KriSp

An R Package for Covariance Tapered Kriging of Large Datasets Using Sparse Matrix Techniques

Tutorial

Reinhard Furrer
Mathematical and Computer Sciences Department
Colorado School of Mines
Golden, CO, 80401

rfurrer@mines.edu
November 21, 2006

1 Introduction

Interpolation of a spatially correlated random process is used in many scientific areas. The best unbiased linear predictor (BLUP), often called kriging predictor in geostatistics, requires the solution of a linear system based on the (estimated) covariance matrix of the observations. Frequently, the most interesting spatial problems involve large datasets and their analysis overwhelms traditional implementations of spatial statistics. Furrer et al. (2006) show that tapering the correct covariance matrix with an appropriate compactly supported covariance function reduces the computational burden significantly and still results in an asymptotic optimal mean squared error. The effect of tapering is to create a sparse approximate linear system that can then be solved using sparse matrix algorithms. This package provides a suite of functions for the R statistical computing software (Ihaka and Gentleman, 1996; R, 2004) to perform interpolation of large or even massive datasets using covariance tapering (R is an open source implementation of the S language, Chambers and Hastie, 1992; Chambers, 1998).

Throughout this document, packages, programs and external functions are written in sans serif font. R input commands, R functions and their arguments are typed in *`slanted typewriter font`*, corresponding output, if any, in `upright typewriter font`.

The package KriSp (Kriging with Sparse matrices) is considered as an add-on to the package fields. KriSp has a similar class to one of fields and uses some of its methods. Further, KriSp uses the package SparseM to handle the sparse matrix techniques. Thus the package is not exhaustive in its functionality compared to other geostatistical packages like geoR. Also, most functions are not fully optimized in order to enhance readability of the code.

The reader should be familiar with R as well as with standard geostatistical terms and modeling (see for instance Cressie, 1993 or Stein, 1999, for a detailed discussion on that topic). Insight of the packages fields and SparseM is beneficial but not mandatory. This tutorial should give insight in how to use the different functions and methods of KriSp. Typically, default arguments are used for the function calls. The user is strongly encouraged to examine the function arguments using the the R functions *`args`* and *`help`* (the help files of major functions included in KriSp are given in the Appendix). The commands used in this tutorial are also available at http://www.mines.edu/~rfurrer/software/KriSp/KriSp.tutorial.R. Please send any comments concerning this document or the package to `rfurrer@mines.edu`.

2 Getting Started

Download the package KriSp_0.4.tar.gz to your local file system and install it using the following command at your Unix prompt

```
$ R CMD INSTALL -l  /path/to/library  KriSp_0.4.tar.gz
```

For Windows, the equivalent command to be executed at the DOS prompt is

```
$ Rcmd INSTALL -l  /path/to/library  KriSp_0.4.tar.gz
```

Alternatively, the package can also be installed within an R session (see the R manual for more details). Start by opening an R session (version ≥1.7) and load the package

```
> library("KriSp", lib.loc="/path/to/library")
```

The required libraries fields and SparseM are automatically loaded, provided they have been previously installed.

The main class of KriSp is *sparse*. In order use the methods defined for the *Krig* class in fields, the class *sparse* has the secondary class *Krig*. The main functions of the package KriSP are

```
> Krig.simple.sparse()
> Krig.sparse()
> predict()
```

The first function performs simple kriging, the second universal kriging. The third function is a method to perform predictions on a given set of locations. In order to simplify the coding, the simple and the universal kriging approach have been separated. Both approaches are illustrated in the subsequent sections. In Section 4, the tapering technique is further illustrated with two examples.

3 Included Spatial Models

3.1 Spatial Model with Zero Mean

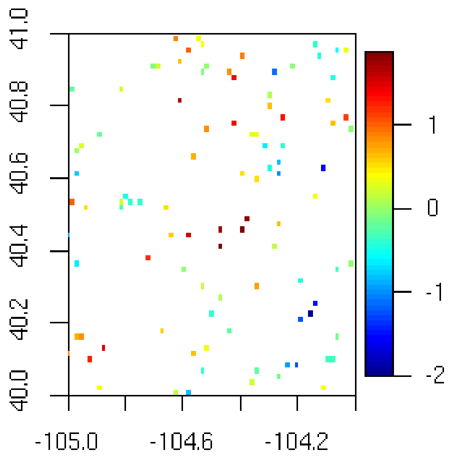
To illustrate the capacity of sparse matrix techniques, we start with the simple spatial model

$$Y(\boldsymbol{x}) = Z(\boldsymbol{x}) + \varepsilon(\boldsymbol{x}),$$

where Z is mean zero process with covariance function K and ε is a white noise with variance σ^2. In other words, we observe the process Z at some locations, say $\boldsymbol{x}_1, \dots, \boldsymbol{x}_n$, with a measurement error of variance σ^2. Then the best linear unbiased prediction (BLUP) of Z at an (unobserved) location $\boldsymbol{x}^*$ is then

$$\hat{Z}(\boldsymbol{x}^*) = \boldsymbol{c}^{*\mathsf{T}}(\mathbf{C} + \sigma^2\mathbf{I})^{-1}\mathbf{Z}, \tag{1}$$

where $\mathbf{Z} = \big(Z(\boldsymbol{x}_1), \dots, Z(\boldsymbol{x}_n)\big)^{\mathsf{T}}$, $\mathbf{C}_{ij} = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$, $\boldsymbol{c}_i^* = K(\boldsymbol{x}_i, \boldsymbol{x}^*)$ and $\mathbf{I}$ is the identity matrix. In geostatistical literature, (1) is referred to as simple kriging (*e.g.* Cressie, 1993).



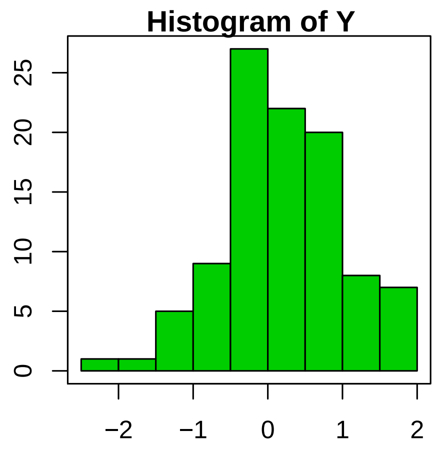


Figure 1: Dataset ***simple.data***: locations on the left, histogram of the observations on the right.

When predicting on many points, a fine regular grid, spatial field or lattice, the vector $\boldsymbol{c}^*$ in (1) is replaced by a matrix $\mathbf{C}^*$ containing as columns the respective vectors $\boldsymbol{c}^*$ for the different points on the grid. The functions ***Krig.simple.sparse*** and ***predict*** calulate the BLUP (1) for large datasets and large spatial fields.

There are a few datasets included in the package distribution. To illustrate the simple kriging approach, we use the dataset ***simple.data***. This artificial dataset consists of 100 locations randomly distributed in a unit longitude-latitude square. The spatial Gaussian process has an exponential covariance structure with a range of 10 miles and a sill of 1 and no measurement error. The data are loaded with

```
> data(simple)
> attach(simple.data)
```

Figure 1 shows the locations and a histogram of the values.

```
> look <- as.image( Y, x=x)
> image.plot( look)
> hist( Y)
```

Kriging in KriSp is performed by creating a ***sparse*** object with a call to a kriging function such as ***Krig.simple.sparse*** and a subsequent call to a prediction function like ***predict***.

```
> obj <- Krig.simple.sparse(x, Y)
```

The variable ***obj*** contains among other quantities the vector $(\mathbf{C} + \sigma^2\mathbf{I})^{-1}\mathbf{Z}$ and the predictions at the observed locations. As there is no measurement error, the predictions are identic to the observations. Prediction of the proces Z at an unobserved location $\boldsymbol{x}^* = (-105.5, 40.5)$ is obtained simply by

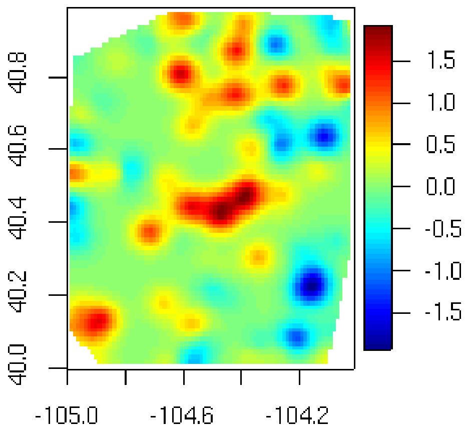


Figure 2: Predictions on a fine grid.

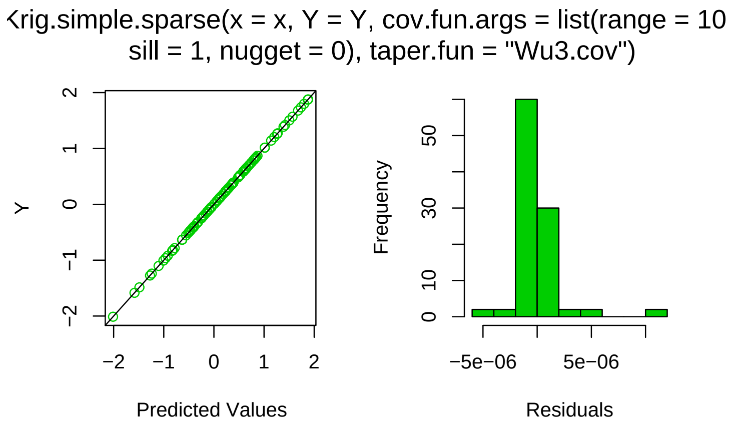


Figure 3: *plot* method for a *sparse* object.

```
> pre <- predict(obj, x=cbind(-105.5,40.5))
```

For prediction at several points, we just specify the arguments *x* with the locations in a $m \times 2$ matrix.

If we want to predict on a fine grid, we use fields function *predict.surface* (Figure 2).

```
> surf <- predict.surface( obj)
> image.plot( surf)
```

There exists a rudimentary *plott* and *print* methods for the *sparse* object (Figure 3).

```
> plot( obj)
> obj
```

```
Call:
  Krig.simple.sparse(x = x, Y = Y)

Covariance: expo.cov
Taper: Wu3.cov with range 10

Number of obs. = 100
```

The method ***summary*** is just a more extended version of ***print***. Other methods for the ***sparse*** class include ***residuals***, ***fitted*** and ***coef***. Those are included, as simple and universal kriging can be considered as a linear model.

KriSp is not about parameter estimation, thus we use our a priori knowledge for the parameter specification in the kriging routines. Typically, the covariance parameters are different to the default values. The exponential covariance and its parameters ***range=10***, ***sill=1*** and ***nugget=0*** are the default values for the argument ***cov.fun="expo.cov"*** and ***cov.fun.args*** respectively. We also used the default taper function and taper parameter. The previous ***Krig.simple.sparse*** is acutally identical to

```
> obj <- Krig.simple.sparse(x, Y, cov.fun = "expo.cov",
+       cov.fun.args = list(range = 10, sill = 1, nugget = 0),
+       taper.fun = "Wu3.cov", taper.fun.args = list(range = 50))
```

Note that the variance of the measurement error is given by the ***nugget*** argument in ***cov.fun.args***.

Too small values for the arguments ***tmpmax*** (working array for ***chol***) or ***nnzmax*** (upper bound of nonzero elements in $\mathbf{C}$) result in a warning or core dump respectively. Too big values do not hurt, just consume some computing time (see alse Section 4.2).

A little clean up, prior to the universal kriging example.

```
> detach(simple.data)
> rm("obj","pre","look","surf")
```

3.2 Spatial Model with a Drift or Trend

In this section we discuss an example where we have a spatial process of the form

$$U(\boldsymbol{x}) = \boldsymbol{m}(\boldsymbol{x})^{\mathsf{T}}\boldsymbol{\beta} + Z(\boldsymbol{x}), \tag{2}$$

where $\boldsymbol{m}$ is a known function in $\mathbb{R}^p$ and $\boldsymbol{\beta}$ is an unknown parameter in $\mathbb{R}^p$. Suppose we observe the process U at n locations $\boldsymbol{x}_1, \dots, \boldsymbol{x}_n$ with a mesurement error having variance σ^2. Similar to equation (1), the BLUP of $U(\boldsymbol{x}^*)$ is given by

$$\hat{U}(\boldsymbol{x}^*) = \boldsymbol{c}^{\mathsf{T}}(\mathbf{C} + \sigma^2\mathbf{I})^{-1}(\mathbf{Y} - \mathbf{M}\hat{\boldsymbol{\beta}}) + \boldsymbol{m}(\boldsymbol{x}_0)^{\mathsf{T}}\hat{\boldsymbol{\beta}}, \tag{3}$$

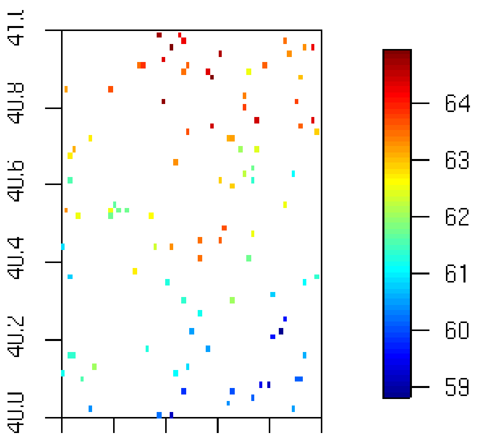


Figure 4: Dataset ***universal.data***.

where

$$\hat{\boldsymbol{\beta}} = (\mathbf{M}^\mathsf{T}(\mathbf{C}+\sigma^2\mathbf{I})^{-1}\mathbf{M})^{-1}\mathbf{M}^\mathsf{T}(\mathbf{C}+\sigma^2\mathbf{I})^{-1}\mathbf{Y} \tag{4}$$

with $\mathbf{M} = \big(\boldsymbol{m}(\boldsymbol{x}_1), \dots, \boldsymbol{m}(\boldsymbol{x}_n)\big)^\mathsf{T}$. In geostatistical literature, (3) is referred to as universal kriging. The sparse matrix approach is used with an iterative procedure illustrated as follows. We estimate the mean structure, *i.e.* the vector $\boldsymbol{\beta}$ in (2), via ordinary least squares (OLS), then $\mathbf{Y} - \mathbf{M}\hat{\boldsymbol{\beta}}^*$ is kriged yielding $\mathbf{Z}^*$. With OLS on $\mathbf{Y} - \mathbf{Z}^*$ we obtain a second estimate $\hat{\boldsymbol{\beta}}^*$ and so forth. This convenient back-fitting procedure converges to the BLUP and a few iterations usually suffice to obtain precise results. Note that (4) is the solution of the weighted least squares. If p is not too big, the BLUP could be also obtained by solving $p+2$ linear systems as given by equation (3) using a sparse approach.

The function Krig.sparse loops over the regression and simple kriging steps until convergence. Consider the example dataset ***universal.data***. The artificial data is similar to ***simple.data*** except that we have a linear trend (Figure 4).

```
> data(universal)
> attach(universal.data)
> look <- as.image( Y, x=x)
> image.plot( look)
```

We create a ***sparse*** object by calling

```
> obj <- Krig.sparse(x, Y,
+            cov.fun.args=list(range=10,sill=.9,nugget=.1))
```

The error norm between the consecutive coefficients $\hat{\boldsymbol{\beta}}^*$ are in ***out$coef***

The universal kriging object ***obj*** contains more information than in the simple kriging case.

```
> summary( obj)

Call:
  Krig.sparse(x = x, Y = Y,
    cov.fun.args = list(range = 10, sill = 0.9, nugget = 0.1))

Covariance: expo.cov
Taper: spher.cov with range 10

Number of obs. =100
nnz(sigma)     =820, (8.2%)
nnz(Chol sigma)=645, (6.45%)

Spatial trend:
  order m=2, betahat=(-0.5179,4.319,-166.8438)
  convergence with MSE=0.008 after 25 steps
  (criterion maxiter>=25, or MSE<0.001)

Residuals:
      Min        1Q    Median        3Q       Max
-0.168041 -0.045088  0.002130  0.047612  0.189869

Timing:
Covariance   Cholesky  Backsolve Iterations
      0.07       0.05       0.00       0.07
```

The predicted surface is in `out$fitted`, which is the sum of the trend, `out$trend`, and the spatial field, `out$spatial`.

As in the simple kriging case, the `predict` method returns a vector, the prediction on some specified locations. If `trend.only=TRUE`, `predict` returns the mean structure of the field only (Figure 5).

```
> image.plot( predict.surface(obj))
> image.plot( predict.surface(obj, trend.only=TRUE))
```

Finally, proper clean up.

```
> detach(universal.data)
> rm("obj","look","surf")
```

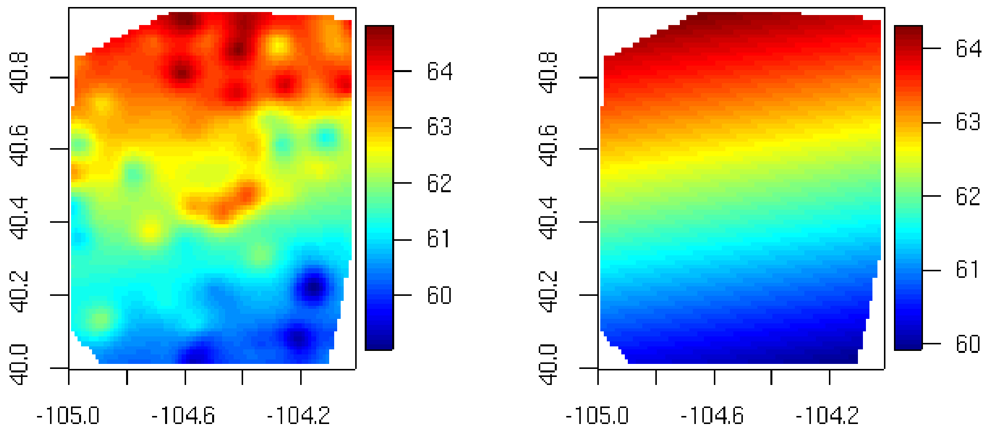


Figure 5: Prediction with universal kriging (left) and fitted trend structure (right) for `universal.data`.

4 Illustration of The Tapering Technique

4.1 Comparisation with Other Interpolation Methods

In this section we use the `universal` dataset to compare interpolation results obtained from different approaches, namely the straightforward naive approach (*i.e.* using equations (3) and (4)), fields' `Krig` and KriSps `Krig.sparse`. Refer to the fields demo for a detailed discussion of the use of `Krig` function.

To get started, we load the data and and set the covariance parameters..

```
> data(universal)
> attach(universal.data)
> range <- 10
> nugget <- 0.1
> sill <- 0.9
```

Prediction is performed on a 50×50 grid (depending on the available computing power, you might want to lower the grid size to 30×30).

```
> nx <- ny <- 50
> xgrid <- make.surface.grid( grid.list=list(lon='x',lat='y'),
+                               X=x, nx=nx, ny=ny)
```

To predict the surface using fields, we type

```
> out.fields <- Krig( x, Y, expo.earth.cov,
+                    sigma2=nugget, rho=sill, lambda=nugget/sill)
> fields <- predict.surface( out.fields, nx=nx, ny=ny)
```

The sparse approach (with taper range 30) is

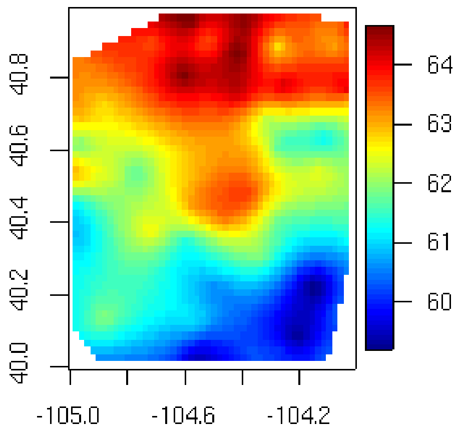
```
> out.sparse <- Krig.sparse( x, Y,cov.fun.args=list(range = range,
+           sill=sill, nugget=nugget),
+           taper.fun.args = list(range = 30),
+           scale.type ="range")
> sparse <- predict.surface( out.sparse, nx=nx, ny=ny)
```

We used the `scale.type` argument to transform the locations to obtain better numerical stability. The naive approach is somewhat longer to calculate.

```
> lagC <- rdist.earth(x)
> lagc <- rdist.earth(x, xgrid)
> C <- expo.cov( lagC, range=range, sill=sill, nugget=nugget)
> c <- expo.cov( lagc, range=range, sill=sill, nugget=nugget)
>
> invC <- solve(C)
> M <- out.sparse$xM
> scaledxgrid <- scale(xgrid, center = out.sparse$x.center,
+                        scale = out.sparse$x.scale)
>
> m0 <- t(cbind(scaledxgrid, 1))
>
> betahat <- solve(t(M) %*% invC %*% M)  %*% t(M) %*% invC  %*% Y
> naive <- c( t(c) %*%  invC %*% (Y-M%*%betahat )+t(m0)%*%betahat )
```

The results and the differences in the predicted fields are obtained with the following commands. (Figures 6 and 7).

```
> fields.naive <- fields
> fields.naive$z <- fields$z - array( naive, c(nx,ny))
> fields.sparse <- fields
> fields.sparse$z <- fields$z - sparse$z
> sparse.naive <- fields
> sparse.naive$z <- sparse$z - array(naive, c(nx,ny))
>
> image.plot( sparse)
> image.plot( fields)
>
> image.plot( fields.naive)
> image.plot( fields.sparse)
> image.plot( sparse.naive)
```



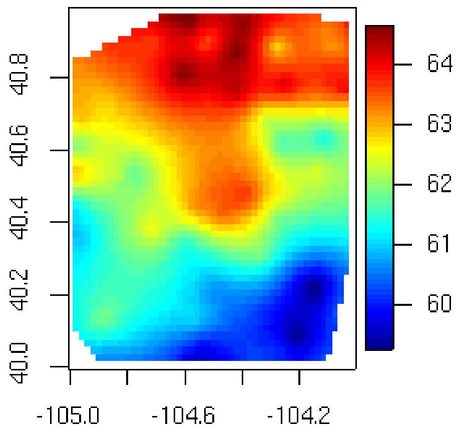


Figure 6: Predicted ozone on a 50 × 50 grid obtained with fields' *Krig* (left) and KriSps *Krig.sparse* (right).

The differences seem nevertheless remarkably high. With higher taper ranges, the difference between approaches KriSp and naive can be considerably lowered. Note the difference in the fitted coefficients for the mean structure.

```
> c(betahat)

[1] -0.1607431  4.0057863 60.2022324

> out.sparse$coef

[1] -0.5928071  4.2595088 60.3194296
```

For this illustrative example we used a small dataset such that the computing performance of KriSp is not as impressive as it could be.

4.2 Large Data

In this section we look at the large dataset (`anomaly.data`) to illustrate the capacity of KriSp. The data are anomalies of aggregated monthly precipitation for April 1948 at 11,918 stations in the US (for more details about the data refer to Johns *et al.*, 2003 and Furrer et al., 2006). We assume that the data is second order stationary. To simplify the document, we also suppose that the underlying isotropic structure is a mixture of two exponential covariance fucntions with range parameters of 40 and 520 miles with respective sill of 0.28 and 0.72.

```
> data(anomaly)
> attach(anomaly.data)
> US()
> points(x, pch=".", col=3)
```

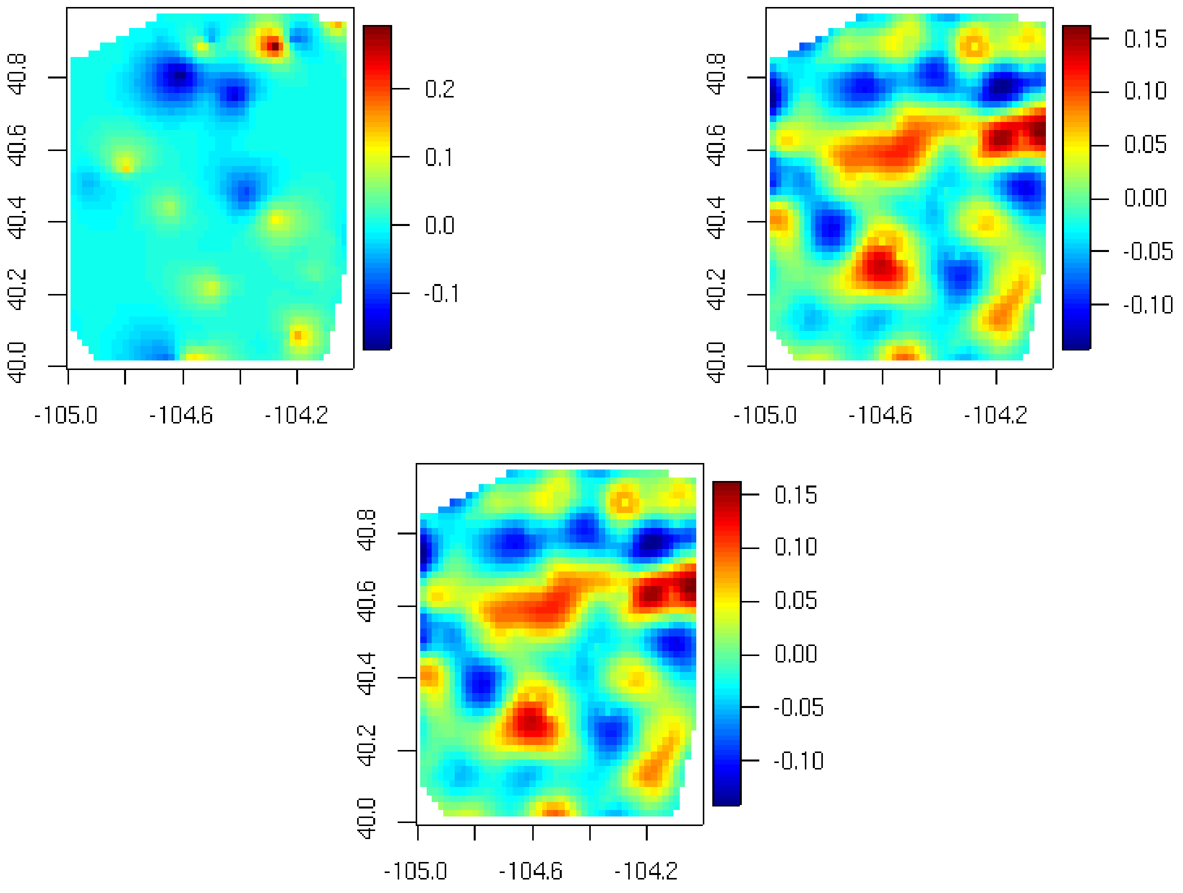


Figure 7: Differences in prediction between approaches fields and naive (left), between approaches fields and KriSp (right) and between approaches KriSp and naive (lower panel).

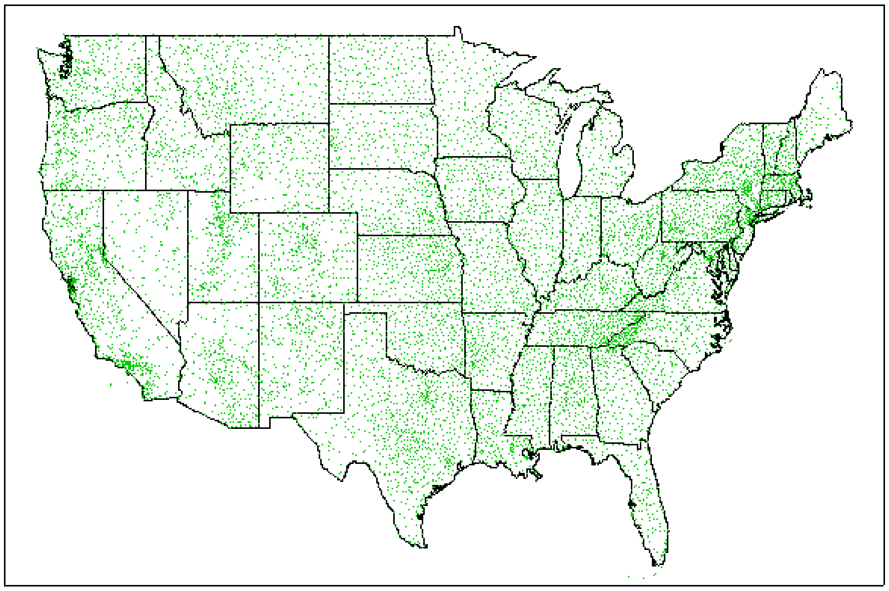


Figure 8: Measurement locations of precipitation anomalies for April 1948.

According to Furrer et al. (2006) a tapering radius with 16 to 24 points within the support is a sufficiently precise approximation. With our observation density, a tapering range of 40 miles is sufficient (*cf.* Figure 8 and 10). We first create the covariance function, being a mixture of two exponential ones. Typical KriSp covariance functions take the arguments ***distance***, ***range*** and ***eps***. If some are not used, just include the "dot" construction ... in the function definition. The mixture covariance could be defined by

```
> exp.mix.cov <- function(distance, ...)
+  0.28 * exp( -abs(distance)/40) + 0.72 * exp( -abs(distance)/520)
```

The ***sparse*** object is created with the following arguments.

```
> timing <- numeric(3)
> dummy <- gc()           # clean up before the heavy work...
> obj <- Krig.simple.sparse(x, Y, cov.fun = "exp.mix.cov",
+                         taper.fun.args = list(range=40), covfun = T,
+                         nnzmax = 320000, tmpmax = 12000)
> timing[1] <- (proc.time()-prtm)[1]
> summary( obj)
```

```
Call:
  Krig.simple.sparse(x = x, Y = Y, cov.fun = "exp.mix.cov",
    taper.fun.args = list(range = 40),
    nnzmax = 320000, tmpmax = 15000, covfun = T)

Covariance: exp.mix.cov
Taper: Wu3.cov with range 40

Number of obs. =11918
nnz(sigma)     =317832, (0.2238%)
nnz(Chol sigma)=720234, (0.5071%)


Residuals:
       Min         1Q     Median         3Q        Max
-2.146e-04 -2.220e-16  0.000e+00  2.220e-16  5.786e-05

Timing:
Covariance   Cholesky  Backsolve    Fitting
     21.38       1.50       0.03      21.06
```

Note that we had to increase the values of `nnzmax` and `tmpmax` to perform the kriging.

Be aware that the next step might be very time consuming. We perform a prediction and visualisation step (Figure 9).

```
> prtm <- proc.time()
> lon=seq( -126, -66, length=500)
> lat=seq( 24, 50, length=500)
> pred.surf <- predict.surface(obj,
+                  grid.list = list(lon=lon, lat=lat), extrap=T)
> timing[2] <- (proc.time()-prtm)[1]
>
> prtm <- proc.time()
> image.plot(pred.surf, xaxt="n", yaxt="n")
> US(add=T)
> timing[3] <- (proc.time()-prtm)[1]
> timing   # in seconds

[1]  43.99 308.37   0.98
```

(The calculations were performed on a Linux powered Xeon processor with 2Gbytes RAM.)

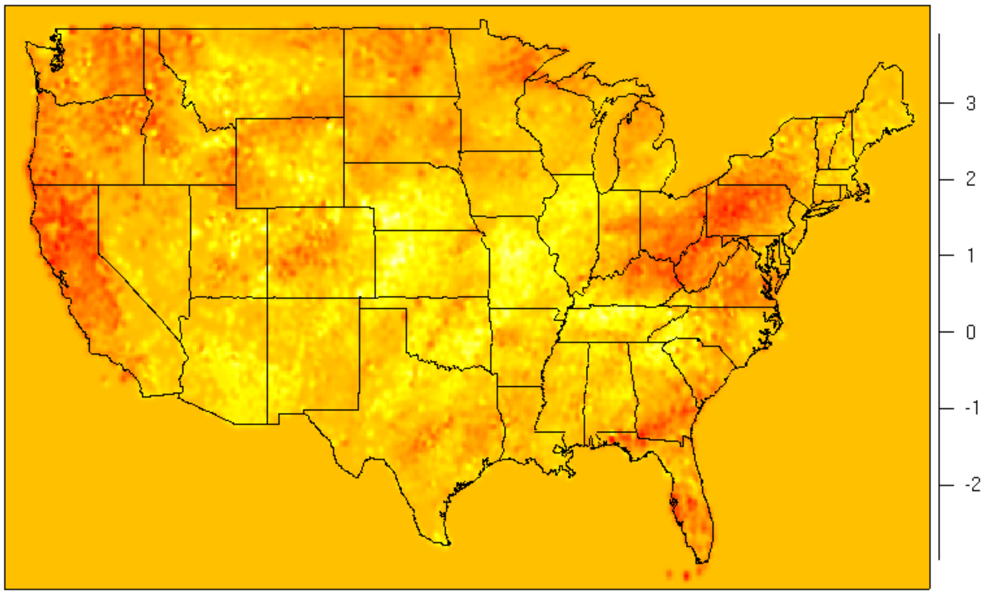


Figure 9: Predicted precipitation anomaly for April 1948 (500 × 350 grid).

Even if ***extrap*** or ***chull.mask*** is altered, the computation time is not reduced as fields' ***predict.surface*** calculates the values for the entire grid and sets the values not falling in the ***chull.mask*** region to ***NA*** afterwards.

For readers that are familiar with the storage structure of sparse matrices, the following lines create Figure 10.

```
> ia <- slot( obj$sigma, "ia")
> n.tap <- diff( ia) - 1
> hist( n.tap)
```

As the we have 11.913 observations, the full covariance matrix takes more than 1136 Mbytes compared to 3.86 Mbytes, if we have "typical" precision with 8-byte reals and 4-byte integers.

5 Computational Issues

The function ***Krig.simple.sparse*** consists basically in calculating the covariance matrix **C** with a Fortran subroutine, using the Cholesky factorization of SparseM and in performing a backsolve operation with the observations. The function ***Krig.sparse*** uses a backfitting approach that loops over a regression and a kriging step. The regression step, a simple ordinary least squares estimation, does not use sparse matrix techniques. The kriging step is essentially identical to a call to ***Krig.simple.sparse***.

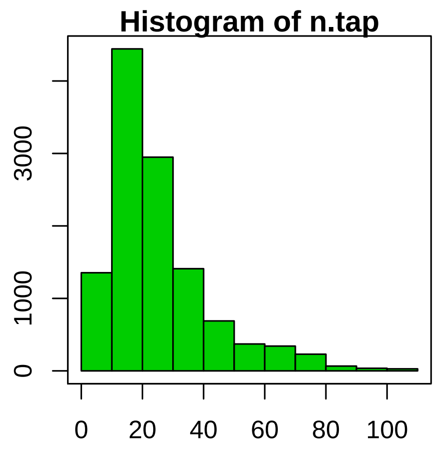


Figure 10: Histogram of the number of observations within the taper range. The median is 22 observations and the mean is slightly over 26. One location had just one, twelve locations had two observations within the range.

The method ***predict*** is essentially a wrapper to a Fortran function, which loops over all points at which to predict. For each point it calculates $\boldsymbol{c}^{*\mathsf{T}}\boldsymbol{v}$, where $\boldsymbol{v} = \mathbf{C}^{-1}\mathbf{Z}$ given by a ***sparse*** object. There are several ways to optimize this costly procedure:

- use a fast Fourier transform (FFT) approach similar to the function ***krig.image*** in fields.
- take account of some a priori knowledge of the locations and rewrite the Fortran functions. For example, if the points are on a regular grid, the operation can be done in (essentially) $O(N)$ instead of $O(N^2)$ operations (N the number of points to predict).

Evaluating the Matérn covariance function is computationally heavy. R essentially uses the Netlib function rkbesl.f. Instead of using the same or similar functions in the Fortran code of KriSp, a linear approximation scheme is used. All covariances are evaluated on a fine grid with R functions such as ***mater.cov***, ***expo.cov***, *etc.* Those function values are then passed to the corresponding Fortran routine in where a linear interpolation is made. Of course, this method could be refined to a cubic spline approximation (using fields' css.f) or some other interpolation scheme. This approach also eliminates the need of providing a large database of covariance functions coded in Fortran. Note that in Furrer et al. (2006), no covariance approximations were made. The the following lines display the logarithm of the maximum error of the linear covariance approximation as a function of the number of grid points (Figure 11).

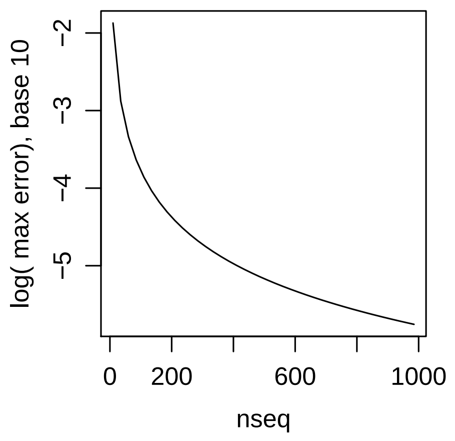


Figure 11: Error of the linear covariance approximation.

```
> nseq <- seq(10,1000,by=25)
> maxerr <- sapply(nseq, "covapprox.error")
> plot( nseq, log(unlist(maxerr[1,]),10), type="l",
+          ylab="log( max error), base 10")
```

The functions of KriSp are written in a linear, sequential way, calling as few functions as possible in order to save memory. Of course, there would be further gain in "unrolling" some of the remaining functions. When dealing with massive data sets, I recommend to use the source of KriSp and to hard code the functions to completely adjust it to the specific problem. In such cases, it is possible to work with datasets involving up to 400,000 locations (Sain and Furrer, 2004).

6 Outlook

The package KriSp should provide an insight on how to interpolate large or massive spatial datasets. Current work focuses on applying tapering techniques to microarray data and to maximum likelihood covariance parameter estimation.

A possible extension might be to uniquely use S4 classes and methods. As the current version of fields uses S3 classes, KriSp does not entirely use the new class concept.

The next major version of fields (version>3.2) will contain a sparse matrix module and extends the capacities of KriSp. Therefore, KriSp will no longer be significantly extended and future releases will most likely consist of bug fixes only.

7 Disclaimer

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

Acknowledgments

Many thanks to Doug Nychka, Stephen R. Sain, Tim Hoar and Eva Maria Furrer for many valuable remarks on the programming of the package and the writing of this document.

References

Chambers, J. M. (1998). *Programming with Data: A Guide to the S Language*. Springer-Verlag. 2

Chambers, J. M. and Hastie T. J. (1992). *Statistical Models in S*. Wadsworth and Crooks/Cole. 2

Cressie, N. A. C. (1993). *Statistics for Spatial Data*. John Wiley & Sons Inc., New York, revised reprint. 2, 3

Furrer, R., Genton, M. G., and Nychka, D. (2006). Covariance Tapering for Interpolation of Large Spatial Datasets. *Journal of Computational and Graphical Statistics*, **15**, 502–523. 2, 11, 13, 16

Ihaka, R. and Gentleman, R. (1996). R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, **5**, 299–314. 2

Johns, C., Nychka, D., Kittel, T., and Daly, C. (2003). Infilling sparse records of spatial fields. *Journal of the American Statistical Association*, **98**, 796–806. 11

R Development Core Team. (2004). R: A language and environment for statistical computing, R Foundation for Statistical Computing, Vienna, Austria, http://www.R-project.org. 2

Sain, S. and Furrer, R., (2004). Fitting Large-Scale Spatial Models with Applications to Microarray Data Analysis. Proceedings Interface 2004. 17

Stein, M. L. (1999). *Interpolation of Spatial Data*. Springer-Verlag, New York. 2

Appendix

`KriSp`	*Tools for interpolating large data sets*

Description

KriSp is a collection of functions for interpolationg large spatial datasets with covariance tapering.

Details

There are also generic functions that support these methods such as

`plot` - diagnostic plots of fit
`summary` - statistical summary of fit
`print` - shorter version of summary
`surface` - graphical display of fitted surface
`predict` - evaluation fit at arbitrary points

To get started, try some of the examples from help files for `KriSp`. See also the manual/tutorial at `http://www.mines.edu/~rfurrer/software/KriSp`.

The theoretical background can be found in the paper:

Furrer, R., Genton, M. G., and Nychka, D. (2006). Covariance Tapering for Interpolation of Large Spatial Datasets. Journal of Computational and Graphical Statistics, 15(3), 502–523.

The structure of the functions `Krig` in `fields` has changed drastically between versions 2.x and 3.x. To keep sample source code simple, we wrote a function `expo.earth.cov` that works like `exp.earth.cov` in versions 2.x but has the required functionality for higher versions.

The next major version of `fields` (version>3.2) will contain a sparse matrix module and extends the capacities of `KriSp`. Therefore, `KriSp` will no longer be significantly extended and future releases will most likely consist of bug fixes only.

Note

DISCLAIMER:

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

`KriSp.methods` *Accessing Sparse Kriging Fits*

Description

These functions are all `methods` for class `sparse` objects.

Usage

```
summary(object, ...)
print(x, digits = max(3, getOption("digits") - 3),...)

residuals(object, ...)
resid(object, ...)

fitted(object, ...)
fitted.values(object, ...)

coef(object, ...)
coefficients(object, ...)
```

Arguments

`object,x`	an object of class `sparse`, typically the result of a call to `Krig.sparse` or `Krig.simple.sparse`.
`digits`	a non-null value for `digits` specifies the minimum number of significant digits to be printed in values. If `digits` is `NULL`, the value of `digits` set by `options` is used.
`...`	further arguments passed to or from other methods.

See Also

`Krig.sparse`, `Krig.simple.sparse`.

`Krig.simple.sparse` *Kriging surface estimate*

Description

Calculating the BLUP or kriging estimate for large two dimensional spatial datasets.

Usage

```
Krig.simple.sparse(x,Y,
        cov.fun = "expo.cov",
            cov.fun.args = list(range = 10, eps = eps),
        taper.fun = "spher.cov",
            taper.fun.args = list(range = 10),
        ncov = 10000, approxhmax = taper.fun.args$range,
        miles = TRUE, R = NULL, eps = 1e-5,
        save.sigma = TRUE, save.chol = FALSE,
        verbose = FALSE, nnzmax = 1e+05, tmpmax = 10000, ...)
```

Arguments

`x`	a `m` times 2 matrix containing the locations.
`Y`	the observed values at `x`.
`cov.fun`	Covariance function in the form of an R function, or its name as a string.
`cov.fun.args`	A list with the arguments to call the covariance function (in addition to the locations).
`taper.fun`	Taper function in the form of an R function, or its name as a string.
`taper.fun.args`	A list with the arguments to call the taper function (in addition to the locations).
`ncov`	Number of knots to evaluate the approximation
`approxhmax`	Maximum distance over which the covariance is approximated
`miles`	logical. If `TRUE` (default) distances are in statute miles if `FALSE` distances in kilometers.

`R`	the radius to use for the sphere to find spherical distances. If `NULL` the radius is either in miles or kilometers of the earth depending on the values of the miles argument. If `R=1` then distances are in radians.
`eps`	small value, everything smaller is considered zero.
`save.sigma`	should the covariance matrix be saved (in `SparseM` format).
`save.chol`	should the Cholesky factor of the covariance matrix be saved (in `SparseM` format).
`verbose`	should timing and convergence results be printed.
`nnzmax`	upper bound of non-zero elements in the covariance matrix.
`tmpmax`	working array for the Cholesky factorisation
`...`	supplementary parameters that can be given as arguments to the function `chol`.

Details

For computational reasons, we do not call simple the `solve` function but use `chol(...)` and `backsolve(...)` form the `SparseM` library. We do not allow missing values. We only consider two-dimensional domains.

Value

`Krig.simple.sparse` returns an object of `class c("sparse","Krig")`. The second is to reuse many handy functions of the library `fields`.

An object of the class `"sparse"` is a list containing at least the following components:

`call`	the matched call.
`fitted`	fitted values at observed values.
`solve`	vector used for prediction on other grid points.
`sigma`	if requested, the covariance matrix.
`sigmachol`	if requested, the Cholesky factor.
`timing`	time needed for the main calculations.
`nnz`	The number of nonzero elements in the covariance matrix and its Cholesky factor.

Additionally, most input arguments are passed to the object.

Note

For REALLY big datasets, it would be wise to dissect the functions.

The radius of the earth is assumed to be 3963.34 miles or 6378.388 kilometers.

`approxhmax` should be at least as big as the taper range or the domain of the field.

If `nnzmax` is too small, R may produce a 'core dumped'.

See Also

`Krig.simple.sparse`, `predict.sparse`, `plot.sparse`;

`chol` and `backsolve` from the `SparseM` library.

Examples

```
data(simple)
attach(simple.data)

obj <- Krig.simple.sparse( x, Y)
pre <- predict( obj, x=cbind(-104.5,40.5))

surf <- predict.surface( obj)
image.plot( surf)
```

`Krig.sparse` *Kriging surface estimate*

Description

Calculating the BLUP or kriging estimate for large two dimensional spatial datasets.

Usage

```
Krig.sparse(x,Y,
        cov.fun="expo.cov", cov.fun.args=list(range=10,eps=eps),
        taper.fun="spher.cov", taper.fun.args=list(range=10),
        ncov=10000,approxhmax=taper.fun.args$range,
        miles=TRUE,R=NULL, eps=1e-5,
        xM=NULL, m=2,scale.type = "user",x.center = rep(0, ncol(x)),
        x.scale = rep(1, ncol(x)),
        maxiter=25,epsiter=1e-3,
        save.sigma=TRUE,save.chol=FALSE,verbose=FALSE,
        nnzmax=100000,tmpmax=10000,...)
```

Arguments

`x`	a `m` times 2 matrix containing the locations.
`Y`	the observed values at `x`.
`cov.fun`	Covariance function in the form of an R function, or its name as a string.
`cov.fun.args`	A list with the arguments to call the covariance function (in addition to the locations).
`taper.fun`	Taper function in the form of an R function, or its name as a string.
`taper.fun.args`	A list with the arguments to call the taper function (in addition to the locations).
`ncov`	Number of knots to evaluate the approximation
`approxhmax`	Maximum distance over which the covariance is approximated
`miles`	logical. If `TRUE` (default) distances are in statute miles if `FALSE` distances in kilometers.
`R`	the radius to use for the sphere to find spherical distances. If `NULL` the radius is either in miles or kilometers of the earth depending on the values of the miles argument. If `R=1` then distances are in radians.
`eps`	small value, everything smaller is considered zero.
`xM`	a `m` times 2 matrix containing the values for the spatial drift. By default, the locations are used.
`m`	A polynomial function of degree (m-1) will be included in the model as the spatial trend (drift) component.
`scale.type`	A character string among: `range`, `unit.sd`, `user`, `unscaled`. The independent variables are scaled to the specified type. See below.
`x.center`	Centering values to be subtracted from each column of the x matrix.
`x.scale`	Scale values that are divided into each column after centering.
`maxiter`	Maximum number of iterations used in the backfitting iteration
`epsiter`	Stop the backfitting as soon as the sum of squares of the coefficients of two consecutive iterations is smaller.
`save.sigma`	should the covariance matrix be saved (in `SparseM` format).

`save.chol`	should the Cholesky factor of the covariance matrix be saved (in `SparseM` format).
`verbose`	should timing and convergence results be printed.
`nnzmax`	upper bound of non-zero elements in the covariance matrix.
`tmpmax`	working array for the Cholesky factorisation
`...`	supplementary parameters that can be given as arguments to the function `chol`.

Details

For computational reasons, we do not call simple the `solve` function but use `chol(...)` and `backsolve(...)` form the `SparseM` library. We do not allow missing values. We only consider two-dimensional domains.

Concerning the scaling for the spatial trend. By default no scaling is done. Scale type of `range` scales the data to the interval (0,1) by forming `(x-min(x))/range(x)` for the x- and y-axis. Scale type of `unit.sd` subtracts the mean and divides by the standard deviation. Scale type of `user` allows specification of an `x.center` and `x.scale` by the user. The default for `user` is mean 0 and standard deviation 1. Scale type of `unscaled` does not scale the data.

Value

`Krig.sparse` returns an object of `class c("sparse","Krig")`. The second is to reuse many handy functions of the library `fields`.

An object of the class `"sparse"` is a list containing at least the following components:

`call`	the matched call.
`iternorm`	norm of coefficients for each iteration.
`trend`	fitted trend surface at observed values.
`coef`	coefficients of trend surface.
`spatial`	spatial part of surface at observed values.
`solve`	vector used for prediction on other grid points.
`sigma`	if requested, the covariance matrix.
`sigmachol`	if requested, the Cholesky factor.
`timing`	time needed for the main calculations.
`nnz`	The number of nonzero elements in the covariance matrix and its Cholesky factor.

Additionally, most input arguments are passed to the object.

Note

For REALLY big datasets, it would be wise to dissect the functions.

The radius of the earth is assumed to be 3963.34 miles or 6378.388 kilometers.

`approxhmax` should be at least as big as the taper range or the domain of the field.

If `nnzmax` is too small, R may produce a 'core dumped'.

See Also

`Krig.simple.sparse`, `predict.sparse`, `plot.sparse`;

`transformx` from the `fields` library; `chol` and `backsolve` from the `SparseM` library.

Examples

```
data(universal)
attach(universal.data)

obj <- Krig.sparse(x, Y,
          cov.fun.args=list(range=10,sill=.9,nugget=.1))

summary(obj)

image.plot( predict.surface(obj))
image.plot( predict.surface(obj, trend.only=TRUE))
```

`predict.sparse` *Evaluation of Krig spatial process estimate*

Description

Provides predictions from the spatial process estimate at arbitrary points

Usage

```
predict.sparse(object, x=NULL, trend.only=FALSE, ...)
```

Arguments

`object`	an object of class `sparse`, typically the result of a call to `Krig.sparse` or `Krig.simple.sparse`.
`x`	Matrix of x-values on which to evaluate the kriging surface. If omitted, the data x-values, i.e. `obj$x` will be used.
`trend.only`	for universal kriging, should only the trend be returned.
`...`	only for compatibility reasons.

Details

We evaluate the kriging surface on the given grid.

Value

Vector of predicted responses.

See Also

`Krig.sparse`, `Krig.sparse`; `predict.surface` from the `fields` package.

Examples

```
data(universal)
attach(universal.data)

obj <- Krig.sparse(x, Y,
          cov.fun.args=list(range=10,sill=.9,nugget=.1))

print( predict( obj, x=cbind(-104.5,40.5)))

xgrid <- expand.grid( lon=seq(min(x[,1]), max(x[,1]), l=50),
                      lat=seq(min(x[,2]), max(x[,2]), l=50))

# older verstions of fields used:
#   xgrid <- make.surface.grid( grid.list=list(lon='x',lat='y'),
#                           X=x, nx=50, ny=50)

surf <- predict( obj, x=xgrid)

image.plot( predict.surface(obj))
image.plot( predict.surface(obj, trend.only=TRUE))
```

`plot.sparse`	*Diagnostic and summary plots of the Krig.sparse object*

Description

Plots a series of two diagnostic plots that summarize the fit from `Krig.sparse`.

Usage

```
plot.sparse(x, main=NA, which=c(TRUE,TRUE), graphics.reset=TRUE,
                ...)
```

Arguments

`x`	an object of class `sparse`, typically the result of a call to `Krig.sparse` or `Krig.simple.sparse`.
`main`	Title of the plot. Default is the function call.
`graphics.reset`	Reset to original graphics parameters after plotting. Default is `TRUE`.
`which`	A vector of 2 logical values. Controls which of the two graphs to plot.
`...`	Optional graphics arguments to pass to each plot.

Details

This function creates two summary plots of the `sparse` object. The default is to put these in a 1 times 2 panel. However, if the screen is already divided in some other fashion the plots will just be added according to that scheme. This option is useful to compare to compare several different model fits.

The first is a scatterplot of predicted value against observed.

The second plot is a histogram of the residuals.

See Also

`Krig.sparse`, `summary.sparse` and `plot.Krig` from the `fields` library.

Examples

```
data(universal)
attach(universal.data)
obj <- Krig.sparse(x, Y)
fit <- predict(obj)
# fitting a surface to ozone measurements
plot(fit)
```

`covapprox.error` *Error of linear covariance approximation*

Description

Evaluates different error measures of the linear covariance approximation used in the kriging approach

Usage

```
covapprox.error(n, cov.fun='expo.cov', cov.fun.args=list(range=1),
          taper.fun='spher.cov', taper.fun.args=list(range=1),
          hmax=taper.fun.args$range,nres=25)
```

Arguments

`n`	Number of knots to evaluate the approximation
`cov.fun`	Covariance function in the form of an R function, or its name as a string.
`cov.fun.args`	A list with the arguments to call the covariance function (in addition to the locations).
`taper.fun`	Taper function in the form of an R function, or its name as a string.
`taper.fun.args`	A list with the arguments to call the taper function (in addition to the locations).
`hmax`	Distance over which the error is calculated.
`nres`	Resolution over which the errors are calculated.

Details

`n=1000` usually gives a maximum error smaller than 0.1 percent of the total sill.

Value

A list with the elements:

`max`	Maximum of the error.
`ise`	Approximation of the integrated squared error.
`iae`	Approximation of the integrated absolute error.
`covapprox`	Linear approximation of the covariance.
`error`	Error committed using the linear interpolation.

Note

To obtain a 'small' tapering effect, the taper range can be set to a large value. In such a case, `hmax` should be set to the diameter of the domain.

Using the `tophat` function as taper, no tapering is done.

See Also

Covariance functions such as `expo.cov`, `mater.cov`, etc.

`Krig.sparse` and `Krig.simple.sparse`.

Examples

```
# plot the error using the default functions.
nres <- 10
n <- 25
h <- seq(0,to=1, l=n*nres)
plot( h, covapprox.error(n,nres=nres)$error, type='l')

# evaluate error for a covariance only.
expoapprox <- covapprox.error(1000,taper.fun=tophat,
               taper.fun.args=list(range=5))
# all values are negative, as expo.cov is convex.
```

covariance *Theoretical Distance Based Covariance Functions*

Description

Computes theoretical covariance function at supplied distance values. Models include exponential, spherical, Matern and compactly supported covariances.

Usage

```
spher.cov(distance, range, sill=1, nugget=0, eps=1.0e-7,...)
expo.cov(distance, range, sill=1, nugget=0, effect=FALSE, eps=1.0e-7,...)
mater.cov(distance, smooth, range, sill=1, nugget=0, eps=1.0e-7,...)
tri.cov(distance, range, sill=1, nugget=0, eps=1.0e-7,...)
Wu1.cov(distance, range, sill=1, nugget=0, eps=1.0e-7,...)
Wu2.cov(distance, range, sill=1, nugget=0, eps=1.0e-7,...)
Wu3.cov(distance, range, sill=1, nugget=0, eps=1.0e-7,...)
tophat(distance, range, sill=1,...)
```

Arguments

`distance`	a vector/matrix of distances to compute the covariance for.
`range`	the range value.
`smooth`	smoothness of the Matern covariance.
`sill`	the partial sill value. The absolute sill (the variance or equivalently the covariance at distance zero) is `sill + nugget`.
`nugget`	the nugget effect.
`effect`	if `TRUE`, `range` is the practial range
`eps`	any distance less than it will be set to `nugget + sill`.
`...`	see below.

Details

The tophat is NOT an actual covariance function. It is included for illustration purposes only.

The triangular, spherical and the Wu type functions are compactly supported covariance functions, vanishing beyond the range. The triangular and the first Wu type are not valid in two dimensions and up.

Value

a vector/matrix of covariance values at the supplied distances.

Note

To all the covariance functions the `...` argument is added to ensure compatibility between different types of covariance functions. Although this would not be necessary, it simplifies internal coding and usage considerably.

There is a difference between `exp.cov` and `expo.cov`, as well as `matern.cov` and `mater.cov` functions in `fields` and `KriSp`.

Examples

```
distance <- seq(0,2,length=150)
plot( distance, mater.cov(distance,smooth=1,range=.4,sill=.8,nugget=.2))
```

`nnz` *Nonzero elements of a sparse matrix*

Description

Returns the nonzero elements of a sparse matrix.

Usage

```
nnz(x)
```

Arguments

`x`	matrix of class `matrix.csr`.

Value

nonzero elements of a sparse matrix `x`.

See Also

`as.matrix.csr` from the `SparseM` library.

Examples

```
nnz( as.matrix.csr( diag( 5)))
```

`cite.KriSp` *Citing Package KriSp in Publications*

Description

How to cite the package `KriSp` in publications.

Usage

```
cite.KriSp()
```

Details

Execute function `cite.KriSp()` for information on how to cite KriSp in publications.

Examples

```
cite.KriSp()
```

`datasets` *Artificial datasets used in the tutorial*

Description

The `simple.data` and `universal.data` are artifical datasets to illustrate the simple and universal kriging in the tutorial.

Usage

```
data(simple)
data(universal)
```

Format

A list containing the two components. `x`: 100 longitude-latitude position of locations, `Y`: the simulated values.

See Also

`anomaly.data`.

Examples

```
library(mvtnorm)

# simple.data:
set.seed(15)
n <- 100
x <- round(cbind(lon=runif(n)-105,lat=runif(n)+40),3)

lags <- rdist.earth(x)
C <- expo.cov( lags, range=10, sill=1)

Y <- round(c(rmvnorm(1,sigma=C)), 3)
simple.data <- list(x=x,Y=Y)

# universal.data:
set.seed(15)
n <- 100
x <- round(cbind(lon=runif(n)-105,lat=runif(n)+40),3)

lagC <- rdist.earth(x)
C <- expo.cov( lagC, range=10, sill=0.9, nugget=0.1)

Y <- round(c(rmvnorm(1,sigma=C))+4*x[,2]-100, 3)
universal.data <- list(x=x,Y=Y)
```

`anomaly` *Precipitation anomalies of April 1948 in the US*

Description

The data are anomalies of aggregated monthly precipitation for April 1948 at 11,918 stations in the US.

Usage

```
data(anomaly)
```

Format

A list containing the following components. `x`: longitude-latitude position of measurement locations, `Y`: are precipitation anomalies.

Source

http://www.image.ucar.edu/GSP/Data/US.monthly.met/

References

Johns, C., Nychka, D., Kittel, T., and Daly, C. (2003). Infilling sparse records of spatial fields. Journal of the American Statistical Association, 98, 796-806.

See Also

simple.data.

Examples

```
data(anomaly)
plot(anomaly.data$x)
```

`KriSp-internal` *KriSp internal and secondary functions*

Description

Listed below are supporting functions for KriSp.

Usage

```
distprep(distance, range, effect)

version.KriSp(verbose=TRUE)

predict.surface.se.sparse(object,...)
predict.se.sparse(object,...)

expo.earth.cov(x1, x2, theta = 1, C = NA)
```

Details

`distprep` is an auxiliary function used in the different covariance functions. `effect` can be used if ‘effective’ ranges should be considered.

As all `sparse` objects are also `Krig` objects, we need as many methods as defined for `Krig` in the `fields` library.

The structure of the functions `Krig` in `fields` has changed drastically between versions 2.x and 3.x. To keep sample source code simple, we wrote a function `expo.earth.cov` that works like `exp.earth.cov` in versions 2.x but has the required functionality for higher versions.

Index