`fields` vignette

Ashton Wiens, Mitchell Krock, Emma Lilly and Doug Nychka

14 September, 2021

Contents

1 `mKrig`

The `mKrig` function is both a Kriging/linear algebra engine and also user-level function for fitting spatial models. The `mKrig` engine is a simple and fast version of `Krig`. The “m” stands for micro, being a succinct (and we hope readable) function. The function focuses on the same computations as in `Krig.engine.fixed` done for a fixed `lambda` parameter, for unique spatial locations and for data without missing values. `mKrig` is optimized for large data sets, sparse linear algebra, and a clear exposition of the computations. The underlying code in `mKrig` is more readable and straightforward, but it does fewer checks on the data than `Krig`. One savings in computation is that Monte Carlo simulations are used to compute the trace of the smoothing matrix $\mathrm{tr}(A(\lambda))$, which can be used for the effective degrees of freedom of the model.

Just as in other functions, we can use `cov.function` and `cov.args` to specify which covariance we want to use in the model. `mKrig` is often used with a compactly supported covariance to take advantage of sparsity in computations. See `fastTps` as an example of how it uses `mKrig`.

There are other arguments that can be specified in `chol.args` for efficiency when dealing with a large data set. For example, `nnzR` is a guess at how many nonzero entries there are in the Cholesky decomposition of the linear system used to solve for the basis coefficients. Specifying this to increase size will help to avoid warnings from the `spam` package.

`mKrig.trace` is an internal function called by `mKrig` to estimate the effective degrees of freedom. The Kriging surface estimate at the data locations is a linear function of the data and can be represented as $A(\lambda)y$.

The trace of A is one useful measure of the effective degrees of freedom used in the surface representation. In particular this figures into the GCV estimate of the smoothing parameter. It is computationally intensive to find the trace explicitly, but there is a simple Monte Carlo estimate that is often very useful. If $\mathbf{E}$ is a vector of iid $N(0,1)$ random variables then the trace of A is the expected value of E^TAE. Note that AE is simply predicting a surface at the data location using the synthetic observation vector E. This is done for $N \cdot \mathrm{tr}(A)$ independent $N(0,1)$ vectors and the mean and standard deviation are reported in the `mKrig` summary. Typically as the number of observations is increased, this estimate becomse more accurate. If $N \cdot \mathrm{tr}(\mathrm{A})$ is as large as the number of observations np, then the algorithm switches to finding the trace exactly based on applying A to np unit vectors.

`mKrig` has been written to handle multiple data sets. Specifically, if the spatial model is the same except for different observed values (such as multiple days of observations), one can pass `y` as a matrix and the computations are done efficiently for each set. Note that this is not a multivariate spatial model just an efficient computation over several data vectors without explicit looping. See the example for the defaults.

<In the examples below we frequently use the S3 methods available to `mKrig` objects. `print`/`summary` and `surface` are used for model diagnostics and quick visualizaiton. `predict` and `predictSurface` can predict the model at the observations or an arbitrary region (derivatives supported with Wendland covariance). `predictSE` gives model uncertainty, and>

Finally, `sim.mKrig.approx` has the capability to quantify the uncertainty in the estimated function using conditional simulation.

Basic Usage

mKrig(x, y, Z = NULL, cov.function = "stationary.cov", lambda = 0, m = 2)

Value

Returns an object of class `mKrig`.

##ozone2

First we show that we can reproduce the `Krig` computations using `mKrig`. The algorithm/parameter estimation is done differently for the two functions, so this is a useful check. We fit an exponential covariance with range equal to 2 with both functions.

```
# Midwest ozone data 'day 16' stripped of missings
data( ozone2)
y<- ozone2$y[16,]
good<- !is.na( y)
y<-y[good]
x<- ozone2$lon.lat[good,]
out<- mKrig( x,y, aRange = 2.0, lambda=.01)
out2 <- Krig( x,y, aRange=2.0, lambda=.01)
grid.list <- fields.x.to.grid(x)
grid <- make.surface.grid(grid.list)
out.p <- matrix(predict(out,grid),nrow=80,ncol=80)
out.p2 <- matrix(predict(out2,grid),nrow=80,ncol=80)
test.for.zero(out.p, out.p2)
```

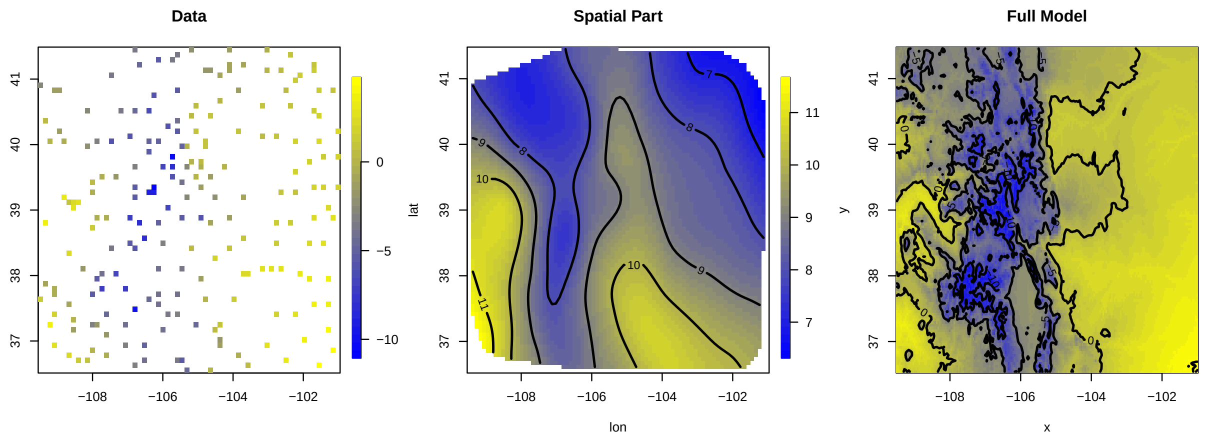
```
## PASSED test at tolerance  1e-08
```

##COmonthlyMet

We use this data set in almost every chapter for completeness. Here we quickly show how to use `mKrig` with a Z covariate.

```
data(COmonthlyMet)
yCO<- CO.tmin.MAM.climate
good<- !is.na(yCO)
yCO<-yCO[good]
xCO<- CO.loc[good,]
Z<- CO.elev[good]
out<- mKrig(xCO,yCO, Z=Z, cov.function="stationary.cov", Covariance="Matern", aRange=4.0, smoothness=1.
pred0 <- predict(out, grid.list=CO.Grid, Z= as.vector(CO.elevGrid$z))
pred <- as.surface(make.surface.grid(CO.Grid) , pred0)
```

```
set.panel(1,3)
quilt.plot(xCO, predict(out), main="Data",smallplot= c(.88,.9,0.2,.8),col=blue2yellow(35)) #colorRamps
surface(out, main="Spatial Part",smallplot= c(.88,.9,0.2,.8),col=blue2yellow(35))
surface(pred, main="Full Model",smallplot= c(.88,.9,0.2,.8),col=blue2yellow(35))
```

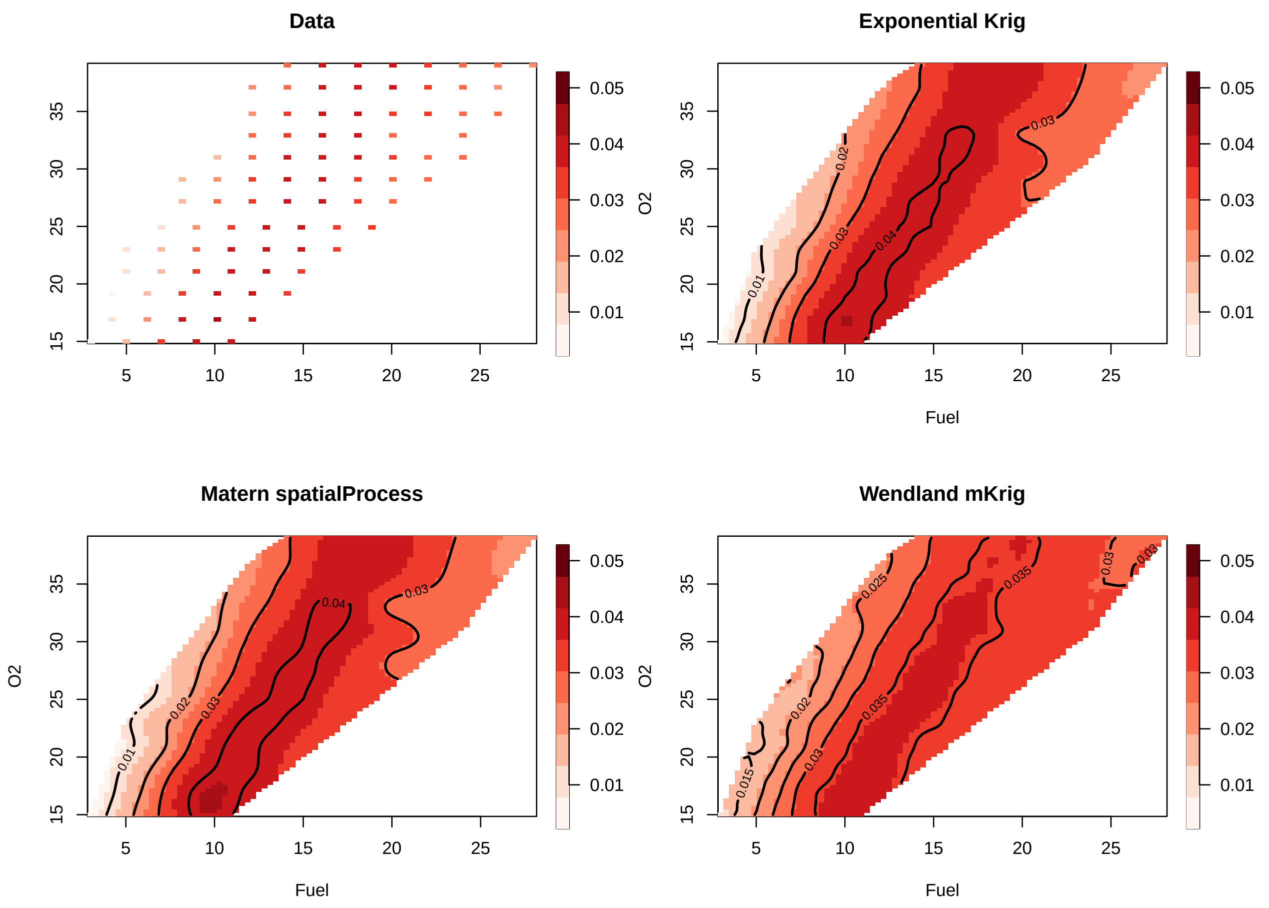


##flame

The `flame` data consists of a 2 column matrix `x` with different fuel and oxygen flow rates for the burner, and vector `y` with the intensity of light at a particular wavelength indicative of Lithium ions. The characteristics of an ionizing flame are varied with the intent of maximizing the intensity of emitted light for lithuim in solution. Areas outside of the measurements are where the mixture may explode! Note that the optimum is close to the boundary.

This example shows the versatility of `fields`, fitting `Krig`, `spatialProcess`, and `mKrig` models to the data. Note how much faster `mKrig` is, and how even with a Wendland it is able to reproduce the other fits' behaviours fairly well.

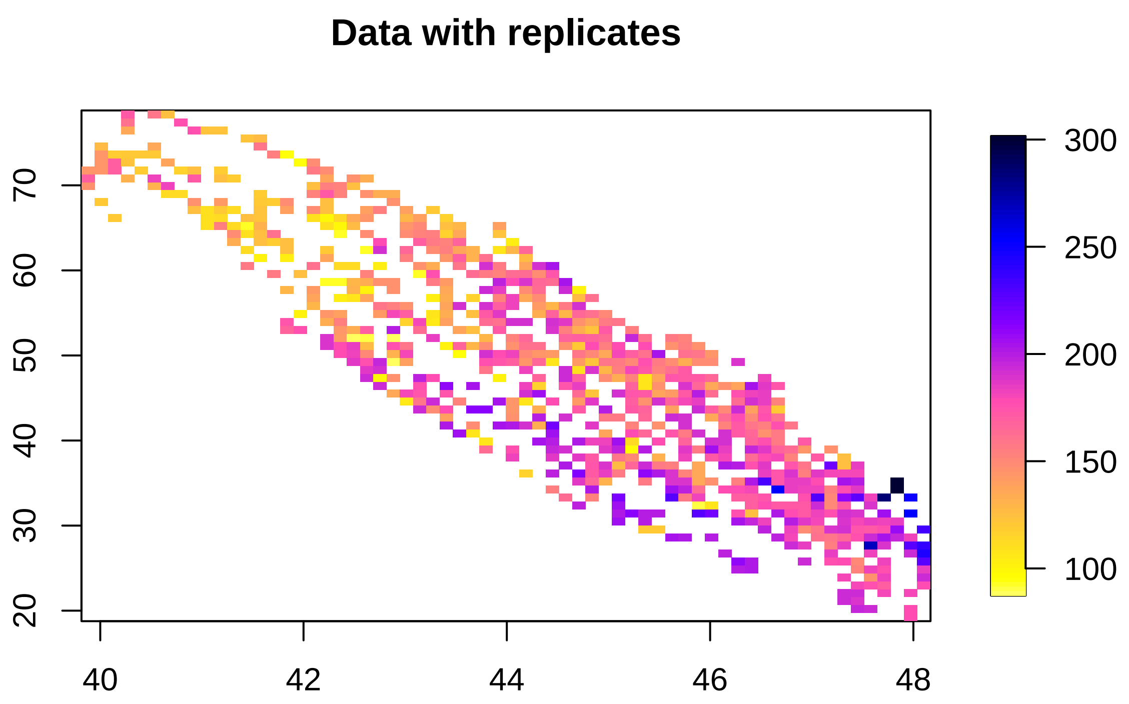
```
x <- flame$x
y <- flame$y
look <- as.image(Z=y, x=x)
fit <- Krig(x=x, Y=y, cov.function="stationary.cov", aRange=5)
fit1 <- spatialProcess(x,y)
fit2 <- mKrig(x, y, cov.function = "wendland.cov", k=2, aRange=4, lambda=0.75)
set.panel(2,2)
image.plot(look, main="Data",smallplot= c(.88,.9,0.2,.8),zlim=c(0.005,0.05),col=brewer.pal(9,'Reds'))
surface(fit, main="Exponential Krig",legend.shrink=0.6,smallplot= c(.88,.9,0.2,.8),zlim=c(0.005,0.05),c
surface(fit1, main="Matern spatialProcess",smallplot= c(.88,.9,0.2,.8),zlim=c(0.005,0.05),col=brewer.pal
surface(fit2, main="Wendland mKrig",smallplot= c(.88,.9,0.2,.8),zlim=c(0.005,0.05),col=brewer.pal(9,'Re
```



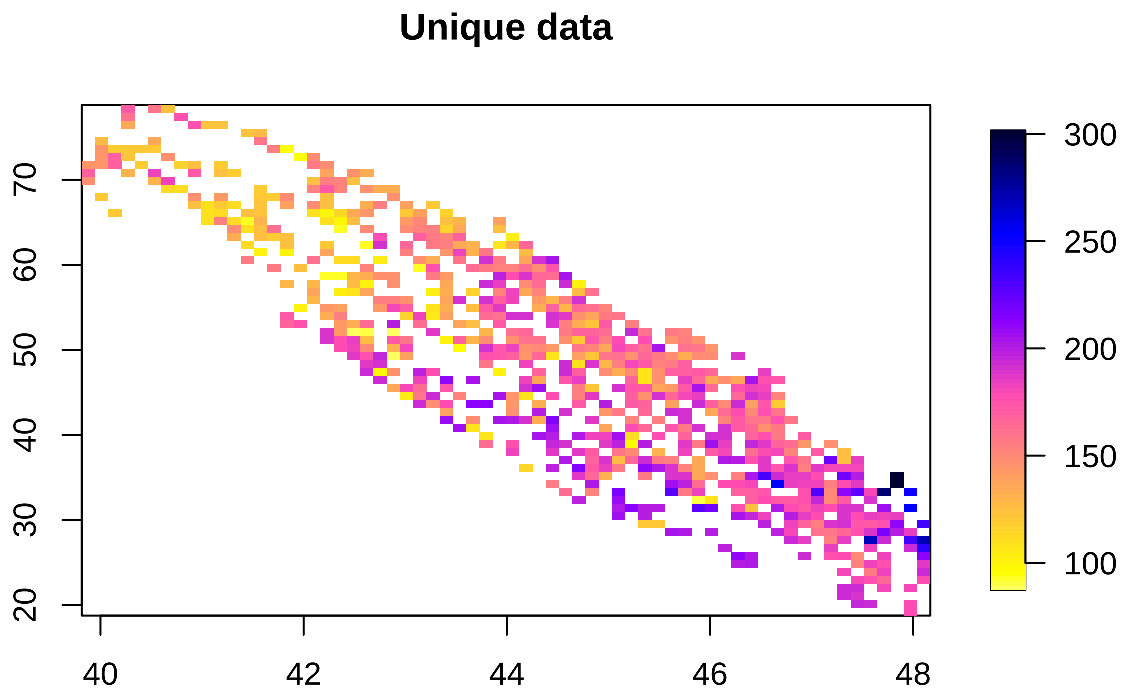
##Krig.replicates with NWSC

We now consider the `NWSC` data set. The data we consider are the temperature and relative humidity as our spatial indexing variables, and our response is power usage. The observations aren't at unique locations, so to use `mKrig` or `spatialProcess`, we will have to collapse the observations to unique locations with the function `Krig.replicates`.

```
library(sp) #for colors
load("NWSC.rda")
x <- cbind(NWSC$RH, NWSC$Otemp)
y <- NWSC$Mpower
quilt.plot(x,y, main="Data with replicates",col=rev(bpy.colors()))
```

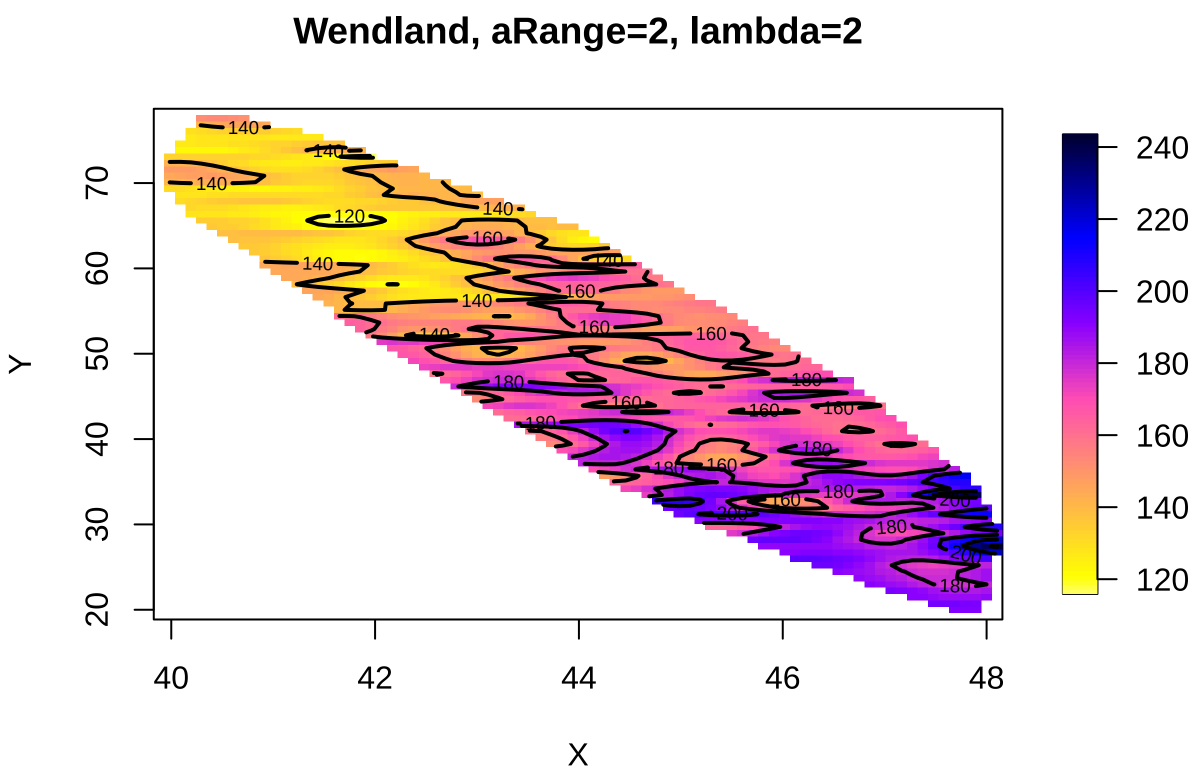


```
new.dat <- Krig.replicates(x=x, y=y)
xM <- new.dat$xM
yM <- new.dat$yM
quilt.plot(xM, yM, main="Unique data",col=rev(bpy.colors()))
```



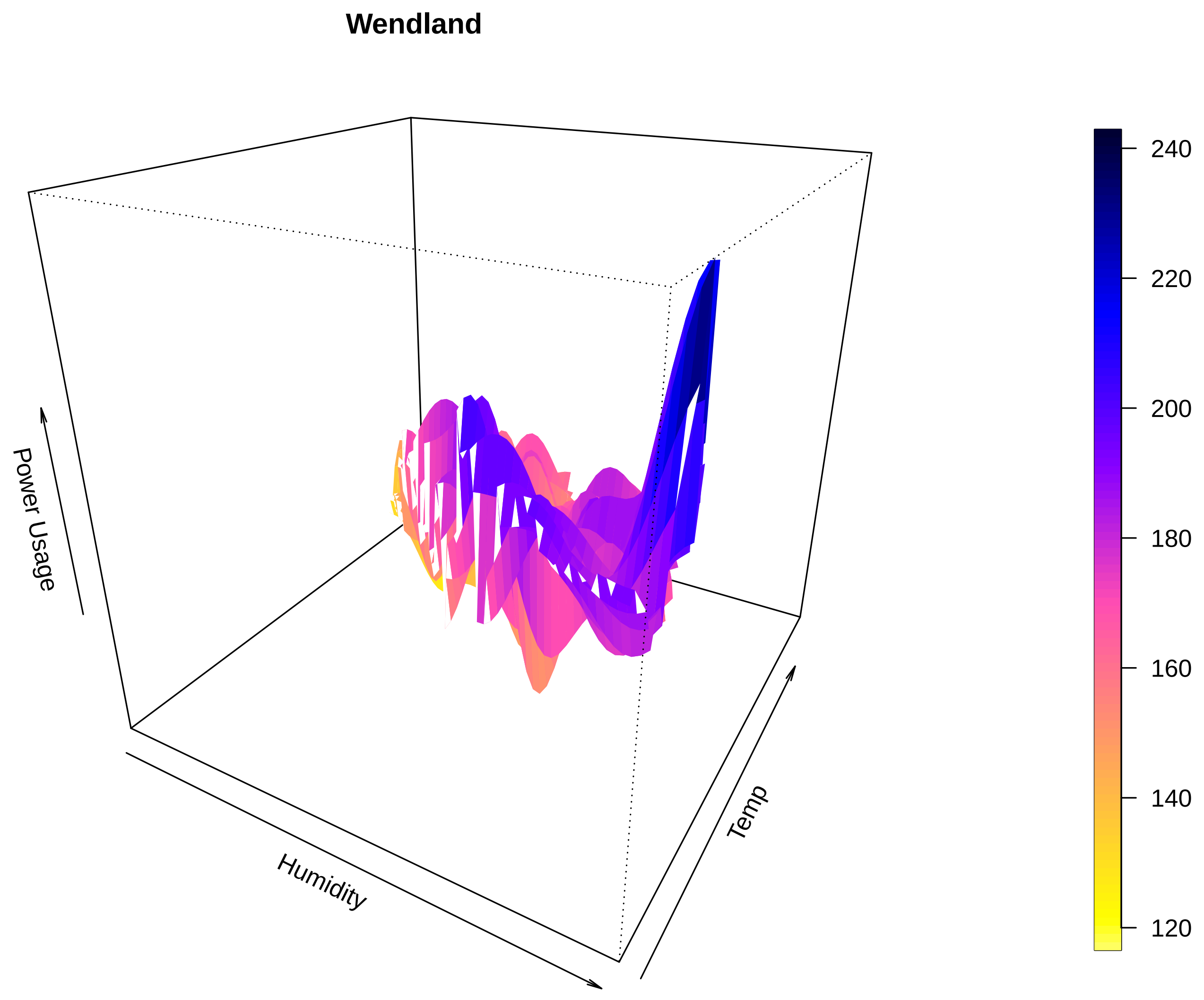
```
###fit <- spatialProcess(xM, yM) takes too long
fit <- mKrig(xM, yM, cov.function="wendland.cov", aRange=2,lambda=2)
#try different aRange and lambda
surface(fit, main="Wendland, aRange=2, lambda=2",
        col=rev(bpy.colors())
```

```
      )
```



```
pred <- predictSurface(fit)
```

```
set.panel()
library(sp) #for colors
drape.plot(pred, border=NA,  phi=25, horizontal=FALSE, xlab="Humidity", ylab="Temp", zlab="Power Usage"
title("Wendland          ")
```



##mKrig and a GCV grid with ozone2

We return to ozone2 for a final time. We use mKrig to interpolate using a tapered version of the exponential. The taper scale is set to 1.5, and the default taper covariance is the Wendland. Tapering will done at a scale of 1.5 relative to the scaling.

```
data( ozone2)
y<- ozone2$y[16,]
good<- !is.na( y)
y<-y[good]
x<- ozone2$lon.lat[good,]
out2 <- mKrig( x,y,cov.function="stationary.taper.cov",aRange = 2.0, lambda=.01,
       Taper="Wendland", Taper.args=list(aRange = 1.5, k=2, dimension=2))
```

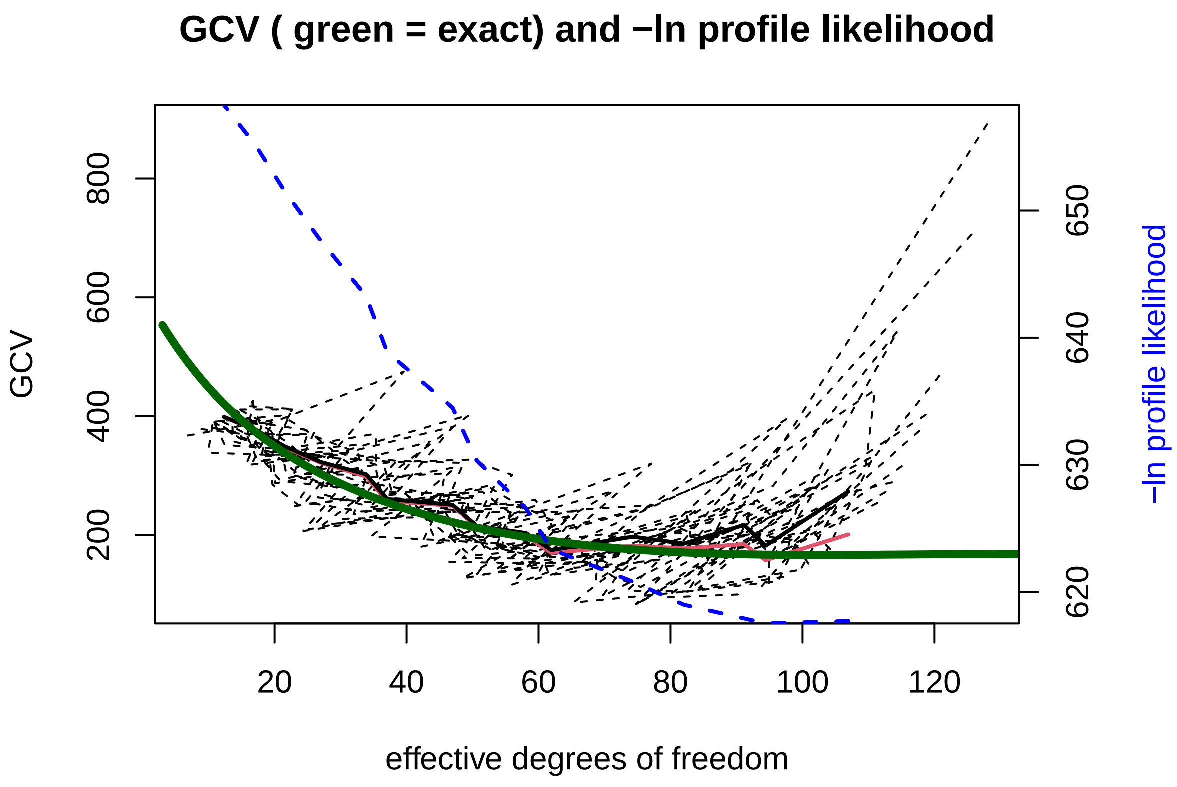
Now, we will compare many lambda's on a grid using GCV for this small data set. One should really just use Krig or Tps, but this is an example of approximate GCV that will work for much larger data sets using sparse covariances and the Monte Carlo trace estimate.

```
lgrid<- 10**seq(-1,1,,15)    # a grid of lambdas
GCV<- matrix( NA, 15,20)
trA<- matrix( NA, 15,20)
```

```
GCV.est<- rep( NA, 15)
eff.df<- rep( NA, 15)
logPL<- rep( NA, 15)
for( k in 1:15){    # loop over lambda's
  out<- mKrig( x,y,cov.function="stationary.taper.cov", aRange = 2.0,
               lambda=lgrid[k], Taper="Wendland", Taper.args=list(aRange = 1.5, k=2, dimension=2))
  GCV[k,]<- out$GCV.info
  trA[k,]<- out$trA.info
  eff.df[k]<- out$eff.df
  GCV.est[k]<- out$GCV
  logPL[k]<- out$summary["lnProfileLike.FULL"]
}
```

For each lambda, we have fitted an `mKrig` object and then extracted `trA` and `GCV`. Remember `trA` is the effective degrees of freedom `df` which is really `lambda` on another scale. The dashed black lines show the effective degrees of freedom versus the approximate Monte Carlo GCV in `GCV.info`. The average of these MC simulations is the red line. Note how close it is to the solid black line, which the Monte Carlo approximation automatically computed by `mKrig`. We fit a `Krig` object to the same data, which computes the exact GCV, which is shown in dark green. Finally, the negative log profile likelihood is shown in blue.

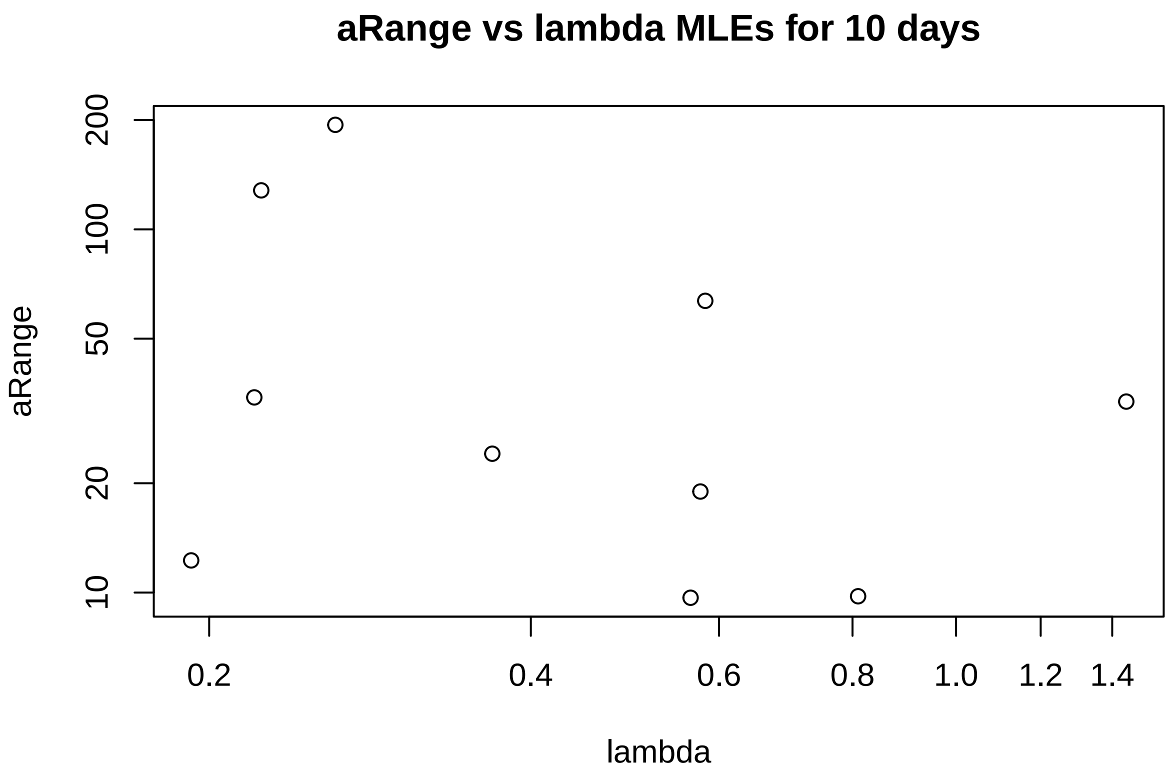
```
par(mar=c(5,4,4,6))
matplot( trA, GCV, type="l", col=1, lty=2,
         xlab="effective degrees of freedom", ylab="GCV")
lines( eff.df, GCV.est, lwd=2, col=2)
lines( eff.df, rowMeans(GCV), lwd=2)
# add exact GCV computed by Krig
out0<- Krig( x,y,cov.function="stationary.taper.cov", aRange = 2.0, Taper="Wendland",
             Taper.args=list(aRange = 1.5, k=2, dimension=2), spam.format=FALSE)
lines( out0$gcv.grid[,2:3], lwd=4, col="darkgreen")
# add profile likelihood
utemp<- par()$usr
utemp[3:4] <- range( -logPL)
par( usr=utemp)
lines( eff.df, -logPL, lwd=2, col="blue", lty=2)
axis( 4)
mtext( side=4,line=3, "-ln profile likelihood", col="blue")
title( "GCV ( green = exact) and -ln profile likelihood", cex=2)
```



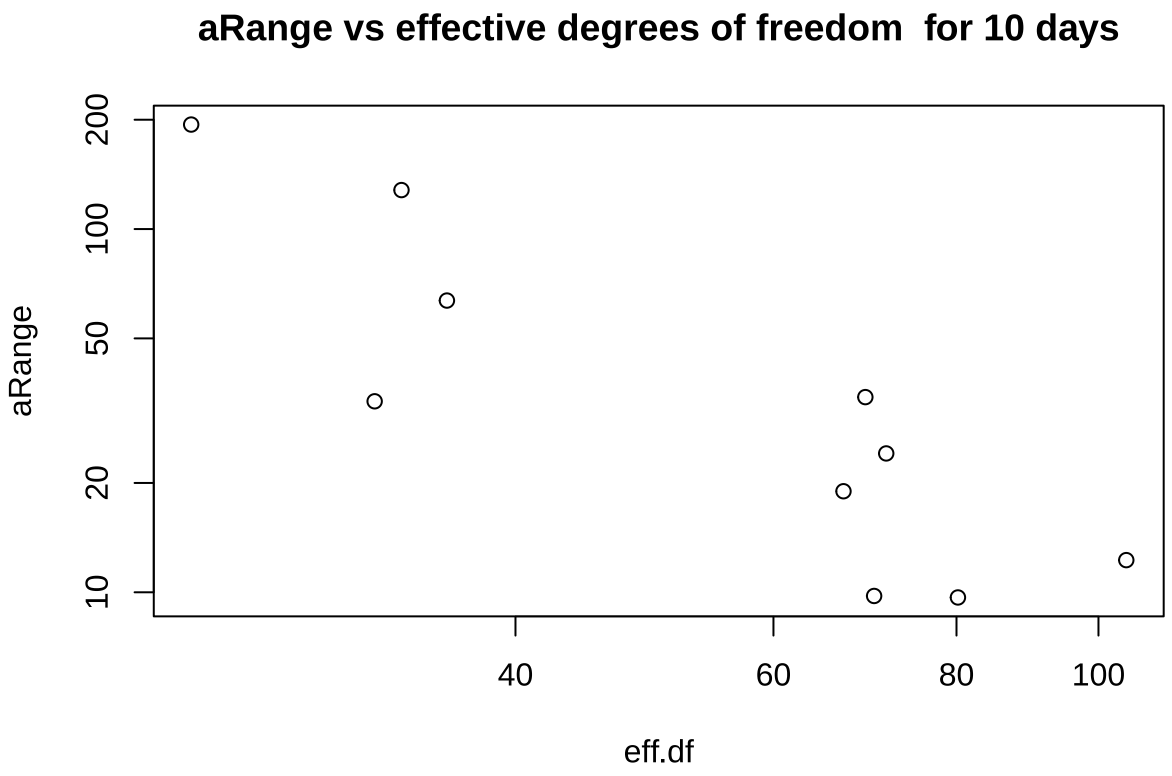
##Collapse fixed effects with ozone2

```
# fits for first 10 days from ozone data
data( ozone2)
NDays<- 10
O3MLE<-NULL
for( day in 1: NDays){
  ind<- !is.na(ozone2$y[day,] )
  x<- ozone2$lon.lat[ind,]
  y<- ozone2$y[day,ind]
  tmpFit<- spatialProcess( x,y,Distance="rdist.earth")
  O3MLE<- rbind( O3MLE, tmpFit$MLESummary)
}
# last two columns are repeated
O3MLE<- O3MLE[, -c(11,12)]
```

```
plot( O3MLE[,"lambda"], O3MLE[,"aRange"], log = 'xy', xlab = "lambda", ylab = "aRange")
title("aRange vs lambda MLEs for 10 days")
```



```
plot( O3MLE[,"eff.df"], O3MLE[,"aRange"], log = 'xy', xlab = "eff.df", ylab = "aRange")
title("aRange vs effective degrees of freedom  for 10 days")
```



An alternative to fitting a different mean part of the model for each day is to pool all of the days and compute the fixed effects by averaging. The function `mKrig.coef` takes care of this for us. One just needs to pass an `mKrig` object, and a vector `y` of new observations. If `y` is a matrix, the columns are treated as multiple realizations of the same field.

Basic Usage

> object<- mKrig(x, y, aRange= 1.0, lambda = .1, collapseFixedEffect = TRUE)
> look<- predict(object, collapseFixedEffect = object$collapseFixedEffect)
> coef<- mKrig.coef(object, y, collapseFixedEffect=TRUE)

Value

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

Since this data set has `NA`'s, we remove any columns (observation locations) with an `NA`. Beware of blindly removing data like this - we are just illustrating the `mKrig.coef` functionality.

```
Y<- t(ozone2$y)
Y<- na.omit( Y)
# reduced locations
s<- ozone2$lon.lat[-attr(Y,"na.action"),]
# always good to check
dim( s)
```

```
## [1] 67  2
```

```
dim( Y)
```

```
## [1] 67 89
```

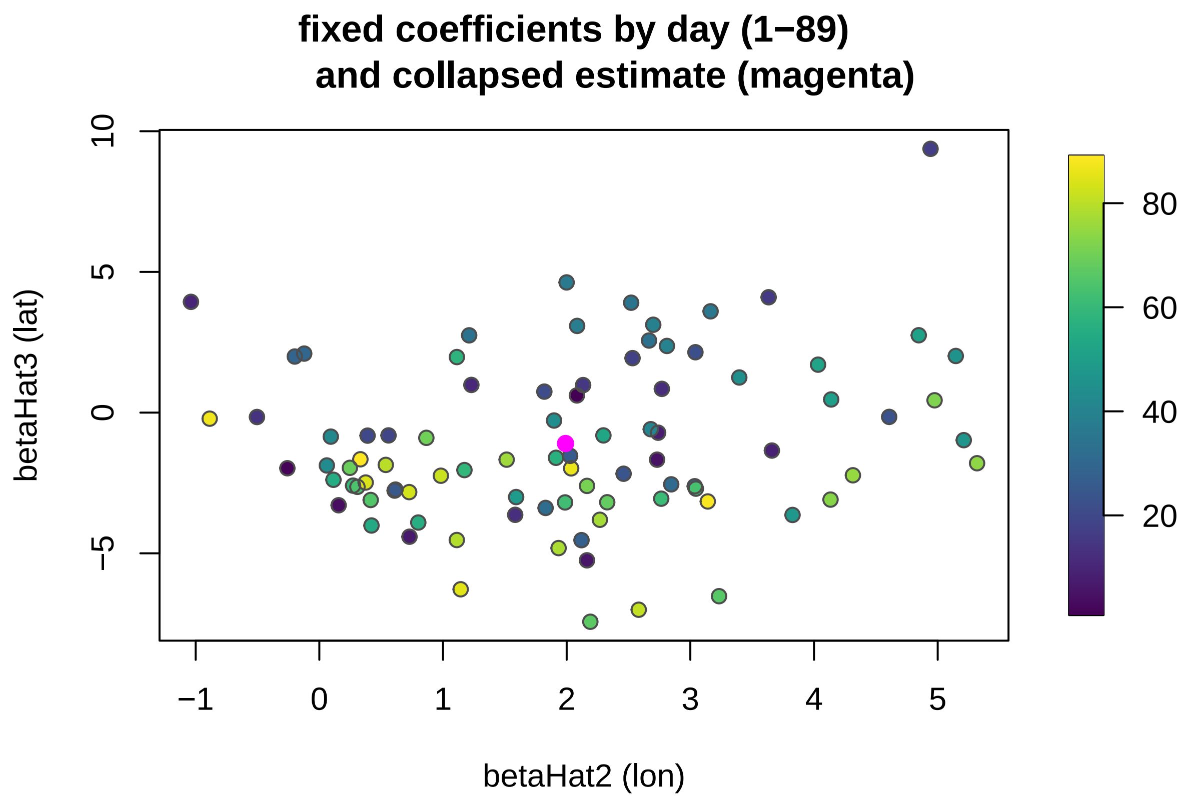
```
# covariance parameters found from spatialProcess
out <- mKrig(s, Y, lambda = .23,
             cov.args=list(Covariance="Matern",
                      aRange= 1.2,smoothness=.5),
                      collapseFixedEffect=TRUE)

out2 <- mKrig(s, Y, lambda = .23,
             cov.args=list(Covariance="Matern",
                      aRange= 1.2,smoothness=.5),
                      collapseFixedEffect=FALSE)
print( out$beta[,1])
```

```
## [1] 267.321876   1.990969  -1.097688
```

```
bubblePlot( t(out2$beta[2:3,]), 1:89,
            xlab="betaHat2 (lon)", ylab="betaHat3 (lat)")
```

```
## NULL
```

```
title("fixed coefficients by day (1-89)
      and collapsed estimate (magenta)")
points( out$beta[2,1], out$beta[3,1], pch=16, col="magenta", cex=1.2)
```



After collapsing the fixed effects, we can predict! We compare the original day 1 fit versus the collapsed fit based on all the days.

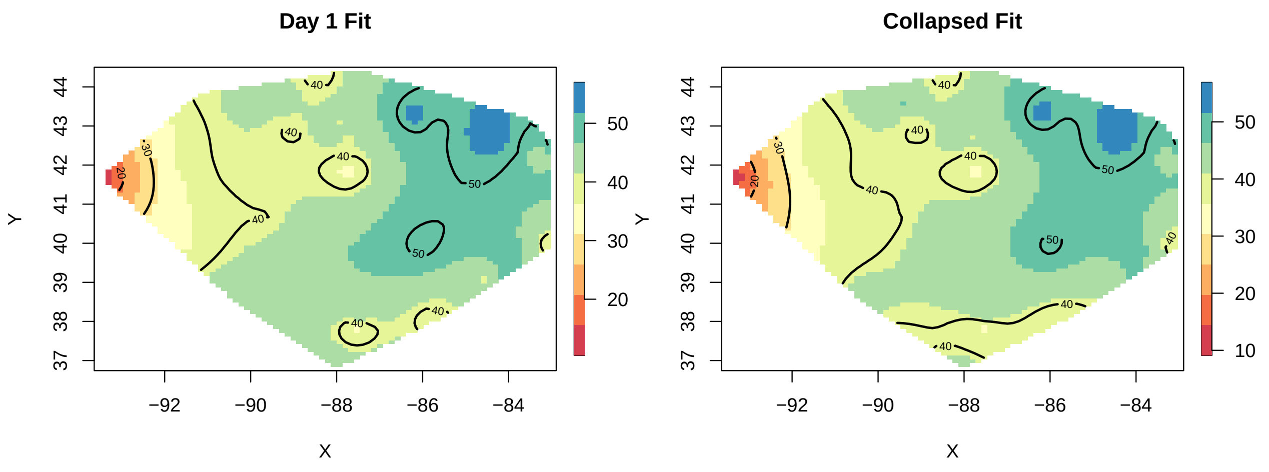
```
xnew <- fields.x.to.grid(x)
pred1 <- predictSurface(out2, xnew, collapseFixedEffect = TRUE)
```

```
## Warning in out[indexGood] <- predict(object, x = xg[indexGood, ], Z = Z[indexGood, : number of items
```

```
pred2 <- predictSurface(out, xnew)
```

```
## Warning in out[indexGood] <- predict(object, x = xg[indexGood, ], Z = Z[indexGood, : number of items
```

```
set.panel(1,2)
surface(pred2, main="Day 1 Fit",col=brewer.pal(9,'Spectral'))
surface(pred1, main="Collapsed Fit",col=brewer.pal(9,'Spectral'))
```



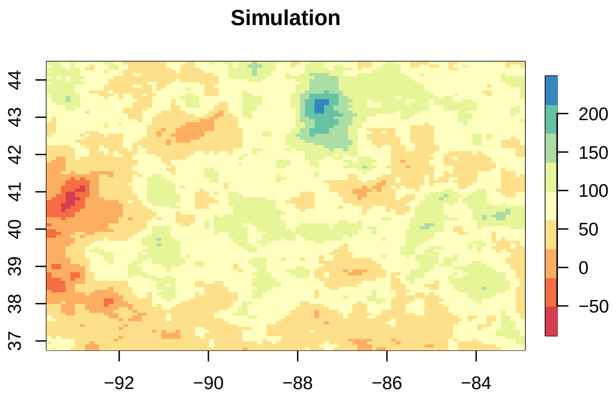
1.1 Approximate standard errors in `mKrig` with `ozone2`

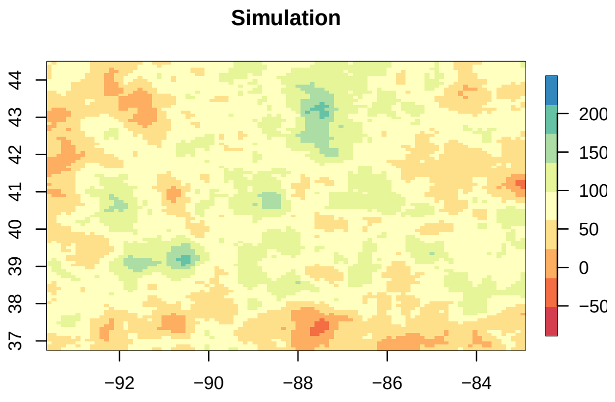
```
data( ozone2)
y<- ozone2$y[16,]
good<- !is.na( y)
y<-y[good]
x<- ozone2$lon.lat[good,]
xMissing<- ozone2$lon.lat[!good,]

O3.fit<- mKrig( x,y, Covariance="Matern", aRange=.5,smoothness=1.0, lambda= .01 )
```

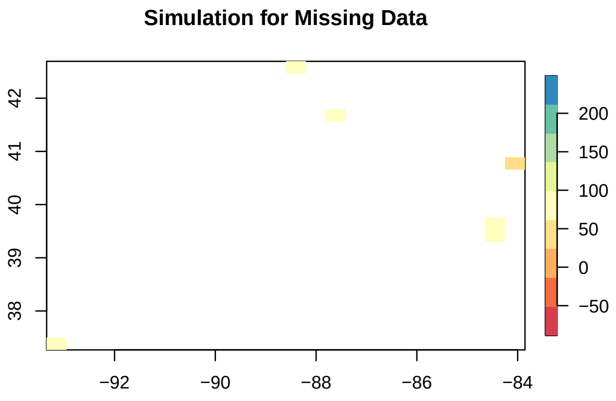
<surface(O3.fit,col=brewer.pal(9,'Spectral'))>

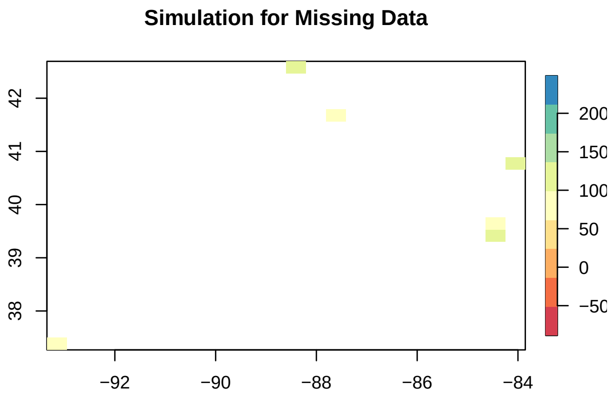
Now, we'll look at some approximate conditional simulations of our random field and the missing data.

```
set.seed(122)

O3.sim<- sim.mKrig.approx( O3.fit, nx=100, ny=100, gridRefinement=3, M=2 )
O3.sim.missing <- sim.mKrig.approx( O3.fit, xMissing, nx=80, ny=80, gridRefinement=3, M=2 )

set.panel(2,2)

  for (k in 1:2){
    image.plot( as.surface( O3.sim$predictionPoints, O3.sim$Ensemble[,k]),zlim=c(-70,230),col=brewer.pa
    title("Simulation")
  }
for(k in 1:2){
    image.plot( as.image(O3.sim.missing$Ensemble[,k],  O3.sim.missing$predictionPoints,nx=24,ny=24) ,zl
    title("Simulation for Missing Data")
}
```





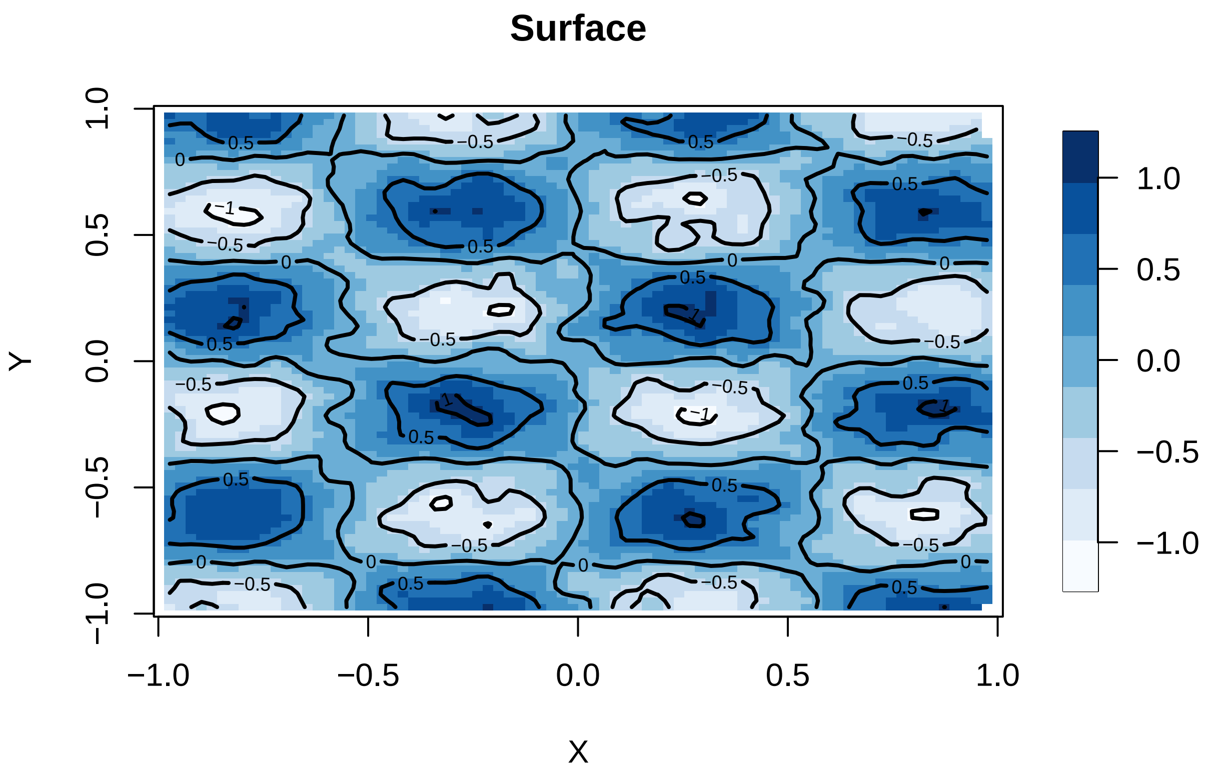




1.2 Finding taper range

Here is a series of examples with bigger datasets that are randomly generated. We use a compactly supported covariance directly.

```r
set.seed(334)
N<- 1500
x<- matrix(2*(runif(2*N)-.5),ncol=2)
values <- sin( 1.8*pi*x[,1])*sin( 2.5*pi*x[,2])

y<- values + rnorm(N)*.1
look2 <- mKrig( x,y, cov.function="wendland.cov",k=2, aRange=.2,lambda=.1)
predictSurface(look2) -> out.p
surface( out.p,col=brewer.pal(9,'Blues'))
title("Surface")
```



Even though there are a large number of points, this works because the number of nonzero elements within distance `aRange` are less than the default maximum allocated size of the sparse covariance matrix. See `options()` for the default values. The following will give a warning for `aRange=.9` because allocation for the covariance matirx storage is too small. Here `aRange` controls the support of the covariance and so indirectly the number of nonzero elements in the sparse matrix.

```
look2<- mKrig(x,y, cov.function="wendland.cov",k=2, aRange=.9, lambda=.1)

## Warning in nearest.dist(structure(c(0.963543639518321, 0.0860745231620967, : You ask for a 'dense' s
## To avoid the iteration, increase 'nearestdistnnz' option to something like
## 'options(spam.nearestdistnnz=c(750000,400))'
## (constructed 1213 lines out of 1500).

## Warning in nearest.dist(structure(c(0.963543639518321, 0.0860745231620967, : You ask for a 'dense' s
## To avoid the iteration, increase 'nearestdistnnz' option to something like
## 'options(spam.nearestdistnnz=c(750000,400))'
## (constructed 1213 lines out of 1500).
```

The warning resets the memory allocation for the covariance matrix according the to values `options(spam.nearestdistnnz=c(416052,400))`. This is inefficient because the preliminary pass failed. The following call completes the computation in "one pass" without a warning and without having to reallocate more memory.

```
options(spam.nearestdistnnz=c(4.2E5,400))
look2<- mKrig( x,y, cov.function="wendland.cov",k=2, aRange=.9, lambda=1e-2)

## Warning in nearest.dist(structure(c(0.963543639518321, 0.0860745231620967, : You ask for a 'dense' s
## To avoid the iteration, increase 'nearestdistnnz' option to something like
## 'options(spam.nearestdistnnz=c(600000,400))'
## (constructed 966 lines out of 1500).

## Warning in nearest.dist(structure(c(0.963543639518321, 0.0860745231620967, : You ask for a 'dense' s
## To avoid the iteration, increase 'nearestdistnnz' option to something like
## 'options(spam.nearestdistnnz=c(600000,400))'
## (constructed 966 lines out of 1500).
```

As a check notice that `print(look2)` reports the number of nonzero elements consistent with the specifc allocation increase in `spam.options`.

Now we generate a new data set of 1500 locations.

```
set.seed( 234)
N <- 1500
x <- matrix( 2*(runif(2*N)-.5),ncol=2)
y <- values + rnorm(N)*.01
```

The following is an example of where the allocation (for `nnzR`) for the cholesky factor is too small. A warning is issued and the allocation is increased by 25. To avoid the warning, use the second call to `mKrig`.

```
look2<- mKrig( x,y,cov.function="wendland.cov",k=2, aRange=.1, lambda=1e2 )

look2<-mKrig( x,y,cov.function="wendland.cov", k=2, aRange=.1,lambda=1e2,
              chol.args=list(pivot=TRUE, memory=list(nnzR = 450000)))
```

Next, we show how to fit multiple data sets observed at the same locations.

```
y1 <- values + rnorm(N)*.01
y2 <- values + rnorm(N)*.01
Y <- cbind(y1, y2)
look3<- mKrig( x,Y,cov.function="wendland.cov",k=2,aRange=.1, lambda=1e2 )
print( look3)
```

```
## [1]  0.013450047 -0.008442535  0.009224268
## Call:
## mKrig(x = x, y = Y, cov.function = "wendland.cov", lambda = 100,
##     k = 2, aRange = 0.1)
##
##  Number of Locations:                          1500
##  Number of data sets fit:                      2
##  Degree of polynomial null space ( base model): 1
##  Total number of parameters in base model      3
##   Estimate Eff. degrees of freedom             18.2
##      Standard Error of Eff. Df                 0.769
##  Smoothing parameter                           100
##  Nonzero entries in covariance                 18614
##
##
## Summary of fixed effects
##     estimate       SE pValue
## d1  0.013450 0.009545 0.1588
## d2 -0.008443 0.016670 0.6125
## d3  0.009224 0.016320 0.5718
##
## Covariance Model: wendland.cov
##    Non-default covariance arguments and their values
##    Argument: k  has the value(s):
## [1] 2
##    Argument: aRange  has the value(s):
## [1] 0.1
```

Note the slight difference in the summary output because two data sets have been fit.

<We continue by showing a method for finding a good choice for `aRange` as a taper. Suppose the target `aRange` is a spatial prediction using roughly 50 nearest neighbors (tapering covariances is effective for roughly 20 or more in the situation of interpolation) see Furrer, Genton and Nychka (2006). We begin with a random sample of 100 points to get an idea of scale and save computation time.>

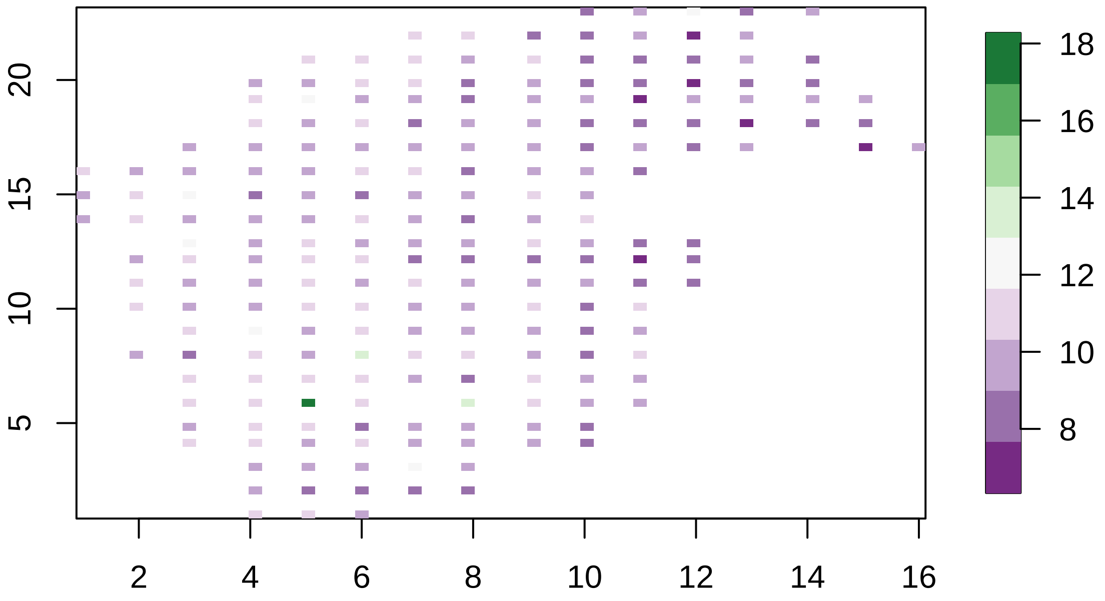
<We declare `min(hold2)` is close enough, so now the following will use no less than 55 nearest neighbors due to the tapering.>

1.3 Using `V`, `mKrig`,`Tps` with `RMprecip`

In this example we return to the Rocky Mountain precipitation data. We make a grid and append our data observations to it, then interpolate using a `Tps` model. We compare this to an `mKrig` model using a Wendland covariance. We also use the `V` argument in `mKrig`, which is a matrix that describes the inverse linear transformation of the coordinates before distances are found. In R code this transformation is `x1 %*% t(solve(V))`. The default is `NULL`, i.e. $V = a \cdot \mathbf{I}_{n \times n}$. If one has a vector of `aRange` values that are the scalings for each coordinate then just express this as `V = diag(aRange)` in the call to this function.

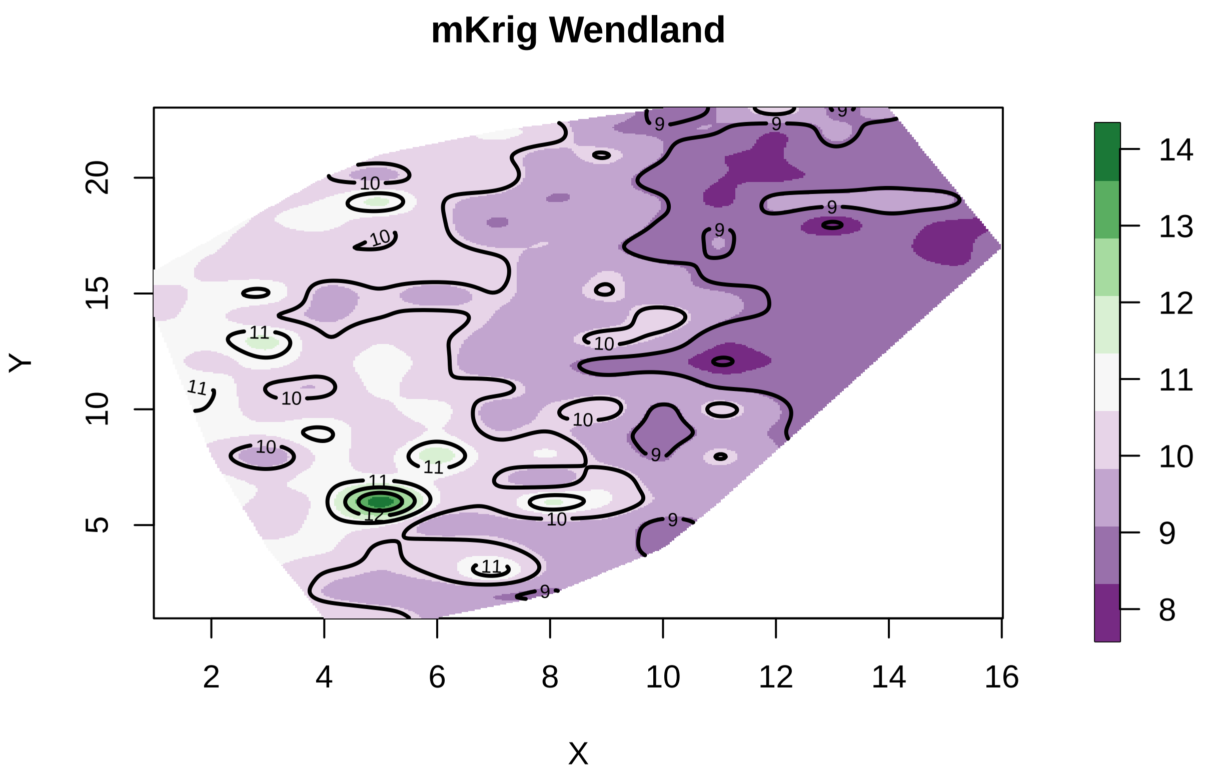
Again, `smallplot` is used strictly for formatting this document.

```
library(gstat)
data(coalash)
x <- cbind(coalash$x, coalash$y)
y <- coalash$coalash

pg <- brewer.pal(9,'PRGn')
quilt.plot(x, y,col=pg)
```



```
#equispaced grid with length 1 in both lon and lat
dlon <- 1
dlat <- 1
V <- diag( c(2.5*dlon, 2.5*dlat) )
```

```r
# this is an interplotation of the values using a compact support but Tps-like covariance.
out2 <- mKrig( x, y, cov.function="wendland.cov",k=4, V=V, lambda=1)
look <- predictSurface( out2, nx=400, ny=400)

surface(look, main="mKrig Wendland",smallplot= c(.88,.9,0.2,.8),col=pg)
US(add=TRUE, col="white")
```


mKrig Wendland
X
Y