Spatial Experiments Doc 2

Antony Sikorski

Comparing the fields functions to raw mathematical calculations

First, we do a little setup:

```
suppressMessages(library( fields))
suppressMessages(library(viridis))

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

#we will constantly be comparing to values from obj below
obj<- spatialProcess( s, y, smoothness=.5)

sigma2<- obj$summary["sigma2"]
tau<- obj$summary["tau"]
aRange<- obj$summary["aRange"]
```

First we explicitly compute the GLS estimate for the parameters:

```
#Some setup
X <- cbind(1,s)
n <- length(y)
K <- sigma2 * exp(-rdist(s,s)/aRange)

#note: bigSigma is the same as S11
bigSigma <- K + diag(tau^2, n)

#Actually computing beta hat
betaHat <- solve((t(X)%*%solve(bigSigma)%*%X))%*%t(X)%*%solve(bigSigma)%*%y
betaHat
```

```
##            [,1]
## [1,] 198.698905
## [2,]   3.103622
## [3,]   3.583734
```

```
test.for.zero(obj$beta,betaHat)
```

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

It appears that our parameters are the same as the ones from obj$beta.

We now compute the predicted values.

```
predVals <- X%*%betaHat + K%*%solve(bigSigma)%*%(y - X%*%betaHat)
test.for.zero(predVals, predict(obj, s)) #this should pass
```

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

```
test.for.zero(predVals, y) #this should fail
```

```
## FAILED test value =  4.58748  at tolerance  1e-08
```

We find that our predicted values are the same as the ones in in the R obj, but do not match the data values exactly. This is because when this prediction is computed, we find a mean surface that represents zero real surfaces, but is simply the best average estimate.

In order to find the standard error, we must first compute the weight matrix, W, which will help us obtain the covariance matrix. After finding the covariance matrix, we will take the square root of its diagonals to find the standard error.

```
#grid provided
sStar<- make.surface.grid(
  list(x = seq( -94,-82,length.out=5),
       y = seq( 36,45, length.out=5)
  )
)

#Xstar needs a constant
Xstar <- cbind(1, sStar)
H <- solve((t(X)%*%solve(bigSigma)%*%X))%*%t(X)%*%solve(bigSigma)
w1 <- Xstar%*%H

#this is the same as S21
k <- sigma2 * exp(-rdist(sStar,s)/aRange)

#will use these later on
S12 <- sigma2 * exp(-rdist(s,sStar)/aRange)
S22 <- sigma2 * exp(-rdist(sStar,sStar)/aRange)

c <- k%*%solve(bigSigma)
w2 <- c%*%(diag(1,nrow(bigSigma)) - X%*%H)

#our W matrix
W <- t(w1 + w2)

#checking to make sure we are correct so far
#dougsW <- makeKrigingWeights(obj, sStar)
#test.for.zero(W,dougsW)

#computing covar and testing SE against the predicted SE
covPredErr <- t(W)%*%bigSigma%*%W - k%*%W - t(W)%*%S12 + S22
predSE <- sqrt(diag(covPredErr))
dougSE <- predictSE(obj, sStar)

test.for.zero(predSE, dougSE)
```

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

It appears that our SE matches the predictSE command for our R obj.

Inspecting Dr. Doug Nychka's makeKrigingWeights function:

```
makeKrigingWeights<- function(obj, sStar){
  n<- length(obj$y)
  IM<- diag( 1, n)

  nPred<- nrow( sStar)
  W<- matrix(NA,n,nPred)
  for( k in 1:n){
    W[k,]<- predict( obj,sStar, ynew=IM[,k])
  }
 # test.for.zero( t(W)%*%obj$y, obj$fitted.values[1:10])
  return(W)
}
```

Everything before the for-loop statement in this function is just setup. We are really just instantiating an identity matrix with the same number of rows/cols as the data and an empty W matrix. The real magic of the function occurs at the for loop. According to the R documentation, the predict function takes in obj as the fit object, and sStar as the locations where it wishes to evaluate at. The ynew argument is described as:

“Evaluate the estimate using the new data vector y (in the same order as the old data). This is equivalent to recomputing the Krig object with this new data but is more efficient because many pieces can be reused. Note that the x values are assumed to be the same.”.

In our case, predict(obj, sStar) is equivalent to $w^T z$, where z is sStar. But rather than evaluating this, the function takes ynew, and evaluates $w^T (ynew)$ in the same order as it did at sStar. In this case, our ynew is actually just a vector of a column of the identity matrix, where the first one would look like: $(1\ 0 \ldots 0)$. Each multiplication by w^T by a column of that identity matrix reconstructs a row of the W matrix, since it is not originally stored when spatialProcess is performed (to reduce computational demands).

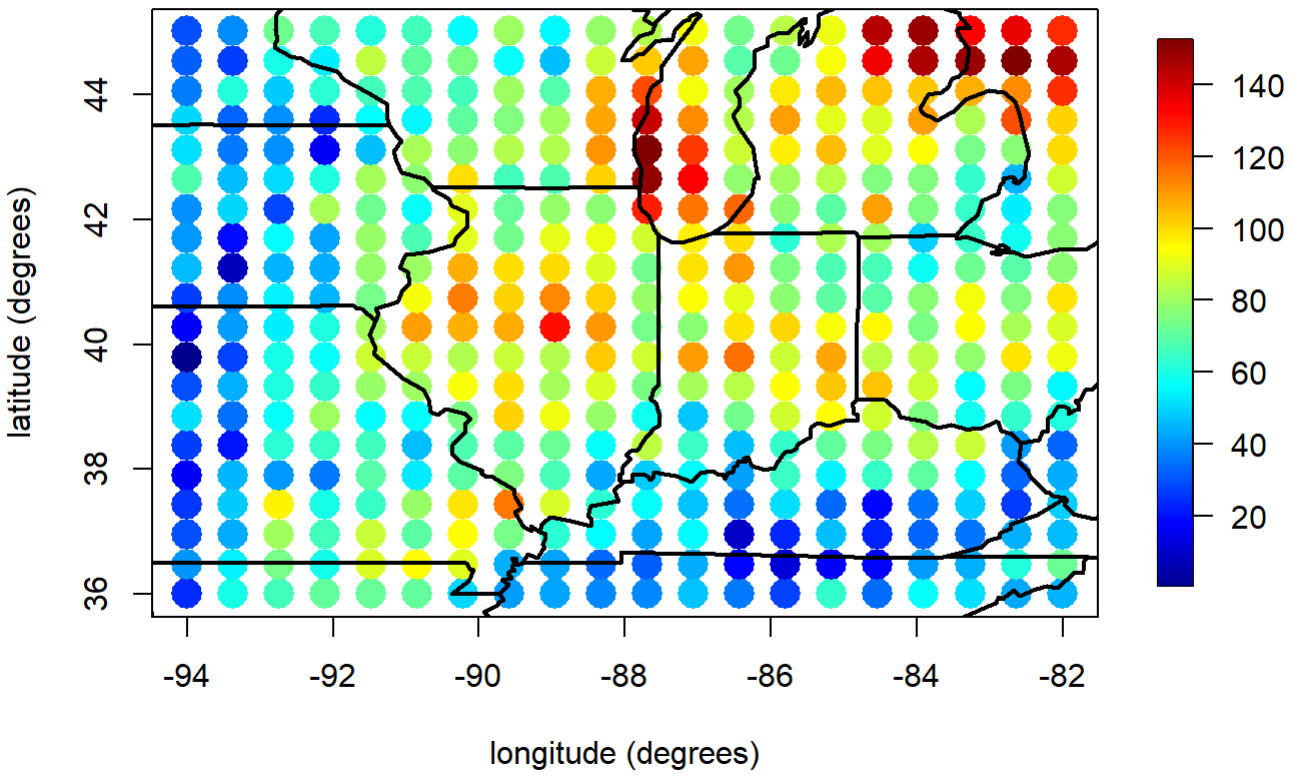
Simulating a conditional field

We have to redo some of the setup from above with minor differences.

```
#new 20x20 grid
sStar<- make.surface.grid(
  list(x = seq( -94,-82,length.out=20),
       y = seq( 36,45, length.out=20)
  )
)

#effectively redoing 1c
Xstar <- cbind(1, sStar)
H <- solve((t(X)%*%solve(bigSigma)%*%X))%*%t(X)%*%solve(bigSigma)
w1 <- Xstar%*%H
k <- sigma2 * exp(-rdist(sStar,s)/aRange)
S12 <- sigma2 * exp(-rdist(s,sStar)/aRange)
S22 <- sigma2 * exp(-rdist(sStar,sStar)/aRange)

c <- k%*%solve(bigSigma)
w2 <- c%*%(diag(1,nrow(bigSigma)) - X%*%H)
W <- t(w1 + w2)

#final covariance matrix
covPredErr <- t(W)%*%bigSigma%*%W - k%*%W - t(W)%*%S12 + S22

#Cholesky decomp
cholPredCov <- chol(covPredErr)

#conditional simulated field
set.seed( 432)
error<- t(cholPredCov)%*% rnorm(400)
condSim<- predict( obj, sStar) + error
```

We create a bubble plot on top of the US map to visualize the conditional simulated field that we created above.

```
#plotting
bubblePlot(sStar, condSim, highlight = FALSE, size = 2.2, col = tim.colors,
           ylab = "latitude (degrees)", xlab = "longitude (degrees)")
US(add = TRUE, lwd = 2)
```


latitude (degrees)
36
38
40
42
44
-94
-92
-90
-88
-86
-84
-82
longitude (degrees)
140
120
100
80
60
40
20