coeffcient_investigation

Daivd Leslie

Sunday, April 12, 2015

The first step in investigating coeffcients will be to setup data storage for generated data. In order to accomplish this, an arrary of matrixes containing generated data will be created. To determind the size of the arrary, a grid size must be provided.

```
# Set grid size
n = 100

# Select desired number of matrices
numMat = 100

# Select number of columns
numCols = 6

# Names of columns
# Must match the number of coulmns
colNames = c('y', 'x1', 'x2', 'x3', 'xCoord', 'yCoord')

# Create arrary filled with NAs
resultsArray = array(1:(numCols*n^2), dim=c(n^2, numCols, numMat), dimnames = list(1:n^2, colNames ,NUL
```

Now that the array has been created, we can now begin to fill the array with data that we will using for the investigation. In order to accomplish this spatailly autocorrelated data will be generated using a modified version of the Spatail leave-one-out method developed by Le Rest.

```
# Import Statement(s)
library(RandomFields)

# Set spatail range for distance between points
spat_range =  seq(0.001, 60, length.out=numMat)

# Determine range of varience
var_range = var_range = seq(1, 10, 10)

for (i in 1:numMat) {
  # Create  a model with spatial dependence
  mod_spat_dep = RMexp(var=var_range, scale=spat_range[i])

  # Create spatially autocorralated predictor variables
  x1 = RFsimulate(mod_spat_dep, x=1:n, y=1:n, grid=T)
  x2 = RFsimulate(mod_spat_dep, x=1:n, y=1:n, grid=T)
  x3 = RFsimulate(mod_spat_dep, x=1:n, y=1:n, grid=T)

  # Create spatail error term
  spat_err = RFsimulate(mod_spat_dep, x=1:n, y=1:n, grid=T)

  # Convert objects variables to vectors and store in columns
  spat_err = as.vector(spat_err)
```

```
  resultsArray[, 2, i] = as.vector(x1)
  resultsArray[, 3, i] = as.vector(x2)
  resultsArray[, 4, i] = as.vector(x3)
  resultsArray[, 1, i] = 2*resultsArray[, 2, i] + resultsArray[, 3, i] + 3*resultsArray[, 4, i] + spat_

  # Coords
  resultsArray[, 5, i] = rep(x=1:n, times=n)
  resultsArray[, 6, i] = rep(x=1:n, each=n)
}

head(resultsArray[,,1])
```

```
##            y         x1          x2         x3 xCoord yCoord
## 1  6.3506312  2.6847384 -0.41317416  0.4455965      1      1
## 2  2.0231717 -0.5116817  1.04992697  0.3109833      2      1
## 3 -0.3708355  0.6690017  0.08732582 -0.4678195      3      1
## 4 -0.3393989  0.3683890 -0.74616419  0.5869910      4      1
## 5  1.5577861  0.3972475  0.38428597 -0.1262380      5      1
## 6  3.6133031 -0.8537624 -0.40349719  1.7900478      6      1
```

```
head(resultsArray[,,2])
```

```
##           y          x1         x2         x3 xCoord yCoord
## 1  2.483913  0.01155508  1.4838024  0.5562255      1      1
## 2 -2.013822 -1.06715069 -1.1948665 -0.2225202      2      1
## 3  1.213622 -0.97857227 -0.8196596  1.3292277      3      1
## 4 -3.636189 -1.22893479  0.8553526 -0.4872828      4      1
## 5  2.646919  0.08938044  0.1998311  0.8689177      5      1
## 6 -7.237107 -2.61562940  0.6081789 -0.9281224      6      1
```

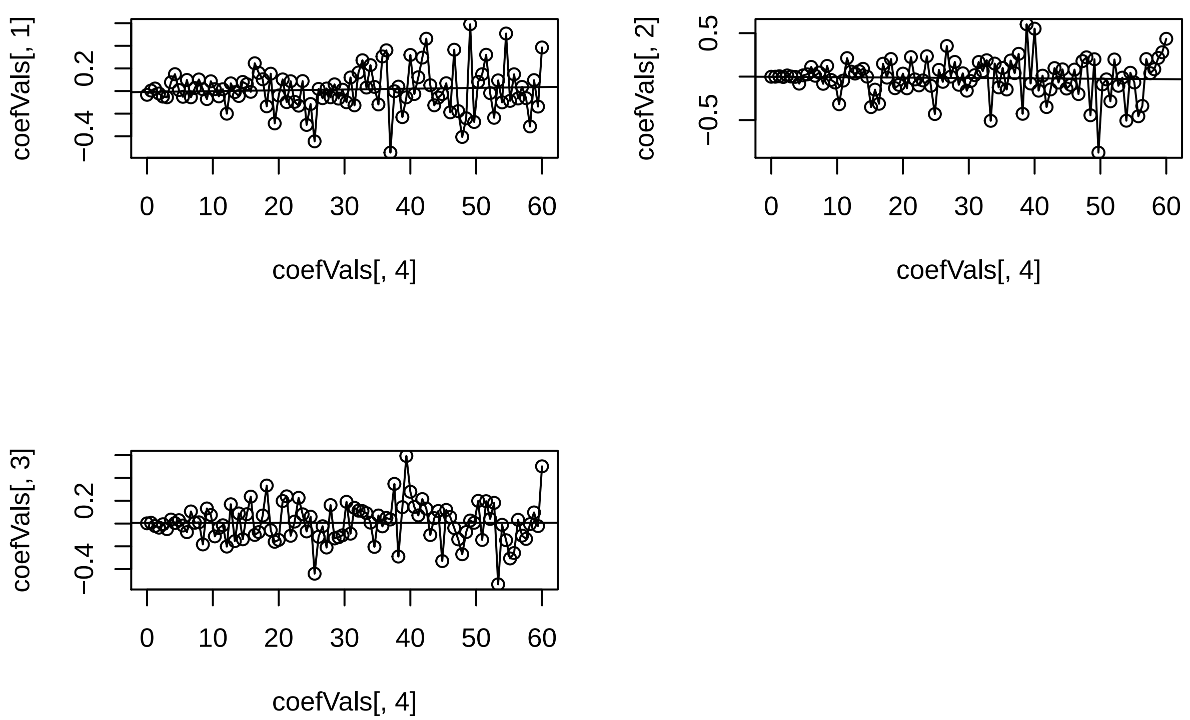
```
head(resultsArray[,,3])
```

```
##            y         x1         x2         x3 xCoord yCoord
## 1 -0.8007126  0.4893490 -0.7454141  0.3583610      1      1
## 2 -3.2217893  0.4020989 -0.7974530 -1.0809301      2      1
## 3  5.2334556  1.8182719 -0.9549836  0.6911030      3      1
## 4 -1.0604952 -0.4866523  0.8607248 -0.5789069      4      1
## 5 -0.7361020 -1.0209484  0.1870013  0.3944556      5      1
## 6 -1.9670030  0.2180257  1.4815691 -1.0523563      6      1
```

Now that we have the array filled with spatially autocorrelated data, let's see how the accurary of the model coeffcients changes as the spatail range increases (Remebering that they should be close to the exspected values of: x1=2, x2=1, and x3=3).

```
coefVals = matrix(NA, nrow=numMat, ncol=3, dimnames=list(1:numMat,c('x1_dif', 'x2_dif', 'x3_dif')))
for(i in 1:numMat) {
  model = glm(y ~ x1 + x2 + x3, data = as.data.frame(resultsArray[,,i]))
  coefVals[i,] = coef(model)[2:4]

}
```

```
coefVals[,1] = 2 - coefVals[,1]
coefVals[,2] = 1 - coefVals[,2]
coefVals[,3] = 3 - coefVals[,3]
coefVals = cbind(coefVals, spat_range)

# Plot accuracy
par(mfrow=c(2,2))
plot(coefVals[,4], coefVals[,1], type='o')
abline(lm(coefVals[,1] ~ coefVals[,4]))


plot(coefVals[,4], coefVals[,2], type='o')
abline(lm(coefVals[,2] ~ coefVals[,4]))

plot(coefVals[,4], coefVals[,3], type='o')
abline(lm(coefVals[,3] ~ coefVals[,4]))
```



From the plots, it appears that the model coeffceints are somewhat accrurate. Now lets look at how the precision changes.

```
coefVals = matrix(NA, nrow=numMat, ncol=3, dimnames=list(1:numMat,c('x1_dif', 'x2_dif', 'x3_dif')))
for(i in 1:numMat) {
  model = glm(y ~ x1 + x2 + x3, data = as.data.frame(resultsArray[,,i]))
  coefVals[i,] = coef(model)[2:4]

}
```

```
coefVals[,1] = abs(2 - coefVals[,1])
coefVals[,2] = abs(1 - coefVals[,2])
coefVals[,3] = abs(3 - coefVals[,3])
coefVals = cbind(coefVals, spat_range)

# Plot precision
par(mfrow=c(2,2))
plot(coefVals[,4], coefVals[,1], type='o')
abline(lm(coefVals[,1] ~ coefVals[,4]))


plot(coefVals[,4], coefVals[,2], type='o')
abline(lm(coefVals[,2] ~ coefVals[,4]))

plot(coefVals[,4], coefVals[,3], type='o')
abline(lm(coefVals[,3] ~ coefVals[,4]))
```