(t(tempB)
        choleskyMemory = choleskyMemory)
```

This generic method uses the low level utility LKrig.quadraticform that evaluates diag(t(PHI) solve(Q.l) PHI) where PHI are the basis functions evaluated at the locations and Q.l is the precision matrix for the lattice coefficients at level l.

For constant a.wght and for the rectangular geometry one can use an eigen decomposition of the Kronecker product of the SAR matrix. This allows for a faster way of calculating the exact marginal variance.

In cases where (2 * NC - 1) < s, where NC is the selected number of basis functions and s is the larger side of a 2-dimensional dataset, an approximate marginal variance with extremely high accuracy (average error 0.001%, maximum error ~2%) can be calculated. This is done by first calculating the marginal variance on a coarser grid, then performing 2-dimensional Fourier Interpolation to upsample to all locations of the dataset.

Value

A vector of weights. The basis are normalized by dividing by the square root of the weights.

Author(s)

Doug Nychka

`LKrigSAR` *Method that creates the spatial autoregressive (SAR) matrix.*

Description

Using the information in LKinfo create the SAR matrix for a given level of the multi-resolution.

Usage

```
LKrigSAR(object, ...)
  ## Default S3 method:
LKrigSAR(object, ...)
  ## S3 method for class 'LKInterval'
LKrigSAR(object, Level, ...)
  ## S3 method for class 'LKRectangle'
LKrigSAR(object, Level, ...)
  ## S3 method for class 'LKBox'
LKrigSAR(object, Level, ...)
  ## S3 method for class 'LKRing'
LKrigSAR(object, Level, ...)
  ## S3 method for class 'LKCylinder'
LKrigSAR(object, Level, ...)
   ## S3 method for class 'LKSphere'
LKrigSAR(object, Level, ...)
```

Arguments

`object`	An LKinfo object.
`Level`	The level of the multi-resolution.
`...`	Any additional arguments to pass to this method.

Details

The model for the Gaussian Markov Random field, c, at a given level is

B c = e,

where B is the SAR matrix computed by this method, and e are uncorrelated N(0,1). The precision matrix for this level is

Q= t(B)%*% B

and so the covariance matrix for c is the inverse of Q:

solve(Q)= solve(B)%*% t(solve(B))

Value

A matrix in the sparse matrix format, spind, with dimensions given by the number of lattice points at Level. Because this construction is geometry dependent the default version of this method just returns an error message.

Author(s)

Doug Nychka

See Also

LKrig.precision,spind2full

Examples

```
    x<- cbind( c(0,1))
LKinfo<- LKrigSetup(x,LKGeometry="LKInterval",
               nlevel=3, NC=3, a.wght=5, alpha=c(1,.5,.2) )
B<- LKrigSAR( LKinfo, Level=2)
B<-spind2full(B)
image.plot( B)

LKinfo<- LKrigSetup(cbind( c(0,360), c(0,1)) ,LKGeometry="LKRing",
               nlevel=1, NC=3, a.wght=5, alpha=1)
B<- LKrigSAR( LKinfo, Level=1)
B<-spind2full(B)
image.plot( B)
```

LKrigSetup *Create or update the LatticeKrig model object (LKinfo) for spatial fitting.*

Description

This function combines some input arguments with defaults for other to create a list describing the LatticeKrig spatial model. A key to specifying the LatticeKrig spatial model is specifying the geometry, e.g. LKRectangle for a 2-d rectangular domain. Each geometry has some parameters that control the basic model setup and these are included through the ... arguments of this function. See the help for this argument below for some examples.

Usage

```
LKrigSetup(x = NULL, nlevel = NULL, alpha = NA, alphaObject =
                 NULL, nu = NULL, a.wght = NA, a.wghtObject = NULL, NC
                 = NULL, NC.buffer = NULL,delta = delta, normalize = TRUE,
                 normalizeMethod =
               "exact", lambda = NA, tau = NA, sigma2 = NA, sigma2.object = NULL,
                 latticeInfo = NULL, basisInfo = NULL, LKGeometry =
```

```
                    "LKRectangle", distance.type = "Euclidean",
                    BasisFunction = "WendlandFunction", overlap = 2.5, V =
                    NULL, BasisType = "Radial", fixedFunction =
                    "LKrigDefaultFixedFunction", fixedFunctionArgs =
                    list(m = 2), collapseFixedEffect = FALSE, max.points =
                    NULL, mean.neighbor = 50, choleskyMemory = NULL,
                    verbose = FALSE, noCheck = FALSE, returnCall = FALSE,
                    dense = FALSE, ...)

LatticeKrigEasyDefaults(argList, nlevel, x)

LKinfoUpdate(LKinfo, ... )
```

Arguments

argList	Argument supplied to the top level LatticeKrig function.
alpha	A vector of length nlevel with the relative variances for the different multi-resolution levels.
alphaObject	For non-stationary models an object to be used with the predict function to give the alpha values at the process locations.Typically this is a list of "predict" objects. See nonstationaryModels for examples how to use this option.
a.wght	This parameter controls the correlation range in the SAR model. In most cases, a scalar value and for the 2-d model by default greater than 4. If a vector this can specify an anisotropic set of weights. To specify a.wght parameters that are different for each level they should be in the form of a list of length nlevel. E.g. a.wght = list(4.5, 5, 10) will specify different a.wghts for 3 different levels. The setup function will check that the values are in a valid range for a geometry and the length of the list agrees with the number of levels. The details of how this is connected to the covariance function varies based on the geometry. However, qualitatively this is related to a range parameter. For the LKRectangle geometry and a stationary model, at level k the center point has weight 1 with the 4 nearest neighbors given weight -1/a.wght[k]. In this case a.wght must be greater than 4 for the fields to be stationary and following Lindgren and Rue the range parameter is approximately 1/sqrt(a.wght-4).
a.wghtObject	For non-stationary models an object to be used with the predict function to give the a.wght values at the lattice locations. See nonstationaryModels for examples how to used option.
basisInfo	A list with extra components the object used to describe the multi-resolution basis. Usually this will not be needed for standard models.
BasisType	A character string indicating the type of basis function. Currently this is either "Radial" or "Tensor".
choleskyMemory	A list that will be used in the spam call to do the Cholesky decomposition. See the memory argument in chol.spam.
collapseFixedEffect	If FALSE the fixed part of the model is found separately for each replicated data set. If TRUE the estimate is polled across replicates.This is largely a modeling

decision whether variation among the replicate fields is due to the spatial component or also include variation in the fixed effects across replicates – guess they are not really fixed then!

`delta` A vector that will be used for the basis center spacings. This is an alternative choice to `NC`. Note that all the spacings need to be specified as a vector with length `nlevel`.

`dense` If FALSE sparse linear algebra is used for the computations . If TRUE the matrices are made "dense" (zeroes are filled in) and the ordinary Lapack functions are used for the linear algebra. This option is primarily for testing and timing sparse verses standard algorithms.

`distance.type` A text string indicate type distance to use between spatial locations when evaluating the basis functions. Default is "Euclidean". Other choices when locations are in degrees longitude and latitude are "Chordal" and "GreatCircle" with the default units being miles. See Details below how to change the radius that is used.

`fixedFunction` A text string that is the name of the function used to find the fixed part of the spatial model based on the locations. The default is a linear (m=2) polynomial in the spatial coordinates. See `LKrigDefaultFixedFunction` for more details. Set this to NULL if you do not want to include a fixed effect in the spatial model.

`fixedFunctionArgs`
A list containing arguments to supply when evaluating the fixed function.

`lambda` The "noise to signal ratio" or also known as the smoothing parameter it is the parameter `lambda = tau^2/sigma2`. If specified then tau and sigma2 typically are estimated in LKrig by maximum likelihood. If lambda is not specified then it is set as `lambda = tau^2/ sigma2`. Note that to evaluate the spatial process model, e.g. using the function `LKrig.cov`, a value of lambda is not needed and this argument can default to NA.

`latticeInfo` Part or all of the object used to describe the Markov random field lattice. In the standard cases this list is created in the setup function and need not be specified. See `LKrigSetupLattice` for details. Note that the contents of this list is concatenated to any additional components supplied by `LKrigSetupLattice`.

`LKGeometry` A text string that gives the names of the model geometry. The default is "LKrectangle" assuming the spatial domain is a rectangle. Other common choices are "LKInterval" (1 d problem) and "LKBox" (for a 3d problem). See LKGeometry for more detials.

`LKinfo` A list that has class "LKinfo".

`mean.neighbor` The average number of nonzero points when each basis function is evaluated at a set of points points in the spatial domain.

`max.points` This is a parameter for the nearest neighbor distance function that sets the maximum array size for the nonzero distances among points. e.g. with 100 points each with 20 nonzero neighbors max.points needs to be 2000 = 100*20. Specifically if the total number of nonzero values when each basis function is evaluated at all the spatial locations. The easier way to specify space is by using the mean.neighbor argument.

NC — For regular grids of lattice points the maximum number of lattice grid points for a spatial coordinate and at the coarsest level of resolution. For a example, for a square region, (and LKGeometry = "LKRectangle") NC=5 results in a 5X5 = 25 lattice points at the first level. Note that the default is that lattice points are also taken to have the same spacing in every dimension. See the delta argument as an alternative way to specify the basis centers.

NC.buffer — Number of extra lattice points added outside the spatial domain for regular grids of lattice points. This helps to reduce boundary effects from the SAR model. NC.buffer=5 and NC=5 for a square region will result in (5+ 2*5) X (5+ 2*5) = 225 lattice locations at the coarsest level of resolution. Note that this number by default is fixed for finer resolutions and so does not contribute as much to the total number of lattice points. This option works for either NC or delta.

nlevel — Number of levels in multi-resolution. Note that each subsequent level increases the number of basis functions within the spatial domain size by a factor of roughly 4. But not exactly 4!

noCheck — If FALSE do not make any checks on the consistency of the different parts of the final LKinfo object. e.q. values of a.wght within the range of a stationary Markov random field.

normalize — If TRUE the basis functions will be normalized to give a marginal variance of one for each level of multi-resolution. (Normalizing by levels makes it easier to interpret the alpha weights.)

normalizeMethod — Options are "exact", "exactKronecker", "fftInterpolation", and "both". The "exact"" option calculates the exact marginal variance but slowly, while "exactKronecker" can offer a speedup in scenarios with a constant a.wght and rectangular geometry. "fftInterpolation" offers an ever faster but approximate calculation, and can only be used in situations where the number of locations significantly exceeds th e number of basis functions. The "both" method automatically uses the FFT approximation for lower levels, and switches to the Kronecker method when the number of basis functions grows to be too large for the approximate method. See LKrigNormalizeBasis for more detail.

nu — A smoothness parameter that controls relative sizes of alpha. Currently this parameter only makes sense for the 2D rectangular domain (LKRectangle)

overlap — Controls the overlap among the radial basis functions and should be in units of the lattice spacing. For the rectangular case the default of 2.5 means that the support of the Wendland basis function will overlap 2.5 lattice units in each direction. See LKrig.basis for how overlap adjusts scaling in the basis function formula.

sigma2 — A scalar, the sill or marginal variance of the process.

BasisFunction — Text string giving the 1-d form for basis function.

sigma2.object — A prediction object to specify part of the marginal variance for the process. Specifically the form is VAR(g(x1))= sigma2*h(x1) Calling predict(sigma2.object,x1) should return a vector with the values of h at the (arbitrary) spatial locations in x1. If omitted assumed to be the constant one – the usual case.

returnCall — If TRUE the call to LKrigSetup is also included as part of the LKinfo object.

tau	The measurement error standard deviation. Also called the nugget variance in geostatistics.
V	See entry in LKrig.
verbose	If TRUE print out intermediate information.
x	Spatial locations that define the spatial domain for prediction. This is only used to determine ranges of the grid for the basis functions so, for example, for a rectangular domain only two points are required that bound the rest of the data locations. E.g. x= rbind(c(0,0), c(1,1)) will set the domain to be the unit square.
...	Specific arguments that will be included in the setupArgs list and also passed to LKrigSetupLattice . For LKinfoUpdate these specify the components of LKinfo to update. For example for a rectangular domain (the default) the argument NC is needed to specify the initial lattice size NC.buffer sets the number of extra points included in the lattice and defaults to 5. NC for the LKSphere geometry is the initial level of the icosahedral grid.

Details

Many of the functions within LKrigSetup are overloaded to adapt to the LKGeometry class. This makes it easy to add new geometries or other models to the LatticeKrig framework. The required components of this object (see below) outline how the latticeKrig model is structured and what should be common features independent of the geometry. The key components are:

basisInfo used to specify the form of the basis functions (see LKrig.basis)

latticeInfo that contains the information used to generate the spatial autoregressive matrix on the lattice. See (LKrigSetupLattice for the top level function that sets up the lattice.

a.wght A component that specifies the weights in the SAR model. (See LKrigSetupAwght)

alpha A component that gives the relative weights of the different layers for a multiresolution model. (See LKrigSetupAlpha.)

Also part of this flexibility is to use different distance functions (metrics). See help on LKDistance for the S4 method to find the distance and LKDist for the lower level functions that actually do the work. To change the radius used for the spherical distances one can add a Radius attribute to the text string. For example to use kilometers for great circle distance (R approximately 6371)

```
dtype <- "GreatCircle"
attr( dtype, which="Radius") <- 6371
```

Now use distance.type = dtypefor the distance type argument.

The function LKrigEasyDefaults is used in the top level function LattticeKrig to make the logic of different default choices more readable and reduces the clutter in this function. Its main purpose is to find a reasonable choice for NC when this is not specified.

As example of how the computation is structured with the complete LKinfo object LKrigSAR can be called to generate the sparse SAR matrix of the basis function coefficients at a given level of resolution:

```
B1<- LKrigSAR(LKinfo, Level=1)
```

The basis function matrix can be evaluated at a matrix of locations $x1 as

```
Phi<- LKrig.basis( x1,LKinfo)
```

The function LKinfoUpdate is more of a utility used for clarity that allows one to update the LKinfo object with particular components without having to recreate the entire object. This function is used in the MLE search when just values of alpha or a.wght are being varied.

Value

An object with class "LKinfo" and also the additional class given by LKGeometry. The required components are:

nlevel Number of levels

alpha alpha parameters as a list that has nlevel components and possibly some attributes.

a.wght a.wght parameters as a list that has nlevel components and possibly some attributes.

nu nu parameter

normalize A logical indicating whether to normalize.

normalizeMethod A string that determines the method for normalization.

lambda Value of lambda.

tau Value of tau.

sigma2 Value for sigma2.

sigma2.object Value for sigma2.object.

latticeInfo A list with specific multi-resolution lattice information

setupArgs All arguments passed in the call and any in in ...

basisInfo A list with basis information.

call The actual call used to create this object.

Author(s)

Doug Nychka

Examples

```
data(ozone2)
# find the ranges of the  data, this is the same as passing
# the entire set of observation locations and is more compact
sDomain<-apply( ozone2$lon.lat, 2,"range")
LKinfo0<- LKrigSetup( sDomain, NC=10, nlevel=2, alpha=c(1,.5),
                      a.wght = 5)
print( LKinfo0)

#Gigantic buffer added note extra basis functions.
LKinfo<- LKrigSetup( sDomain, NC=10, NC.buffer= 15, nlevel=2,
alpha=c(1,.5),a.wght = 5)
print( LKinfo)
```

```
  LKinfo2<- LKinfoUpdate( LKinfo,  a.wght=4.1, NC=12)
  LKinfo3<- LKrigSetup( sDomain, NC=12, nlevel=2, alpha=c(1,.5),
                        a.wght=4.1)
# LKinfo2 and LKinfo3 should be the same.
```

LKrigSetupAlpha *Creates the alpha parameter list in LatticeKrig covariance.*

Description

This function is called by LKrigSetup and creates the list for the alpha parameters based on the information from the call to LKrigSetup and additional information and conditions related to the geometry.

Usage

```
LKrigSetupAlpha(object, ...)
## Default S3 method:
LKrigSetupAlpha(object, ...)

LKFindAlphaVarianceWeights(x1, LKinfo, level)
```

Arguments

x1	Locations to evaluate the alpha weights.
LKinfo	An LKinfo object that is usually created by LKrigSetup.
level	Level of the multiresolution.
object	The partial LKinfo object created within LKrigSetup
...	Any additional arguments to this method

Details

The main function of this method is to fill in the sequence of alpha values for a parametric model and convert those values to a list instead of a vector. In the case that the scalar nu is supplied it is used to create the list according to:

```
alpha <- 2**(-2 * (1:nlevel) * nu)
    alpha <- alpha/sum(alpha)
    as.list( alpha)
```

Value

A list with nlevel components each representing the alpha values at that level. In the simplest case a vector of alpha values is converted to a list.

```
  LKinfo<- LKrigSetup( x, alpha=c( 1,.2,.01),
                   nlevel=3, a.wght=4.5, NC= 3)
  LKrigSetupAlpha( LKinfo)
[[1]]
[1] 1
[[2]]
[1] 0.2
[[3]]
[1] 0.01
```

The lower level function LKFindAlphaVarianceWeights is used to supply weights as in the default model but also evaluate the weights at arbitrary locations. This second role is part of specifying a nonstationary model where the weights vary over the spatial domain.

Author(s)

Doug Nychka

See Also

LKrigSetup, LKrigSetupAwght, LKrigSAR, LKrig

Examples

```
# an x that is just the limits of the domain
  x<- cbind( c(0,1), c(0,1))

  LKinfo<- LKrigSetup( x, alpha=c( 1,.2,.01),
                   nlevel=3, a.wght=4.5, NC= 3)
  alphaList<- LKrigSetupAlpha( LKinfo)

  LKinfo<- LKrigSetup( x, nu=1, nlevel=4, a.wght=4.5, NC= 4)
  alphaList<- LKrigSetupAlpha( LKinfo)
```

LKrigSetupAwght *Method to create a.wght component from the* LKinfo *object.*

Description

This method takes a vector or more complicated object and based on the geometry creates a list with the a.wght information.

Usage

```
LKrigSetupAwght(object, ...)
## Default S3 method:
LKrigSetupAwght(object, ...)

## S3 method for class 'LKRectangle'
LKrigSetupAwght(object, ...)

LKrigSetupAwghtObject(object)
```

Arguments

object	The partial or complete LKinfo object created within LKrigSetup.
...	Any additional arguments to this method.

Details

The simplest function of this method is to convert the a.wght value into a list that has the length of the number of levels. If only a scalar a.wght value is supplied then the default method just repeats this for each level.

The function LKrigSetupAwghtObject uses the a.wghtObject component in the LKinfo object to fill in a.wght parameters for the different levels. This is convenient because the lattice locations are different at each level. The parameters are filled in at level, Level according to

```
latticeLocations<- make.surface.grid(
                          object$latticeInfo$grid[[Level]])
a.wght<- predict( object$a.wghtObject, latticeLocations )
```

here the predict function is whatever is supplied according to the class for a.wghtObject. Note that since the returned set of parameters will be in the format used internally a.wght here will be a list with each component being a matrix. Number of rows are each to the number of lattice points (or basis functions) at that level. This is easier implement that it may seem and see the examples in nonstationaryModels.

The attribute fastNormalize (either TRUE or FALSE) is attached to this list to indicate how the marginal variance of the process should be found.

```
LKinfo<- LKrigSetup( x,LKGeometry="LKInterval", alpha=c( 1,.2,.01),
                nlevel=3, a.wght=4.5, NC= 3)
LKrigSetupAwght( LKinfo)

[[1]]
[1] 4.5

[[2]]
[1] 4.5
```

```
[[3]]
[1] 4.5

attr(,"fastNormalize")
[1] FALSE
```

Currently the only geometry with fastNormalization being TRUE is for a rectangular domain.

For the LKRectangle geometry, however, more complicated anisotropic and non-stationary a.wght specifications are possible. See LKrig for details. Also in the case that the fastNormalization is TRUE for rectangles several more attributes are added to the `a.wght` list that precompute some matrices of the SAR.

Value

A list with `nlevel` components. The attribute `fastNormalize` is added to this list. In the case that the geometry is LKRectangle several more attributes are added indicating the type of covariance model and possibly an eigen decomposition of the SAR matrix exploiting Kronecker products.

Author(s)

Doug Nychka

See Also

LKrigSetup, LKrigSetupAlpha, LKrigSAR, LKrig

Examples

```
  x<- cbind( c(0,1))
  LKinfo<- LKrigSetup( x,LKGeometry="LKInterval", alpha=c( 1,.2,.01),
                  nlevel=3, a.wght=4.5, NC= 3)
  a.wghtList<- LKrigSetupAwght( LKinfo)

  x<- cbind( c(0,1), c(0,1))
  LKinfo<- LKrigSetup( x, alpha=c( 1,.2,.01),
                  nlevel=3, a.wght=4.5, NC= 3)
  a.wghtList<- LKrigSetupAwght( LKinfo)
# see
  names(attributes( a.wghtList))
```

LKrigSetupLattice *Creates the lattice information for a specific geometry.*

Description

Given a specific geometry and the initial information supplied in an LKinfo list create the information that is needed to define the lattice for a given model. This function is required for any new geometry added to LatticeKrig.

Usage

```
LKrigSetupLattice(object, ...)
## Default S3 method:
LKrigSetupLattice(object, ...)

## S3 method for class 'LKBox'
LKrigSetupLattice(object, verbose,...)

LKBoxCreateLattice(LKinfo, verbose = FALSE)

## S3 method for class 'LKRectangle'
LKrigSetupLattice(object, verbose,...)

LKRectangleCreateLattice(LKinfo, verbose = FALSE)

## S3 method for class 'LKInterval'
LKrigSetupLattice(object, verbose,...)

LKIntervalCreateLattice(LKinfo, verbose = FALSE)

 ## S3 method for class 'LKRing'
LKrigSetupLattice(object, verbose,...)

## S3 method for class 'LKCylinder'
LKrigSetupLattice(object, verbose,...)

 ## S3 method for class 'LKSphere'
LKrigSetupLattice(object, x = NULL, verbose,...)
```

Arguments

LKinfo	The LKinfo object created by LatticeKrig or by LKrigSetup.
object	A list that is an LKinfo object.
verbose	If TRUE print out intermediate information for debugging.

x	Locations of the observations that define spatial domain. For LKSphere locations are in lon/lat degrees.
...	Any additional arguments.

Details

This method takes the LKInfo object and the other arguments and computes the lattice information needed for a specific geometry. Three lower level functions that actually create the lattices for regular grids in 1D, 2D, and 3D were written to make the code eaiser to follow and hanlde both the case when the number of basis functions is specified (`NC`) or where the grid spacing is specified (`delta`).

These functions are called from within LKrigSetup and the results are added as a component to`latticeInfo` as part of the LKinfo object. The way to design what should be in `latticeInfo` is to keep in mind that creating the spatial AR matrix (LKrigSAR) and determining the multi-resolution lattice points (LKrigLatticeCenters) use the information in the LKinfo object.

Because the lattice must depend on the geometry the default method just prints out an error message.

NC and NC.buffer for Cartesian geometries (e.g. `LKInterval`, `LKRectangle`, `LKBox`, `LKRing`) are parameters to specify to number of the grid points in the largest dimension of the spatial domain and for the coarsest lattice. These should already be a component in the LKinfo object, object specified in the usage.

For `LKInterval` this is just of lattice points at the coarsest level. For LKRectangle and LKBox if the spatial domain has the same size in all dimensions then the number of lattice points within the spatial domain are NC^2 and NC^3 respectively. Note that the total number of lattice points at a given level is also effected the size of NC.buffer (see below).

The number of lattice points to add to the margins beyond the spatial domain is controlled by `NC.buffer`. Thus in the largest spatial dimension there are a total of NC + NC.buffer*2 grid points. This is the number of lattice points for LKInterval.

Value

For the `LKInterval`, `LKRectangle`, `LKBox` geometries. A list with required components:

m The total number of lattice points. i.e. the total number of basis functions.

mx A matrix giving the number of lattice points in each coordinate (columns) and at each level (rows) of the multi-resolution.

offset When the lattice points are unrolled as a single array, a vector of indexes giving the start of each lattice level in the coefficient vector.

delta A vector of the lattice point spacings for each level.

rangeLocations Limits of spatial domain.

mLevel A vector giving number of lattice points at each level.

In addition, the methods for Cartesian and cartesian-like spatial domains (LKInterval, LKRectangle, LKBox, LKRing, LKCylinder, LKSphere) include the additional components:

mx A matrix giving number of grid points at each level and for each dimension.

mLevelDomain Same as mLevel but restricted to lattice points within the spatial domain.

mxDomain Same as mx but restricted to the points within the spatial domain.

NC Passed value.

NC.buffer Passed value.

grid A list where each component is also list giving the grid points of the lattice in each coordinate.

LKSphere also returns the additional component `grid3d` – the direction Cosines of the grid points.

Author(s)

Doug Nychka

See Also

`LatticeKrig` `LKGeometry` `LKrigSetup`, `LKrigSAR`, `LKrigLatticeCenters`

`nonstationaryModels` *Specifying non-stationary models*

Description

An overview of specifying non-stationary and anisotropic models with some specific 2-d examples showing how to vary the alpha, sigma2 and a.wght parameters.

Details

The Lattice Krig model can be extended in a natural way to have a non-stationary covariance that has a multi-scale structure.

The default and process model has the form

$$g(x) = \sum_{l=1}^{L} g_l(x)$$

where g_l(x) are processes defined by fixed basis functions and with coefficients that follow a mean zero multivariate normal distribution and with dependence described by a spatial autoregression (SAR).

Under the model with approximate stationary the SAR is parameterized by a set of a.wght values that are applied in the same way to every lattice point and its neighbors. When the process is normalized to have constant marginal variance, the variance of g_l(x) is given by sigma2* alpha_l. We recommend that the alpha parameters sum to one and so the marginal variance of g(x) is given by sigma2. This package allows for the parameters a.wght, sigma2, and alpha to vary over the spatial domain. In this way the variance of g_l(x) is given by sigma2(x)alpha_l(x) and g(x) by sigma2(x) provided alpha_l sums to one. The a.wght parameters can different values at each lattice point and possibly at each level. Some preliminary results suggest that having a.wght vary differently for different levels may be too much flexibility and may not needed.

We describe the formatting for these features in the LKinfo object below. Essentially they involve specifying one or more of the arguments: `a.wghtObject`, `sigma2.object` or `alphaObject`, in the call to `LKrigSetup`.

a.wght The general form of this object is as a list of matrices. In this case length(a.wght) = nlevel and a.wght[[l]] is a matrix where the number of rows is equal to the number of lattice points at level l and in the order that they are indexed for the SAR matrix. The number of columns depends on the particular geometry but we explain how this works for the rectangular spatial domain, LKRectangle. Here a.wght[[l]] has either 1 or 9 columns depending if anisotropy is specified. The weights for the LKRectangle anisotropy case are organized as

```
1 4 7
2 5 8
3 6 9
```

So an isotropic model with the center value as 4.5 looks like

```
 0   -1    0
-1   4.5  -1
 0   -1    0
```

and to encoded with the anisotropic extension the row in a.wght[[l]] corresponding to this lattice point would be

```
c( 0, -1, 0, -1, 4.5, -1, 0,-1, 0)
```

Specifying the matrices of a.wghts directly can be involved because one needs to known the lattice information. An easier way to accomplish specifying these models is to use a function on the spatial domain to define the a.wght values based on the locations of the lattice points. In this case one would pass an object, a.wghtObject, that describes the function. This object needs to be defined so that predict(a.wghtObject, x) will evaluate the function at the locations x. (See example below.). With this object, LKrigSetupAwght will evaluate the function at the lattice points and so create the correct list of matrices.

alpha. To specify spatial varying alpha parameters one specifies `alphaObject` as a list of objects where the predict function works:

```
predict(alphaObject[[l]], x)
```

The result should give the values for alpha_l(x) with `x` a matrix of arbitrary locations.

One should also set the vector of usual alpha parameters equal to all ones so only the spatially varying values as used for the variance. E.g. alpha= rep(1, nlevel)

sigma2 The object `sigma2.object` is used define a spatially varying sigma2(x). Like alpha and a.wght this object must work with the predict function.

```
predict(sigma2.object, x)
```

Examples

```
#########################################################
##### This is an extended example showing how to define
##### spatially varying sigma2 parameter
#########################################################
```

```
# Define some useful predict functions.
#########################################################
  predict.surfaceGrid<- function(object,x){
    interp.surface( object, x)
    }

  predict.multivariateSurfaceGrid<- function(object,x){
    dimZ<- dim( object$z)
    L<- dimZ[3]
    out<- matrix( NA, nrow= nrow(x), ncol=L)
    for (  l in 1:L){
     out[,l]<- interp.surface(
     list( x=object$x,y=object$y, z=object$z[,,l]) , x)
     }
     return( out)
  }

  predict.constantValue<- function(object,x){
   n<- length(object$values)
   m<- nrow( x)
   return( matrix( object$values, nrow=m, ncol=n, byrow=TRUE ) )
    }

###############################################
##### Non-stationary examples
###############################################
# spatial domain
sDomain<- rbind( c(-1.2,-1.2),
                 c(1,1))

# we will use this coarse grid to define any
# surfaces of parameters
# (unrelated to the lattice grids and plotting grid!)
# this is larger than the sDomain to accommodate buffer points
# (with larger ranges when NC is small)
  gridList<- list( x = seq( -3, 3,,50),
                   y = seq( -3, 3,,75) )
  xPoints<- make.surface.grid( gridList)
  fineGrid<- make.surface.grid(
                 list( x = seq(-1, 1, ,45),
                       y = seq(-1, 1, ,60)
                       )
                               )

################################################
### end of setup
################################################
# sigma2 increases across the domain as a function of first coordinate.
  sigma2Temp<-  .01 +  10* pnorm( xPoints[,1], mean=.25, sd =.3 )
  sigma2.object<- as.surface( xPoints, sigma2Temp)
  class( sigma2.object)<- "surfaceGrid"

  LKinfo<- LKrigSetup( sDomain, NC= 4, nlevel = 3,
```

```
                a.wght=4.5, nu=1, sigma2.object=sigma2.object)
# simulate a field from this model
  set.seed(123)
  look<- LKrig.sim( fineGrid, LKinfo)
  image.plot( as.surface( fineGrid, look))
  xline( .25, col="grey30")
# see also
# temp<- as.surface( fineGrid, look)
# I<- 20
# matplot(temp$x, temp$z, type="l", xlab="x", ylab="GP slice" )

#########################################################
##### spatially varying alpha parameters
#########################################################

# the alpha surface at each level will just be the result of
# bi-linear interpolation of values specified on a small grid.
# To keep things identified the alpha weights at
# any grid location are
# normalized to sum to 1.
#
# create a 3 column matrix with  (proportional) alpha weights
# at each grid point
#
  taper<- pnorm( xPoints[,1], mean = .4, sd=.02)
  alphaTemp<- cbind( taper,
        rep( 1, length( taper)),
                        1-taper)
# normalize to sum to one
  alphaTemp <- alphaTemp/rowSums(alphaTemp)

# pack as a list
# convert from a vector to the image/list format  $x $y $z
# give this object a class so that predict.surfaceGrid is called.
# accumulate these objects in a list
# (yes this is a "list of lists")
  alphaObject<- list()
  for( k in 1:3){
     hold<- as.surface( xPoints, alphaTemp[,k])
     class( hold)<- "surfaceGrid"
     alphaObject<- c( alphaObject, list( hold))
  }

# define the 2-d LatticeKrig model
  LKinfo<- LKrigSetup(sDomain, NC = 4, a.wght=4.5,
              alpha = c(1,1,1), nlevel = 3,
              alphaObject =  alphaObject )
# simulate a field

  set.seed(123)
  look<- LKrig.sim( fineGrid, LKinfo)
  image.plot( as.surface( fineGrid, look))
```

```
#######################################################
##### spatially varying a.wght parameters
##### See above comments and setup
##### for steps that are the same
#######################################################
  taper<- pnorm( xPoints[,1] + xPoints[,1],
                   mean = 0, sd=.15)
  a.wghtTemp<- 4.001*taper +  10*(1-taper)
# pack up as a list
# convert from a vector to the image/list format  $x $y $z
# give this object a class so that predict.surfaceGrid is called.
# accumulate these objects in a list (yes this is
# a "list of lists")

     a.wghtObjectA <- as.surface( xPoints, a.wghtTemp)
     class( a.wghtObjectA)<- "surfaceGrid"


# define the 2-d LatticeKrig model

  LKinfo2<- LKrigSetup(sDomain, NC = 5, NC.buffer=0,
              alpha = c(1, .5, .125), nlevel = 3,
              a.wghtObject =  a.wghtObjectA)

  set.seed(123)
  look<- LKrig.sim( fineGrid, LKinfo2)
  image.plot( as.surface( fineGrid, look))
#############################################
###### 1-d example
#############################################
  xCoarse1<- seq( -.5,1.5,, 40)
  y<-  pnorm( xCoarse1, mean=.4, sd=.05)*5 + 2.2
  a.wghtObject<- Tps(xCoarse1, y, lambda=0)
  alphaTemp<-c(.5, .3, .2)
  LKinfoTEST<- LKrigSetup( rbind(0,1), NC=10,
                           LKGeometry="LKInterval",
                           nlevel=3, alpha=alphaTemp,
                           a.wghtObject = a.wghtObject,
                           NC.buffer=2
                           )
  xFine1<- cbind(seq( 0,1,length.out= 200))
  set.seed( 123)
  look<- LKrig.sim( xFine1, LKinfoTEST, M=5)
  matplot( xFine1, look, type="l", lty=1)
#################################################
######## Anisotropy in a.wght
#################################################
#### stationary example
  a.wghtM2<- c( rbind( c(  0,   0, -1.5),
                       c(-.5, 4.5,  -.25),
                       c(-1.5,  0,    0)
                  )
                  )
```

```
  LKinfo3<- LKrigSetup(sDomain, NC = 5,
      a.wght= list( a.wghtM2),
              alpha = c(1, .5, .125), nlevel = 3,
              a.wghtObject =  NULL, normalize=TRUE )


  set.seed(123)
  look<- LKrig.sim( fineGrid, LKinfo3)
  image.plot( as.surface( fineGrid, look))

## Not run:

#### Anisotropy varying over space
#### First check that the constant model can be reproduced
  a.wghtM2<- c( rbind( c(  0,   0, -1.5),
                       c(-.5, 4.5,  -.5),
                       c(-1.5,  0,    0)
                  )
                  )

  a.wghtObject<- list( values=a.wghtM2)
  class(a.wghtObject )<- "constantValue"

  LKinfo4<- LKrigSetup(sDomain, NC = 5,
              alpha = c(1,.5, .125), nlevel = 3,
              a.wghtObject =  a.wghtObject, normalize=TRUE )
  set.seed(123)
  look<- LKrig.sim( fineGrid, LKinfo4)
  image.plot( as.surface( fineGrid, look) )

###### non-stationary anisotropy
 a.wghtA <- c( rbind( c(    0,   0, -2),
                       c( 0, 4.5,  0),
                       c(-2,  0,     0)
                  )
                  )
 a.wghtB <- c( rbind( c(  -2,   0,     0),
                       c(  0, 4.5,  0),
                       c(    0,    0, -2)
                  )
                  )
# Now create multivariate prediction object.
  gridList<-  attributes( xPoints)$grid.list
  m1<- length(gridList$x)
  m2<- length(gridList$y)
  z<- array( NA, c( m1,m2,9))
  alpha<- (xPoints[,1] + 1 )/2
  alpha<- ifelse( alpha <= 0, 0, alpha)
  alpha<- ifelse( alpha >= 1, 1, alpha)
# A for loop over the 9 pixels
  for(j in 1:9) {
# linear combination of two a.wght matrices
```

```
#
    zTemp<- a.wghtA[j] * (1-alpha) +  a.wghtB[j]*(alpha)
# coerce into an image format
    z[,,j]<- as.surface( xPoints, zTemp)$z
  }

  a.wghtObject<- list( x= gridList$x,  y= gridList$y, z=z )
  class( a.wghtObject)<- "multivariateSurfaceGrid"

  LKinfo5<- LKrigSetup(sDomain, NC = 25, NC.buffer=0,
              alpha = c(1,.5),
              nlevel = 2,
              a.wghtObject =  a.wghtObject )
  set.seed(122)
  fineGrid<- make.surface.grid(
                 list( x = seq(-1, 1, ,150),
                       y = seq(-1, 1, ,180)
                       )
                              )
  look<- LKrig.sim( fineGrid, LKinfo5)
  image.plot( as.surface( fineGrid, look), col = terrain.colors(256) )

## End(Not run)
```

`Radial.basis` *Two dimensional radial and tensor basis functions based on a Wendland function.*

Description

Two dimensional radial basis and tensor functions based on a Wendland function and using sparse matrix format to reduce the storage.

Usage

```
Radial.basis(x1, centers, basis.delta, max.points = NULL,
                  mean.neighbor = 50,
                  BasisFunction = "WendlandFunction",
                  distance.type = "Euclidean",
                        verbose = FALSE)

Tensor.basis(x1, centers, basis.delta, max.points = NULL, mean.neighbor = 50,
   BasisFunction = "WendlandFunction", distance.type = "Euclidean")

WendlandFunction(d)

triWeight(d)
```

Arguments

x1	A matrix of locations to evaluate the basis functions. Each row of x1 is a location. For spherical distances these locations are assumed to be in degrees.
centers	A matrix specifying the basis function centers. For spherical distances these locations are assumed to be in degrees.
d	A vector of distances.
basis.delta	A vector of scale parameters for the basis functions.
max.points	Maximum number of nonzero entries expected for the returned matrix.
distance.type	A text string indicate type distance to use between spatial locations when evaluating the basis functions. Default is "Euclidean". Other choices when locations are in degrees (e.g. longitude and latitude) are "Chordal" and "GreatCircle" with the default units being miles. See Details below how to change the radius that is used.
mean.neighbor	Average number of centers that are within delta of each x1 location. For centers on a regular grid this is often easy to estimate.
BasisFunction	A function that will take a non-negative argument and be zero outside [0,1]. This is applied to distance(s) to generate the basis functions. For tensor basis functions, the function is applied to the distance components for each dimension.
verbose	Print out debugging information if TRUE.

Details

This function finds the pairwise distances between the points x1 and centers and evaluates the function RadialBasisFunction at these distances scaled by delta. In most applications delta is constant, but a variable delta could be useful for lon/lat regular grids. The Wendland function is for 2 dimensions and smoothness order 2. See WendlandFunction for the polynomial form. This code has a very similar function to the fields function wendland.cov.

In pseudo R code for delta a scalar Radial.basis with the Wendland "shape" evaluates as

```
BigD<- rdist( x1,centers)
WendlandFunction(BigD/basis.delta)
```

and is comparable to

```
Radial.basis(x1, centers,basis.delta)
```

The actual code uses a FORTRAN subroutine to search over distances less than delta and also returns the matrix in sparse format.

Part of the flexibility in these function is to use different distance functions (metrics). See help on LKDistance for the S4 method to find distance and LKDist for the lower level functions that actually do the work. To change the radius used for the spherical distances one add a Radius attribute to the text string. For example to use kilometers for great circle distance (R approximately 6371)

```
dtype <- "GreatCircle"
attr( dtype, which="Radius") <- 6371
```

dtype is still a text string but has the extra bit of information attached to it. Now use distance.type = dtype for the distance type argument.

The function Tensor.basis has similar function as the radial option. The main difference is that a slightly different distance function is used to return the component distances for each dimension. In pseudo R code for delta a scalar and for just two dimensions Tensor.basis evaluates as

```
BigD1<- rdist( x1[,1],centers[,1])
BigD2<- rdist( x1[,2],centers[,2])
WendlandFunction(BigD1/basis.delta) *WendlandFunction(BigD1/basis.delta)
```

The user is also encourage to consider using the higher level function LKrig.basis paired with an LKinfo object to define the centers. But besure to use normalize = FALSE to avoid strange interactions of the SAR model, irrelevant in this case, with the basis function form.

Value

For Radial.basis and Tensor.basis a matrix in sparse format with number of rows equal to nrow(x1) and columns equal to nrow(center) and value are the basis functions evaluated at these combinations For WendlandFunction and triweight a vector with the values for this function.

Author(s)

Doug Nychka

See Also

LKrig.basis, LKDist

Examples

```
set.seed(12)
x<- cbind( runif(100), runif(100))
center<- expand.grid( seq( 0,1,,5), seq(0,1,,5))
# coerce to matrix
center<- as.matrix(center)

  PHI1<- Radial.basis(x, center, basis.delta = .5)
  PHI2<- Tensor.basis( x, center, basis.delta = .5 )
# similarity of radial and tensor product forms
  plot( c(0,1.1), c(0,1.1), type="p")
  for( k in 1:25){
points( PHI1[,k], PHI2[,k])
}

# LKrig with a different radial basis function.
#
  data(ozone2)
  x<-ozone2$lon.lat
  y<- ozone2$y[16,]
# Find locations that are not 'NA'.
# (LKrig is not set up to handle missing observations.)
```

```
  good <-  !is.na( y)
  x<- x[good,]
  y<- y[good]
  obj<- LKrig(x,y,NC=30,nlevel=1, alpha=1, lambda=.01, a.wght=5)

  obj1<- LKrig(x,y,NC=30,nlevel=1, alpha=1,
    lambda=.01, a.wght=5, BasisFunction="triWeight", overlap=1.8)

# It is usually better to create the LKinfo object first and then call LKrig
LKinfo <- LKrigSetup( x,NC=30,nlevel=1, alpha=1,
            lambda=.01, a.wght=5, BasisFunction="triWeight", overlap=1.8 )
obj1B<- LKrig(x,y, LKinfo=LKinfo)
#########################################################
###  a radial basis on the sphere
#########################################################
# icosohedral grid at the second generation
allLevels<- IcosahedronGrid(2)
# extract second level
centers3d<- allLevels[[2]]
centers<- toSphere(centers3d)
# take a look
plot( centers, xlim=c(-180,180), ylim=c( -90, 90), pch=16)
lonlat<- make.surface.grid( list( x= seq( -180,180,,80), y= seq( -90,90,,40)))
delta<- 5000 # default distance is in miles
# evaluate all basis functions on the grid
bigX<- Radial.basis( lonlat, centers, basis.delta= delta, distance.type="GreatCircle")
# look at the 12th basis function
b12<- bigX[,12]
image( as.surface( lonlat, b12), col=c( "white",tim.colors(256)), add=TRUE,
xlim="longitude", ylim="latitude")
points( centers,  pch=16, col="magenta", cex=2)
```

`registeredFORTRAN` *Internal FORTRAN routines for working with grids and finding distances.*

Description

These are objects of class `FortranRoutine` and also `NativeSymbolInfo` They provide information for compiled functions called with `.Call`, or `.Fortran`. Ordinarily one would not need to consult these and they are used to make the search among dynamically loaded libraries (in particular the fields library) have less ambiguity and also be faster. These are created when the package/library is loaded are have their definitions from the compilation of `init.c` in the package source (src) directory.

Format

The format is a list with components:

name The (registration ?) name of the C function.

address See NativeSymbolInfo.

dll Dynamically linked library information.

numParameters Number of calling arguments in function.

Details

Registered routines are

findnorm Finds marginal variance quickly for the case of a rectangular lattice, a specific Wendland basis, and constant a.wght at a given level. Called from `LKRectangleFastNormalization`

lkdist Euclidean distance between two sets of coordinates but restricted to a maximum distance.Called from `LKDist`.

lkdistcomp Same as lkdist but returns the component distances for each coordinate.Called by `LKDistComponents`

lkdistgrid Finds all distances in x1 that are within delta of a set of grid points. Results are returned in the row, column, value sparse matrix format.called from `LKDistGrid`

lkdistgridcomp Same as lkdistgrid but the component distances are found for each coordinate.`LKDistGridComponents`

lkdiag Fills in diagonal and off diagonal elements in sparse matrix format.

See `package_native_routine_registration_skeleton` for the utility used to create these data objects.

References

For background on registering C, C++ and Fortran functions see 5.4 of Writing R Extensions.

Examples

```
print(lkdistgridcomp)
```

---

setDefaultsLKinfo *Method for including default information in the LKinfo object.*

Description

This method is used to define the various parts of the LatticeKrig model by including specific components in the LKinfo object. The specific instance of the function that is used depends on the geometry that has been specified. Typically one would would not need to use this method for a new geometry.

Usage

```
setDefaultsLKinfo(object, ...)
## Default S3 method:
setDefaultsLKinfo(object, ...)
## S3 method for class 'LKRectangle'
setDefaultsLKinfo(object, ...)
## S3 method for class 'LKBox'
setDefaultsLKinfo(object, ...)
## S3 method for class 'LKRing'
setDefaultsLKinfo(object, ...)
## S3 method for class 'LKCylinder'
setDefaultsLKinfo(object, ...)
## S3 method for class 'LKInterval'
setDefaultsLKinfo(object, ...)
## S3 method for class 'LKSphere'
setDefaultsLKinfo(object,...)
```

Arguments

`object`	An object of class LKinfo.
`...`	Additional arguments to a specific method.

Details

This method is used to include some default settings or components in the LKinfo object and is called from `LKrigSetup` with the initial (or partial) LKinfo list that is formed from the information passed by the user to LKrigSetup.

In creating a new geometry this method is not required but can be used to include convenient default values and also some checks on the arguments passed to LKrigSetup. Often there are several arguments for a geometry that make sense to set to simplify the use or to avoid bad things ....

For the LKRing geometry the defaults values follow the logic:

- alpha set to 1.0 if not specified and nlevel is 1
- a.wght set to 4.01 if not specified.
- the fixed part of the model (fixedFunction) uses the function LKrigPeriodicFixedFunction which insures the first component is periodic.
- the transformation matrix, V has the default diag(c(1,1)). Also in this function if V[1,1] is not 1 then an error is given.

For the LKSphere geometry the defaults values follow the logic:

- alpha set to 1.0 if not specified and nlevel is 1
- a.wght set to 6.01 if not specified.
- the fixed part of the model (fixedFunction) just includes a constant (m=1)
- Default distance function is great circle distance, with sphere radius of 1.0.

Author(s)

Doug Nychka

See Also

LKrigSetup

`Spatial models for data on spherical regions.`

Geometries to represent 2-d and 3-d spherical data.

Description

These models include a completely spherical geometry based on a nearly regular node layout on the sphere. Also a simpler and perhaps more efficient versions are implemented where the first coordinate is periodic in the interval [0,360] and the remaining coordinates are regular (Euclidean). This might be used to approximate a section of spherical data that excludes the polar caps. These approximations are useful because one can take advantage of faster methods based on rectangular grids rather the more complex grids on a sphere. The disadvantage is that the mapping from these coordinates to the sphere is distorted as one gets close to the poles.

Details

These geometries are specified with the `LKGeometry` argument either in LKrigSetup or LatticeKrig. The first coordinate is longitude either from [0,360] or [-180,180] and the second is latitude [-90, 90].

They each have the four specific methods: `LKrigLatticeCenters`, `LKrigSAR`, `LKrigSetupLattice`, `setDefaultsLKinfo` and the source code is consolidated in the source files ModelRing.R and ModelCylinder.R in the R sub-directory of this package source. For the spherical grid the a.wght is handled a but differently please read the note below.

`LKSphere` Recall that the core of the LatticeKrig model is the placement of the basis function centers and how the spatial autoregression is constructed from these node locations. Here the centers are generated according to a multi-resolution based on the triangles from an icosahedron. `IcosahedronGrid` creates a grid by taking the first level as the 12 points from a regular icosahedron. The subsequent levels generate a finer set of points by subdividing each triangular face into 4 new triangles. The three new mid points from the subdivision are added to the previous nodes to give the new level of resolution. The triangles tend to roughly equilateral and so the nodes will tend to be roughly equally spaced. This regularly improves for the finer generations of triangles, however, the 12 original nodes from the icosahedron will have 5 nearest neighbors rather than 6.

The setup argument `startingLevel` specifies the first level of the lattice and `nlevel` the total number.

NOTE: Because the distances between nodes are not perfectly equally spaced and nearest neighbors can be either 5 or 6 some adjustment is made to the weights when forming the SAR matrix. The net result is that it makes more sense to have the off diagonal rows sum to 1 and so **a.wght must be greater than 1.0** for a stationary model.

See the help file on `link{IcosahedronFaces}` for code on visualizing these. The first 12 centers will have 5 close neighbors and remaining centers will have 6. Currently the SAR weights are roughly equal among all the neighbors but are adjusted so that a locally linear function will be in the "null" space of these weights. For each node the neighbors are projected onto the tangent plane to the sphere at this node location. Now consider a linear function on the coordinates in this tangent plane. The idea is to find weights applied to the neighbors that will give a perfect linear prediction for the value at the node. The negatives of these values are used as the SAR weights. Specifically let w_k be the weights and Y_k the values of the field at the these locations, and Y_0 the value at the node. Then by construction

$$Y_0 - sumw_kY_k = 0$$

for any field that is linear in the tangent plane to the node. Specifically in the function `LKrigSAR.LKSphere` the weights follow the code fragment

```
x1<- grid3d[J,]
    x0<- grid3d[I,]
    u<- projectionSphere( x0,x1)
 # u are local 2 d coordinates on tangent plane to sphere at x0
 # x0 projects to (0,0)
   X<- cbind( rep( 1,nJ), u )
 # find weights to predict a linear function
 # at node from the nearest neighbors.
   W<- c( (X)
```

`I` is the index of the node at lattice point `x0`, `J` is the index of the neighbors at lattice points `x1`, and `-W` the weights used in the SAR.

The basis functions have their default as using great circle distance to determine the values between the center and other points. See Radial.basis for an example of the basis functions. Because the distances between centers are not equal some adjustment is made to the `delta` parameter for the basis function support. Currently the number of basis functions in each level are

Level	1	2	3	4	5	6	7	8
Number Basis functions	12	42	162	642	2562	10242	40962	163842

So if `startingLevel=2` and `nlevel=3` there will be a total of `42 + 162 + 642 = 846` basis functions in the model if the spatial domain includes the entire sphere.

`LKRing` This model follows the Mercator projection for a sphere where longitude and latitude are treated as Euclidean coordinates except that longitude is periodic. So the actual coordinates represent the surface of cylinder which is one way of visualizing the Mercator projection. To keep things simple the first coordinate is essentially hardwired to be in the scale of degrees (sorry for all you fans of radians) and wrapping 0 to 360 (or periodic in [-180,180]). It is important to scale the second coordinate in this geometry to be comparable in spatial scale to degrees (use the `V` argument in LKrigSetup). However, if the second coordinate can be interpreted as a latitude it is often reasonable to assume the spatial scales are the same in these two coordinates.

Note this geometry can also be used to represent an equatorial section of a spherical volume. Here the first coordinate is longitude but the second can be interpreted as a radius from the sphere's

center. This is a case where care needs to taken to scale the radial component to make sense with the degrees in the first.

`LKCylinder` This is just the three dimensional extension of LKRing with the first coordinate being periodic in (0,360) and the remaining two treated as Euclidean

The periodicity in the first coordinate is implemented in two places. First in setting up the spatial autoregression (SAR) weights, the weights reflect the wrapping. Second in finding distances between coordinates the nodes in the lattice has the first coordinate tagged as being periodic. Specifically in LKrigSetupLattice the gridList for each lattice has an attribute vector that indicates which coordinates are periodic. This information is used in the distance function LKrigDistance when called with arguments of a matrix and a gridList.

Why is this so complicated? This structure is designed around the fact that one can find the pairwise distance matrix quickly between an arbitrary set of locations and a rectangular grid (a gridList object in this package). The grid points within a delta radius of an arbitrary point can be found by simple arithmetic and indexing. Because these two geometries have regular lattice spacings is it useful to exploit this. See `LKrigDistance` for more details about the distance function.

Finally, we note that for just patches of the sphere one can use the usual LKRectangle geometry and change the distance function to either chordal or great circle distance. This gives a different approach to dealing with the inherent curvature but will be awkward as the domain is close to the poles.

Author(s)

Doug Nychka and Zachary Thomas

See Also

`LatticeKrig`, `LKrigSetup`, `LKrigSAR`, `LKrigLatticeCenters`

Examples

```
#
# fit the CO2 satellite data with a fixed lambda
# (but use a very small, unrealistic number of basis functions and levels so example
#  runs quickly)
 data(CO2)
# to look at raw data: quilt.plot(CO2$lon.lat, CO2$y, nx=288, ny=165)
# Use two levels starting at the second generation of lattice points from the triangulation
  LKinfo0<- LKrigSetup( CO2$lon.lat, startingLevel=1 ,nlevel=2,
                        a.wght=1.1, alpha=c(1,.25),
                        LKGeometry="LKSphere" )
# Take a look at Model summary
  print( LKinfo0)

# Use arbitrary  lambda  (.01)
  obj0<- LKrig( CO2$lon.lat,CO2$y,LKinfo=LKinfo0, lambda=0.01)
## Not run:
# Surface plot of estimate
  surface(obj0, nx=288, ny=165)
  world( add=TRUE)
```

```
## End(Not run)
## Not run:
data(CO2)
# estimate lambda ( should be around 0.003)
# NOTE: lambda values will tend to be sensitive to the model choice
  LKinfo0<- LKrigSetup( CO2$lon.lat, startingLevel=2 ,nlevel=2,
                        a.wght=1.1, alpha=c(1,.25),
                        LKGeometry="LKSphere")
  obj0B<-  LatticeKrig( CO2$lon.lat,CO2$y,LKinfo=LKinfo0)
 surface( obj0B, col=terrain.colors(256))
 world( add=TRUE, col="magenta")

# use chordal distance
LKinfo1<- LKrigSetup( CO2$lon.lat, startingLevel=2 ,nlevel=2,
                        a.wght=1.1, alpha=c(1,.25),
                        LKGeometry="LKSphere", distance.type="Chordal")

 obj0C<-  LatticeKrig( CO2$lon.lat,CO2$y,LKinfo=LKinfo1)
 surface( obj0C, col=terrain.colors(256))
 world( add=TRUE, col="magenta")

# a more serious model this uses about 3300 basis functions
LKinfo0<- LKrigSetup( CO2$lon.lat, startingLevel=3, ,nlevel=3,
                        a.wght=1.1, alpha=c(1, .5, .25),
                        LKGeometry="LKSphere" )

obj0B<-  LatticeKrig( CO2$lon.lat,CO2$y,LKinfo=LKinfo0)
# takes about 1 minute on a Macbook air
# setting findAwght = TRUE  takes about 8 minutes with
# lambda = 1.737 and a.wght = 15.8

## End(Not run)
#####################################
# The ring geometry
#####################################
## Not run:
  data(CO2)
  LKinfo1<- LKrigSetup(CO2$lon.lat, NC=8 ,nlevel=1, lambda=.2,
                        a.wght=5, alpha=1,
                        LKGeometry="LKRing" )
  obj1<- LatticeKrig( CO2$lon.lat,CO2$y,LKinfo=LKinfo1)
# take a look:
surface( obj1)
world( add=TRUE)

## End(Not run)
# compare to fitting without wrapping:
## Not run:
  LKinfo2<- LKrigSetup(CO2$lon.lat, NC=8 ,nlevel=1,
                   lambda=.2, a.wght=5, alpha=1 )
  obj2<- LKrig( CO2$lon.lat,CO2$y,LKinfo=LKinfo2)
 # NOTE: not periodic in longitude:
 surface(obj2)
```

```
## End(Not run)

# a synthetic example and larger example
## Not run:
 set.seed(124)
 N<- 1e4
  x0<- matrix( rnorm(3*N), ncol=3)
  x0<- x0/ sqrt( rowSums( x0^2))

  x<-  toSphere( x0 )

# the true function for testing -- a bump at the direction alpha
  fun<- function(X){
    alpha<-  c( .1,.1,1)
    alpha<- alpha/ sqrt( sum( alpha^2))
    4*( 1 + c(( X)%*%alpha) )^2
  }

  ytrue <- fun(x0)
  y<- ytrue + .05*rnorm( length(ytrue))
# this defines about 3300 basis functions
  LKinfo1<- LKrigSetup( x,
                        startingLevel=3,
                        LKGeometry="LKSphere",
                        a.wght=1.01,
                        nlevel=3, alpha = c(1.0,.5,.25)^2,
                        choleskyMemory=list(nnzR= 20E6),
                        normalize=TRUE)
  out<- LatticeKrig( x,y, LKinfo=LKinfo1, lambda=.01)
surface( out)

## End(Not run)
```

`VignetteExamples` *Data examples for the LatticeKrig Vignette*

Description

Data sets used for the R scripts in the vignette

Usage

```
data(EquatorData)
data(PolarData)
```

Format

See Section 4 in the LatticeKrig Vignette for the use of these data.

EquatorData: `equatorLocations` `equatorValues` `equatorGrid` `equatorGridValues`

PolarData: `polarLocations` `polarValues` `polarGrid` `polarGridValues`

Index