Analysing Spatial Data in R: Worked example: geostatistics

Roger Bivand

Department of Economics
Norwegian School of Economics and Business Administration
Bergen, Norway

31 August 2007

Worked example: geostatistics

- Geostatistics is a bit like the alchemy of spatial statistics, focussed more on prediction than model fitting
- Since the reason for modelling is chiefly prediction in pre-model-based geostatistics, and to a good extent in model-based geostatistics, we'll also keep to interpolation here
- Interpolation is trying to make as good guesses as possible of the values of the variable of interest for places where there are no observations (can be in 1, 2, 3, ... dimensions)
- These are based on the relative positions of places with observations and places for which predictions are required, and the observed values at observations

Geostatistics packages

- The **gstat** package provides a wide range of functions for univariate and multivariate geostatistics, also for larger datasets, while **geoR** and **geoRglm** contain functions for model-based geostatistics
- A similar wide range of functions is to be found in the **fields** package. The **spatial** package is available as part of the **VR** bundle (shipped with base R), and contains several core functions
- The **RandomFields** package provides functions for the simulation and analysis of random fields. For diagnostics of variograms, the **vardiag** package can be used
- The **sgeostat** package is also available; within the same general topical area are the **tripack** for triangulation and the **akima** package for spline interpolation

Meuse soil data

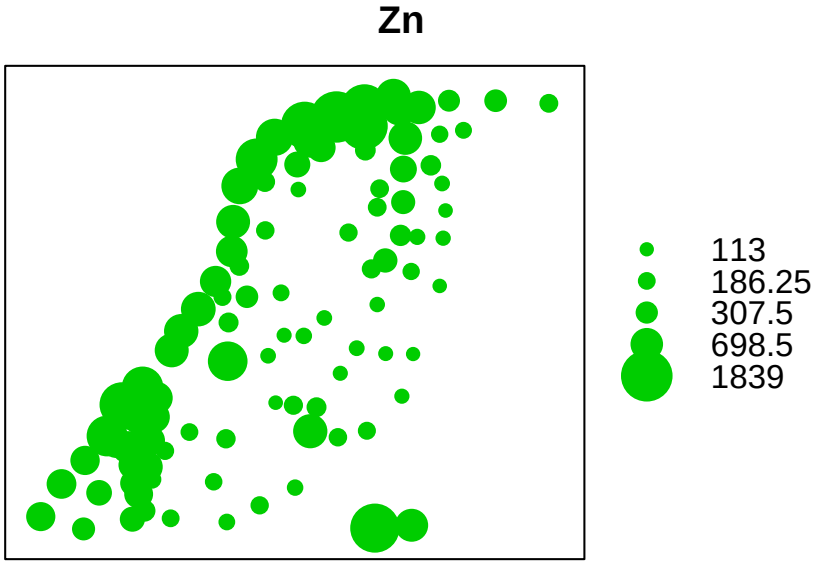
- The Maas river bank soil pollution data (Limburg, The Netherlands) are sampled along the Dutch bank of the river Maas (Meuse) north of Maastricht; the data are those used in Burrough and McDonnell (1998, pp. 309–311)
- These are a subset of the data provided with **gstat** and **sp**, but here we use the same subset as the very well regarded GIS textbook, in case cross-checking is of interest
- The data used here are a shapefile named BMcD.shp with its data table with the zinc ppm measurements we are interested in interpolating, and an ASCII grid of flood frequencies for the part of the river bank we are interested in, giving the prediction locations

Reading the data

```
> library(rgdal)
> BMcD <- readOGR(".", "BMcD")

OGR data source with driver: ESRI Shapefile
Source: ".", layer: "BMcD"
with  98  rows and  15  columns

> BMcD$Fldf <- factor(BMcD$Fldf)
> names(BMcD)

 [1] "x"       "y"       "xl"
 [4] "yl"      "elev"    "d_river"
 [7] "Cd"      "Cu"      "Pb"
[10] "Zn"      "LOI"     "Fldf"
[13] "Soil"    "lime"    "landuse"

> proj4string(BMcD) <- CRS("+init=epsg:28992")
```

Although rgdal is used here, the maptools function readShapePoints could be used. Since a variable of interest — flood frequency — is a categorical variable but read as numeric, it is set to factor

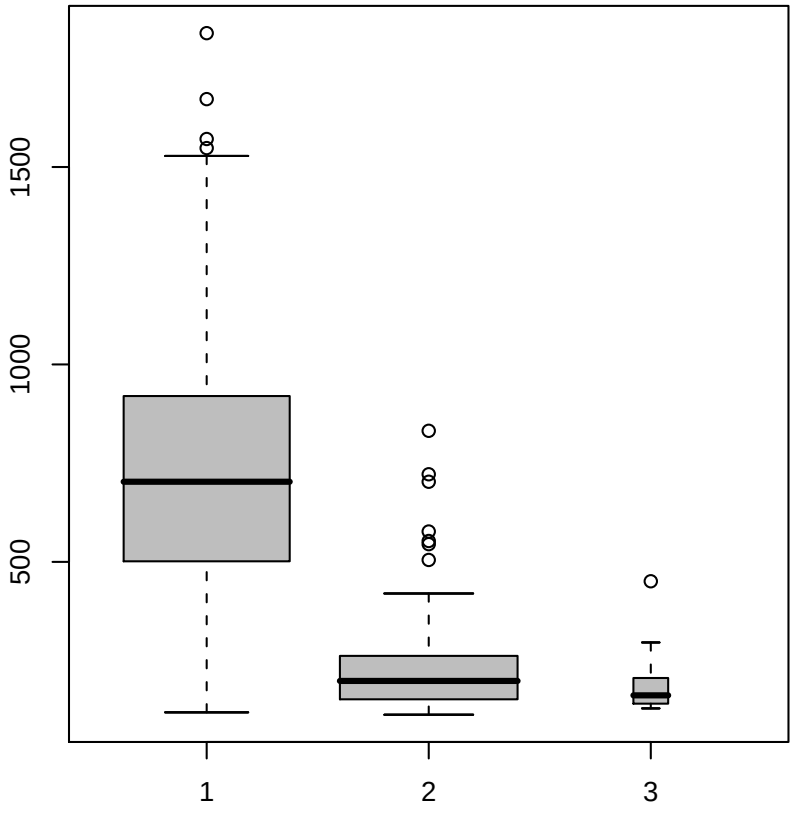
Observed zinc ppm levels



The zinc ppm values are rather obviously higher near the river bank to the west, and at the river bend in the south east; the pollution is from upstream industry in the watershep, and is deposited in silt during flooding

```
> bubble(BMcD, "Zn")
```

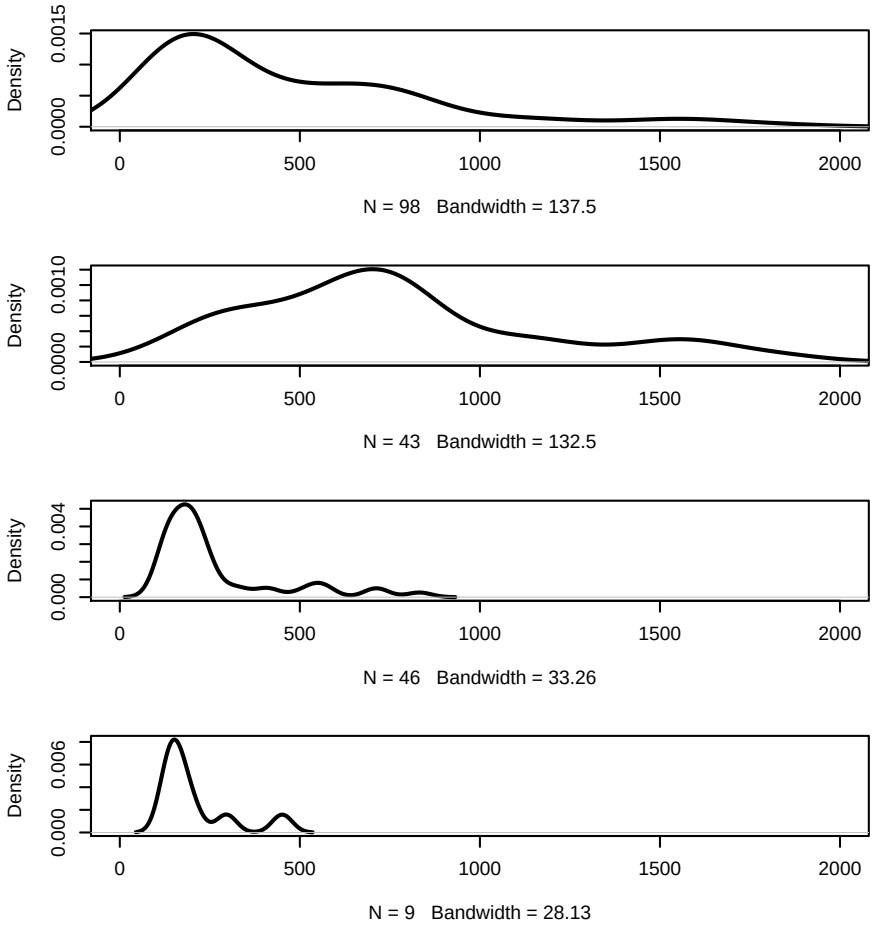
Flood frequency boxplots



Boxplots of the zinc ppm values by flood frequency suggest that the apparent skewness of the values may be related to heterogeneity in environmental "drivers"

```
> boxplot(Zn ~ Fldf, BMcD, width = table(BMcD$Fldf),
+     col = "grey")
```

Densities of zinc ppm



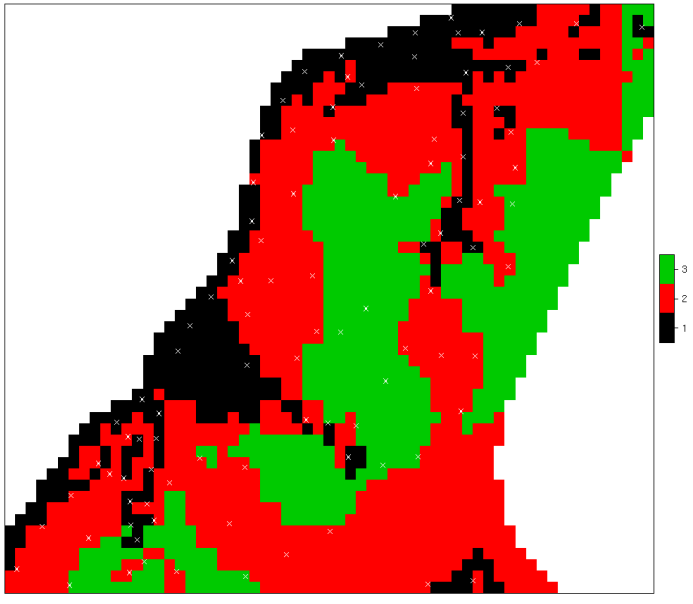
This impression is supported by dividing density plots up into one pooled, and three separate flood frequency classes — the at least annual flooding class has higher values than the others

```
> plot(density(BMcD$Zn), main = "",
+     xlim = c(0, 2000), lwd = 2)
> by(as(BMcD, "data.frame"), BMcD$Fldf,
+     function(x) plot(density(x$Zn),
+         main = "", xlim = c(0,
+             2000), lwd = 2))
```

Reading the prediction locations

Reading the prediction locations:

```
> BMcD_grid <- as(readGDAL("BMcD_fldf.txt"),
+     "SpatialPixelsDataFrame")

BMcD_fldf.txt has GDAL driver AAIGrid
and has 52 rows and 61 columns

> names(BMcD_grid) <- "Fldf"
> BMcD_grid$Fldf <- as.factor(BMcD_grid$Fldf)
> proj4string(BMcD_grid) <- CRS("+init=epsg:28992")

> pts = list("sp.points", BMcD,
+     pch = 4, col = "white")
> spplot(BMcD_grid, "Fldf", col.regions = 1:3,
+     sp.layout = list(pts))
```

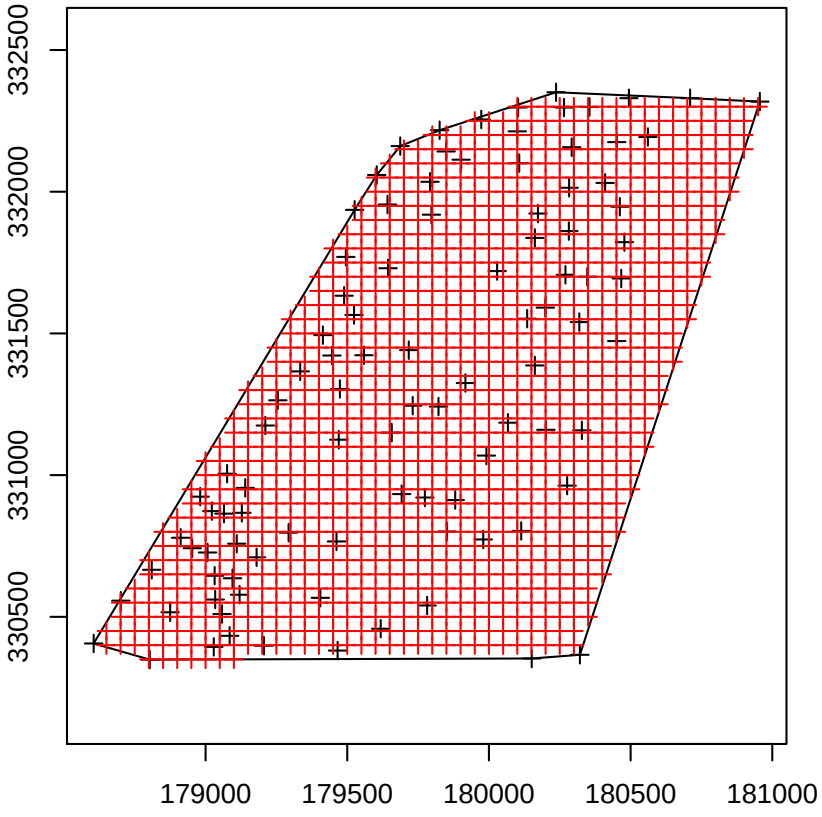


Roll-your-own boundaries

In case there are no such study area boundaries for prediction, we can make some:

```
> crds <- coordinates(BMcD)
> poly <- crds[chull(crds), ]
> poly <- rbind(poly, poly[1, ])
> SPpoly <- SpatialPolygons(list(Polygons(list(Polygon(poly)), ID = "poly")))
> bbox(BMcD)

             min    max
coords.x1 178605 180956
coords.x2 330349 332351

> (apply(bbox(BMcD), 1, diff)%/%50) + 1

coords.x1 coords.x2
       48        41

> grd <- GridTopology(c(178600, 330300), c(50, 50), c(48, 41))
> SG <- SpatialGrid(grd)
> inside <- overlay(SG, SPpoly)
> SGDF <- SpatialGridDataFrame(grd, data = data.frame(list(ins = inside)))
> SPDF <- as(SGDF, "SpatialPixelsDataFrame")
```

Roll-your-own boundaries



Plotting the new boundaries shows how flexible the overlay method and the SpatialPixels class can be

```
> plot(BMcD, axes = TRUE)
> plot(SPpoly, add = TRUE)
> plot(SPDF, col = "red", add = TRUE)
```

Set up class intervals and palettes

Setting up class intervals and palettes initially will save time later; note the use of `colorRampPalette`, which can also be specified from **RColorBrewer** palettes:

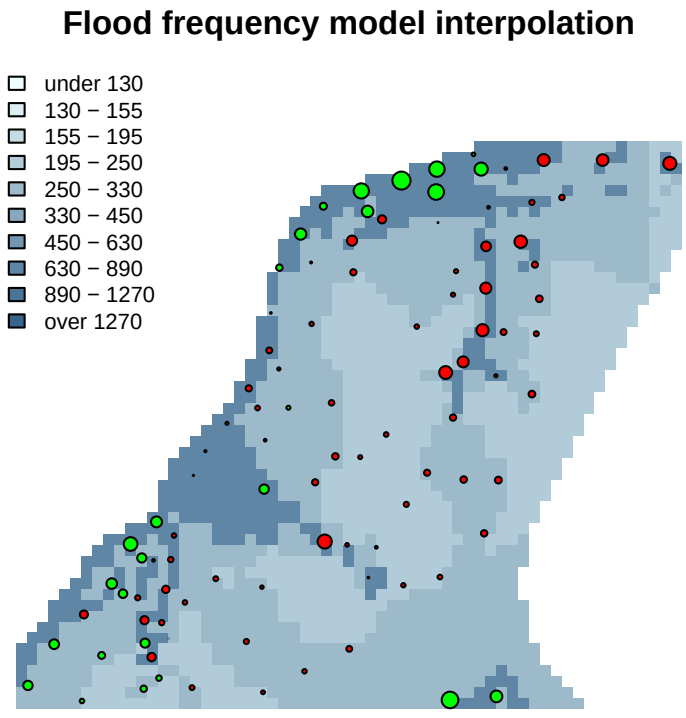
```
> bluepal <- colorRampPalette(c("azure1", "steelblue4"))
> brks <- c(0, 130, 155, 195, 250, 330, 450, 630, 890, 1270, 1850)
> cols <- bluepal(length(brks) - 1)
> sepal <- colorRampPalette(c("peachpuff1", "tomato3"))
> brks.se <- c(0, 240, 250, 260, 270, 280, 290, 300, 350, 400, 1000)
> cols.se <- sepal(length(brks.se) - 1)
> scols <- c("green", "red")
```

Aspatial flood frequency model

Since we have seen how the zinc ppm values seem to be distributed in relationship to flood frequencies, and because we have flood frequencies for the prediction locations, we can start with a null model, then an aspatial model (using leave-one-out cross validation to show us how we are doing):

```
> library(ipred)
> res <- errorest(Zn ~ 1, data = as(BMcD, "data.frame"), model = lm,
+     est.para = control.errorest(k = nrow(BMcD), random = FALSE,
+         predictions = TRUE))
> round(res$error, 2)

[1] 400.86

> fres <- lm(Zn ~ Fldf, data = BMcD)
> anova(fres)

Analysis of Variance Table

Response: Zn
          Df  Sum Sq Mean Sq F value    Pr(>F)
Fldf       2 6413959 3206979    33.8 8.196e-12
Residuals 95 9013656   94881

> eres <- errorest(Zn ~ Fldf, data = as(BMcD, "data.frame"), model = lm,
+     est.para = control.errorest(k = nrow(BMcD), random = FALSE,
+         predictions = TRUE))
> round(eres$error, 2)

[1] 310.74
```

Aspatial flood frequency model



And the messy bits (once):

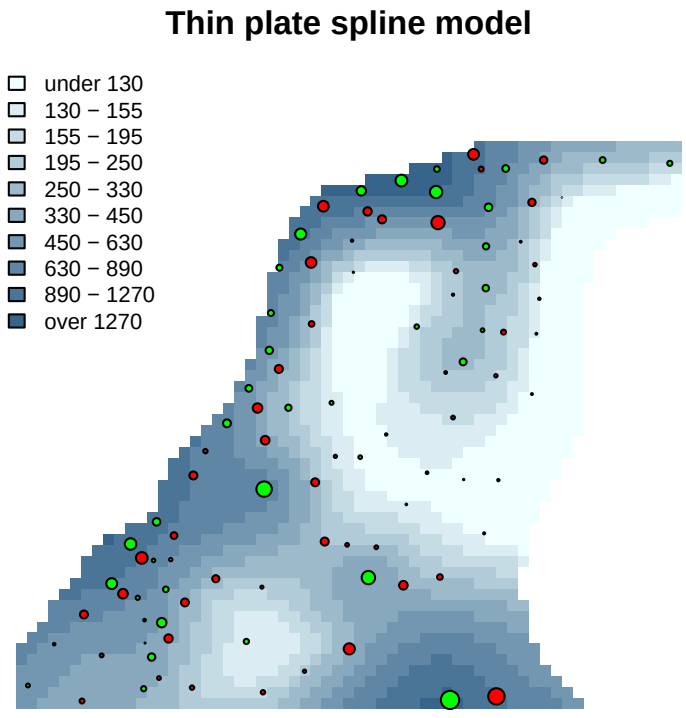
```
> library(maptools)
> BMcD_grid$lm_pred <- predict(fres,
+     newdata = BMcD_grid)
> image(BMcD_grid, "lm_pred",
+     breaks = brks, col = cols)
> title("Flood frequency model interpolation")
> pe <- BMcD$Zn - eres$predictions
> symbols(coordinates(BMcD), circles = sqrt(abs(pe)),
+     fg = "black", bg = scols[(pe <
+         0) + 1], inches = FALSE,
+     add = TRUE)
> legend("topleft", fill = cols,
+     legend = leglabs(brks),
+     bty = "n", cex = 0.8)
```

Thin plate spline interpolation

The next attempt uses `tps` from **fields** to do thin plate spline interpolation, first in a loop to do LOO CV:

```
> library(fields)
> pe_tps <- numeric(nrow(BMcD))
> cBMcD <- coordinates(BMcD)
> for (i in seq(along = pe_tps)) {
+     tpsi <- Tps(cBMcD[-i, ], BMcD$Zn[-i])
+     pri <- predict(tpsi, cBMcD[i, , drop = FALSE])
+     pe_tps[i] <- BMcD$Zn[i] - pri
+ }
> round(sqrt(mean(pe_tps^2)), 2)

[1] 263.69

> tps <- Tps(coordinates(BMcD), BMcD$Zn)
```

Thin plate spline interpolation



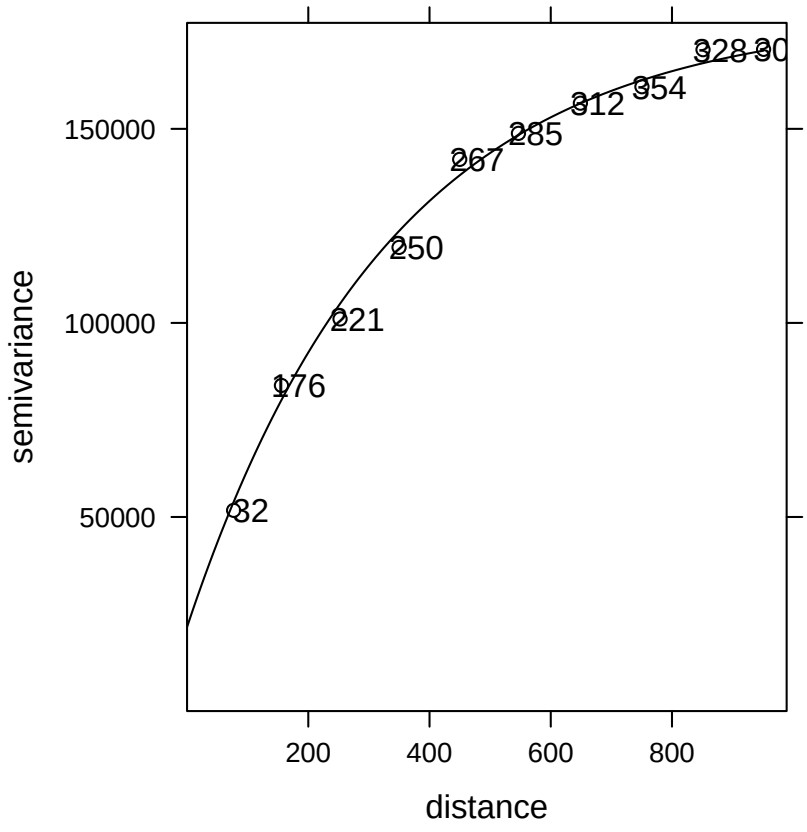
We have a slight problem of undershooting zero on the east, but thin plate splines yield a generally "attractive" smoothed picture of zinc ppm:

```
> BMcD_grid$spl_pred <- predict(tps,
+     coordinates(BMcD_grid))
> image(BMcD_grid, "spl_pred",
+     breaks = brks, col = cols)
```

Modelling the local smooth

If we choose to use geostatistical methods, we need a model of local dependence, and conventionally fit an exponential model to the zinc ppm data:

```
> library(gstat)
> cvgm <- variogram(Zn ~ 1, data = BMcD,
+     width = 100, cutoff = 1000)
> efitted <- fit.variogram(cvgm,
+     vgm(psill = 1, model = "Exp",
+         range = 100, nugget = 1))
> efitted

  model     psill   range
1   Nug  21652.99   0.000
2   Exp 157840.74 336.472
```

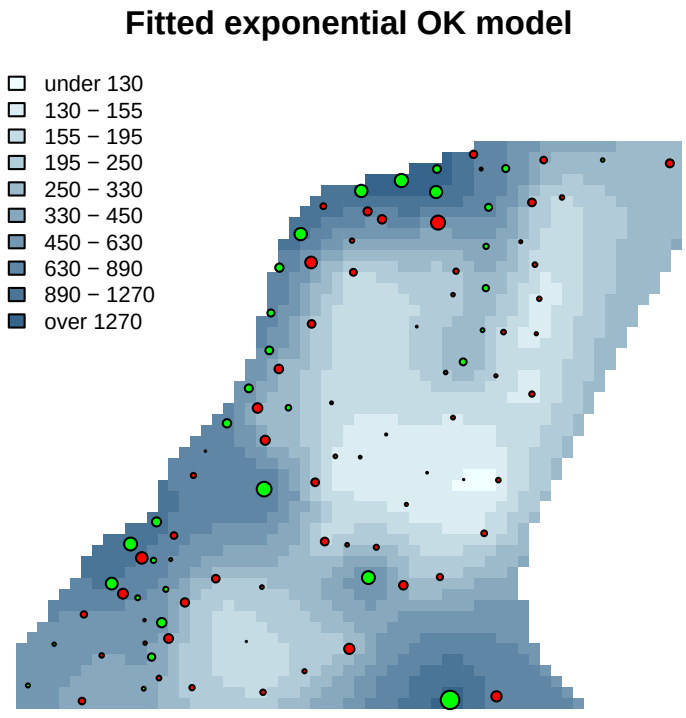


Ordinary kriging

Using the fitted variogram, we define the geostatistical model and use it both for LOO cross validation and for predictions, also storing the prediction standard errors:

```
> OK_fit <- gstat(id = "OK_fit", formula = Zn ~ 1, data = BMcD, model = efitted)
> pe <- gstat.cv(OK_fit, debug.level = 0, random = FALSE)$residual
> round(sqrt(mean(pe^2)), 2)

[1] 261.55

> z <- predict(OK_fit, newdata = BMcD_grid, debug.level = 0)
> BMcD_grid$OK_pred <- z$OK_fit.pred
> BMcD_grid$OK_se <- sqrt(z$OK_fit.var)
```

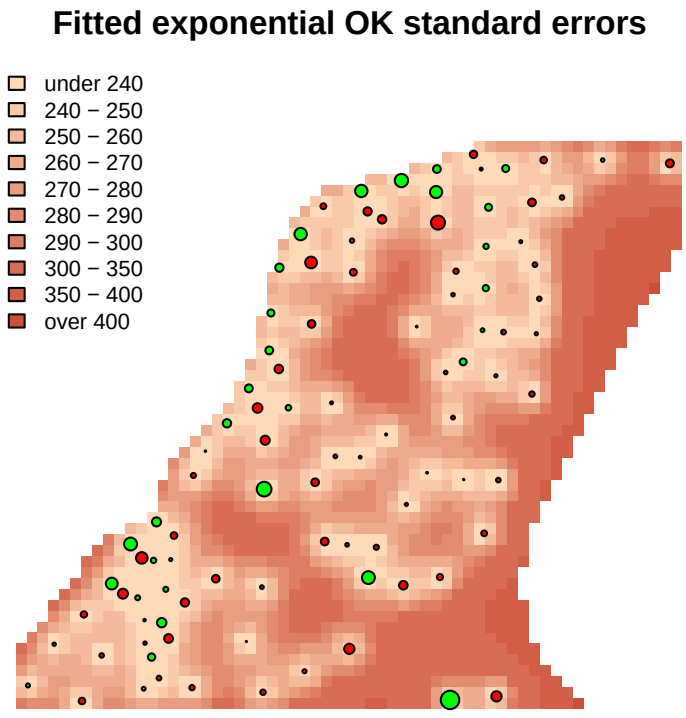
Ordinary kriging predictions



By now, the typical idiom of adding constructed variables to the SpatialPixels data frame object, and displaying them by name, should be familiar:

```
> image(BMcD_grid, "OK_pred",
+     breaks = brks, col = cols)
```

Ordinary kriging standard errors



For the standard errors, we use a different palette, but the procedure is the same:

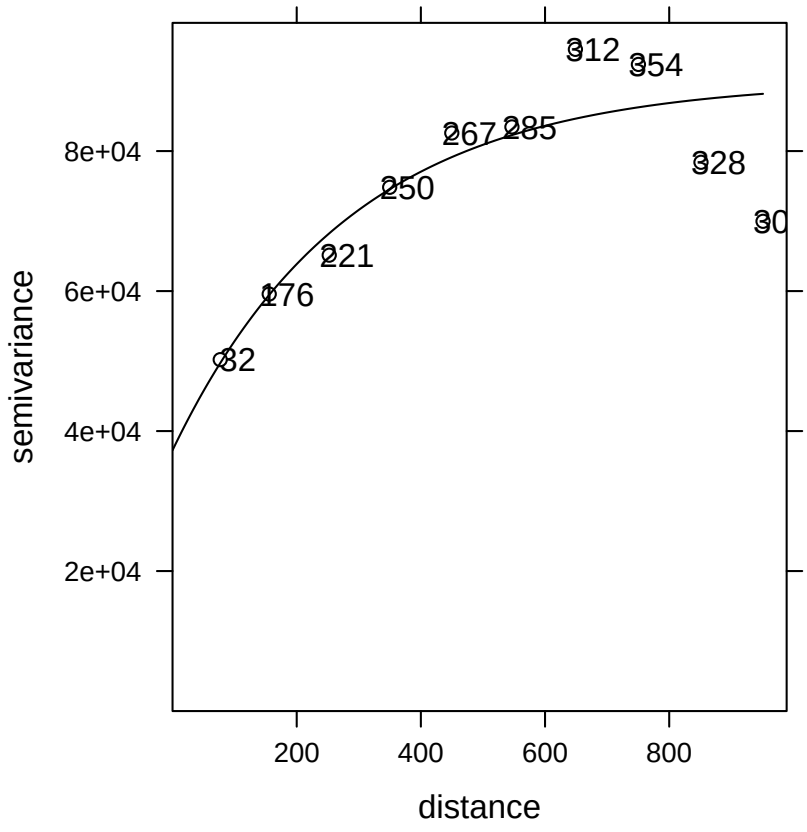
```
> image(BMcD_grid, "OK_se", breaks = brks.se,
+     col = cols.se)
```

Universal kriging — adding flood frequencies

We know that flood frequencies make a difference — can we combine the local smooth with that global smooth?

```
> cvgm <- variogram(Zn ~ Fldf,
+     data = BMcD, width = 100,
+     cutoff = 1000)
> uefitted <- fit.variogram(cvgm,
+     vgm(psill = 1, model = "Exp",
+         range = 100, nugget = 1))
> uefitted
```

```
  model    psill    range
1   Nug 37259.01   0.0000
2   Exp 52811.94 285.6129
```

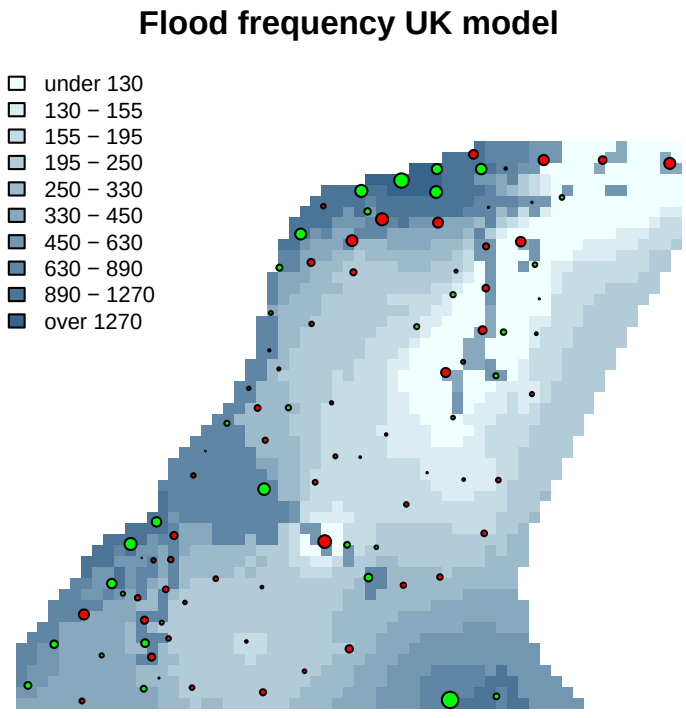


Universal kriging

The geostatistical packages, like **gstat**, use formula objects in standard ways where possible, which allows for considerable flexibility, as in this case, where we do really quite well in terms of LOO CV — and reach the same conclusion as Burrough and McDonnell about the choice of model:

```
> UK_fit <- gstat(id = "UK_fit", formula = Zn ~ Fldf, data = BMcD, model = uefitted)
> pe_UK <- gstat.cv(UK_fit, debug.level = 0, random = FALSE)$residual
> round(sqrt(mean(pe_UK^2)), 2)

[1] 225.8

> z <- predict(UK_fit, newdata = BMcD_grid, debug.level = 0)
> BMcD_grid$UK_pred <- z$UK_fit.pred
> BMcD_grid$UK_se <- sqrt(z$UK_fit.var)
```

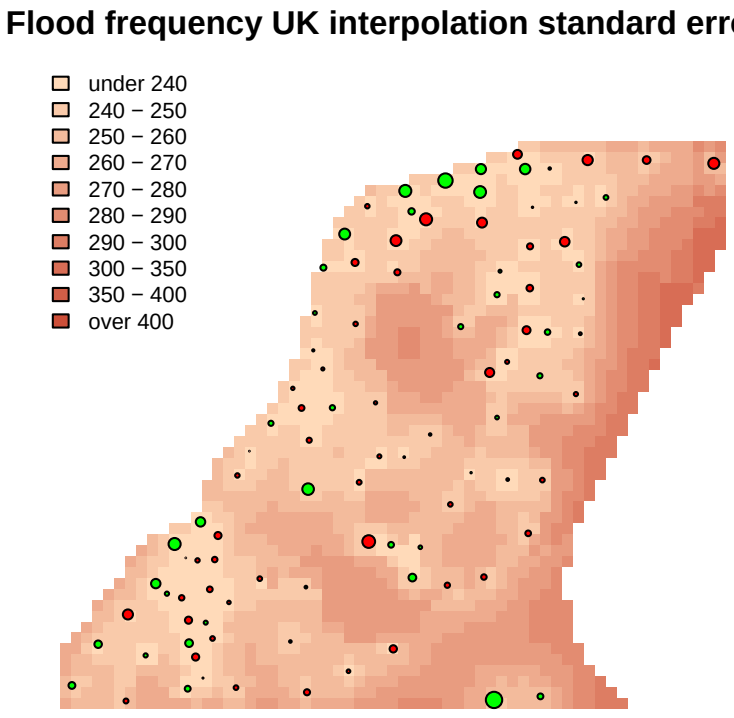
Universal kriging predictions



Of course, the resolution of the grid of prediction locations means that the shift from flood frequency class 1 to the others is too "chunky", but the effect of flood water "backin up" creeks seems to be captured:

```
> image(BMcD_grid, "UK_pred",
+     breaks = brks, col = cols)
```

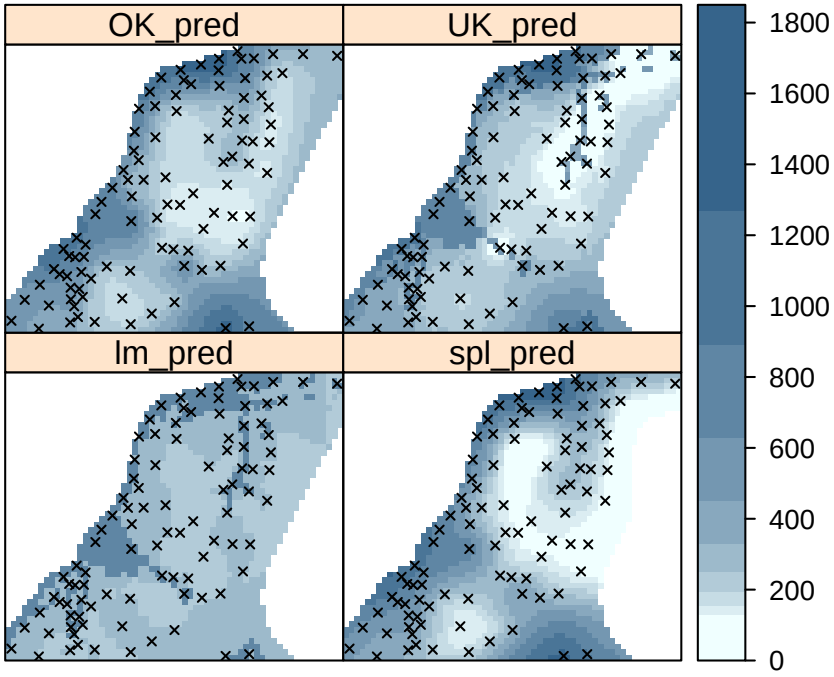
Universal kriging standard errors



The standard errors are also improved on the ordinary kriging case:

```
> image(BMcD_grid, "UK_se", breaks = brks.se,
+     col = cols.se)
```

Putting it all together



Using spplot, we can display all the predictions together, to give a view of our progress:

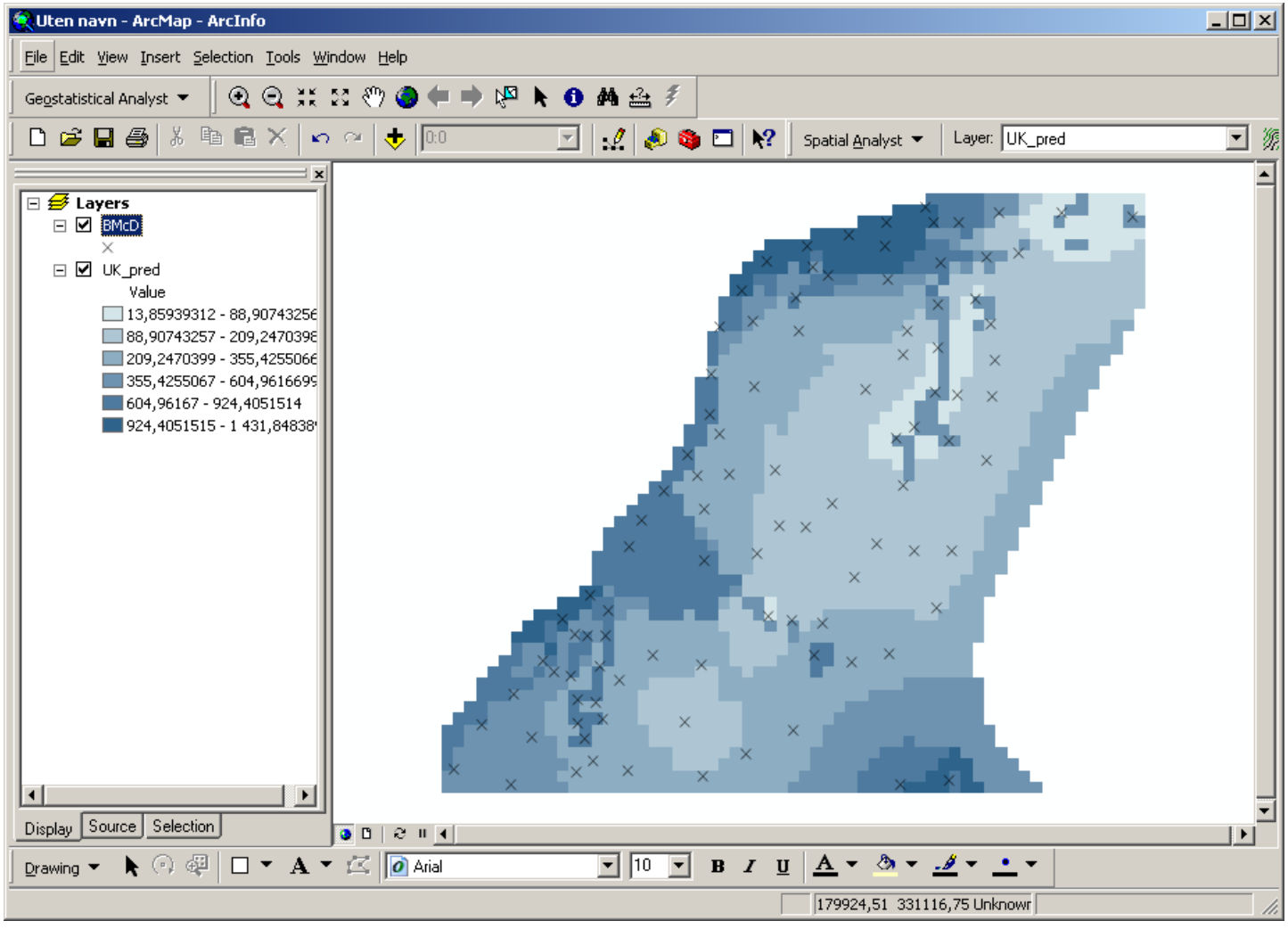
```
> pts = list("sp.points", BMcD,
+     pch = 4, col = "black",
+     cex = 0.5)
> spplot(BMcD_grid, c("lm_pred",
+     "spl_pred", "OK_pred", "UK_pred"),
+     at = brks, col.regions = cols,
+     sp.layout = list(pts))
```

Exporting a completed prediction

We will finally try to export the universal kriging predictions as a GeoTiff file, and read it into ArcGIS. In practice, this requires using Toolbox → Raster → Calculate statistics, and then right-clicking on the layer: Properties → Symbology → Classified:

```
> writeGDAL(BMcD_grid["UK_pred"], "UK_pred.tif")
```

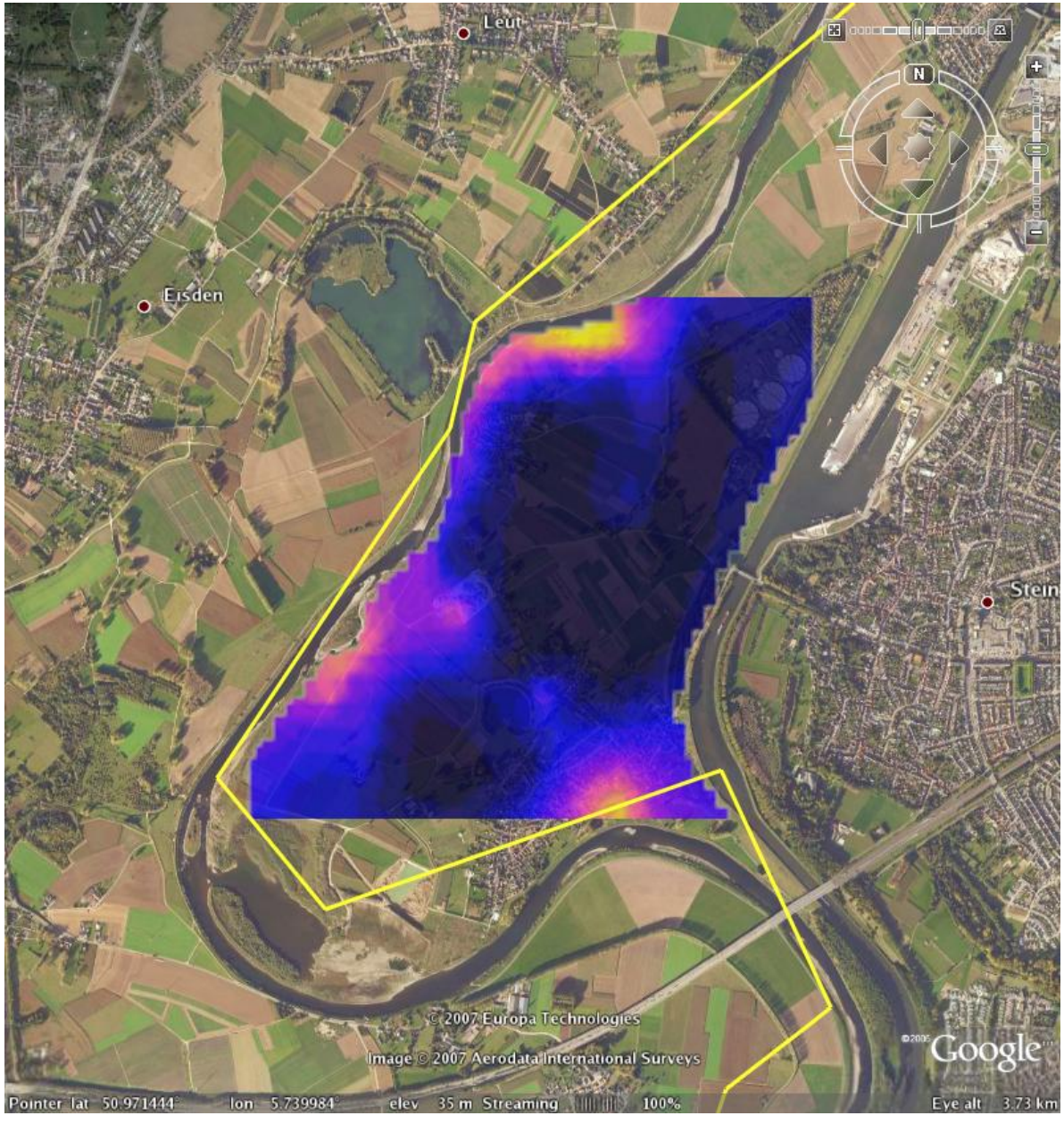
The exported raster viewed in ArcGIS



Writing a GE image overlay

```
> library(maptools)
> grd <- as.SpatialPolygons.SpatialPixels(BMcD_grid)
> proj4string(grd) <- CRS(proj4string(BMcD))
> grd.union <- unionSpatialPolygons(grd, rep("x", length(slot(grd, "polygons"))))
> grd.union.ll <- spTransform(grd.union, CRS("+proj=longlat"))
> llGRD <- GE_SpatialGrid(grd.union.ll, maxPixels = 100)
> llGRD_in <- overlay(llGRD$SG, grd.union.ll)
> llSPix <- as(SpatialGridDataFrame(grid = slot(llGRD$SG, "grid"), proj4string = CRS(proj4string(llGRD$SG)),
+     data = data.frame(in0 = llGRD_in)), "SpatialPixelsDataFrame")
> SPix <- spTransform(llSPix, CRS("+init=epsg:28992"))
> z <- predict(OK_fit, newdata = SPix, debug.level = 0)
> llSPix$pred <- z$OK_fit.pred
> png(file = "zinc_OK.png", width = llGRD$width, height = llGRD$height,
+     bg = "transparent")
> par(mar = c(0, 0, 0, 0), xaxs = "i", yaxs = "i")
> image(llSPix, "pred", col = bpy.colors(20))
> dev.off()
> kmlOverlay(llGRD, "zinc_OK.kml", "zinc_OK.png")
```

The image overlay viewed in GE



Conclusions

- The **sp** classes can be used (more or less) like data frames in many contexts
- The display methods on generated predictions and standard errors can be used directly, with spatial position being handled within the **sp** class objects
- Generating output for interfacing with other software is a bit picky (Arc prefers single-band GeoTiffs, while ENVI will digest multi-band files with no apparent discomfort)
- And we are still just at the beginning of making predictions — there are far more sophisticated methods out there, but they also benefit from ease of standardised data import, export, and display