Surface Interpolation in R

```
install.packages(c("ggplot2", "gstat", "sp", "maptools"), contriburl =
"http://cran.ma.imperial.ac.uk/")
```

```
## Installing packages into 'C:/Users/Adam/Documents/R/win-library/3.0' (as
## 'lib' is unspecified)
```

```
## Warning: cannot open: HTTP status was '404 Not Found' Warning: cannot
## open: HTTP status was '404 Not Found' Warning: unable to access index for
## repository http://cran.ma.imperial.ac.uk/ Warning: packages 'ggplot2',
## 'gstat', 'sp', 'maptools' are not available (for R version 3.0.1)
```

```
setwd("C:/Users/Adam/Dropbox/R/Practical8")

library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.0.2
```

```
library(gstat)
```

```
## Warning: package 'gstat' was built under R version 3.0.2
```

```
library(sp)
```

```
## Warning: package 'sp' was built under R version 3.0.2
```

```
library(maptools)
```

```
## Warning: package 'maptools' was built under R version 3.0.2
```

```
## Checking rgeos availability: TRUE
```

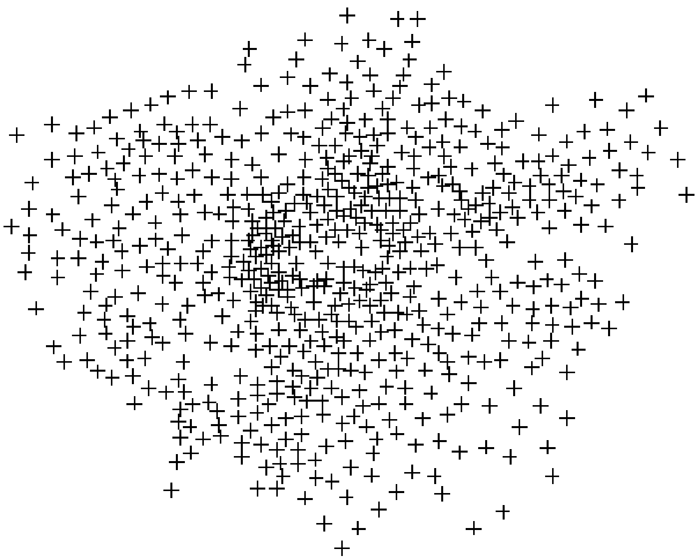
Your London Wards dataframe should already have x (British National Grid Eeastings) and y (Northings) data for the geometric centroid of each ward attached to it. Read in this data frima and then convert it to a spatial points data frame using the coordinates function in the sp package.

```
# read in some data that already has x and y coordinates attached to it
LondonWards1 <- read.csv("LondonData.csv")

# In our case the coordinates columns are already called x and y, so we will
# use these

# convert this basic data frame into a spatial points data frame
coordinates(LondonWards1) = ~x + y

plot(LondonWards1)
```

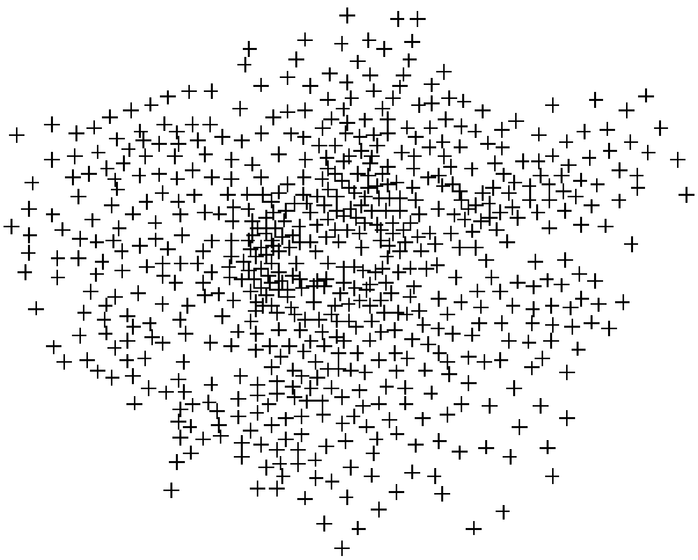


We will now create a grid onto which we will interpolate - this can be done in 2 ways. The first will set the grid using the range of the points data. This is OK, but our interpolation grid will not cover the full extent of our area.

```
## 1. Create a grid from the values in your points dataframe first get the
## range in data
x.range <- as.integer(range(LondonWards1@coords[, 1]))
y.range <- as.integer(range(LondonWards1@coords[, 2]))
```

The second method lets you set your own grid extent using the locator() function. After entering (4) into the function, you should click the four corners of a bounding box larger than the extent of London. These four points will be printed to the console and you can then enter them into your x.range and y.range variables:

```
## 2. Create a grid with a slightly larger extent
plot(LondonWards1)
# use the locator to click 4 points beyond the extent of the plot and use
# those to set your x and y extents
locator(4)
```



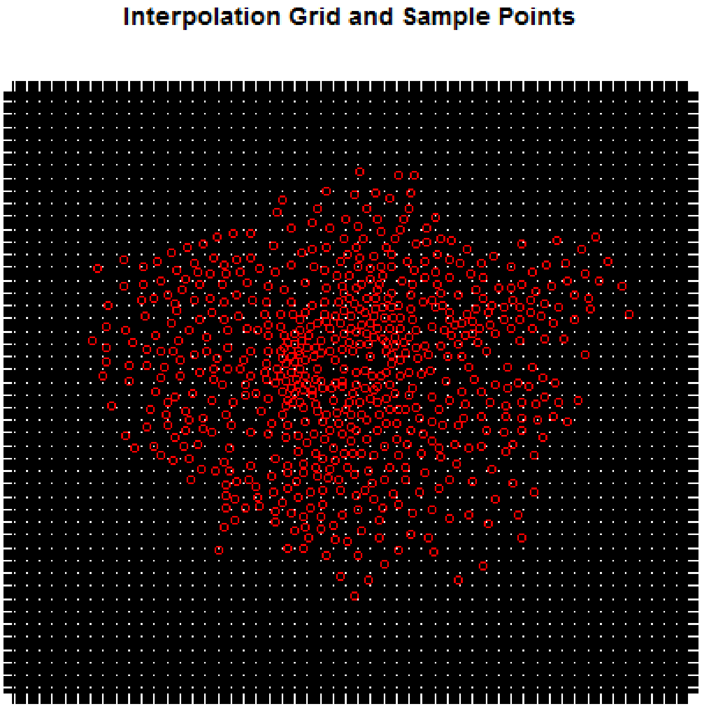
```
x.range <- as.integer(c(497509.5, 563596.8))
y.range <- as.integer(c(148358.5, 207326.7))
```

We will now create a grid with the extent set to our x and y ranges. It is at this point that we set the resolution of the grid. Higher resolution (smaller grid cells) will create a smoother looking grid, but will involve more computation. As British National Grid is in metres, the range is set in metres.

```
## now expand your range to a grid with spacing that you'd like to use in
## your interpolation here we will use 200m grid cells:
grd <- expand.grid(x = seq(from = x.range[1], to = x.range[2], by = 200), y =
seq(from = y.range[1],
    to = y.range[2], by = 200))

## convert grid to SpatialPixel class
coordinates(grd) <- ~x + y
gridded(grd) <- TRUE

## test it out - this is a good way of checking that your sample points are
## all well within your grid. If they are not, try some different values in
## you r x and y ranges:
plot(grd, cex = 1.5)
points(LondonWards1, pch = 1, col = "red", cex = 1)
title("Interpolation Grid and Sample Points")
```



Inverse Distance Weighting

Now we have set up our points and a grid to interpolate onto, we are ready carry out some interpolation. The first method we will try is inverse distance weighting (IDW) as this will not require any special modelling of spatial relationships.

To generate a surface using inverse distance weighting, use the IDW function in gstat. Check the help file for IDW - `?idw` - for information about what this formula is doing.

The surface being generated here smooths the average GCSE score for individual Wards across the whole London area. Feel free to experiment with smoothing alternative variables.

```
idw <- idw(formula = AvgGCSE2011 ~ 1, locations = LondonWards1, newdata = grd)
```

```
## [inverse distance weighted interpolation]
```

```
idw.output = as.data.frame(idw)
names(idw.output)[1:3] <- c("long", "lat", "var1.pred")
```

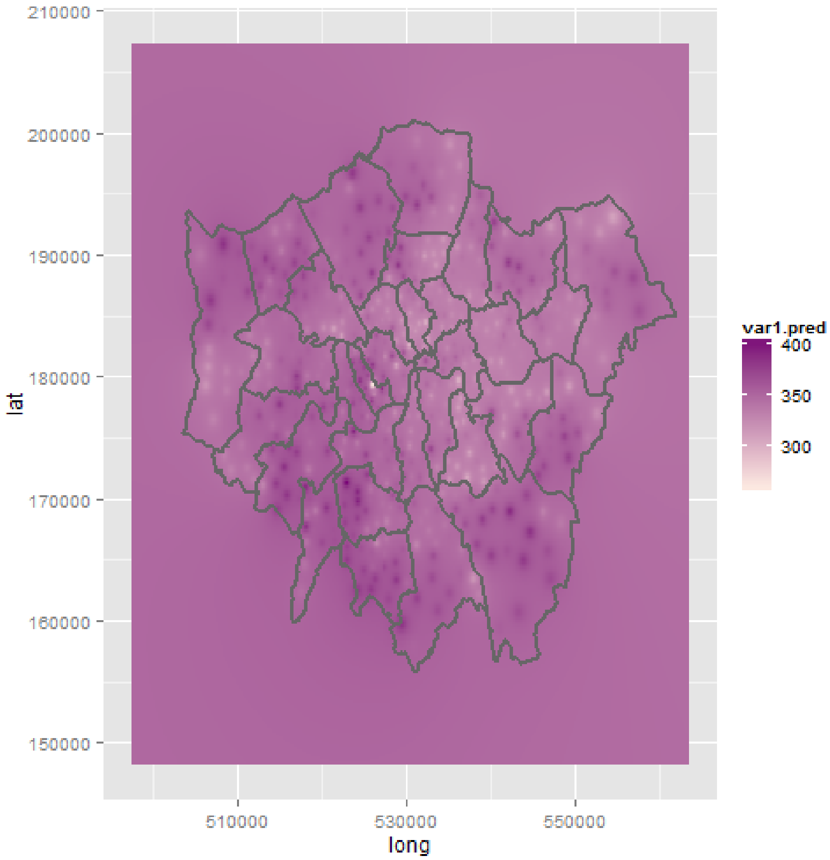
Now we have our IDW output we can plot it using ggplot2. First, however, read in some outline boundary data for London Boroughs in order to contextualise your results.

```
boroughs <- readShapePoly("C:/Users/Adam/Boundary
Data/LondonBoroughs/london_boroughs.shp")
boroughoutline <- fortify(boroughs, region = "name")
```

```
## Loading required package: rgeos rgeos version: 0.2-19, (SVN revision 394)
## GEOS runtime version: 3.3.8-CAPI-1.7.8 Polygon checking: TRUE
```

Now plot your IDW results and your boundary layer

```
plot <- ggplot(data = idw.output, aes(x = long, y = lat))  #start with the base-plot
layer1 <- c(geom_tile(data = idw.output, aes(fill = var1.pred)))  #then create a tile
layer and fill with predicted values
layer2 <- c(geom_path(data = boroughoutline, aes(long, lat, group = group),
    colour = "grey40", size = 1))  #then create an outline layer
# now add all of the data together
plot + layer1 + layer2 + scale_fill_gradient(low = "#FEEBE2", high = "#7A0177")
```



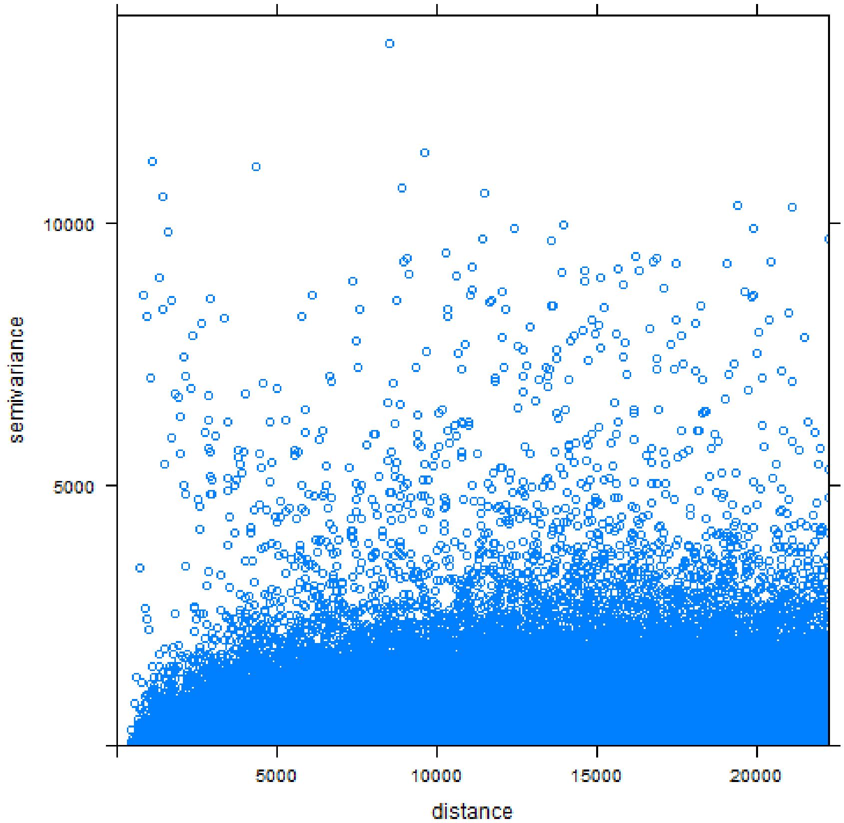
You might want to experiment with a few of the parameters in IDW or different grid resolutions.

Kriging

As you heard in the lecture, Kriging is a little more involved than IDW as it requires the construction of a semivariogram model to describe the spatial autocorrelation pattern for your particular variable.

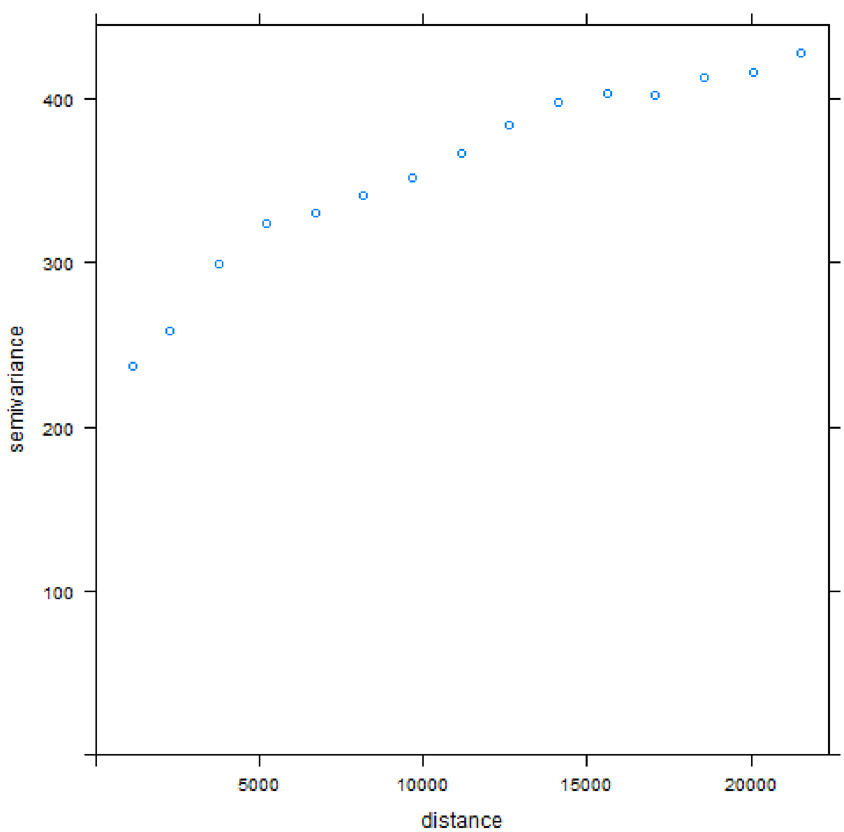
We'll start with a variogram cloud for the Average GCSE Scores

```
variogcloud <- variogram(AvgGCSE2011 ~ 1, locations = LondonWards1, data =
LondonWards1,
    cloud = TRUE)
plot(variogcloud)
```



The values in the cloud can be binned into lags with and plotted with a very similar function

```
semivariog <- variogram(AvgGCSE2011 ~ 1, locations = LondonWards1, data =
LondonWards1)
plot(semivariog)
```




```
semivariog
```

```
##       np  dist gamma dir.hor dir.ver   id
## 1    883  1159 237.7       0       0 var1
## 2   3419  2296 258.6       0       0 var1
## 3   5304  3758 299.4       0       0 var1
## 4   7049  5237 323.9       0       0 var1
## 5   8499  6712 330.7       0       0 var1
## 6   9967  8193 341.4       0       0 var1
## 7  10998  9676 352.1       0       0 var1
## 8  11803 11156 367.2       0       0 var1
## 9  12287 12635 383.7       0       0 var1
## 10 12558 14117 397.7       0       0 var1
## 11 12487 15602 403.6       0       0 var1
## 12 12243 17088 401.8       0       0 var1
## 13 11654 18573 412.8       0       0 var1
## 14 11008 20054 415.9       0       0 var1
## 15 10165 21536 428.5       0       0 var1
```

From the empirical semivariogram plot and the information contained in the semivariog gstat object, we can estimate the sill, range and nugget to use in our model semivariogram.

In this case, the range (the point on the distance axis where the semivariogram starts to level off) is around the value of the last lag - 21535.773 - so we'll use Range = 21500

The Sill (the point on the y axis where the semivariogram starts to level off) is around 420.

The nugget looks to be around 200 (so the partial sill is around 220).

Using this information we'll generate a model semivariogram using the vgm() function in gstat.

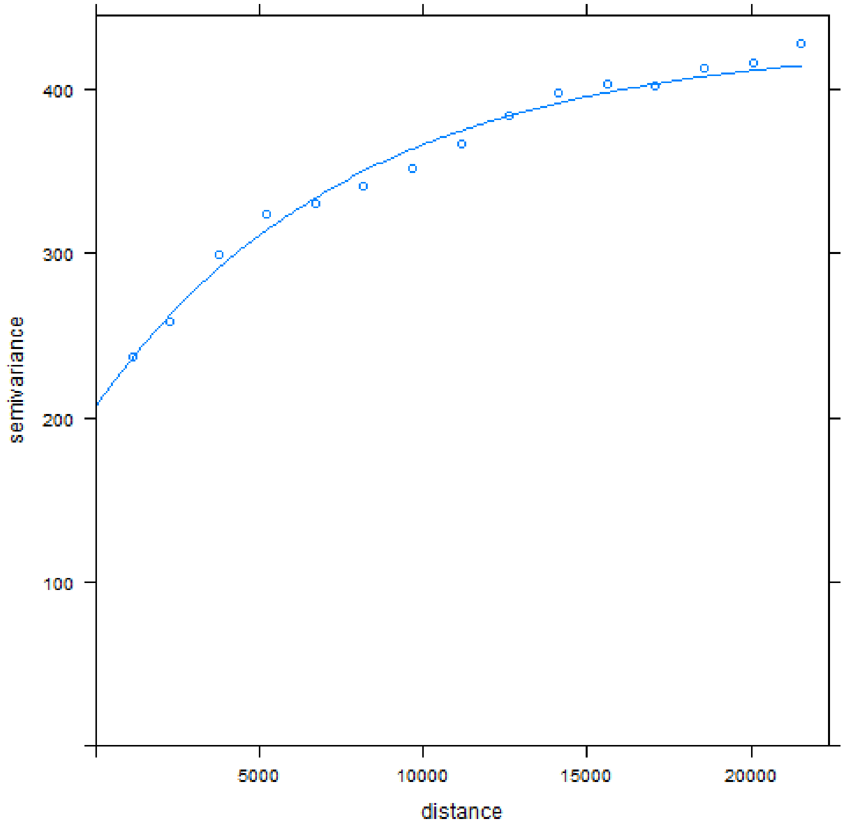
```
# first check the range of model shapes available in vgm
vgm()
```

```
##    short                                      long
## 1    Nug                              Nug (nugget)
## 2    Exp                         Exp (exponential)
## 3    Sph                           Sph (spherical)
## 4    Gau                            Gau (gaussian)
## 5    Exc               Exclass (Exponential class)
## 6    Mat                              Mat (Matern)
## 7    Ste Mat (Matern, M. Stein's parameterization)
## 8    Cir                            Cir (circular)
## 9    Lin                              Lin (linear)
## 10   Bes                              Bes (bessel)
## 11   Pen                      Pen (pentaspherical)
## 12   Per                            Per (periodic)
## 13   Wav                                Wav (wave)
## 14   Hol                                Hol (hole)
## 15   Log                         Log (logarithmic)
## 16   Pow                               Pow (power)
## 17   Spl                              Spl (spline)
## 18   Leg                            Leg (Legendre)
## 19   Err                   Err (Measurement error)
## 20   Int                           Int (Intercept)
```

```
# the data looks like it might be an exponential shape, so we will try that
# first with the values estimated from the empirical
model.variog <- vgm(psill = 220, model = "Exp", nugget = 200, range = 21500)
```

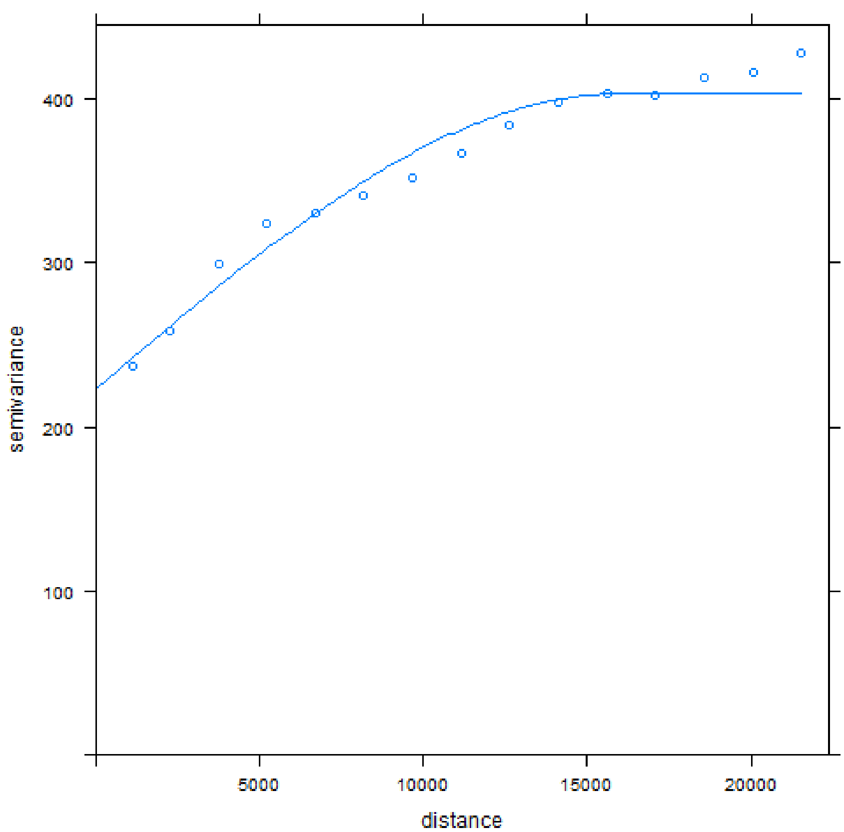
We can now fit this model to a sample variogram to see how well it fits and plot it

```
fit.variog <- fit.variogram(semivariog, model.variog)
plot(semivariog, fit.variog)
```

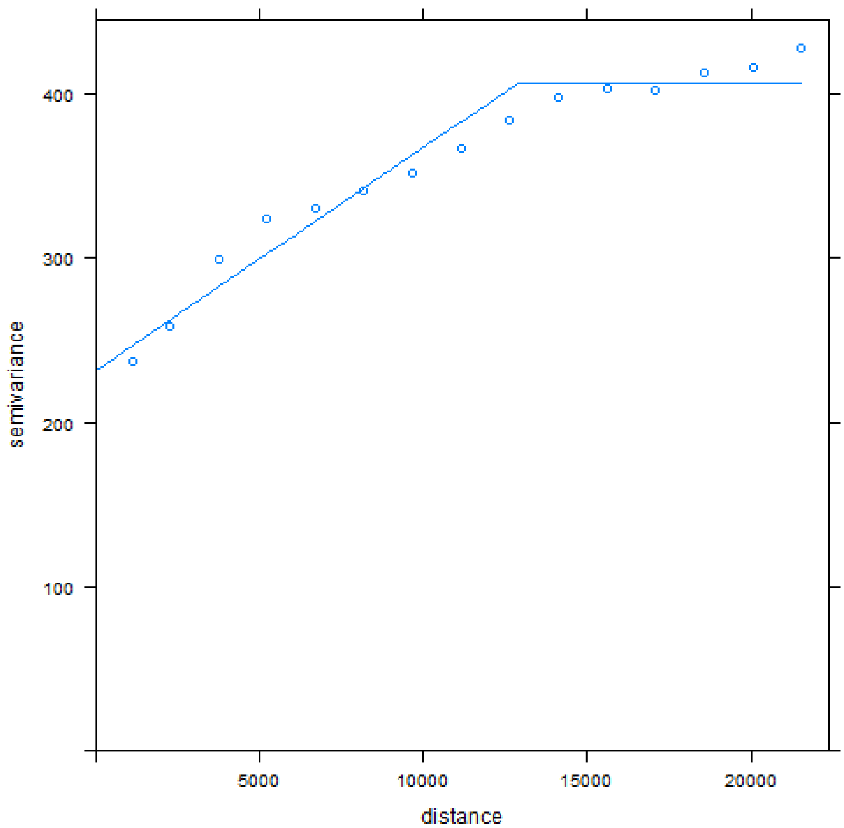


If you like, try some alternative models to see if the fit is any better

```
model.variog <- vgm(psill = 220, model = "Sph", nugget = 200, range = 21500)
fit.variog <- fit.variogram(semivariog, model.variog)
plot(semivariog, fit.variog)
```



```
model.variog <- vgm(psill = 220, model = "Lin", nugget = 200, range = 21500)
fit.variog <- fit.variogram(semivariog, model.variog)
plot(semivariog, fit.variog)
```



The original exponential model seems like a good fit, so we will proceed with that model

```
model.variog <- vgm(psill = 220, model = "Exp", nugget = 200, range = 21500)
```

Use the krige() function in gstat along with the model semivariogram just generated to generate an ordinary/simple Kriged surface - again, check `?krige` to see what the various options in the function are.

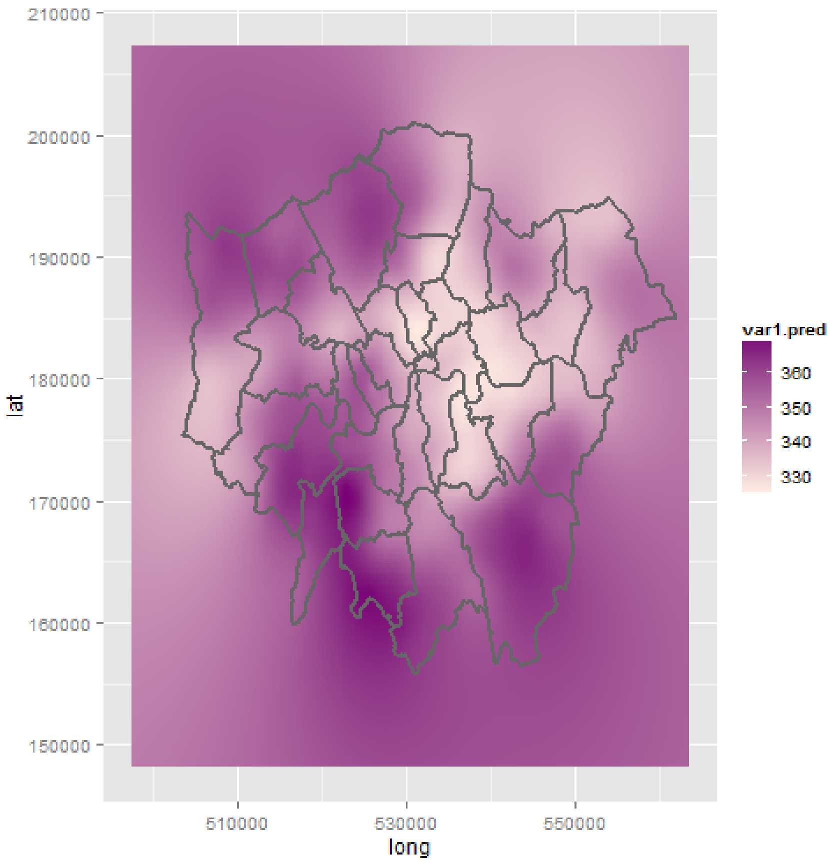
```
krig <- krige(formula = AvgGCSE2011 ~ 1, locations = LondonWards1, newdata = grd,
    model = model.variog)
```

```
## [using ordinary kriging]
```

```
krig.output = as.data.frame(krig)
names(krig.output)[1:3] <- c("long", "lat", "var1.pred")
```

Generate a plot of the kriged surface in ggplot2 in a similar way to before

```
plot <- ggplot(data = krig.output, aes(x = long, y = lat))  #start with the base-plot
and add the Kriged data to it
layer1 <- c(geom_tile(data = krig.output, aes(fill = var1.pred)))  #then create a
tile layer and fill with predicted
layer2 <- c(geom_path(data = boroughoutline, aes(long, lat, group = group),
    colour = "grey40", size = 1))  #then create an outline
plot + layer1 + layer2 + scale_fill_gradient(low = "#FEEBE2", high = "#7A0177")
```



This completes your very short guide to creating spatial surfaces in R. Using your new knowledge about constucting and interpreting semivariograms and, you should try and replicate these surfaces in ArcGIS.

In Spatial Analyst Tools > Interpolation, there are various tools that will carry out IDW, Kriging and a suite of other surface interpolations. In Geostatistical Analyst Tools > Interpolation there are even more tools which will do the same thing in a slightly different way. You may wish to try out cross-validation (in Geostatistical Analyst Tools > Utilities) to check your results.

When in ArcGIS, you can construct semivariograms using the Geostatistical Analyst Tool bar. Go to Customise > Toolbars > Geostatistical Analyst. Then in the toolbar, click the Geostatistical Analyst Wizard to start analysing your data. A guide to semivariogram analysis in ArcGIS can be found here:
http://resources.arcgis.com/en/help/main/10.1/index.html#/Fitting_a_model_to_the_empirical_semivariogram/0031000000n4000000/