Using R with FOSS4G, in particular with GRASS: The R— GRASS 6 interface

Roger Bivand

Norges Handelshøyskole
Bergen, Norway;
Roger.Bivand@nhh.no; Roger.Bivand@R-project.org

15:10 — 15:40, 12 September 2006

Introduction

- Initial insight that R can be run from the GRASS prompt, which is a standard shell prompt including GRASS environment variables
- GRASS is a GPL'ed development of a US Army raster GIS with vector capabilities; GRASS 5.0 introduced support for NULL/NA and floating point raster values
- First versions used intermediate text files, later versions added calls to a local, frozen, copy of the GRASS libgis C functions called from R
- The GRASS 5.0 interface supports raster and sites data, but not vector, **GRASS** is on CRAN

Legacy GRASS 5 interface

- The current window should determine the way in which raster data are retrieved and transferred; this should also apply to vector/site data with regard to extent
- Data moved to GRASS over the interface should be furnished with attributes expected by GRASS, including category labels and colours
- These criteria are important when the interface is used in BATCH mode to supplement GRASS shell scripts

GRASS 6 interface

- The GRASS 6 interface differs from the GRASS 5 in a number of ways, in particular it does not use compiled code, just temporary files (**spgrass6** is on CRAN)
- If the GDAL/OGR GRASS plugins are used, **rgdal** functions can be used to read GRASS data directly into R, but not to write to GRASS
- So intermediate temporary files are the chosen solution, using shapefiles for vector data and BIL binaries for raster data; as yet category labels are passed from GRASS to R, but not R to GRASS for raster data
- Support for GRASS under Cygwin is provided, but native Windows GRASS is untested (a Windows binary is on CRAN); Mac OSX is treated as Unix, and **spgrass6** is installed from source (like **rgdal**)

Using the GRASS 6 interface

Provided that the **spgrass6** package has been installed following the packages it depends upon, **sp**, **maptools**, **foreign** and **rgdal**, the interface is used more or less as before. R is started from within a GRASS session from the command line, and the **spgrass6** loaded with its dependencies:

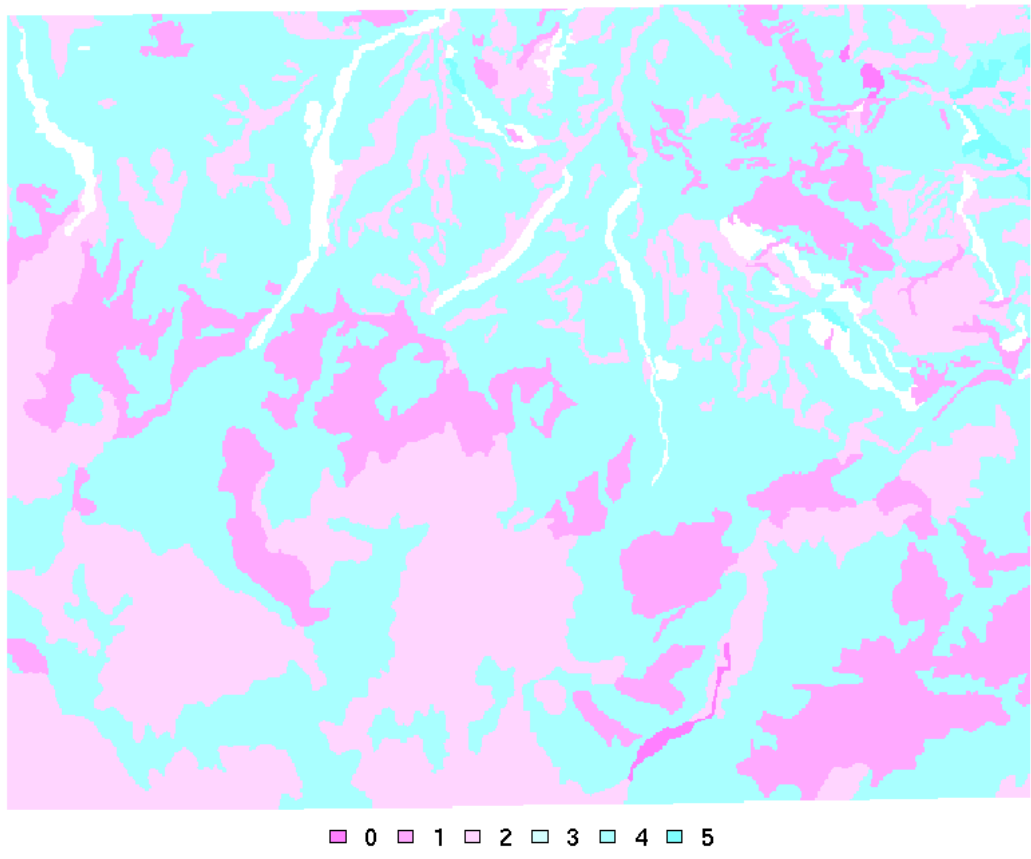
```
> if (nchar(Sys.getenv("GISRC")) == 0) stop("Start R inside GRASS")
> library(spgrass6)
> G <- gmeta6()
> .LOC <- G$LOCATION_NAME
> if (length(grep("^spearfish", .LOC)) == 0) stop("not in spearfish")
```

Raster data transfer

At the present stage of the interface, raster data transfer is done layer by layer, and uses temporary binary files. The following command reads the soil pH values into a `SpatialGridDataFrame` object, treating the values returned as floating point (double in R):

```
> soilsph <- readRAST6("soils.ph", ignore.stderr = TRUE)
> summary(soilsph)
> image(soilsph, "soils.ph", col = rev(cm.colors(6)))
> legend("bottom", legend = 0:5, fill = rev(cm.colors(6)),
+     cex = 0.8, bty = "n", horiz = TRUE)
Object of class SpatialGridDataFrame
Coordinates:
              min     max
coords.x1  589980  609000
coords.x2 4913700 4928010
Is projected: TRUE
proj4string : [+proj=utm]
 proj4string : [+zone=13]
 proj4string : [+a=6378206.4]
 proj4string : [+rf=294.9786982]
 proj4string : [+no_defs]
 proj4string : [+nadgrids=/home/rsb/topics/grass62_0820/grass-6.2.cvs/etc/nad/conus]
Number of points: 2
Grid attributes:
  cellcentre.offset cellsize cells.dim
1            589995       30       634
2           4913715       30       477
Data attributes:
    soils.ph
 Min.   :    1.000
 1st Qu.:    3.000
 Median :    4.000
 Mean   :    3.380
```

Soil pH values for Spearfish



Moving multiple rasters

A feature of the legacy interface was the ability to move category labels to R; this is at present emulated by matching the labels reported by GRASS. This is implemented in the `readRAST6` function, when the `cat=` argument is set to `TRUE` as appropriate:

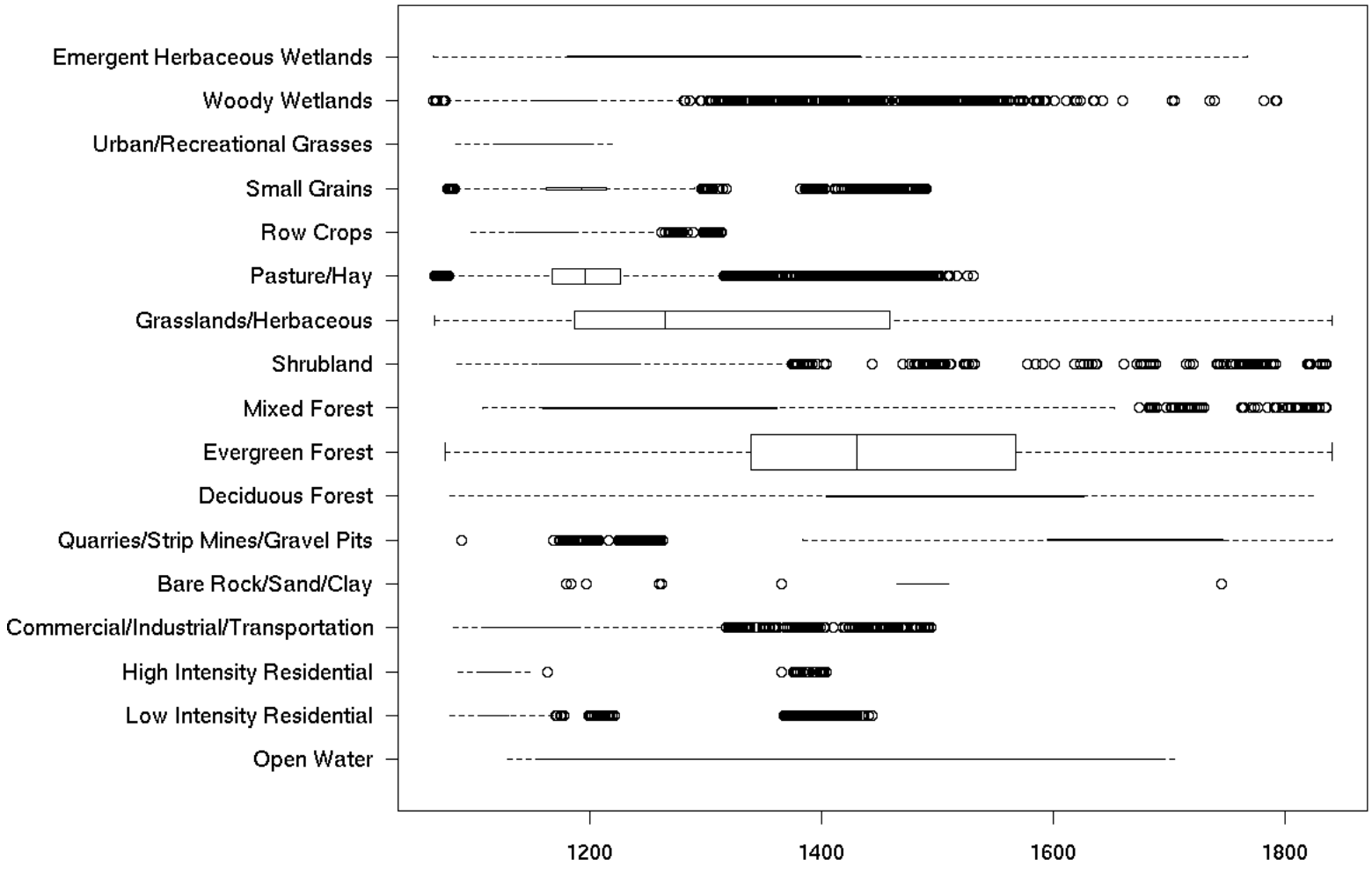
```
> spear <- readRAST6(c("elevation.dem", "landcover.30m"), cat = c(FALSE,
+     TRUE), ignore.stderr = TRUE)
> summary(spear)
Object of class SpatialGridDataFrame
Coordinates:
              min     max
coords.x1  589980  609000
coords.x2 4913700 4928010
Is projected: TRUE
proj4string : [+proj=utm]
 proj4string : [+zone=13]
 proj4string : [+a=6378206.4]
 proj4string : [+rf=294.9786982]
 proj4string : [+no_defs]
 proj4string : [+nadgrids=/home/rsb/topics/grass62_0820/grass-6.2.cvs/etc/nad/conus]
Number of points: 2
Grid attributes:
  cellcentre.offset cellsize cells.dim
1            589995       30       634
2           4913715       30       477
Data attributes:
 elevation.dem                        landcover.30m
 Min.   : 1066   Evergreen Forest            :135375
 1st Qu.: 1200   Grasslands/Herbaceous       : 67396
 Median : 1316   Pasture/Hay                 : 56263
 Mean   : 1354   Small Grains                :  9023
 3rd Qu.: 1488   Emergent Herbaceous Wetlands:  5673
 Max.   : 1840   (Other)                     : 18587
```

The Spearfish boxplot: elevation by landcover

The figure shows how much of the functionality of the legacy raster interface has been maintained, by relating elevation and landcover categories:

```
> grd <- slot(spear, "grid")
> lcovareas <- table(spear$landcover.30m) * (prod(slot(grd,
+     "cellsize"))/10000)
> oopar <- par(mar = c(3, 16, 2, 2) + 0.1)
> boxplot(elevation.dem ~ landcover.30m, data = spear, medlwd = 1,
+     horizontal = TRUE, las = 1, width = lcovareas)
> par(oopar)
```

The Spearfish boxplot: elevation by landcover



Vector data

The package provides functions to move vector features and associated attribute data to R and back again:

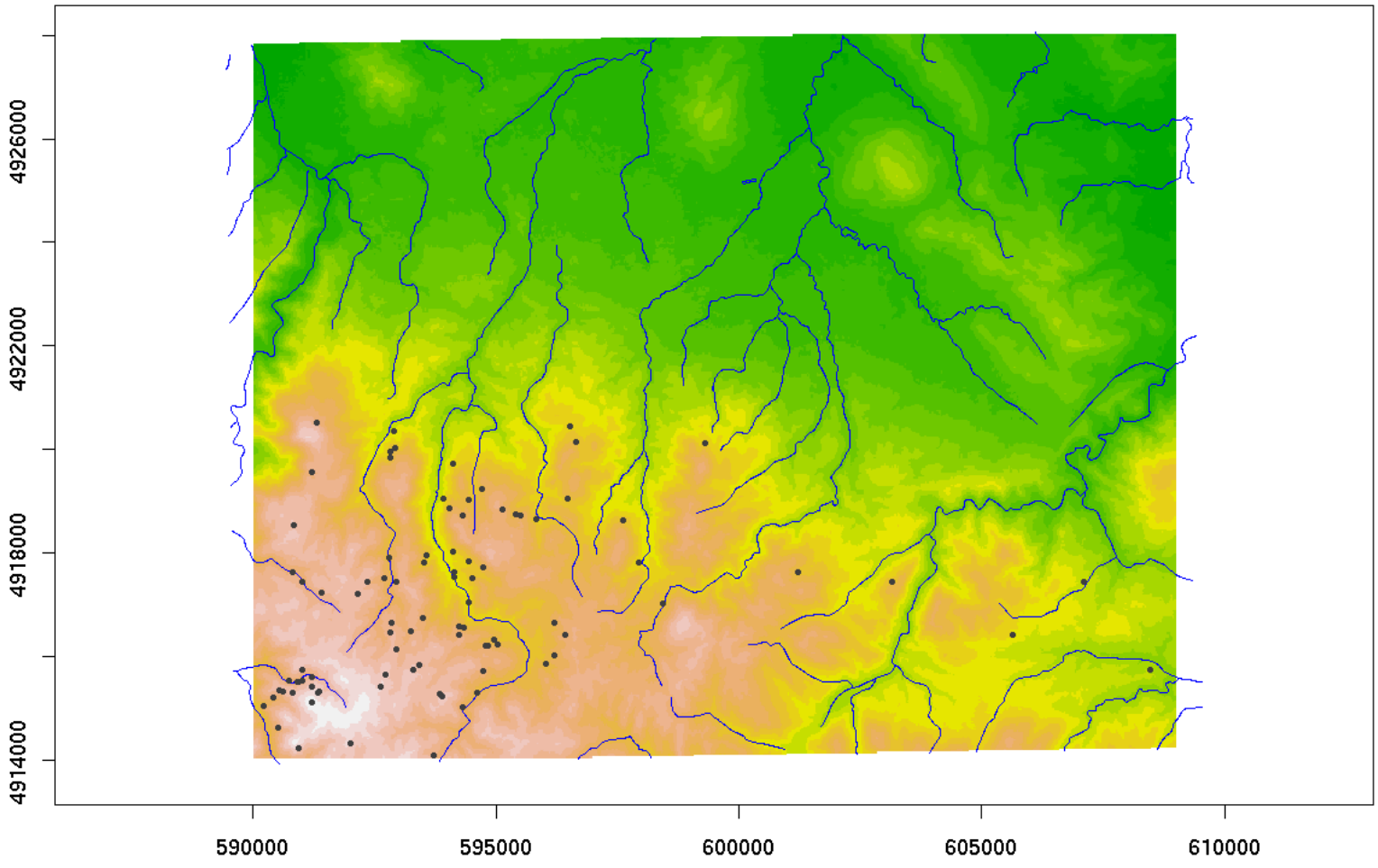
```
> bugsDF <- readVECT6("bugsites", ignore.stderr = TRUE)
> vInfo("streams", ignore.stderr = TRUE)

    points      lines boundaries  centroids      areas    islands
         0        104         12          4          4          4
     faces    kernels
         0          0

> streams <- readVECT6("streams", type = "line,boundary", remove.duplicates = FALSE,
+     ignore.stderr = TRUE)
```

Vector data

The `remove.duplicates=` argument is set to TRUE when there are only for example lines or areas, and the number present is greater than the data count (the number of rows in the attribute data table). The `type=` argument is used to override type detection when multiple types are non-zero, as here, where we choose lines and boundaries, but the function guesses areas, returning just filled water bodies.

```
> image(spear, "elevation.dem", col = terrain.colors(20), axes = TRUE)
> plot(streams, col = "blue", add = TRUE)
> plot(bugsDF, pch = 19, col = "grey25", cex = 0.5, add = TRUE)
```

Streams and bugsites



Interpolation: The Burrough/McDonnell Meuse bank data set

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):

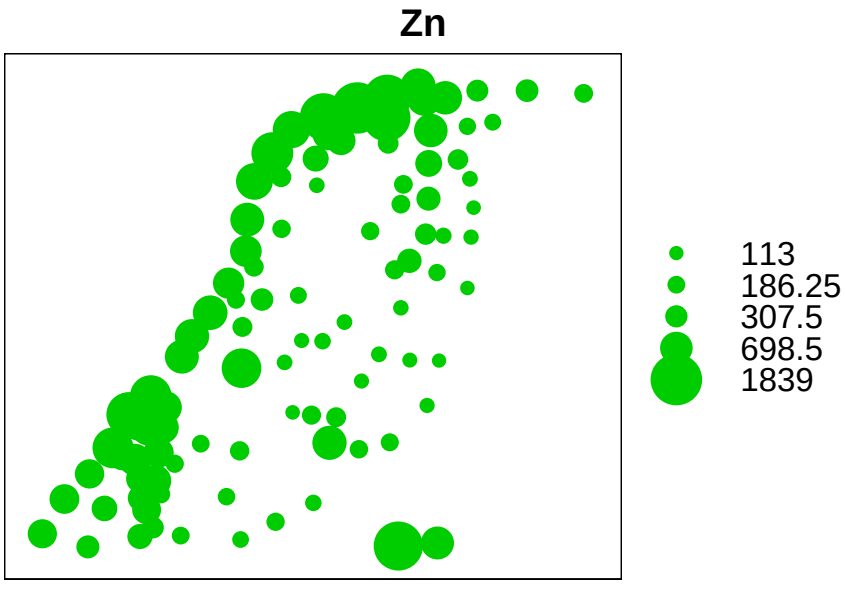
```
> system("r.in.gdal input=BMcD_fldf.txt output=BMcD_fldf location=BMcD")

> system("g.gisenv set='LOCATION_NAME=BMcD'")
> library(gstat)

> system("v.in.ogr dsn=. layer=BMcD output=BMcD")
```

Reading the data

```
> BMcD <- readVECT6("BMcD", ignore.stderr = TRUE)
> BMcD$Fldf <- factor(BMcD$Fldf)
> names(BMcD)

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

Since a variable of interest — flood frequency — is a categorical variable but read as numeric, it is set to factor

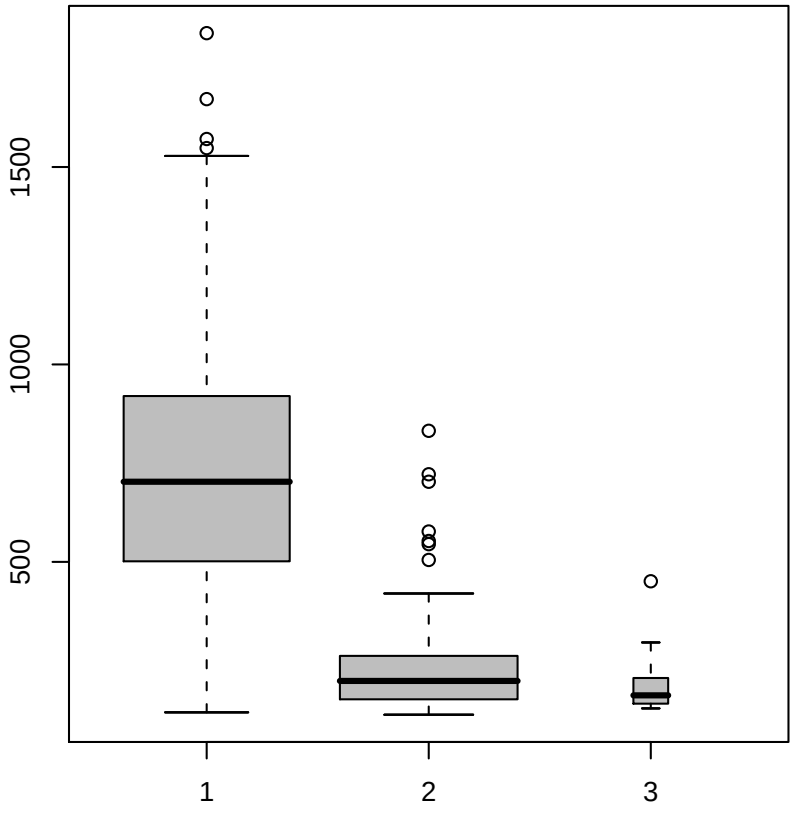
Observed zinc ppm levels



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



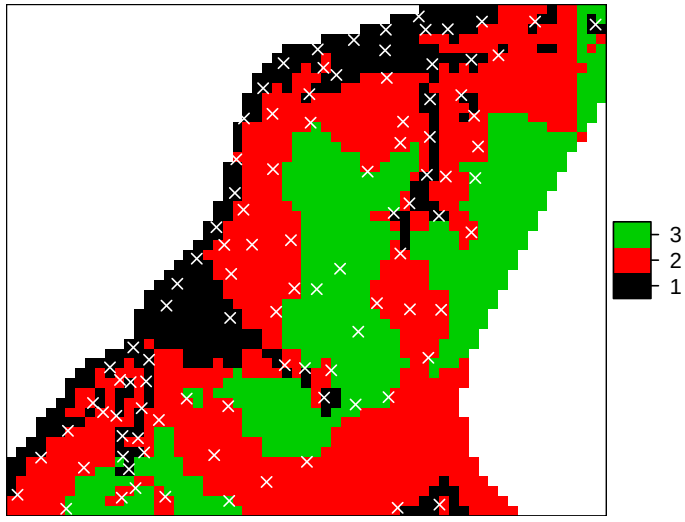
Boxplots of the zinc ppm values by flood frequency suggest that the apparent skewness of the values may be related to heterogeneity in environmental "drivers"

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

Reading the prediction locations

Reading the prediction locations (the helper function `gmeta2grd` can be used to generate a prediction grid for the GRASS computation region) — we also impose the same representation of the projection:

```
> BMcD_grid <- as(readRAST6("BMcD_fldf",
+     ignore.stderr = TRUE), "SpatialPixelsDataFrame")
> names(BMcD_grid) <- "Fldf"
> BMcD_grid$Fldf <- as.factor(BMcD_grid$Fldf)
> proj4string(BMcD) <- CRS(proj4string(BMcD_grid))

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



Set up class intervals and palettes

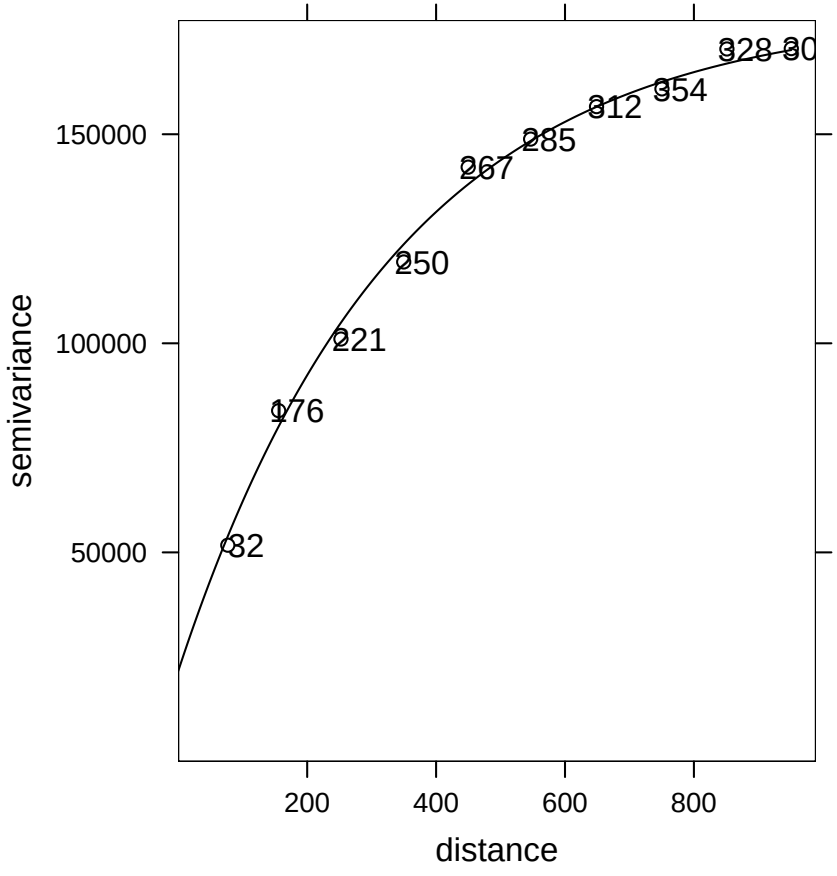
Setting up class intervals and palettes initially will save time later; note the use of `colorRampPalette`, which can also be specified from **RColorBrewer** palettes:

```
> 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")
```

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
```

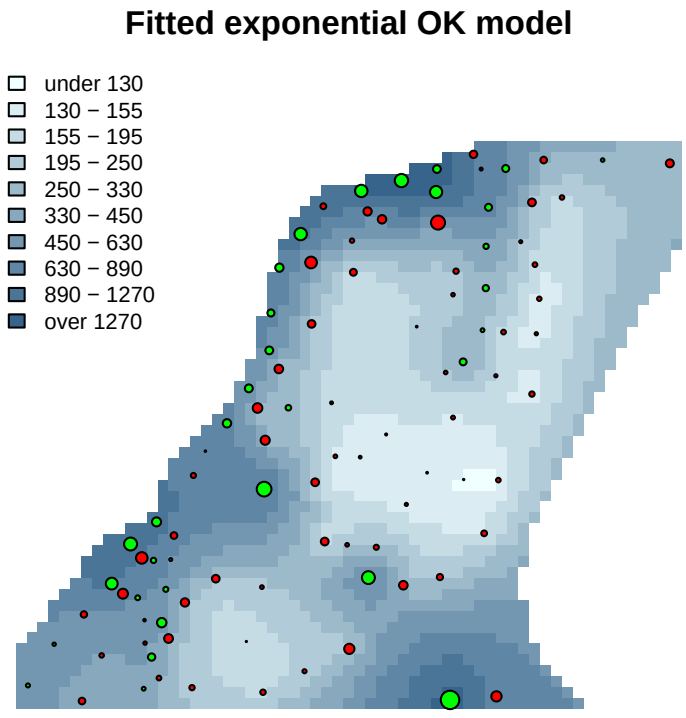


Ordinary kriging

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)
```

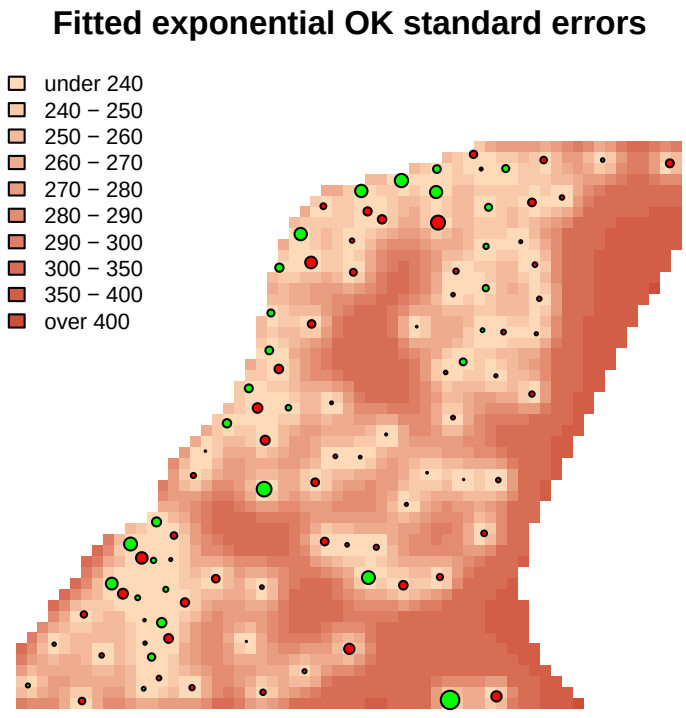
Ordinary kriging predictions



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:

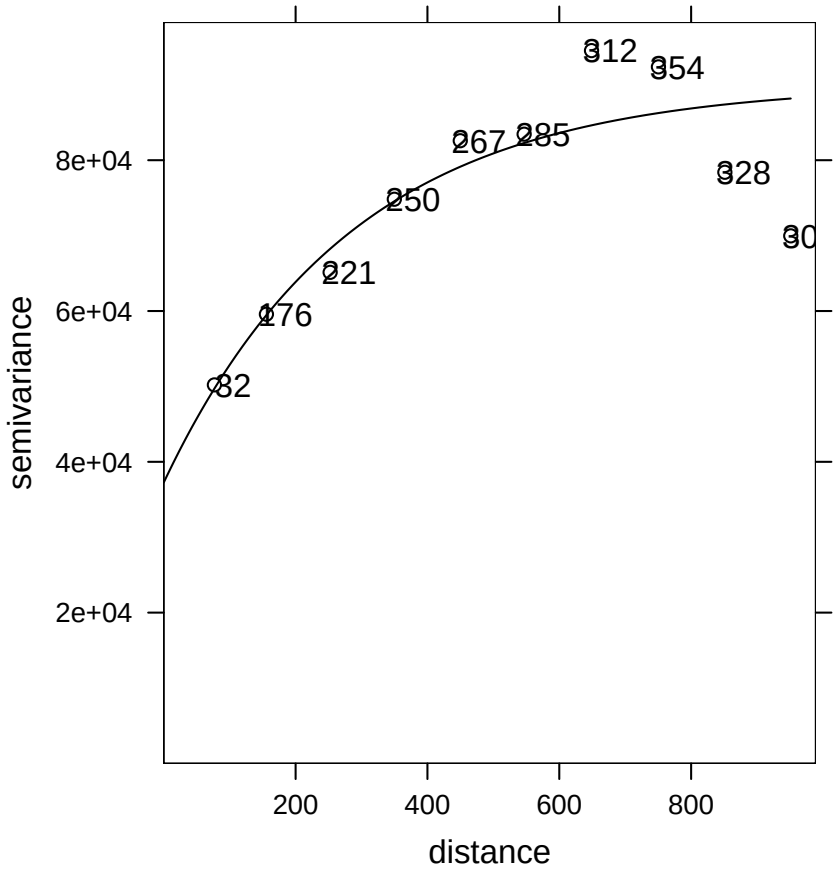
```
> image(BMcD_grid, "OK_se", breaks = brks.se,
+     col = cols.se)
```

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
```

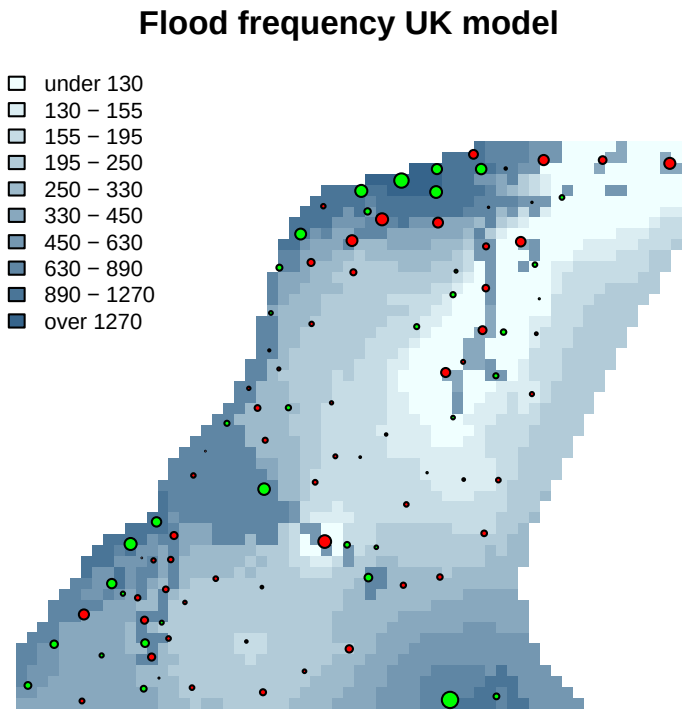


Universal kriging

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)
```

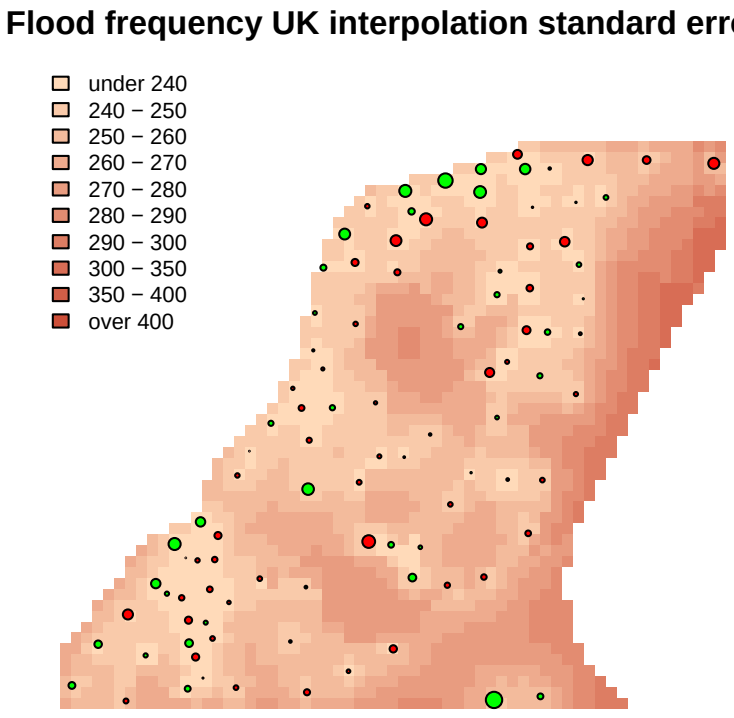
Universal kriging predictions



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)
```

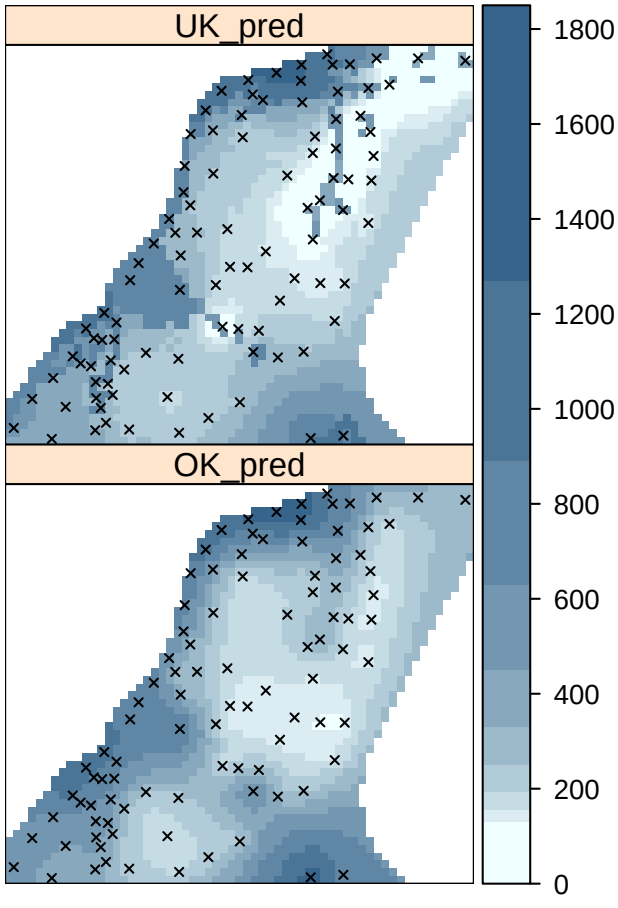
Universal kriging standard errors



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("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 to GRASS:

```
> writeRAST6(BMcD_grid, vname = "UK_pred", zcol = "UK_pred", ignore.stderr = TRUE)
> cat(system("r.info UK_pred", intern = TRUE, ignore.stderr = TRUE), sep = "\n")
```

```
 +----------------------------------------------------------------------------+
 | Layer:    UK_pred                         Date: Fri Sep  1 22:37:16 2006    |
 | Mapset:   rsb                             Login of Creator: rsb             |
 | Location: BMcD                                                              |
 | DataBase: /home/rsb/topics/grassdata                                        |
 | Title:     ( UK_pred )                                                      |
 | timestamp: none                                                             |
 |-----------------------------------------------------------------------------|
 |                                                                             |
 |   Type of Map:  raster               Number of Categories: 255              |
 |   Data Type:    FCELL                                                       |
 |   Rows:         52                                                          |
 |   Columns:      61                                                          |
 |   Total Cells:  3172                                                        |
 |        Projection: sterea (zone 0)                                          |
 |             N:     332400    S:     330320   Res:    40                    |
 |             E:     181000    W:     178560   Res:    40                    |
 |   Range of data:    min = 13.859393  max = 1431.848389                      |
 |                                                                             |
 |   Data Source:                                                              |
 |                                                                             |
 |                                                                             |
 |                                                                             |
 |   Data Description:                                                         |
 |    generated by r.in.bin                                                    |
 |                                                                             |
 |                                                                             |
 +----------------------------------------------------------------------------+
```

Conclusion

- There is still a good deal to do to suit the different software components to one another
- **sp** classes are being used within R more and more to provide a shared platform, and packages like **rgdal** with GDAL/OGR bindings provide an attractive way of building on community progress
- **spgrass6** provides an interpreted interface between GRASS and R with R run above GRASS, but R scripts can be "canned" if need be — see the GRASS book for details
- Finally, initiatives like **aRT** linking R **sp** classes and MySQL/Terralib provide extra possibilities for software development