HW06 Spatial Statistics

Isabella Chittumuri

2023-10-16

Some setup

```
suppressMessages(library( fields))
source("makeKrigingWeights.R")
library(fields)
```

Problem 1

This problem will revisit the ozone data set example and give you the practice in reproducing the Universal Kriging computation. The idea is do the computations "by hand" and compare to the fields functions that implement universal Kriging.

To setup the data set load the fields package and

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

This assignment will make use of the function **spatialProcess** and its supporting methods to serve as a benchmark. We will cover estimating the covariance parameters later in the course and so for this assignment assume they are fixed and take their values from the spatialProcess MLE fit listed below.

The covariance model used here is the exponential with scale parameter **aRange**. In places in class we have also referred to this parameter as θ. See Section 4.5 in the text for the details of this covariance and the Matern family. Here we use the fact that a Matern covariance with smoothness .5 is just the exponential.

```
obj<- spatialProcess( s, y, smoothness=.5)
sigma2<- obj$summary["sigma2"]
tau<- obj$summary["tau"]
aRange<- obj$summary["aRange"]
```

Note that the parameter `tau` refers to the measurement error standard deviation. Be sure to use `tau^2` in your expressions that involve the variance and covariances.

The linear model (aka the fixed part) in this fit has three parameters a constant and linear terms in longitude and latitude. That is, the "X" matrix for the linear part is

```
X <-  cbind(1, s)
```

1(a)

Using the parameters specified above compute explicitly the GLS estimate for the parameters. These should match obj$beta in the R object.

```
d = rdist(s,s)
K = sigma2*exp(-d/aRange)
M = K + tau^2*diag(147)

betaHat = solve(t(X)%*%solve(M)%*%X)%*%t(X)%*%solve(M)%*%y

betaHat
```

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

```
obj$beta
```

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

1(b)

- Compute explictly the predicted values at the observations. See Equation (6.2) in the text book for a concise formula for this but noting that $\hat{\beta}$ is the GLS estimate.
- For this case explain why the predicted values at the observed locations to not match the data values exactly.

```
x <- t(X)
M = K + tau^2*diag(147)
d = rdist(s,s)
k = sigma2*exp(-d/aRange)

preds <- t(x)%*%betaHat + t(k)%*%solve(M)%*%(y-X%*%betaHat)

cbind(y,preds)
```

```
##                  y
##   [1,]  75.00000  74.572172
##   [2,]  84.25000  84.457834
##   [3,]  90.87500  95.283218
##   [4,] 127.42857 117.510165
##   [5,] 104.50000  97.824316
##   [6,]  86.25000  90.562749
##   [7,]  93.25000  91.554708
##   [8,]  95.50000  99.475139
##   [9,]  89.87500  89.883614
##  [10,] 100.87500  99.480177
##  [11,]  87.25000  93.293686
##  [12,] 112.37500 114.723589
##  [13,]  91.37500  90.695761
##  [14,]  88.25000  86.651037
```

```
## [15,]  89.12500  90.237908
## [16,] 115.12500 113.571542
## [17,] 140.87500 138.557245
## [18,] 115.75000 116.587399
## [19,]  98.00000  98.074833
## [20,]  87.37500  87.372282
## [21,]  89.37500  88.762400
## [22,]  82.87500  82.449485
## [23,]  76.75000  75.236562
## [24,]  78.75000  77.627790
## [25,]  78.87500  79.978815
## [26,]  59.00000  60.040587
## [27,]  87.75000  91.166766
## [28,] 100.75000  96.043122
## [29,]  76.00000  78.252925
## [30,]  58.00000  61.035478
## [31,]  89.50000  89.288340
## [32,]  91.00000  89.971636
## [33,]  83.50000  83.521621
## [34,]  83.50000  83.619547
## [35,]  60.50000  63.867205
## [36,]  73.62500  71.687166
## [37,]  73.75000  73.743831
## [38,]  48.00000  53.366194
## [39,]  78.85714  81.461174
## [40,]  86.50000  86.666571
## [41,]  88.33333  89.378132
## [42,]  76.00000  71.747093
## [43,]  48.00000  55.272278
## [44,]  67.87500  74.199939
## [45,]  92.00000  91.333526
## [46,]  97.00000  94.077213
## [47,]  88.71429  90.221533
## [48,]  96.87500  94.594683
## [49,]  95.00000  93.271920
## [50,]  95.37500  91.248105
## [51,]  76.87500  80.043140
## [52,]  71.25000  77.141023
## [53,]  98.25000  92.626478
## [54,]  72.62500  74.210037
## [55,]  65.25000  60.497411
## [56,]  67.00000  66.120224
## [57,]  96.28571  93.509645
## [58,]  69.85714  66.114718
## [59,]  79.66667  76.195638
## [60,]  55.75000  59.252963
## [61,]  66.50000  65.408665
## [62,]   0.00000   3.768484
## [63,]   5.25000   6.434692
## [64,]  87.62500  85.175752
## [65,]  96.87500  95.359445
## [66,]  55.75000  55.492462
## [67,]  89.62500  93.856499
## [68,]  61.25000  61.148509
```

```
##  [69,]  40.85714  42.111604
##  [70,]  71.87500  69.240002
##  [71,]  59.37500  63.326462
##  [72,]  72.75000  70.675109
##  [73,]  22.75000  33.352792
##  [74,]  53.87500  55.289773
##  [75,]  60.62500  58.130208
##  [76,] 100.50000  98.441855
##  [77,]  58.12500  56.280090
##  [78,]  57.50000  56.780438
##  [79,]  35.25000  36.873841
##  [80,]  68.25000  66.404948
##  [81,]  30.37500  32.861547
##  [82,]  85.33333  82.635853
##  [83,]  73.00000  73.292979
##  [84,]  68.40000  69.835833
##  [85,]  70.40000  74.146157
##  [86,] 100.75000 100.142995
##  [87,] 100.50000  99.215010
##  [88,]  66.75000  65.605875
##  [89,]  96.37500  98.834494
##  [90,] 103.16667 102.283197
##  [91,]  63.75000  61.863961
##  [92,]  95.75000  88.722351
##  [93,]  78.37500  79.637777
##  [94,]  52.87500  54.600501
##  [95,]  51.62500  55.113534
##  [96,]  62.75000  62.372700
##  [97,]  54.75000  58.794537
##  [98,]  58.28571  57.135760
##  [99,]  84.00000  84.527364
## [100,]  79.85714  75.016910
## [101,]  82.00000  82.499673
## [102,]  82.50000  80.286584
## [103,]  91.25000  86.652645
## [104,]  95.87500  90.681898
## [105,]  60.62500  64.141765
## [106,]  65.25000  69.028183
## [107,]  60.00000  62.440437
## [108,]  68.37500  71.373249
## [109,]  93.00000  91.213888
## [110,]  81.50000  82.128753
## [111,]  89.14286  88.922339
## [112,]  87.87500  87.797917
## [113,] 106.12500 100.413468
## [114,]  84.25000  92.468129
## [115,] 108.37500 104.049549
## [116,]  86.57143  88.334410
## [117,]  96.75000  97.009889
## [118,] 106.50000 101.117643
## [119,]  62.57143  64.223183
## [120,]  64.62500  64.609294
## [121,]  73.50000  76.286150
## [122,]  86.28571  85.799780
```

```
## [123,]  88.87500  87.826886
## [124,]  80.87500  80.840642
## [125,]  84.37500  83.863517
## [126,]  82.87500  83.069509
## [127,]  96.00000  93.433596
## [128,]  80.37500  83.892883
## [129,]  89.62500  88.908137
## [130,] 162.57143 154.053510
## [131,] 112.62500 113.197395
## [132,]  66.62500  66.982779
## [133,] 137.75000 134.102457
## [134,] 157.62500 146.404833
## [135,]  87.33333 103.653478
## [136,] 107.75000 114.261092
## [137,] 157.37500 148.912408
## [138,] 137.37500 135.401529
## [139,]  72.50000  76.020132
## [140,] 148.50000 143.877254
## [141,] 144.00000 145.102401
## [142,]  82.87500  83.198444
## [143,]  84.87500  85.173944
## [144,] 144.62500 143.041973
## [145,]  94.75000  96.713710
## [146,]  88.00000  90.681740
## [147,]  81.62500  81.047723
```

The predicted values aren't exactly the same as the observed values because they're based on the model. If they matched perfectly everywhere, our model might be over fit the data and would could struggle with new data.

1(c)

The function `makeKrigingWeights` will create the weight matrix referred to in equation (6.3) of the text. For example the code

```r
sStar<- s
W<- makeKrigingWeights(obj, sStar)
test.for.zero( t(W)%*%y, predict(obj, sStar))
```

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

will find the predictions at the observation locations and then test that they agree with the predictions from the spatialProcess fit. `t(W)` is also the smoother matrix referred to in the text although `makeKrigingWeights` will work for any set of locations, not just the observation locations.

Evaluate the formula in 6.2.6 for the covariance of the prediction errors and for the locations on the 5X5 grid defined below:

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

x <- rbind(1, t(sStar))

k = sigma2*exp(-rdist(sStar,s)/aRange)
```

```r
reverse_k = sigma2*exp(-rdist(s,sStar)/aRange)

W <- makeKrigingWeights(obj, sStar)

cov_errors <- t(W)%*%M%*%W - k%*%W - t(W)%*%reverse_k +
  sigma2*exp(-rdist(sStar, sStar)/aRange)
```

Find the square root of the diagonal elements of this covariance matrix and compare to

```r
SE<- predictSE( obj, sStar)

cbind(SE, sqrt(diag(cov_errors)))
```

```
##              SE
##  [1,] 25.76856 25.76856
##  [2,] 25.61328 25.61328
##  [3,] 21.31617 21.31617
##  [4,] 25.16027 25.16027
##  [5,] 29.27596 29.27596
##  [6,] 24.06139 24.06139
##  [7,] 18.95603 18.95603
##  [8,] 15.68801 15.68801
##  [9,] 14.46021 14.46021
## [10,] 25.77280 25.77280
## [11,] 23.18160 23.18160
## [12,] 18.19918 18.19918
## [13,] 16.21271 16.21271
## [14,] 16.06134 16.06134
## [15,] 22.51133 22.51133
## [16,] 23.67797 23.67797
## [17,] 18.93288 18.93288
## [18,] 10.11084 10.11084
## [19,] 15.02431 15.02431
## [20,] 22.81000 22.81000
## [21,] 28.65729 28.65729
## [22,] 23.83354 23.83354
## [23,] 19.41196 19.41196
## [24,] 24.96896 24.96896
## [25,] 27.74508 27.74508
```

They should match!

1(d) GRAD

Explain why the function `makeKrigingWeights` works.

The function makeKrigingWeights works by calculating the weights used in kriging predictions. In kriging, predictions at unsampled locations are made by combining the weighted values of nearby observed locations. The makeKrigingWeights function computes these weights based on the spatial distances between the prediction location and observed points, as well as the covariance parameters obtained from the spatial model. It captures the essence of spatial autocorrelation: nearby points are more similar than distant ones. It works because by using these calculated weights, the function enables the creation of accurate predictions at unsampled locations, incorporating both the observed data and the spatial patterns present in the dataset

Problem 2

Modify the code in 1(c) to be a 20X20 grid, so now sStar is 400 points. Compute the prediction error covariance matrix, find the cholesky decomposition, and call this `cholPredCov` Now generate a conditional simulation for the predicted field according to

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

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

x <- rbind(1, t(sGrid))
k = sigma2*exp(-rdist(sGrid,s)/aRange)
reverse_k = sigma2*exp(-rdist(s,sGrid)/aRange)

W <- makeKrigingWeights(obj, sGrid)

cov_errors <- t(W)%*%M%*%W - k%*%W - t(W)%*%reverse_k +
  sigma2*exp(-rdist(sGrid,sGrid)/aRange)

cholPredCov <- chol(cov_errors)

set.seed(432)
error <- t(cholPredCov)%*%rnorm(400)
condSim <- predict(obj, sGrid) + error
```

2(a)

Use `bubblePlot` to visualize the simulated field, add the data locations, and also the outline of the US states. You may want to adjust the size of the bubble points so that they are almost touching and use `highlight ==FALSE` to create a more finished and readable plot.

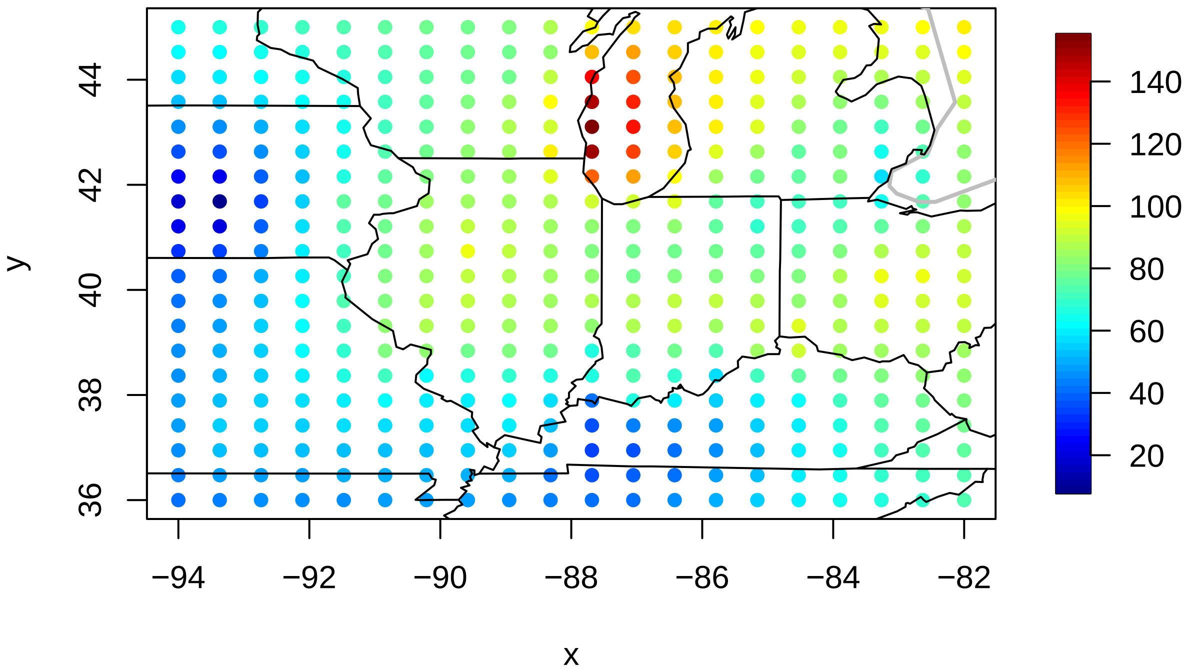
```
XStar <- cbind(1, sGrid)

kStar <- sigma2*exp(-rdist(sGrid, s)/aRange)

yHat <- kStar%*%solve(M)%*%(y-X%*%betaHat)

fHat <- XStar%*%betaHat + yHat

bubblePlot(sGrid, fHat, highlight = FALSE, col=tim.colors())
world(add=TRUE, col="grey", lwd=2)
US(add=TRUE, col="black")
```



2(b)

Here is an easy final question just to fix concepts.

For the random field generated above and called `condSim`, what is its mean conditional on the data and what is its covariance?

condSim is distributed multivariate normal with mean $\hat{f}*$ and covariance Σ