Spatial Statistics 2021 - 4H and 5M

Professor Duncan Lee

University of Glasgow

Computer lab 1 - geostatistical modelling

1. Introduction

In recent years there has been a great deal of interest in monitoring nitrogen levels in the Chesapeake Bay, the largest estuary in the United States. Parts of six states make up the Chesapeake Bay watershed: Delaware, Maryland, New York, Pennsylvania, Virginia, and West Virginia, as well as the District of Columbia. Increased nitrogen levels increase algal growth. The blanketing effect of algal growth, along with the decomposition of these algae, in turn decreases the amount of dissolved oxygen in the water. As a consequence, increasing nitrogen levels pose a substantial threat to the wildlife and ecology of the Bay and the surrounding watershed. In a recent agreement between the Chesapeake Bay Program partners, every state is mandated to regulate the nitrogen levels (both from tributaries and from the air) in their part of the Bay (Chesapeake Bay Program Partnership, 2000). We consider a spatial dataset of 49 measurements of the surface-nitrogen concentration (in mg/l) taken in May 1995 throughout the Chesapeake Bay. The data are stored in two files:

- **chesbay.csv** contains the data; and
- **chesbayboundary.csv** contains the boundary of the region.

Both files are stored on Moodle, and we will use the *geoR* library to perform the geostatistical analysis of these data.

Aim

In analysing these data are aims are to:

- Predict and visualise the pollution levels across the watershed.
- Estimate and visualise the uncertainty in the predictions across the watershed.
- Determine whether the exponential or spherical model has better predictive ability for these data.

2. Data input and visualisation

The first step is to read in the data, which as outlined above contain the data and the boundary of the watershed. These data can be read in using the code

```
data1 <- read.csv(file="chesbay.csv")
head(data1)
```

```
  id longitude latitude nitrogen
1  1  -76.0817  39.5467   1.3185
2  2  -76.0250  39.4400   1.2230
3  3  -76.1750  39.3483   1.1770
4  4  -76.2400  39.2500   1.0830
5  5  -76.3083  39.1650   1.1320
6  6  -76.3600  38.9958   1.0375
```

```
data2 <- read.csv(file="chesbayboundary.csv")
head(data2)
```

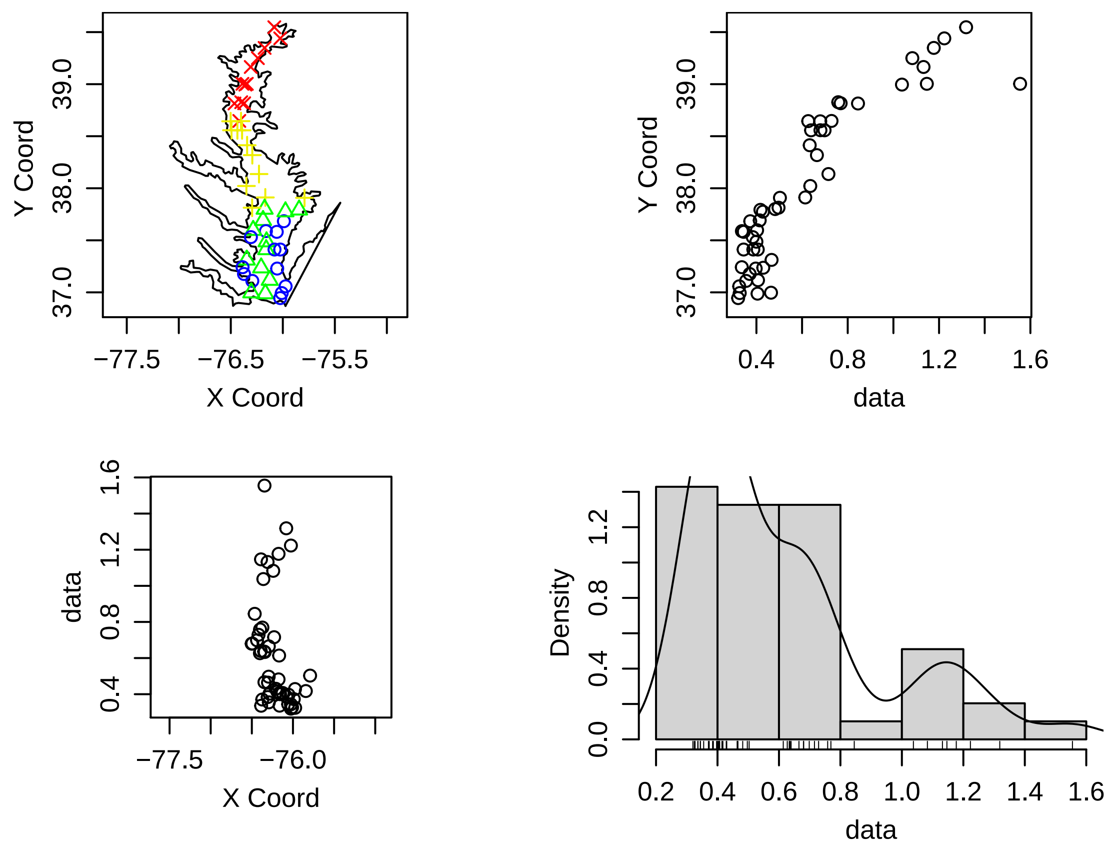
```
  longitude latitude
1 -75.97421 36.86983
2 -75.99139 36.90421
3 -76.02577 36.92140
4 -76.08879 36.89275
5 -76.15755 36.92140
6 -76.20339 36.92713
```

As you can see, the data set contains a column of unique identifiers (*id*), the coordinates (*longitude, latitude*) and the nitrogen levels (*nitrogen*), while the boundary data set contains coordinates (*longitude, latitude*). The *geoR* package like many others requires the data in a specific form, which in this case is a *geodata* object. The data can be transformed to this form using the *as.geodata()* function via the code below.

```
library(geoR)
data.geodata <- as.geodata(data1, coords.col=2:3, data.col=4, borders=TRUE)
data.geodata$borders <- data2
```

The function simply requires one to specify where the data are stored, as well as which column contains the data and which ones contain the geographical coordinates. Finally, the study region border is added in two stages, first by saying *borders=TRUE*, and then manually specifying the border data in the second line above. This object can then be visualised using the *plot()* function as shown below.

```
plot(data.geodata)
```



The plot shows a number of key points:

- The data have a prominant north/south trend.
- No east/west trend is present, most likely because the stuy region is very narrow in this direction.
- The data also appear to be skewed from the histogram.

To visualise these data one can overlay them on a OpenStreetMap. To do this we first need to load the libraries via the code:

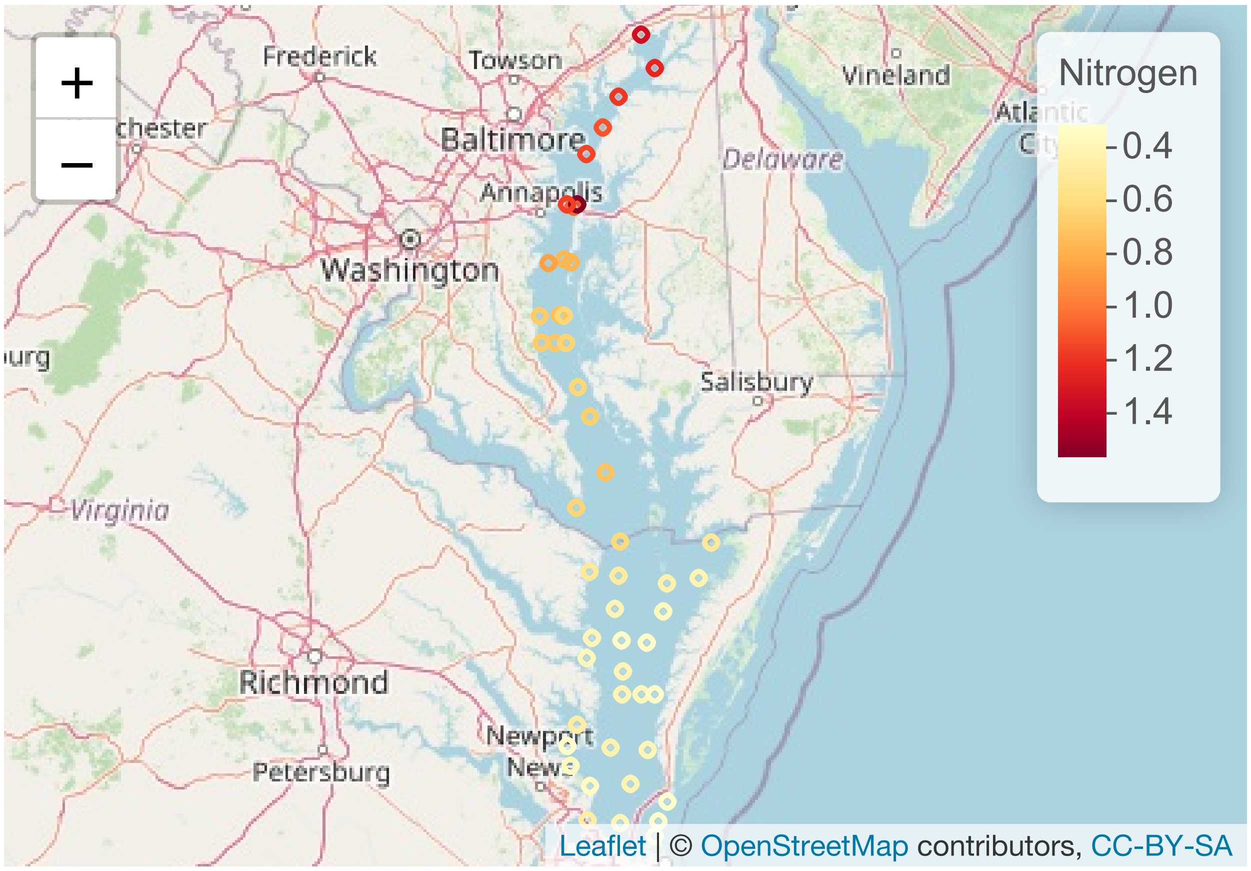
```
library(leaflet)
library(RColorBrewer)
library(sp)
```

Then the data need to be turned into a *SpatialPointsDataFrame* purely for the plotting, which can be done via the code:

```
sp.dat <- SpatialPointsDataFrame(coords=data1[ ,2:3], data=data1)
proj4string(sp.dat) <- CRS("+proj=longlat +datum=WGS84 +no_defs")
```

where the last line specifies that the spatial coordinates are in longitude and latitude. Then finally the map can be drawn using:

```
colours <- colorNumeric(palette = "YlOrRd", domain = sp.dat@data$nitrogen,
                        reverse=FALSE)
leaflet(data=sp.dat) %>%
    addTiles() %>%
    addCircles(~longitude, ~latitude, color = ~colours(nitrogen), opacity = 1,
               radius=80) %>%
    addLegend(pal = colours, values = sp.dat@data$nitrogen,
              opacity = 1, title="Nitrogen")
```



Here the first line specifies the colour scale, while the remaining lines draw the map.

3. Spatial trend modelling

The exploratory plots above suggest fitting a linear trend to the data in the (longitude, latitude) spatial coordinates, which is achieved using the following code:

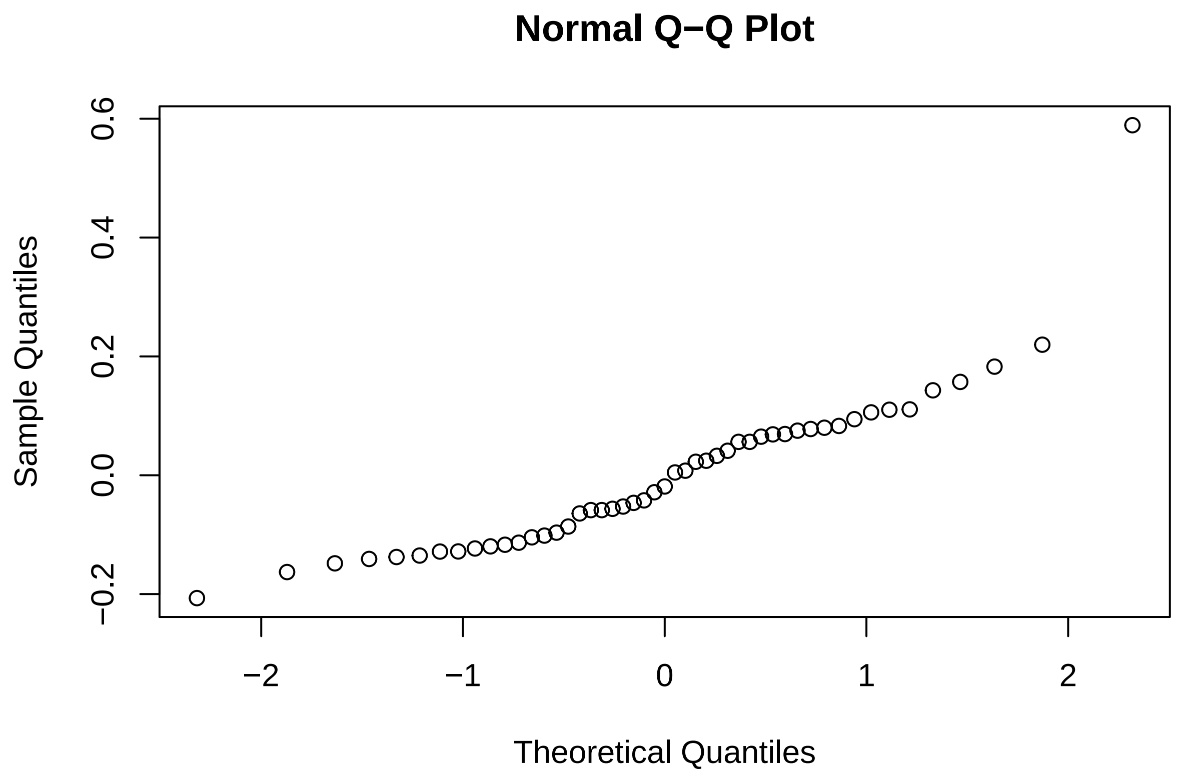
```
model1 <- lm(nitrogen~longitude+latitude, data=data1)
summary(model1)
```

```

Call:
lm(formula = nitrogen ~ longitude + latitude, data = data1)

Residuals:
     Min       1Q   Median       3Q      Max
-0.20675 -0.10450 -0.01891  0.07505  0.58912

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -9.15204    9.15132  -1.000    0.323
longitude    0.05176    0.12442   0.416    0.679
latitude     0.36072    0.02756  13.090   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1371 on 46 degrees of freedom
```

```
Multiple R-squared:  0.8081,	Adjusted R-squared:  0.7997
F-statistic: 96.82 on 2 and 46 DF,  p-value: < 2.2e-16
```

The results show that latitude is significantly related but longitude is not, which is not surprising given the exploratory analysis above. However, to make sure the model is rotationally invariant we keep both variables in the model. To assess the normality of the residuals we draw a qq-plot.

```
qqnorm(residuals(model1))
```

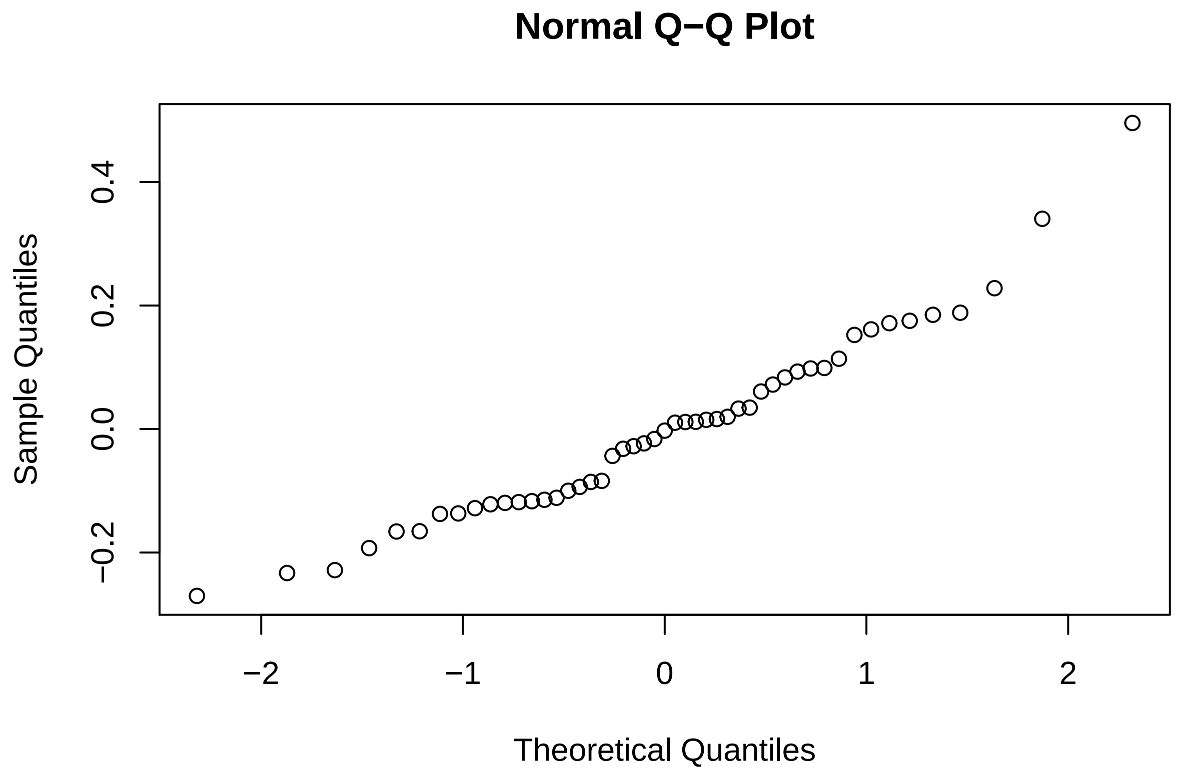


However, here we see the residuals look non-normal, which together with the density estimate in the exploratory analysis suggests a log transformation or similar may be beneficial. Therefore we re-fit the trend model on the log scale and reassess normality.

```
model2 <- lm(log(nitrogen)~longitude+latitude, data=data1)
summary(model2)


Call:
lm(formula = log(nitrogen) ~ longitude + latitude, data = data1)

Residuals:
     Min       1Q   Median       3Q      Max 
-0.27026 -0.11685 -0.00262  0.09296  0.49565 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept) -31.05565   10.31572  -3.011  0.00422 ** 
longitude    -0.13919    0.14025  -0.992  0.32620    
latitude      0.52239    0.03106  16.817  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 0.1546 on 46 degrees of freedom
Multiple R-squared:  0.8815,    Adjusted R-squared:  0.8764
F-statistic: 171.1 on 2 and 46 DF,  p-value: < 2.2e-16
```

```
qqnorm(residuals(model2))
```



The model diagnostics and the normality assumption for the residuals look more reasonable now, so we continue modelling on the log scale.

4. Assessing residual spatial autocorrelation

The next step is to examine the residuals from the model for the presence of spatial autocorrelation using variogram analysis. First, we have to create a *geodata* object of the residuals, which is achieved using the following code:

```
resid.dat <- data.frame(residuals=residuals(model2),
            longitude=data1$longitude, latitude=data1$latitude)
resid.geodata <- as.geodata(resid.dat, coords.col=2:3, data.col=1,
            borders=TRUE)
resid.geodata$borders <- data2
```

Next, an empirical semi-variogram with 95% Monte-Carlo envelopes can be constructed using the following code:

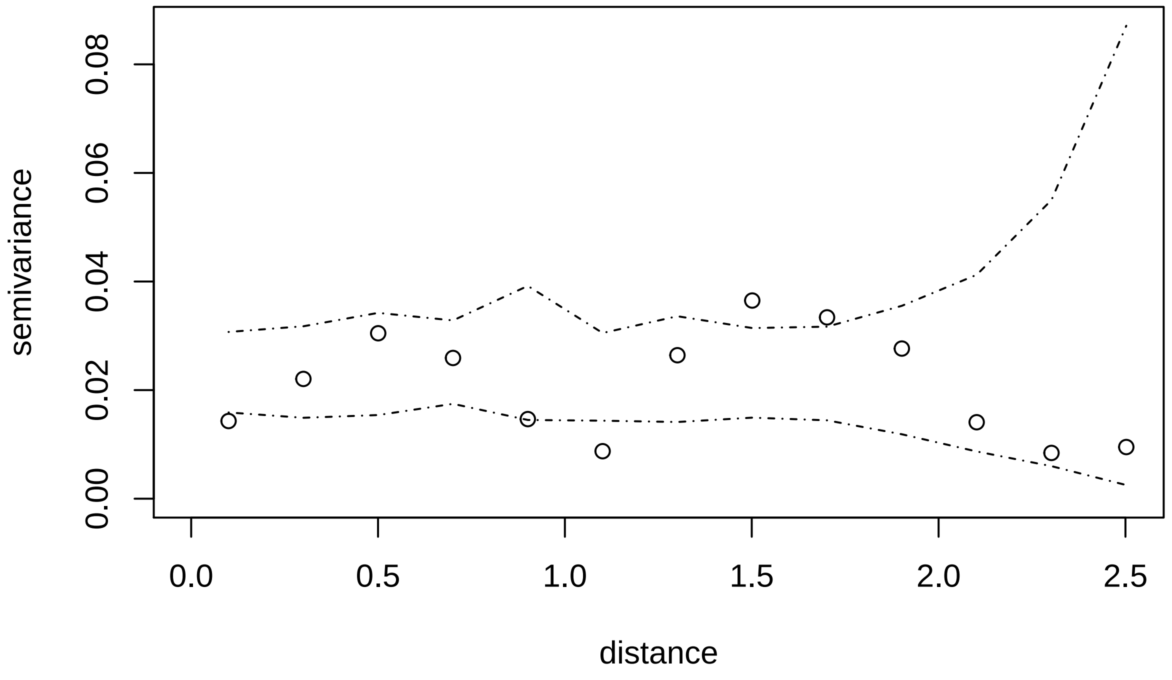
```
vari <- variog(resid.geodata)
```

```
variog: computing omnidirectional variogram
```

```
vari.mc <- variog.mc.env(resid.geodata, obj.variog=vari)
```

```
variog.env: generating 99 simulations by permutating data values
variog.env: computing the empirical variogram for the 99 simulations
variog.env: computing the envelops
```

```
plot(vari, envelope.obj=vari.mc)
```



The plot in general is slightly inconclusive, as the semi-variogram ordinates on the left side of the plot (small distances) lie on the border of the 95% Monte Carlo envelopes developed under independence. In fact if you re-run the code above a number of times you will find that sometimes the points are inside and sometimes outside the envelopes. Therefore as spatial autocorrelation is marginally present, it is worth accounting for it.

5. Fitting a spatial autocorrelation model to the data

The next step is to fit a spatial autocorrelation model to the data, and here we choose to fit a model with an exponential covariance function. The general model fitting function (using maximum likelihood) is *likfit()*, which has a number of key arguments.

- *geodata* - The `geodata` object that contains the data.
- *trend* - The trend model to assume.
- *ini.cov.pars* - Initial values for the partial sill and correlation range, which can be estimated from the empirical semi-variogram plot. From the plot the total variation (vertical axis) is around 0.04 and the range (horizontal axis) is around 0.5.
- *fix.nugget* - Should the nugget effect be fixed, e.g. at zero.
- *lambda* - The value for a Box-Cox transformation of the data. Here 0 corresponds to natural log which is what we will use.
- *cov.model* - The covariance model to use.

Thus the model can be fitted using the following code, where the *summary()* function prints out a summary of the model.

```
model3 <- likfit(geodata=data.geodata, trend="1st", ini.cov.pars=c(0.04, 0.5),
        fix.nugget=FALSE, lambda=0, cov.model="exponential")
```

```
kappa not used for the exponential correlation function
---------------------------------------------------------------
likfit: likelihood maximisation using the function optim.
likfit: Use control() to pass additional
         arguments for the maximisation function.
        For further details see documentation for optim.
likfit: It is highly advisable to run this function several
        times with different initial values for the parameters.
likfit: WARNING: This step can be time demanding!
---------------------------------------------------------------
likfit: end of numerical maximisation.
```

```
summary(model3)
```

```
Summary of the parameter estimation
-----------------------------------
Estimation method: maximum likelihood

Parameters of the mean component (trend):
   beta0    beta1    beta2
-34.5313  -0.1841   0.5245

Parameters of the spatial component:
   correlation function: exponential
      (estimated) variance parameter sigmasq (partial sill) =  0.0144
      (estimated) cor. fct. parameter phi (range parameter)  =  0.2381
   anisotropy parameters:
      (fixed) anisotropy angle = 0  ( 0 degrees )
      (fixed) anisotropy ratio = 1

Parameter of the error component:
      (estimated) nugget =  0.0066

Transformation parameter:
      (fixed) Box-Cox parameter = 0 (log-transformation)

Practical Range with cor=0.05 for asymptotic range: 0.7134337

Maximised Likelihood:
   log.L n.params      AIC      BIC
 "60.86"      "6" "-109.7" "-98.37"

non spatial model:
   log.L n.params      AIC      BIC
 "52.42"      "4" "-96.84" "-89.27"

Call:
likfit(geodata = data.geodata, trend = "1st", ini.cov.pars = c(0.04,
    0.5), fix.nugget = FALSE, lambda = 0, cov.model = "exponential")
```

The results from this model reveal a number of key points:

- The parameters of the trend model are similar to those estimated using *lm()* as expected.
- The partial sill ($\sigma^2 = 0.0144$), range ($\phi = 0.2381$) and nugget parameters ($\tau^2 = 0.0066$) are given.

- The asymptotic range where the correlation falls to 0.05 is around 0.7 degrees which looks reasonable from the empirical semi-variogram plot.
- The model fit criteria in terms of AIC and BIC show the fitted spatial model fits better than a non-spatial model.

6. Spatial Kriging

The next step is to predict the pollution concentrations across the watershed, and the first step is to set up a regular prediction grid. The basic idea is to first create a rectangular grid of prediction locations covering the grid, and then subset it to only use those predictions inside the study region boundary. The rectangular prediction grid can be set up with the following code:

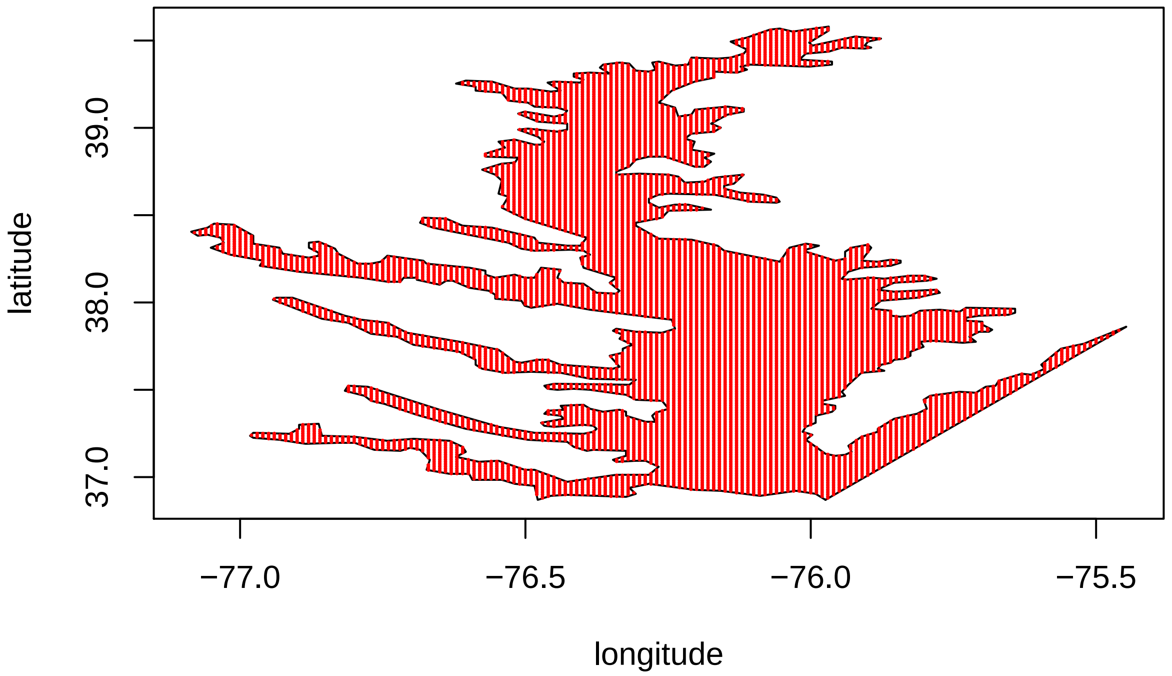
```
predgrid <- expand.grid(x=seq(-77.5, -75, 0.01), y=seq(36.8, 40.2, 0.01))
```

Then this regular grid can be subset to only include those elements that are inside the boundary using the function *in.out()* from the *mgcv* package.

```
library(mgcv)
which.in <- in.out(bnd=as.matrix(data.geodata$borders),x=as.matrix(predgrid))
predgrid2 <- predgrid[which.in, ]
```

These predictions locations can be viewed via the following code

```
plot(data.geodata$borders, type="l")
points(predgrid2, pch=19, col="red", cex=0.1)
```



The choice of the number of prediction points is up to the user, but there is a tradeoff between getting a detailed prediction surface and the time taken to make the predictions. Once the prediction grid has been set up, the predictions can be made using the *krige.conv()* function. However, before that you need to specify some extra options for the Kriging operation via the *krige.control()* function. These are:

- *type.krige* - The type of Kriging to use, here Ordinary Kriging.
- *trend.d* - The trend model to assume for the data locations, here “1st” for a linear trend.
- *trend.l* - The trend model to assume for the prediction locations, here “1st” for a linear trend.
- *obj.model* - The geostatistical model object used to fit the data.

- *lambda* - The value for a Box-Cox transformation of the data. Here 0 corresponds to natural log which is what we will use.

```
control <- krige.control(type.krige="OK", trend.d="1st", trend.l="1st",
            obj.model=model3, lambda=0)
```

Then Kriging can be done using the following code

```
kriging <- krige.conv(geodata=data.geodata, locations=predgrid2, krige=control)
```

```
krige.conv: results will be returned only for prediction locations inside the borders
krige.conv: model with mean given by a 1st order polynomial on the coordinates
krige.conv: performing the Box-Cox data transformation
krige.conv: back-transforming the predicted mean and variance
krige.conv: Kriging performed using global neighbourhood
```

where the arguments are the *geodata* object, the prediction locations, and the options from the *krige.control()* function. The *kriging* object contains the following main elements:

```
summary(kriging)
```

```
             Length Class  Mode
predict      12424  -none- numeric
krige.var    12424  -none- numeric
beta.est         3  -none- numeric
distribution     1  -none- character
message          1  -none- character
call             4  -none- call
```

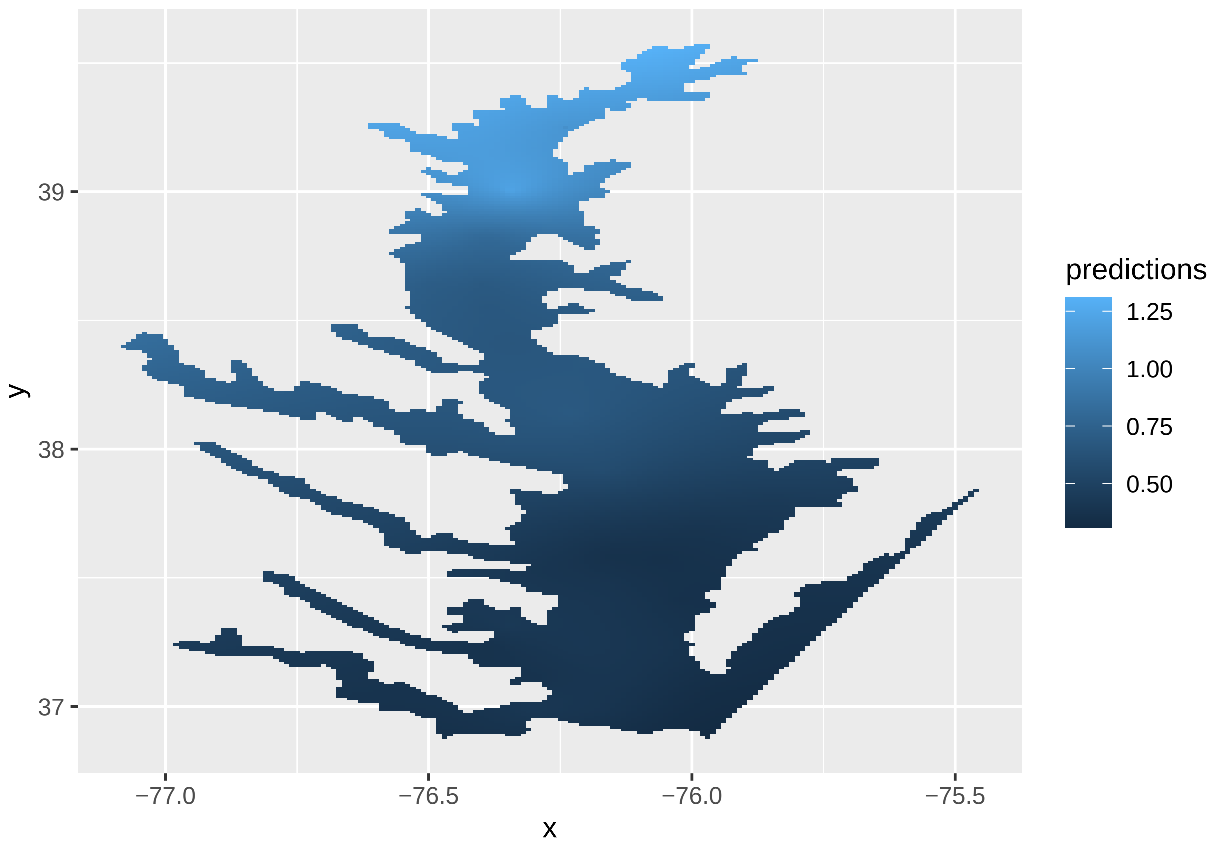
- *predict* - The predictions made.
- *krige.var* - The variances of the predictions made.

The final step is to visualise both surfaces, which can be done using the *ggplot2* package. This package operates on *data.frame* objects, so we first need to create a *data.frame* of the prediction locations, the predictions and their standard deviations. This can be done as

```
pred.data <- data.frame(x=predgrid2[ ,1], y=predgrid2[ ,2],
            predictions=kriging$predict, sd=sqrt(kriging$krige.var))
```

Next the predictions can be visualised using the *ggplot2* package, which in its simplest form is achieved by:

```
library(ggplot2)
ggplot(aes(x = x, y = y), data = pred.data) +
    geom_tile(aes(fill = predictions))
```

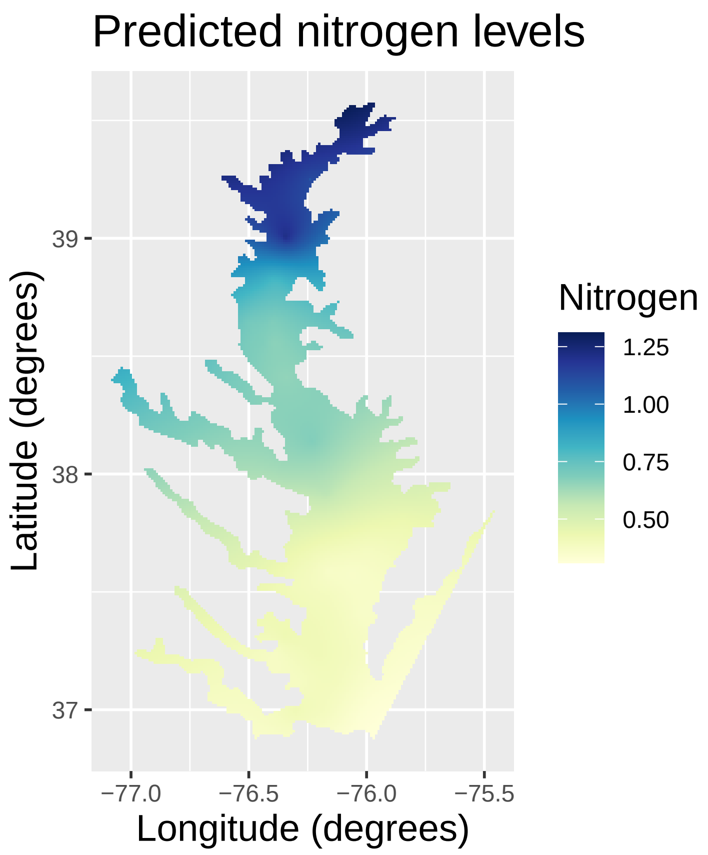


which gives a rather horrible map. Here

- *ggplot()* - specifies the data frame, the two variables (columns) to be used to create the plotting area.
- *geom_tile()* - adds the variable to be mapped.

However, this map is unsatisfactory in a number of ways, and can be improved by adding additional commands to the *ggplot()* function separated by the + sign as shown below.

```
library(RColorBrewer)
ggplot(aes(x = x, y = y), data = pred.data) +
    geom_tile(aes(fill = predictions)) +
    coord_equal() +
    xlab("Longitude (degrees)") +
    ylab("Latitude (degrees)") +
    labs(title = "Predicted nitrogen levels", fill = "Nitrogen") +
    theme(title = element_text(size=14)) +
    scale_fill_gradientn(colors=brewer.pal(n=9, name="YlGnBu"))
```

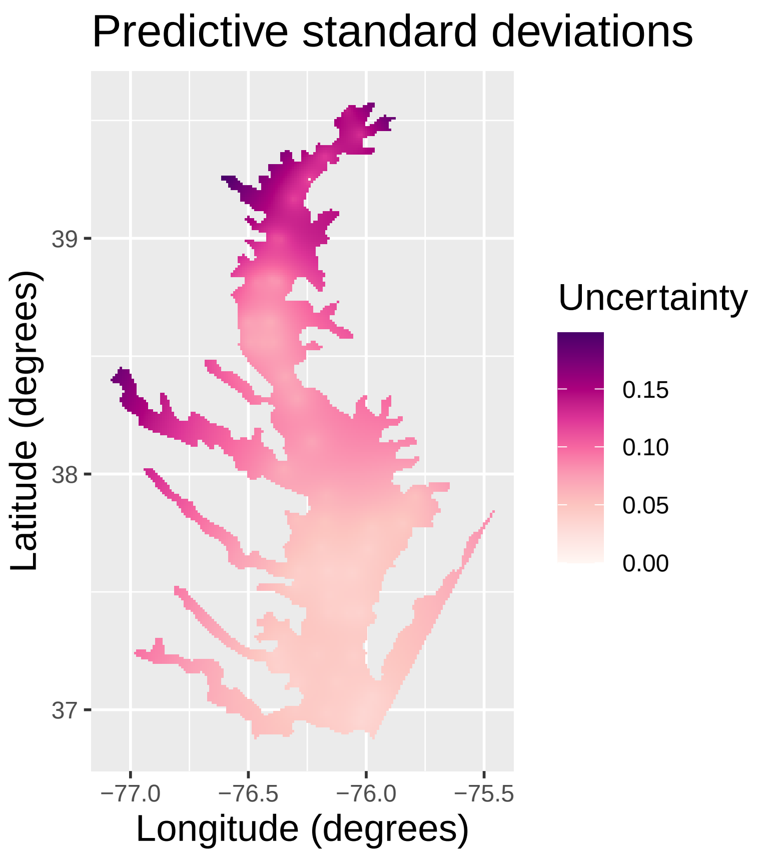


Here the new lines make the following changes:

- *coord_equal()* - makes sure that 1 unit in the x axis direction is the same size as 1 unit in the y axis direction.
- *xlab()* - specifies the horizontal axis label for the plot.
- *ylab()* - specifies the vertical axis label for the plot.
- *labs()* - adds titles to the plot and to the colour key.
- *theme()* - changes the typeface and size of the text on the plot.
- *scale_fill_gradientn()* - changes the colours using the colour palette from the *RColorBrewer* library. Possible colour schemes are available from http://colorbrewer2.org/.

Finally, the prediction standard deviation can be mapped using the following code:

```
ggplot(aes(x = x, y = y), data = pred.data) +
    geom_tile(aes(fill = sd)) +
    coord_equal() +
    xlab("Longitude (degrees)") +
    ylab("Latitude (degrees)") +
    labs(title = "Predictive standard deviations", fill = "Uncertainty") +
    theme(title = element_text(size=14)) +
    scale_fill_gradientn(colors=brewer.pal(n=9, name="RdPu"))
```



7. Comparing the predictive accuracy of different models

Here we compare the exponential covariance model we have used above with the spherical covariance model, to see which gives the best predictive performance. To do this we employ a leave-one-out cross validation exercise as discussed in class. First, we create a matrix to store the predictions.

```
m <- nrow(data1)
results <- array(NA, c(m,3))
colnames(results) <- c("data", "exponential", "spherical")
results[ ,1] <- data1$nitrogen
```

Here the real values are stored in column 1, whilst the predicted values are stored in columns 2 (exponential model) and 3 (spherical model). Then the leave-one-out cross validation exercise is implemented by the code below, which uses a *for()* loop to predict each observation in turn from a model fitted to all the remaining observations.

```
for(i in 1:m)
{
    ## Set up the test data
    test <- as.geodata(data1[-i, ], coords.col=2:3, data.col=4, borders=TRUE)

    ## Run the exponential model
    model.exp <- likfit(geodata=test, trend="1st", ini.cov.pars=c(0.04, 0.5),
                        fix.nugget=FALSE, lambda=0, cov.model="exponential")
    control <- krige.control(type.krige="OK", trend.d="1st", trend.l="1st",
                             obj.model=model.exp, lambda=0)
    kriging <- krige.conv(geodata=test, locations=data1[i, 2:3], krige=control)
    results[i, 2] <- kriging$predict
    ## Run the spherical model
    model.sph <- likfit(geodata=test, trend="1st", ini.cov.pars=c(0.04, 0.5),
                        fix.nugget=FALSE, lambda=0, cov.model="spherical")
    control <- krige.control(type.krige="OK", trend.d="1st", trend.l="1st",
                             obj.model=model.sph, lambda=0)
    kriging <- krige.conv(geodata=test, locations=data1[i, 2:3], krige=control)
    results[i, 3] <- kriging$predict
}
```

Then compute the root mean square prediction error for each model as follows.

```
## Exponential model
sqrt(mean((results[ ,1] - results[ ,2])^2))
```

```
[1] 0.09366253
```

```
## Spherical model
sqrt(mean((results[ ,1] - results[ ,3])^2))
```

```
[1] 0.09257702
```

Thus, you can see that in terms of predictive performance, the spherical model is marginally better than the exponential model.