```
                x2 = aupoints.mds$points[,2])
```

Notice that the distance between Sydney and Canberra has shrunk quite severely.

# 11
# *Spatial Display, Modeling and Interpolation*

In the past several years, there have been spectacular advances in the abilities that are available using R. The range of abilites that are available provide a spectacular demonstration of the benefits of using R as a unfied framewotk for using abilities that were designed to run independently of R, and independently of one another. For an overview of what is available under the R umbrella, see the CRAN Task View:

`http://cran.ms.unimelb.edu.au/web/views/Spatial.html`

The R geo website, at `http://www.r-project.org/Rgeo/` has extensive information. The Wiki page `http://spatial-analyst.net/wiki/index.php?title=Software` has extensive information about installation of relevant geostatistical software.

Below, we will use R interfaces to GDAL and to PROJ.4. For this, various software and scripts (including GDAL and to PROJ.4) must be installed outside of R. For Windows and Linux 32-bit use, the easiest recourse is to download and install the relevant binary toolkit from `http://fwtools.maptools.org/`

From the R command line, the relevant R packages can be installed thus: [1]

```
install.packages(c("rgdal","gstat","sp"), dependencies=TRUE)
```

Ensure also that you have `rJava`.

Hengl(2009) provides a useful introduction to software that is useful for geostatistical and other mapping. Note especially Table 3.1 on page 89, which compares the spatio-temporal abilities of some popular statistics and GIS packages. There are columns for R+*gstat* and R+*geoR*.

[1] For MacOS X with version 2.13.x of R, `rgdal` must be installed from CRAN extras. Either type `setRepositories()` and choose CRAN extras as well perhaps as CRAN, or type `setRepositories(ind=1:2)`, prior to typing the `install.packages()` command.

## *11.1 Reading and Processing Raster (Image) Files*

The function `readGDAL()` in the *rgdal* package is intended for reading GDAL grid maps. Among the large number of possible formats are several that are widely used for image files more generally: BMP, various JPEG formats, GIF, PNG, XPM, etc. Georeferencing (spatial reference data can be included with the file) is available for BMP, JPEG 2000 formats, and TIFF.

The following, loosely based on the example code in the help pages for `readGDAL`, requires the packages *rgdal*, *sp*, and *grid*. We begin by using the function `readGDAL()` to input, as a grid, the R logo file that is supplied with the R package `rgdal`:

```
> library(rgdal)
> logofile <- system.file("pictures/Rlogo.jpg", package = "rgdal")[1]
> rlogo <- readGDAL(logofile)
C:/PROGRA~1/R/R-212~1.0/library/rgdal/pictures/Rlogo.jpg has GDAL
driver JPEG and has 77 rows and 101 columns
Warning message:
In readGDAL(system.file("pictures/Rlogo.jpg", package = "rgdal")[1]) :
  GeoTransform values not available
```

Notice the warning message "GeoTransform values not available". This is not surprising. Try however reading in an image from a camera that records GPS data. The GeoTransform values should be included.

The R logo is to be found in many places, even sometimes on a T-shirt!

Now examine the input object:

```
> class(rlogo)
[1] "SpatialGridDataFrame"
attr(,"package")
[1] "sp"
> names(rlogo)
[1] "band1" "band2" "band3"
```

Notice that the file is input as a `SpatialGridDataFrame` The function `image` has a method for objects of this class.

The image comes out more or less as expected using:

```
image(rlogo, red="band1", green="band2", blue="band3")
```

The function `spplot.grid()` is called to do the plotting In turn, it calls function `levelplot()` from the *lattice* package.

Note also the functions `spplot.polygons()` and `spplot.points()`. These are all documented on the same page as the generic function `spplot()`.

Another possibility is to use the function `spplot()` to examine the red green and blue layers separately, thus:

```
spplot(rlogo, zcol=1:3,
       names.attr=c("red","green","blue"),
       col.regions=grey(0:100/100),
       as.table=TRUE,
       main=paste("example of three-layer",
       "(RGB) raster image"))
```

*A genuine spatial image*

On this occasion we ask for information about the file before inputting it:

```
> sp27 <- system.file("pictures/SP27GTIF.TIF", package = "rgdal")[1]
> GDALinfo(sp27)
projection  +proj=tmerc +lat_0=36.66666666666666
+lon_0=-88.33333333333333 +k=0.999975
+x_0=152400.3048006096 +y_0=0 +ellps=clrk66 +datum=NAD27
+units=us-ft +no_defs
file        C:/PROGRA~1/R/R-212~1.0/library/rgdal/pictures/SP27GTIF.TIF
apparent band summary:
  GDType Bmin Bmax
1   Byte    0  255
Metadata:
TIFFTAG_XRESOLUTION=72
TIFFTAG_YRESOLUTION=72
TIFFTAG_RESOLUTIONUNIT=1 (unitless)
```

```
AREA_OR_POINT=Area
```

Now use `readGDAL()` to create a GDAL grid map from the image file, and plot the grid map:

```
> SP27GTIF <- readGDAL(sp27, output.dim=c(100,100))
C:/PROGRA~1/R/R-212~1.0/library/rgdal/pictures/SP27GTIF.TIF
has GDAL driver GTiff and has 929 rows and 699 columns
> class(SP27GTIF)
[1] "SpatialGridDataFrame"
attr(,"package")
[1] "sp"
> spplot(SP27GTIF)
```

**Note:** Data files that can simplify specification of projections can be found in the **epsg** database, included with the *rgdal* package. To locate it, type:

```
> normalizePath(system.file("proj/epsg", package="rgdal"))
[1] "C:\\Program Files\\R\\R-2.12.0\\library\\rgdal\\proj\\epsg"
```

To view it, type:

```
file.show(system.file("proj/epsg", package="rgdal"))
```

The numeric codes, given in diamond brackets, can simplify specification of the projection. Projections become important when spatial data, e.g., on metal concentrations, is overlaid on map data, e.g., from Google maps.

### *Overlaying information on plots*

Section 5.3 showed how to use a bubble plot to display the `meuse` data from the `sp` package. The function `bubble()` uses the abilities of the *lattice* package. As a consequence, the layering abilities of the *latticeExtra* package, described earlier in Subsection 7.3. can be used to overlay additional information on the plot.

Figure 11.1 adds river boundaries, using data from the dataset `meuse.riv`. (This is a matrix, with Eastings in column 1 and Northings in column 2.)

Code is:

```
library(sp)
data(meuse); data(meuse.riv)
coordinates(meuse) <- ~ x + y
library(latticeExtra)
gph <- bubble(meuse, "zinc", pch=1,
              key.entries =  100 * 2^(0:4),
              main = "Zinc(ppm)",
              scales=list(axes=TRUE, tck=0.4)) +
layer(panel.lines(meuse.riv[,1], meuse.riv[,2],
      col="gray"))
print(gph)
```

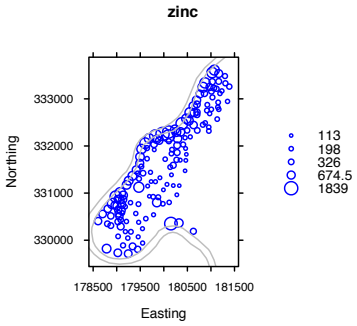


Figure 11.1: Bubble plot for `zinc`, with area of bubbles proportional to concentration. River Meuse boundaries are in gray.

*Combining information from sample sites with data from a regular grid*

The web page http://spatial-analyst.net/book/system/files/ has a zip file **meuse.zip** that holds a substantial number of files, including:

**ffreq.asc** (flooding frequency),
**dist.asc** (distance to river),
**soil.asc** (soil type),
**ahn.asc** (elevation),
**topomap2m.tif** (2m topomap).

These data are on a grid of 78 by 104 points, giving a total of 8112 values.

The following code demonstrates the extraction of the file topomap2m.tif, in the first place into a temporary directory, then copying it to the working directory:

```
tmpfile <- zip.file.extract(file="topomap2m.tif",
                            zipname="meuse.zip")
file.copy(tmpfile, "./topomap2m.tif", overwrite=TRUE)
```

The following uses the same procedure for the four files that will be of immediate interest:

```
grid.list <- c("ffreq.asc", "dist.asc", "soil.asc",
               "ahn.asc")
# unzip the maps in a loop:
for(j in grid.list){
tmpfile <- zip.file.extract(file=j,
                            zipname="meuse.zip")
file.copy(tmpfile, paste("./", j, sep=""),
          overwrite=TRUE)
}
```

Now use `readGDAL()` to load the grids into R:

```
# load grids to R:
meuse.grid <- readGDAL(grid.list[1])
# fix the layer name:
names(meuse.grid)[1] <- sub(".asc", "", grid.list[1])
for(i in grid.list[-1]) {
   meuse.grid@data[sub(".asc", "", i[1])] <-
     readGDAL(paste(i))$band1
}
names(meuse.grid)
proj4string(meuse.grid) <- CRS("+init=epsg:28992")
# reformat maps where needed:
meuse.grid$ffreq <- as.factor(meuse.grid$ffreq)
# pixels per class:
table(meuse.grid$ffreq)
meuse.grid$soil <- as.factor(meuse.grid$soil)
table(meuse.grid$soil)
## str(meuse.grid)
```

Figure 11.2 shows the images in the 2 by 2 layout. The soil type class image has the river boundaries added.

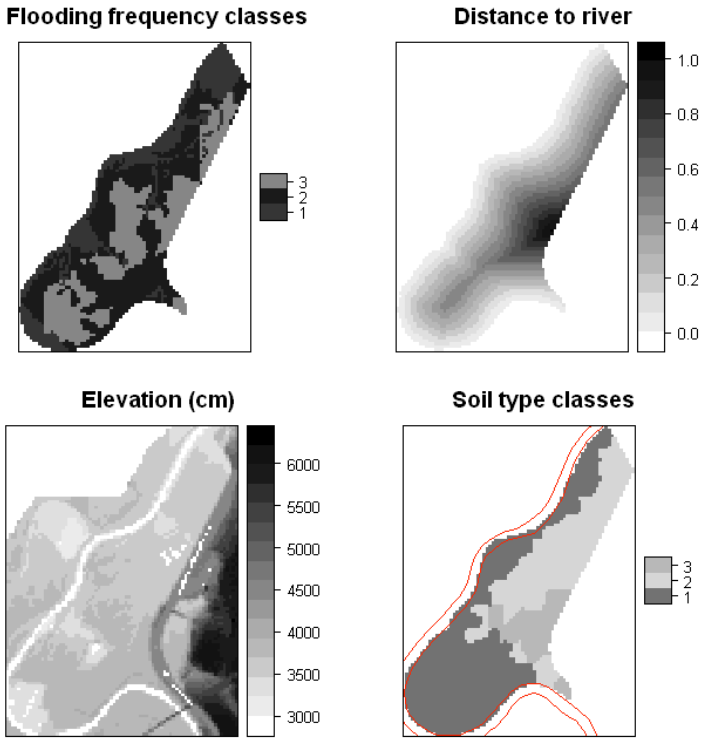


Figure 11.2: Plots show various features of the Meuse river plain data.

The code that plots the images is:

```
# plot the maps in a 2 columns by 2 rows layout:
ffreq.plt <- spplot(meuse.grid["ffreq"],
      col.regions=grey(runif(length(levels(meuse.grid$ffreq)))),
                   main="Flooding frequency classes")
dist.plt <- spplot(meuse.grid["dist"],
                  col.regions=grey(rev(seq(0,1,0.025))),
                  main="Distance to river")
ahn.plt <- spplot(meuse.grid["ahn"],
                 col.regions=grey(rev(seq(0,1,0.025))),
                 main="Elevation (cm)")
soil.plt <- spplot(meuse.grid["soil"],
       col.regions=grey(runif(length(levels(meuse.grid$ffreq)))),
                  main="Soil type classes")
print(ffreq.plt, split=c(1,1,2,2), more=T)
print(dist.plt, split=c(2,1,2,2), more=T)
print(ahn.plt, split=c(1,2,2,2), more=T)
```

```
print(soil.plt +
layer(panel.lines(meuse.riv[,1], meuse.riv[,2], col="red", type="l")),
      split=c(2,2,2,2), more=F)
```

Notice the addition of the river boundaries to the map of soil type classes.

*Overlaying onto new data*

The following overlays the data in `meuse.grid` onto the points in `meuse`, i.e., data at a resolution of 78 by 104 is overlaid onto the 155 locations that were sampled:

```
# overlay points and grids:
meuse.ov <- overlay(meuse.grid, meuse)
meuse.ov@data <- cbind(meuse.ov@data, meuse[c("zinc", "lime")]@data)
```

The following plots the result:

```
ahn.plt <- spplot(meuse.grid["ahn"], col.regions=grey(rev(seq(0,1,0.025))),
                  main="Elevation (cm)")
print(ahn.plt +
      spplot(meuse.ov["ahn"], col.regions=grey(rev(seq(0,1,0.025)))))
```

## *11.2 An Interface to Google maps*

This requires the packages *raster* and *dismo*. The *dismo* package has the function `gmap()` that downloads the map data.

*Basic syntax, accepting defaults:* The following is a simple example of what is possible:

```
cbr <- gmap("Canberra, ACT")
plot(cbr)
acton <- gmap("Acton, ACT")
plot(acton)
```

The argument `type` can be `'roadmap'`, `'satellite'`, `'hybrid'` or `'terrain'`. The argument `exp` can be used to specify an expansion factor.

*Specify map longitude/latitude extent; overlay onto map:* The data frame `possumsites` (*DAAG*) holds the latitudes and longitudes of sites from which possums were taken for study. In the following, a map is created that takes in all the sites. The site names are then overlaid on to the map:

```
## Ranges of latitude and longitude, slightly extended
lonlat <- with(possumsites, c(range(longitude)+c(-2,2),
                              range(latitude)+c(-3,3))
                              )
## Obtain map, as a ‘‘RasterLayer’’ object
googmap <- gmap(lonlat)
```

```
plot(googmap, inter=TRUE)
## Convert latitude/longitude data to Mercator projection
xy <- Mercator(with(possumsites, cbind(latitude, longitude)))
## Points show location of sites on the map
points(xy)
## Add labels that give the names
text(xy, labels=row.names(possumsites), pos=c(1,4,4,1,3,4,4))
```

*Specify map locality; vary vary map boundaries:* The following takes the map that is supplied by Google and varies the boundary limits. The arguments left, right, bottom and top are fractions of the extent in the relevant direction:

```
domap <- function(x="Acton, ACT", left=0, right=1,
                  bottom=0, top=1,
                  type="terrain"){
  if(is.character(x))
  x <- gmap(x, type=type)@extent
    x0 <- x@xmin
    x1 <- x@xmax
    y0 <- x@ymin
    y1 <- x@ymax
    x@xmin <- x0+left*(x1-x0)
    x@xmax <- x0+right*(x1-x0)
    x@ymin <- y0+bottom*(y1-y0)
    x@ymax <- y0+top*(y1-y0)
  map <- gmap(x, type=type)
  plot(map)
}
```

## 11.3 Other software – QGIS

Note in particular QGIS, whoich has an interface via *manageR* to R, which however has still (May 2011) to be updated to work with versions of R that are later than R-2.11.x Go to http://www.ftools.ca/plugins.html.

To obtain Windows and Linux installers for QGIS, go to http://www.qgis.org/wiki/Download. The standalone installer for Windows includes GRASS. For MacOSX, go to http://www.kyngchaos.com/software/qgis. For Leopard and Snow Leopard installations, the QGIS 1.7 developer builds seem relatively stable. GRASS must be installed separately.

## 11.4 References

Bivand R, Pebesma E J, Gomez-Rubio, V. 2008. Applied Spatial Data Analysis with R. Springer.

Diggle, Peter J. & Ribeiro Jr, Paulo J 2007. Model-Based Geostatistics. Springer.

Hengl, T. 2009, A Practical Guide to Geostatistical Mapping.

[To download (free) or purchase ($US16.78), go to: http://www.lulu.com/product/download/a-practical-guide-to-geostatistical-mapping/6379057]

Hijmans, R J. 2011. Introduction to the âĂŹrasterâĂŹ package.

[With the R package *raster* attached, type `vignette("Raster")`.

Hijmans, R J and Elith J. 2011. Species distribution modeling with R.

[With the R package *dismo* attached, type `vignette("sdm")`. The vignette appears to be an outline for a book. Later chapters are very incomplete.]

Maindonald, J H 2011. Generalized Additive Models in Spatial Statistics – Linear Models with a Twist (slides). maths.anu.edu.au//~johnm/r/spatial/

[This offers a perspective on spatial interpolation.]

Quantum GIS Development Team 2010. Quantum GIS User Guide( Version 1.6.0 âĂŹCopiapĂşâĂŹ). Obtain from http://www.qgis.org/en/documentation/manuals.html

See also the vignettes that accompany the package *sp*, describing classes and methods for spatial data, and overlay and aggregation.