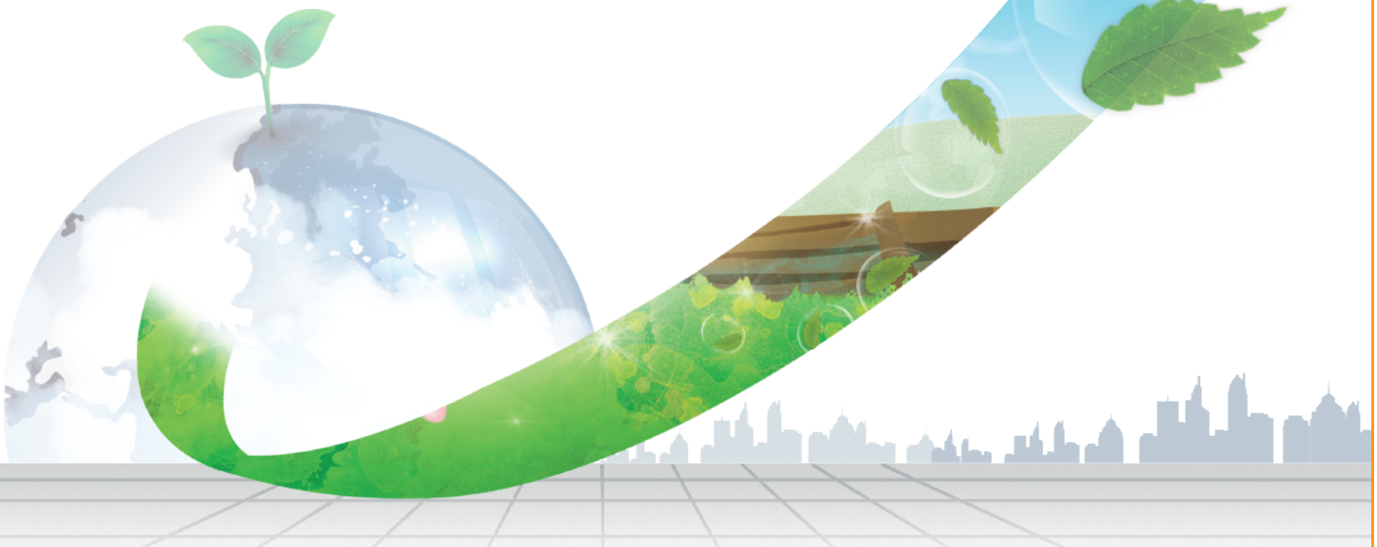
The International Workshop on
Agromet and GIS Applications
for Agricultural Decision Making

Statistical Downscaling Tutorial



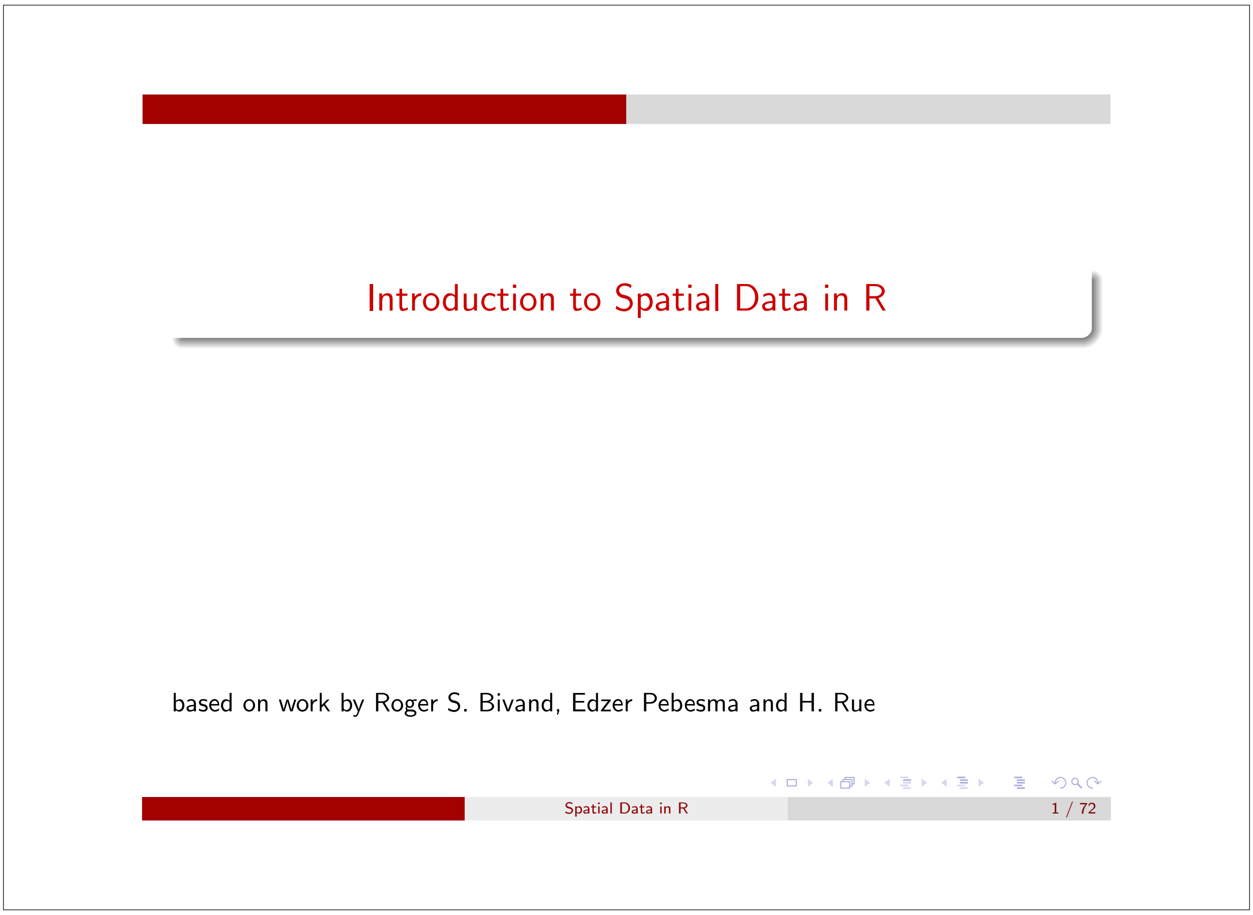
Date : December 5(Mon)~9(Fri), 2016
Place : MSTAY Hotel JEJU
Hosted by : Korea Meteorological Administration(KMA)
Organized by : National Institute of Meteorological Sciences(NIMS)
Sponsored by : WMO CAgM / NCAM / APCC / OSGeo / PKNU / DU

Korea Meteorological Administration

National Institute of Meteorological Sciences

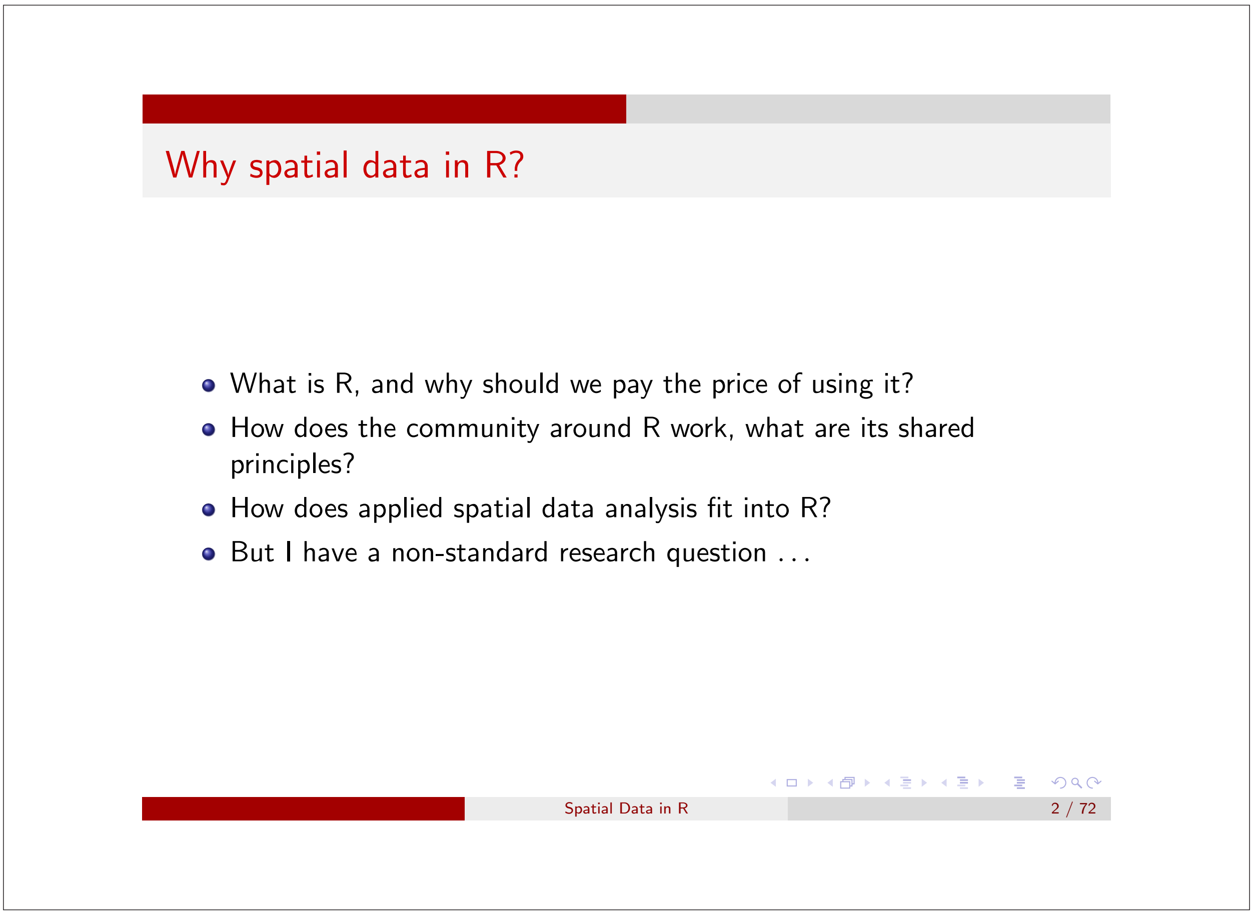
contents

Introduction to Spatial Data in R



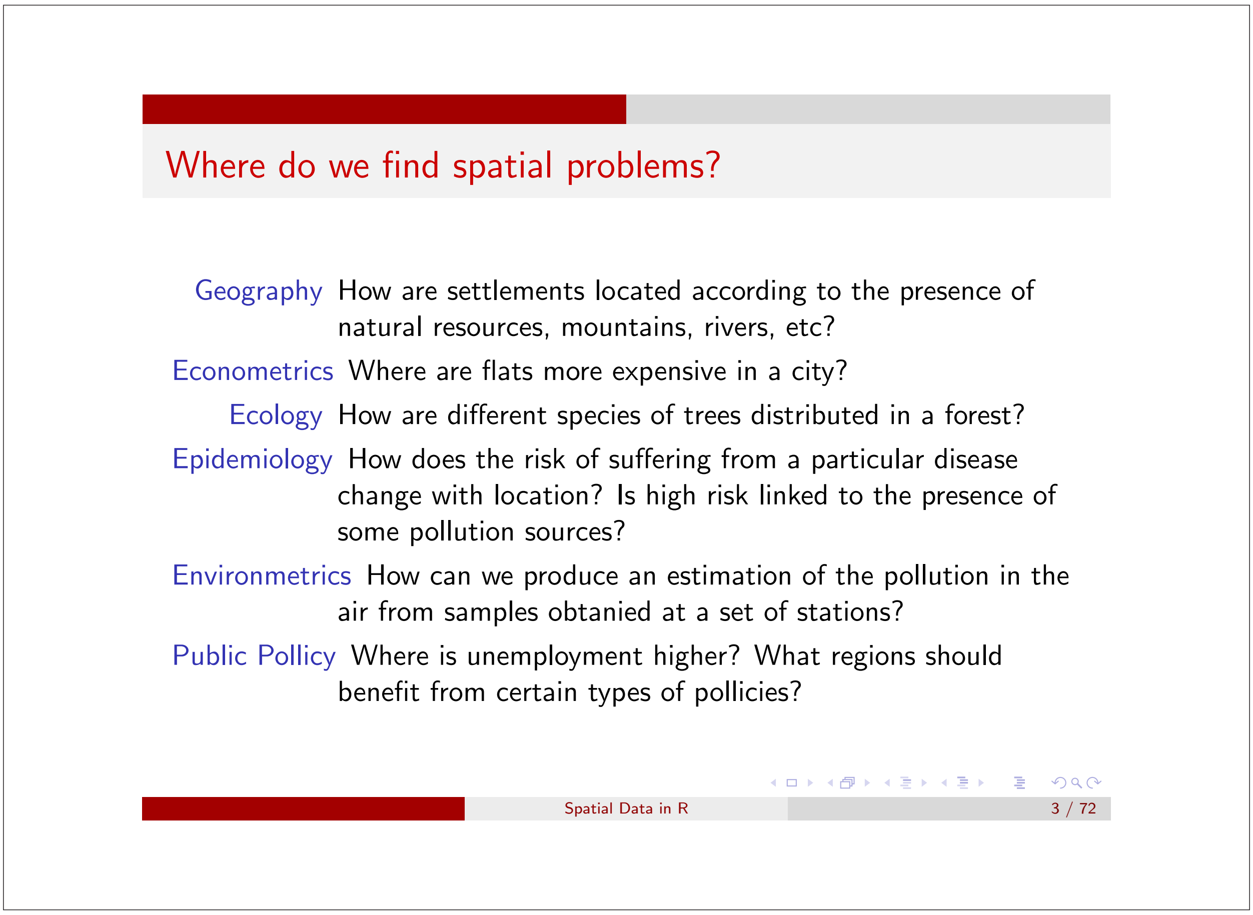
Introduction to Spatial Data in R

based on work by Roger S. Bivand, Edzer Pebesma and H. Rue



Why spatial data in R?

- What is R, and why should we pay the price of using it?
- How does the community around R work, what are its shared principles?
- How does applied spatial data analysis fit into R?
- But I have a non-standard research question . . .



Where do we find spatial problems?

Geography How are settlements located according to the presence of natural resources, mountains, rivers, etc?

Econometrics Where are flats more expensive in a city?

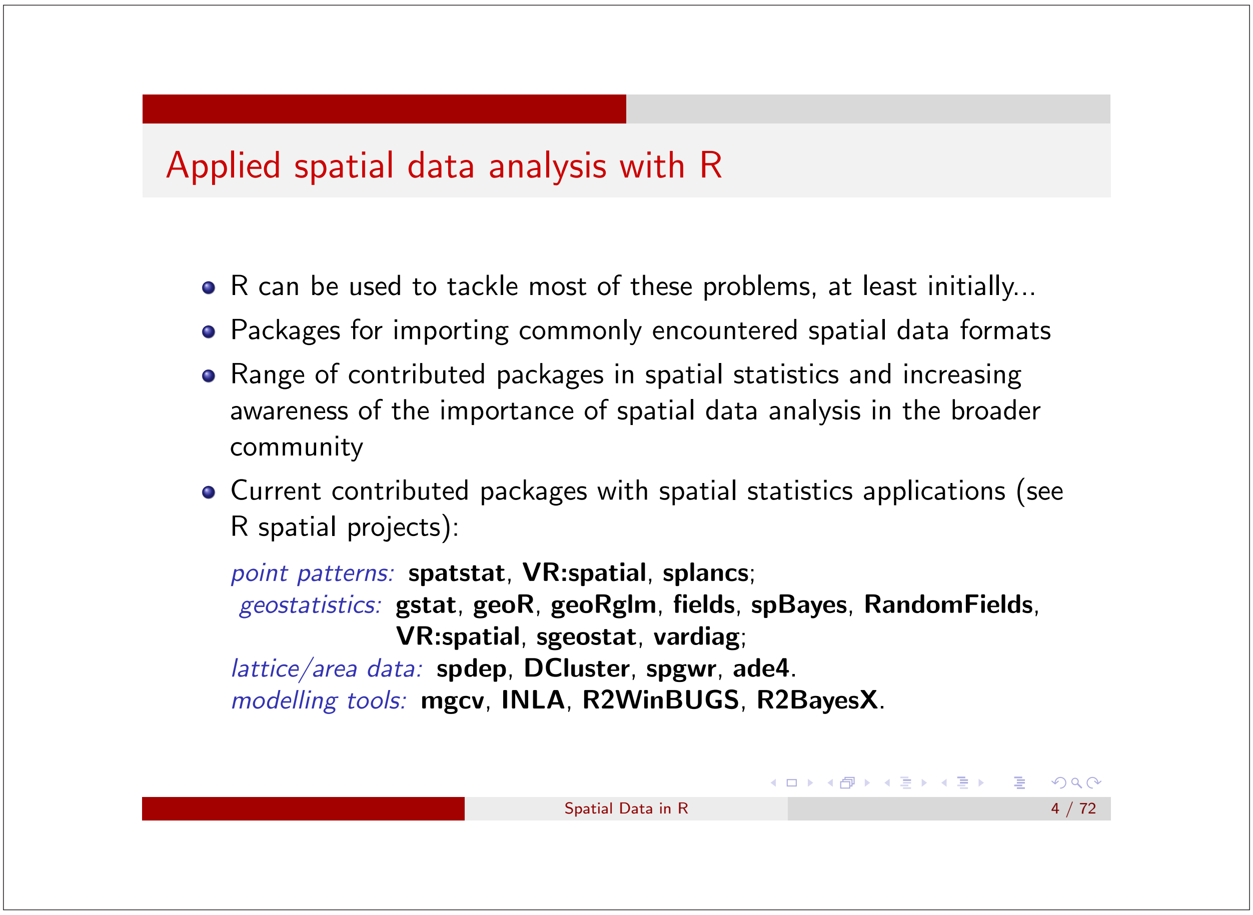
Ecology How are different species of trees distributed in a forest?

Epidemiology How does the risk of suffering from a particular disease change with location? Is high risk linked to the presence of some pollution sources?

Environmetrics How can we produce an estimation of the pollution in the air from samples obtanied at a set of stations?

Public Pollicy Where is unemployment higher? What regions should benefit from certain types of pollicies?

Spatial Data in R 3 / 72

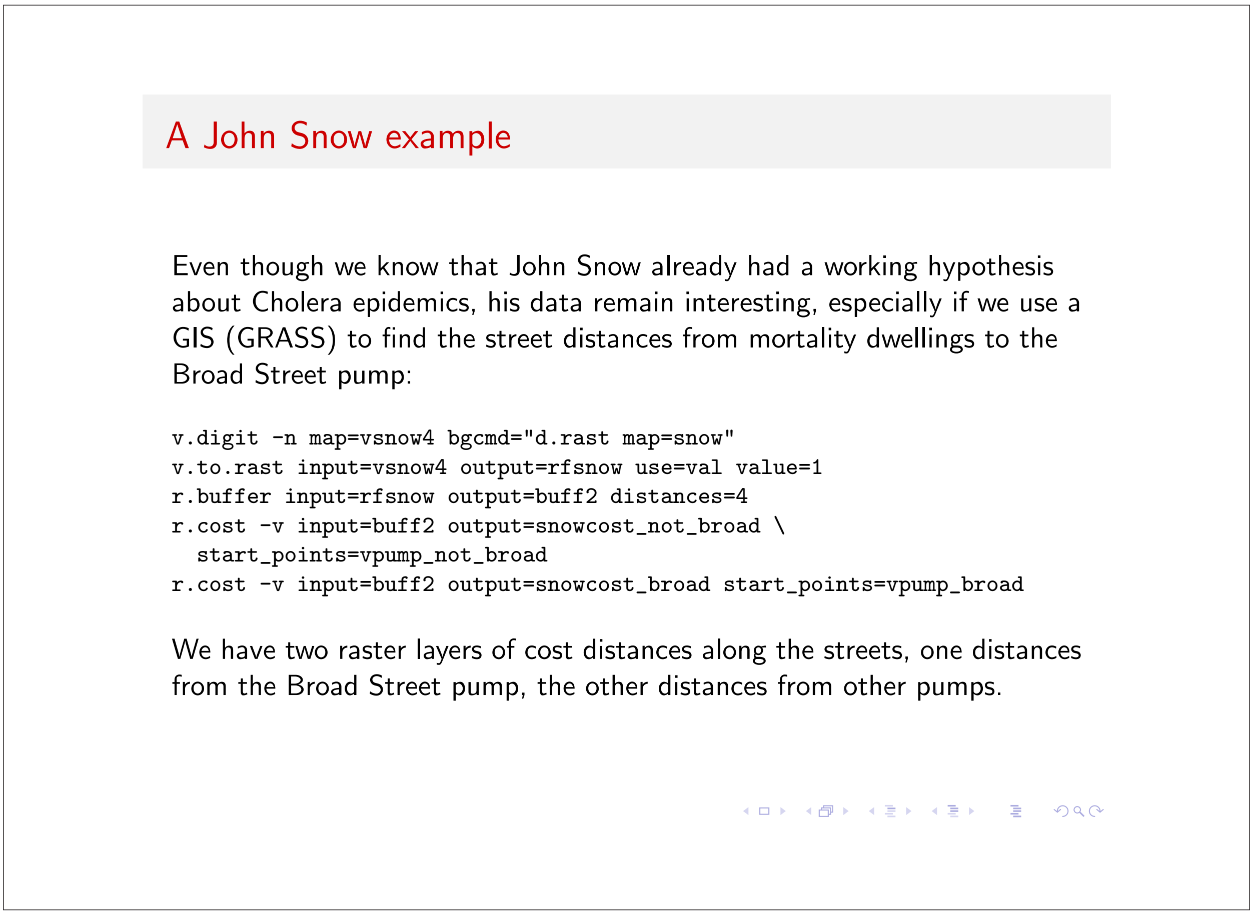


Applied spatial data analysis with R

- R can be used to tackle most of these problems, at least initially...
- Packages for importing commonly encountered spatial data formats
- Range of contributed packages in spatial statistics and increasing awareness of the importance of spatial data analysis in the broader community
- Current contributed packages with spatial statistics applications (see R spatial projects):

point patterns: **spatstat**, **VR:spatial**, **splancs**;
geostatistics: **gstat**, **geoR**, **geoRglm**, **fields**, **spBayes**, **RandomFields**, **VR:spatial**, **sgeostat**, **vardiag**;
lattice/area data: **spdep**, **DCluster**, **spgwr**, **ade4**.
modelling tools: **mgcv**, **INLA**, **R2WinBUGS**, **R2BayesX**.

Spatial Data in R 4 / 72

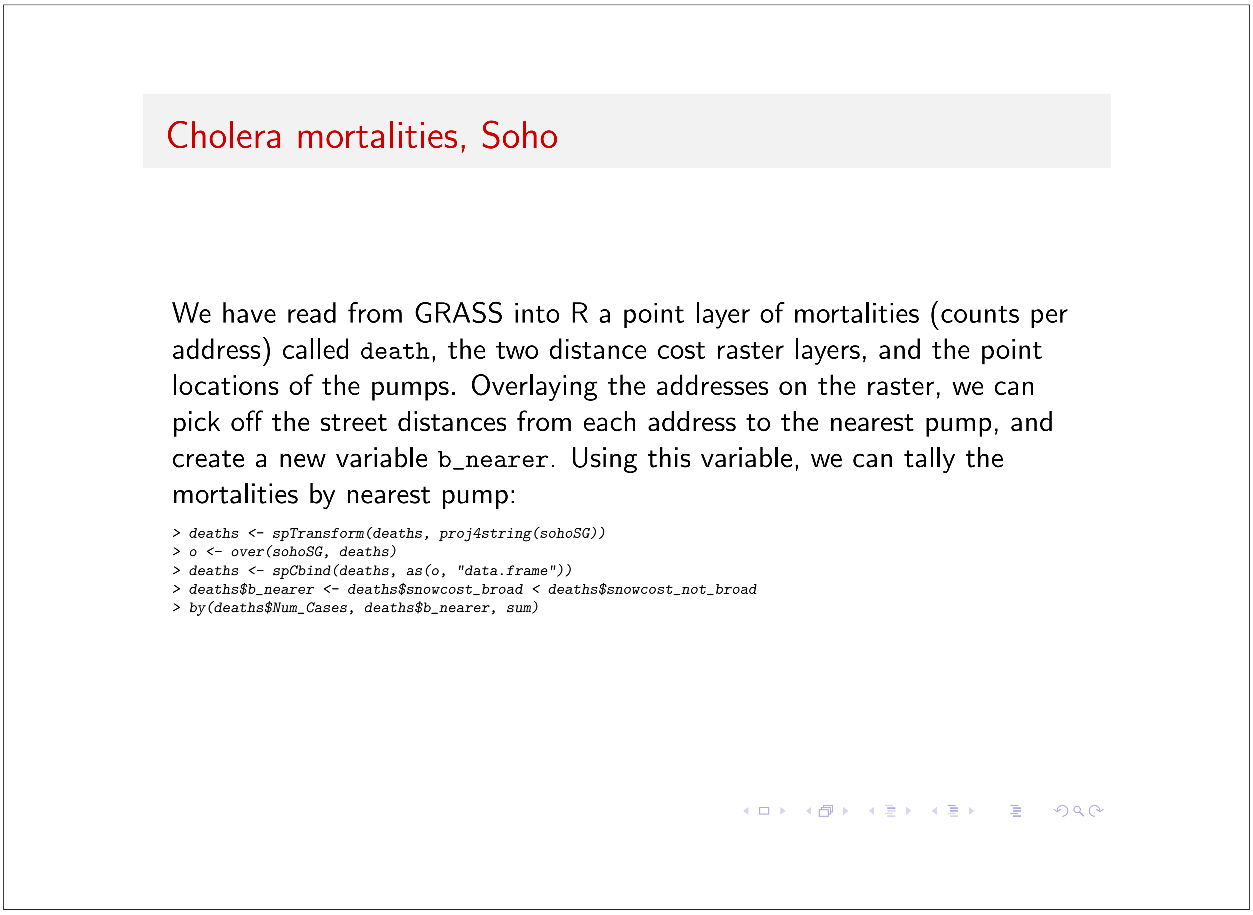


A John Snow example

Even though we know that John Snow already had a working hypothesis about Cholera epidemics, his data remain interesting, especially if we use a GIS (GRASS) to find the street distances from mortality dwellings to the Broad Street pump:

```
v.digit -n map=vsnow4 bgcmd="d.rast map=snow"
v.to.rast input=vsnow4 output=rfsnow use=val value=1
r.buffer input=rfsnow output=buff2 distances=4
r.cost -v input=buff2 output=snowcost_not_broad \
  start_points=vpump_not_broad
r.cost -v input=buff2 output=snowcost_broad start_points=vpump_broad
```

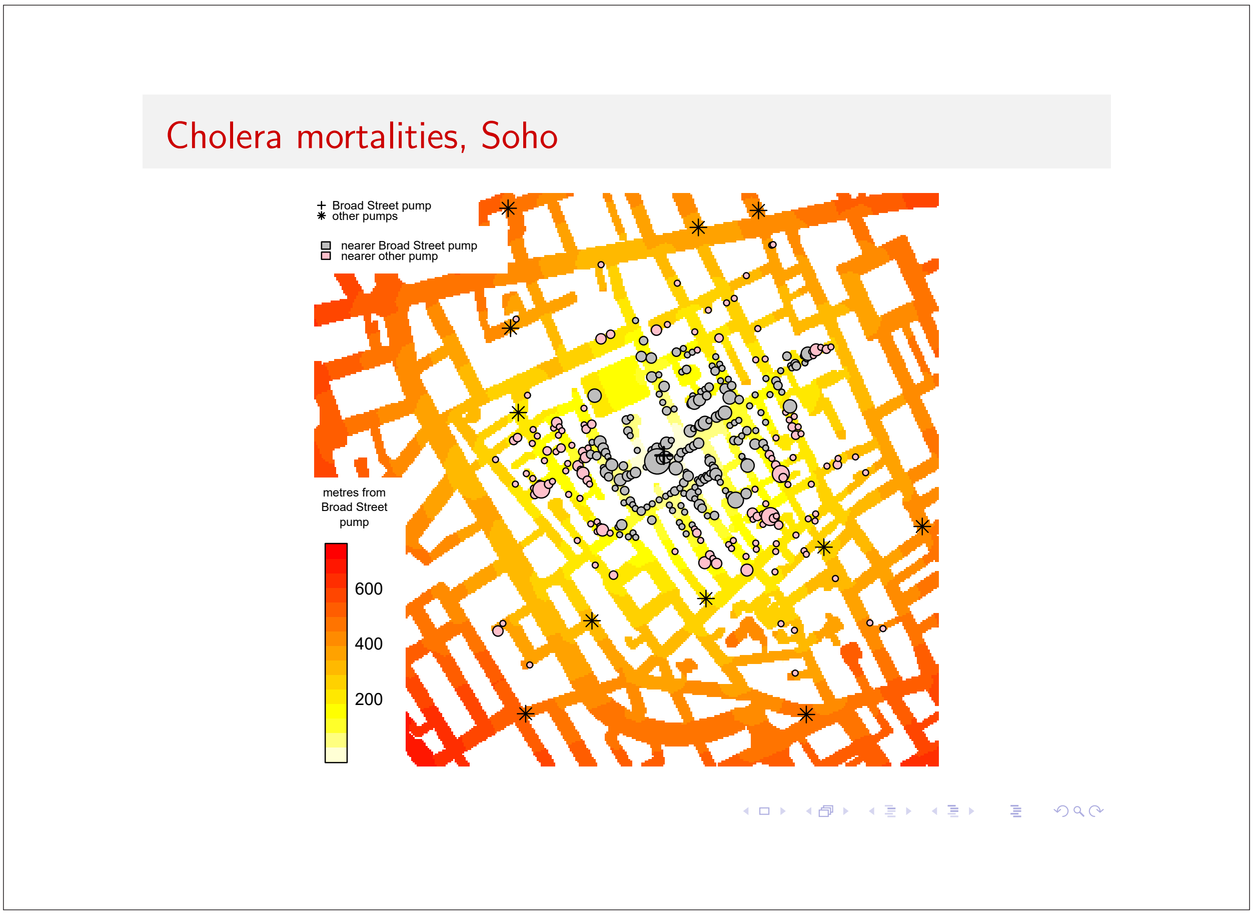
We have two raster layers of cost distances along the streets, one distances from the Broad Street pump, the other distances from other pumps.



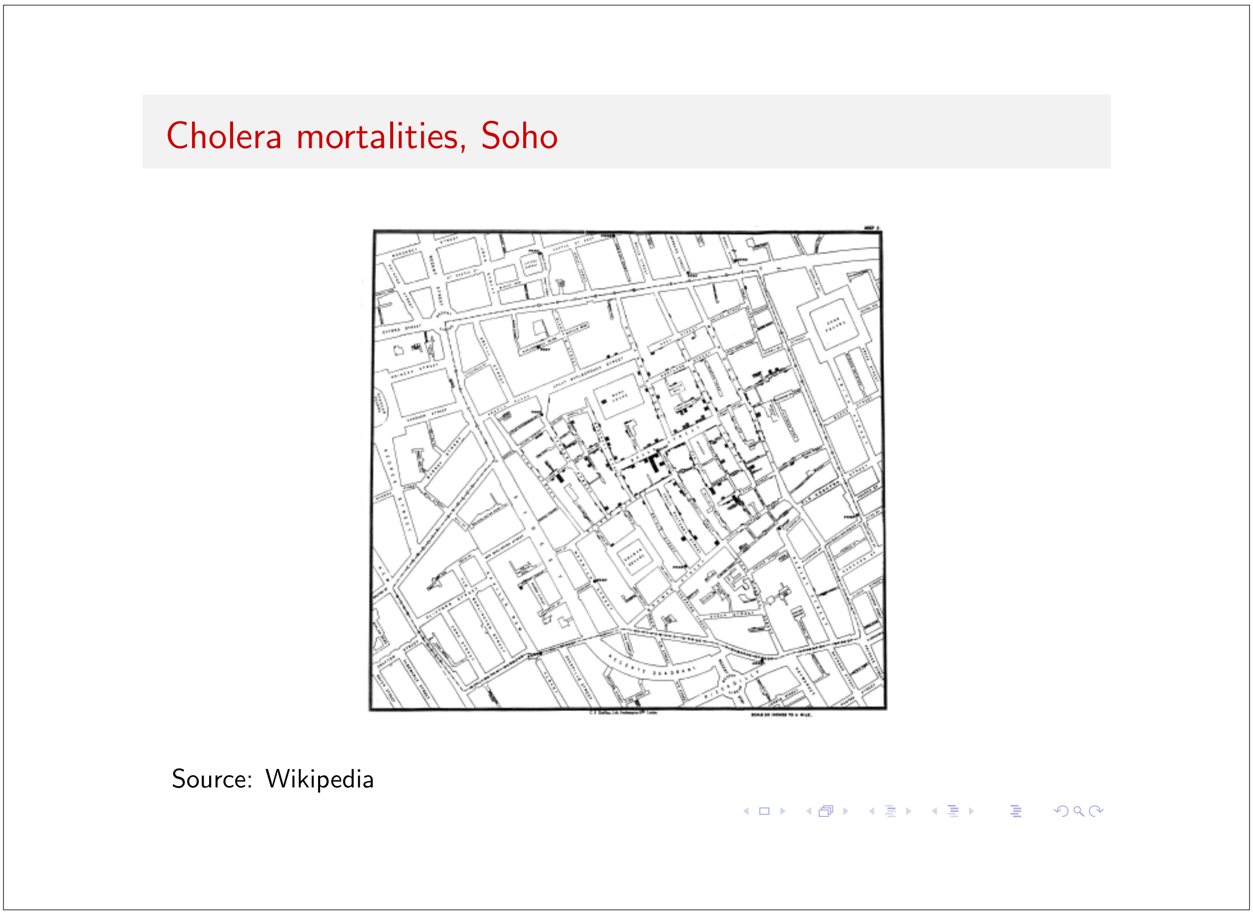
Cholera mortalities, Soho

We have read from GRASS into R a point layer of mortalities (counts per address) called `death`, the two distance cost raster layers, and the point locations of the pumps. Overlaying the addresses on the raster, we can pick off the street distances from each address to the nearest pump, and create a new variable `b_nearer`. Using this variable, we can tally the mortalities by nearest pump:

```
> deaths <- spTransform(deaths, proj4string(sohoSG))
> o <- over(sohoSG, deaths)
> deaths <- spCbind(deaths, as(o, "data.frame"))
> deaths$b_nearer <- deaths$snowcost_broad < deaths$snowcost_not_broad
> by(deaths$Num_Cases, deaths$b_nearer, sum)
```

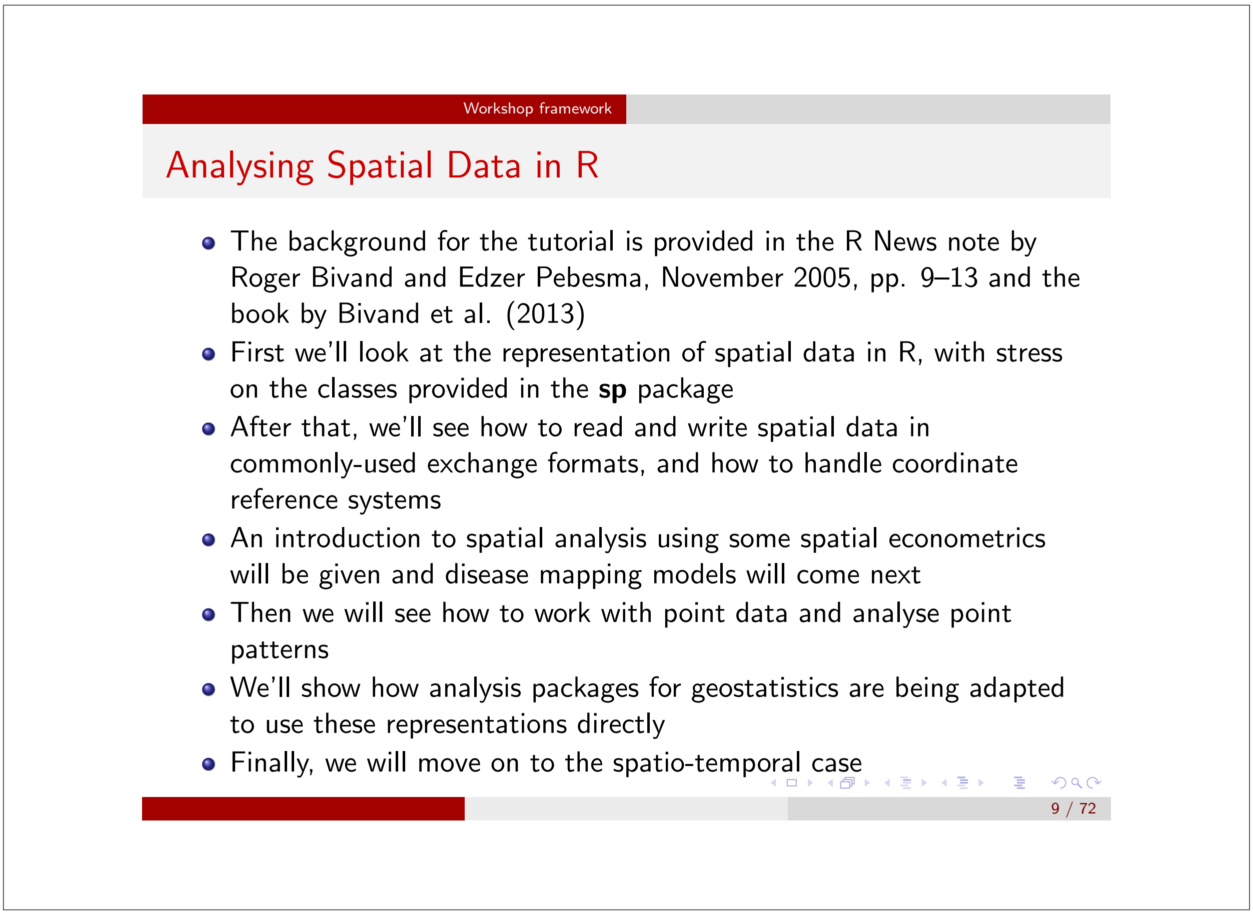


Cholera mortalities, Soho



Cholera mortalities, Soho

Source: Wikipedia

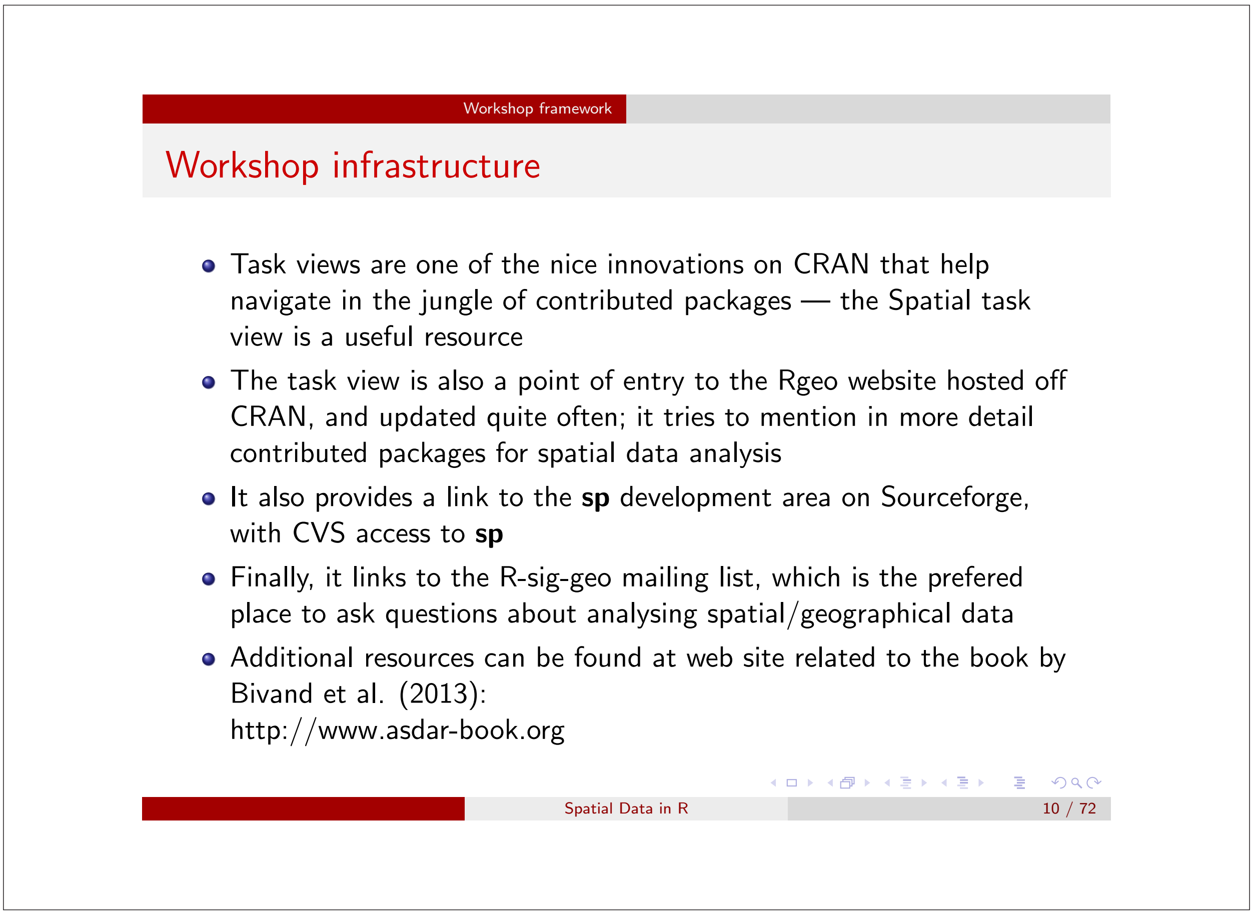


Workshop framework

Analysing Spatial Data in R

- The background for the tutorial is provided in the R News note by Roger Bivand and Edzer Pebesma, November 2005, pp. 9–13 and the book by Bivand et al. (2013)
- First we'll look at the representation of spatial data in R, with stress on the classes provided in the **sp** package
- After that, we'll see how to read and write spatial data in commonly-used exchange formats, and how to handle coordinate reference systems
- An introduction to spatial analysis using some spatial econometrics will be given and disease mapping models will come next
- Then we will see how to work with point data and analyse point patterns
- We'll show how analysis packages for geostatistics are being adapted to use these representations directly
- Finally, we will move on to the spatio-temporal case

9 / 72

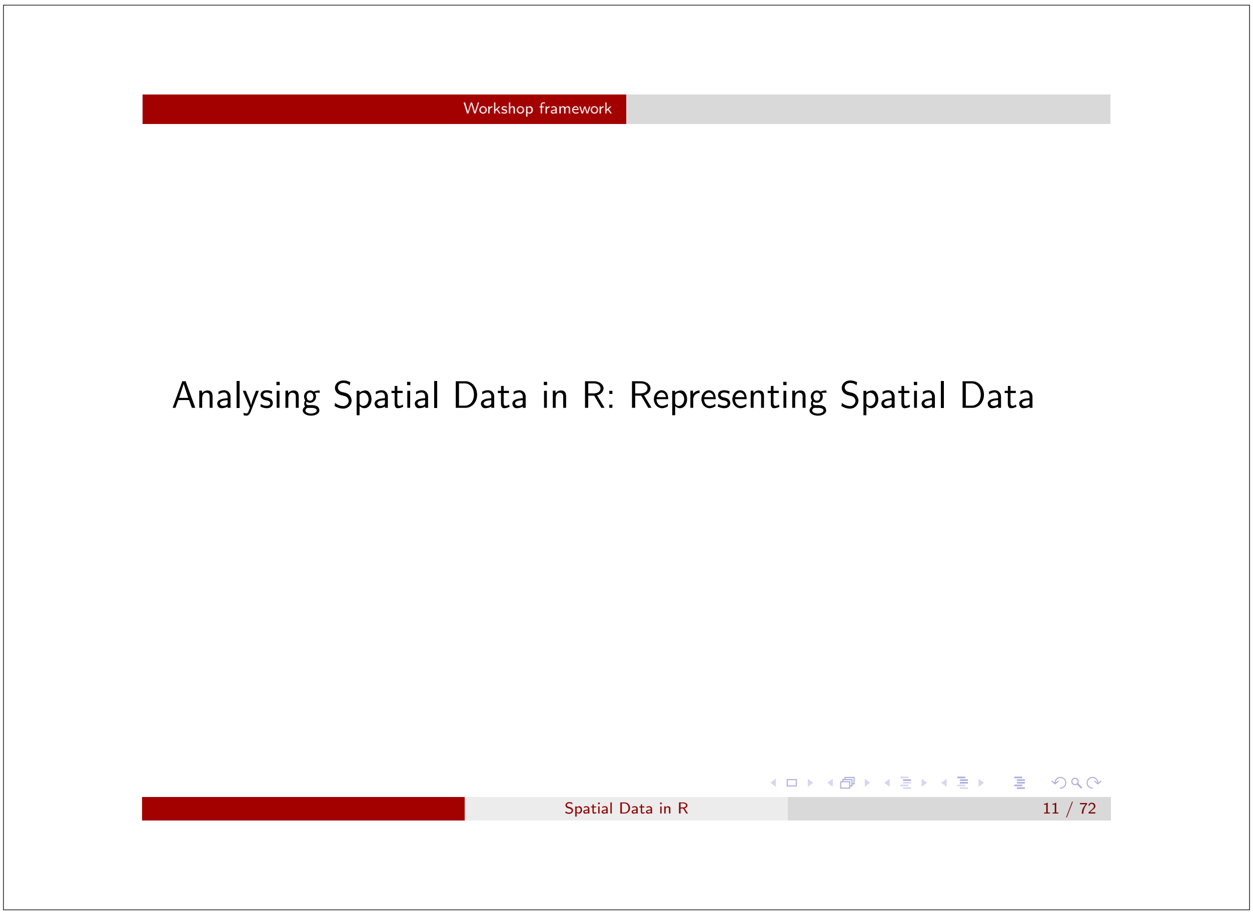


Workshop framework

Workshop infrastructure

- Task views are one of the nice innovations on CRAN that help navigate in the jungle of contributed packages — the Spatial task view is a useful resource
- The task view is also a point of entry to the Rgeo website hosted off CRAN, and updated quite often; it tries to mention in more detail contributed packages for spatial data analysis
- It also provides a link to the **sp** development area on Sourceforge, with CVS access to **sp**
- Finally, it links to the R-sig-geo mailing list, which is the prefered place to ask questions about analysing spatial/geographical data
- Additional resources can be found at web site related to the book by Bivand et al. (2013): http://www.asdar-book.org

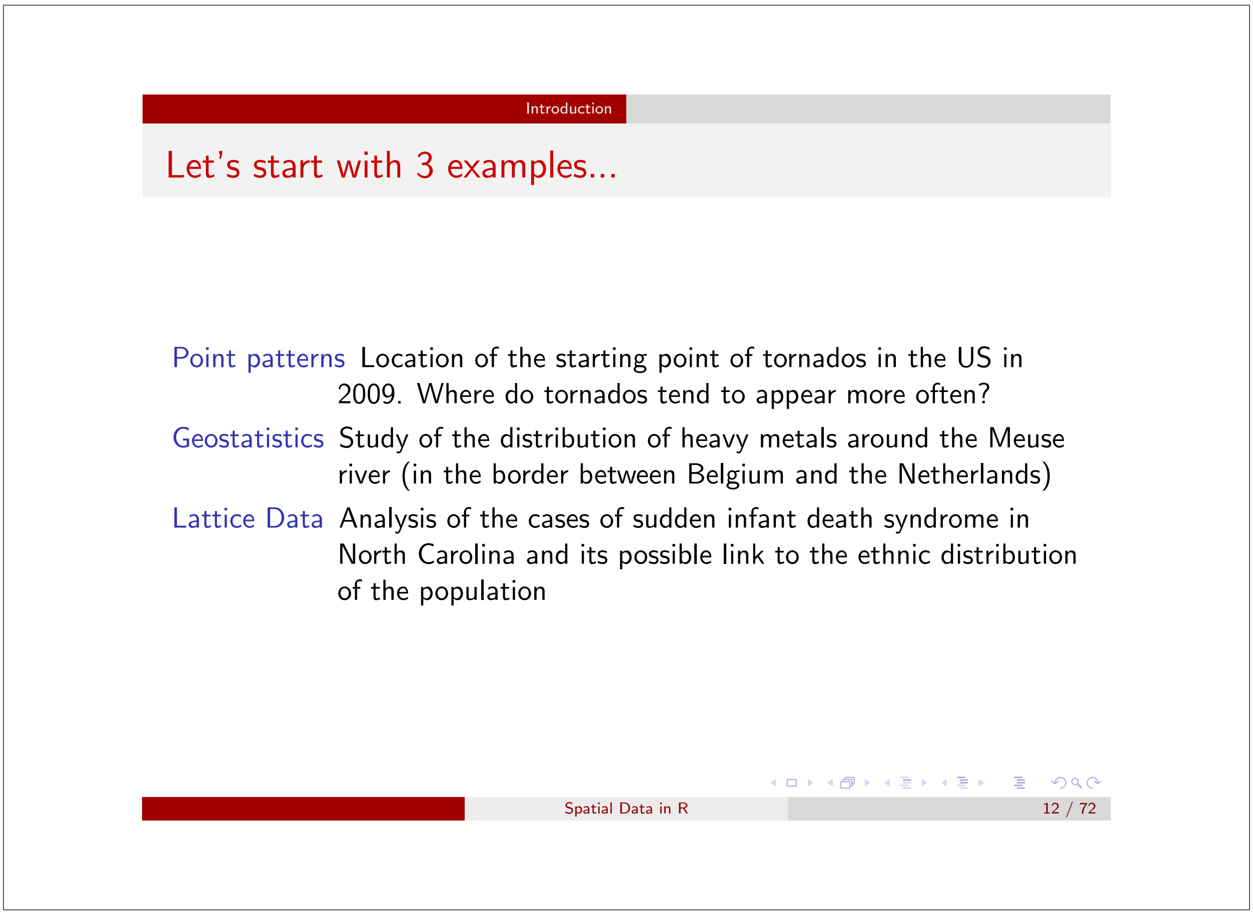
Spatial Data in R 10 / 72



Workshop framework

Analysing Spatial Data in R: Representing Spatial Data

Spatial Data in R 11 / 72



Introduction

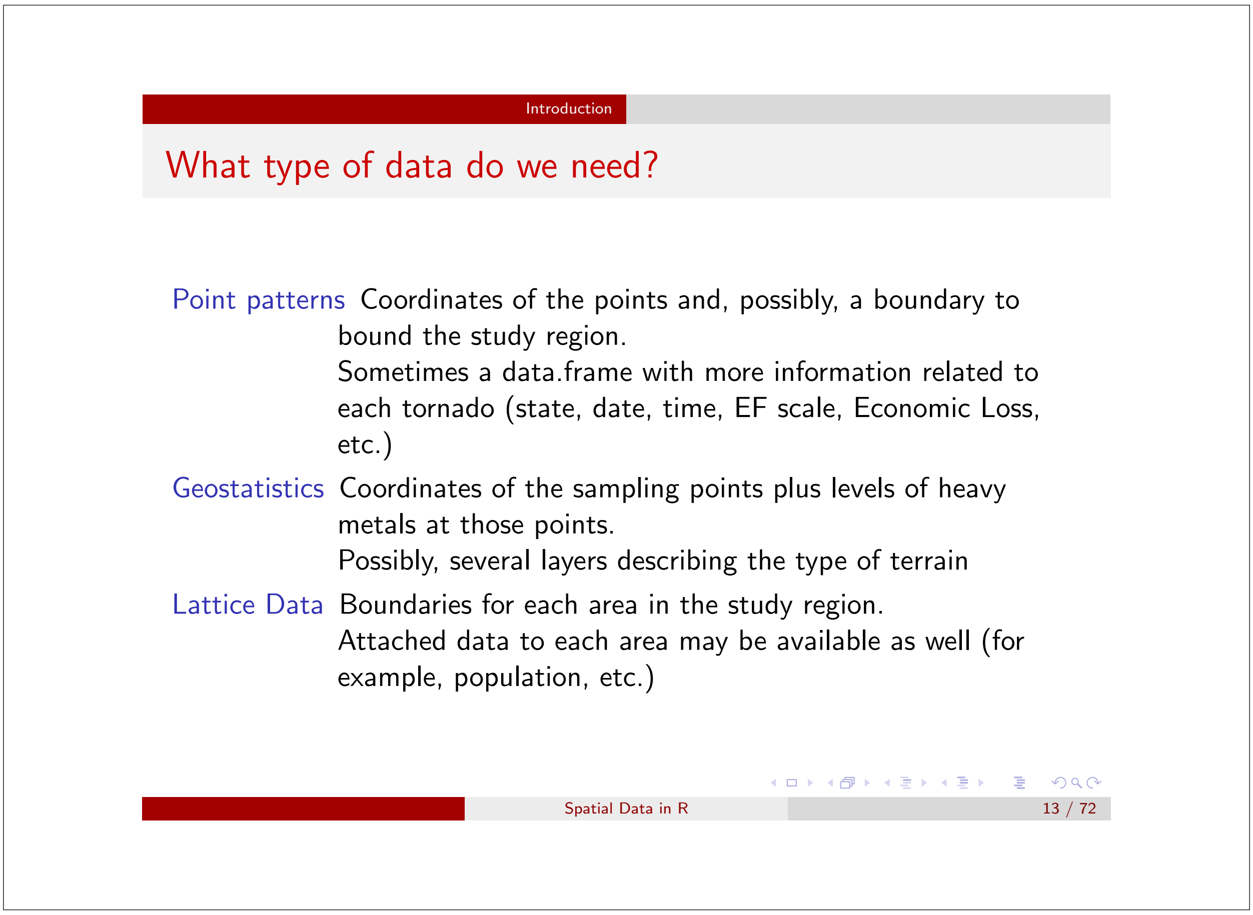
Let's start with 3 examples...

Point patterns Location of the starting point of tornados in the US in 2009. Where do tornados tend to appear more often?

Geostatistics Study of the distribution of heavy metals around the Meuse river (in the border between Belgium and the Netherlands)

Lattice Data Analysis of the cases of sudden infant death syndrome in North Carolina and its possible link to the ethnic distribution of the population

Spatial Data in R 12 / 72

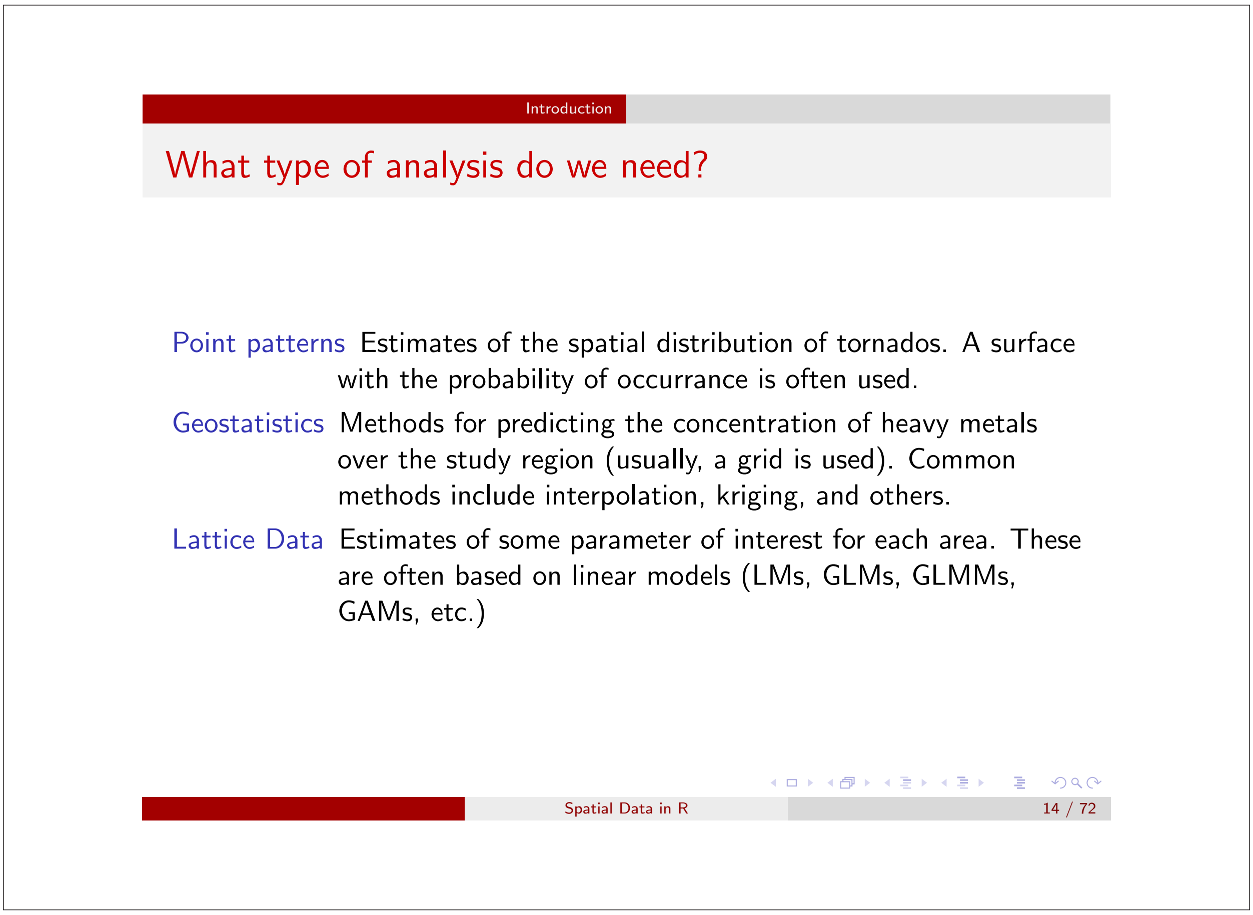


What type of data do we need?

Point patterns Coordinates of the points and, possibly, a boundary to bound the study region.
Sometimes a data.frame with more information related to each tornado (state, date, time, EF scale, Economic Loss, etc.)

Geostatistics Coordinates of the sampling points plus levels of heavy metals at those points.
Possibly, several layers describing the type of terrain

Lattice Data Boundaries for each area in the study region.
Attached data to each area may be available as well (for example, population, etc.)

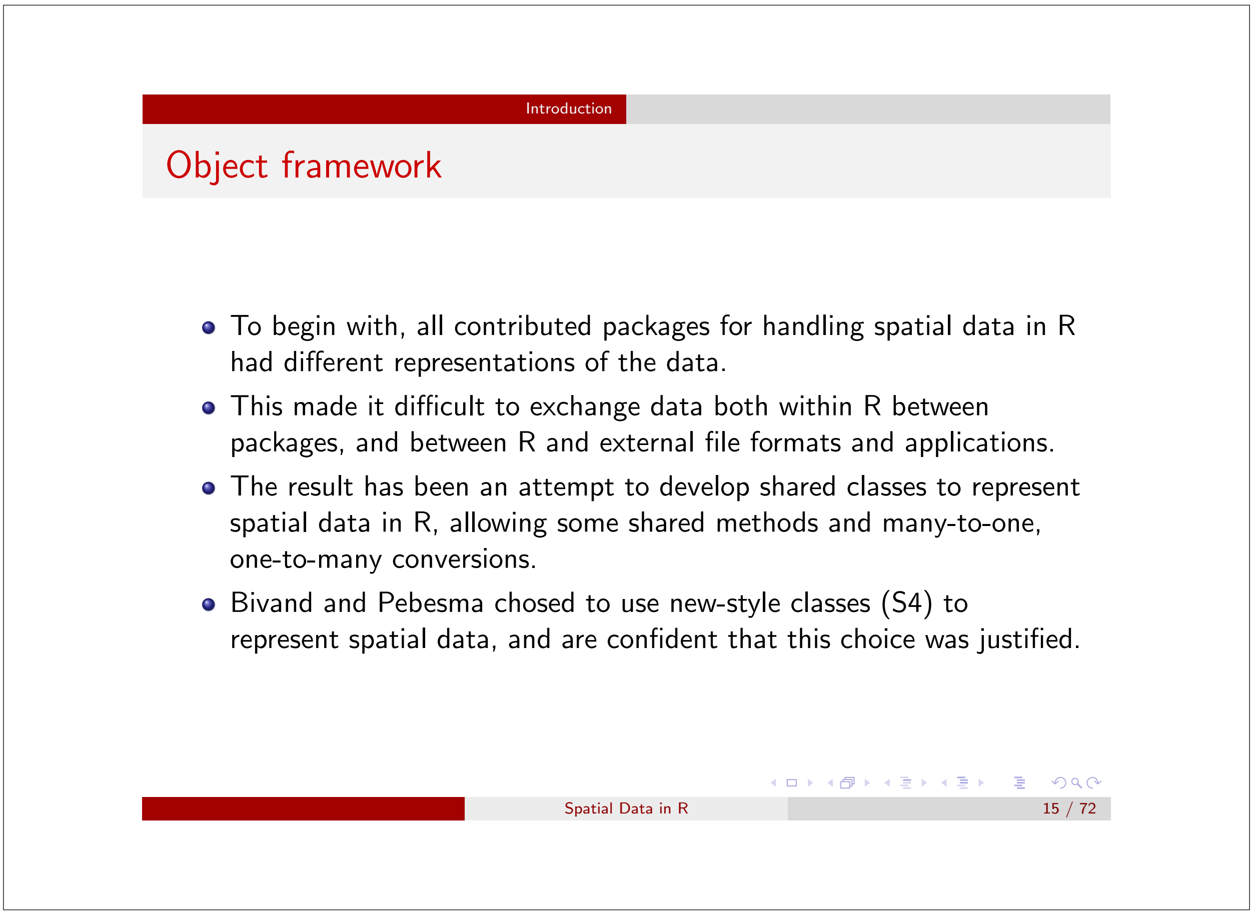


What type of analysis do we need?

Point patterns Estimates of the spatial distribution of tornados. A surface with the probability of occurrance is often used.

Geostatistics Methods for predicting the concentration of heavy metals over the study region (usually, a grid is used). Common methods include interpolation, kriging, and others.

Lattice Data Estimates of some parameter of interest for each area. These are often based on linear models (LMs, GLMs, GLMMs, GAMs, etc.)

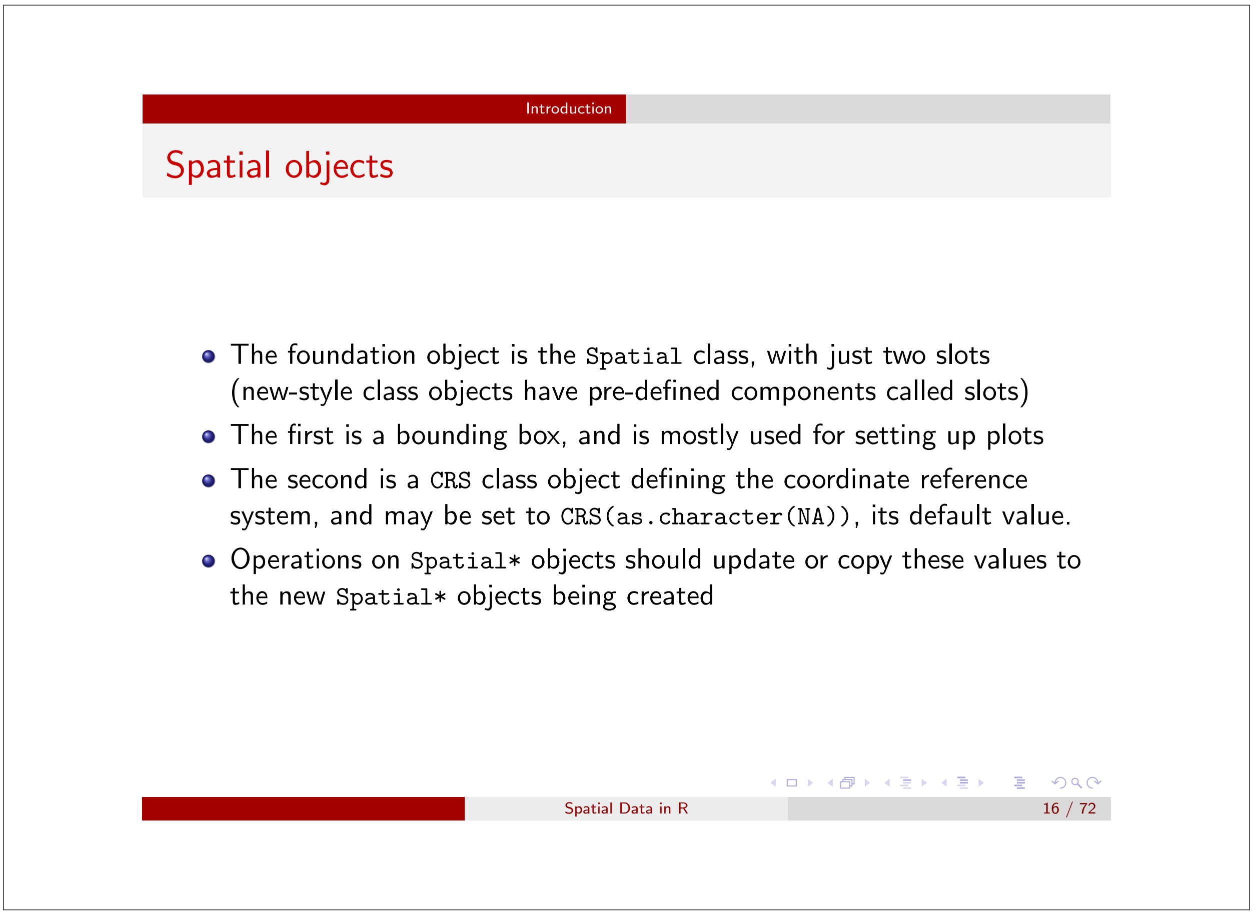


Introduction

Object framework

- To begin with, all contributed packages for handling spatial data in R had different representations of the data.
- This made it difficult to exchange data both within R between packages, and between R and external file formats and applications.
- The result has been an attempt to develop shared classes to represent spatial data in R, allowing some shared methods and many-to-one, one-to-many conversions.
- Bivand and Pebesma chosed to use new-style classes (S4) to represent spatial data, and are confident that this choice was justified.

Spatial Data in R 15 / 72



Introduction

Spatial objects

- The foundation object is the `Spatial` class, with just two slots (new-style class objects have pre-defined components called slots)
- The first is a bounding box, and is mostly used for setting up plots
- The second is a `CRS` class object defining the coordinate reference system, and may be set to `CRS(as.character(NA))`, its default value.
- Operations on `Spatial*` objects should update or copy these values to the new `Spatial*` objects being created

Spatial Data in R 16 / 72

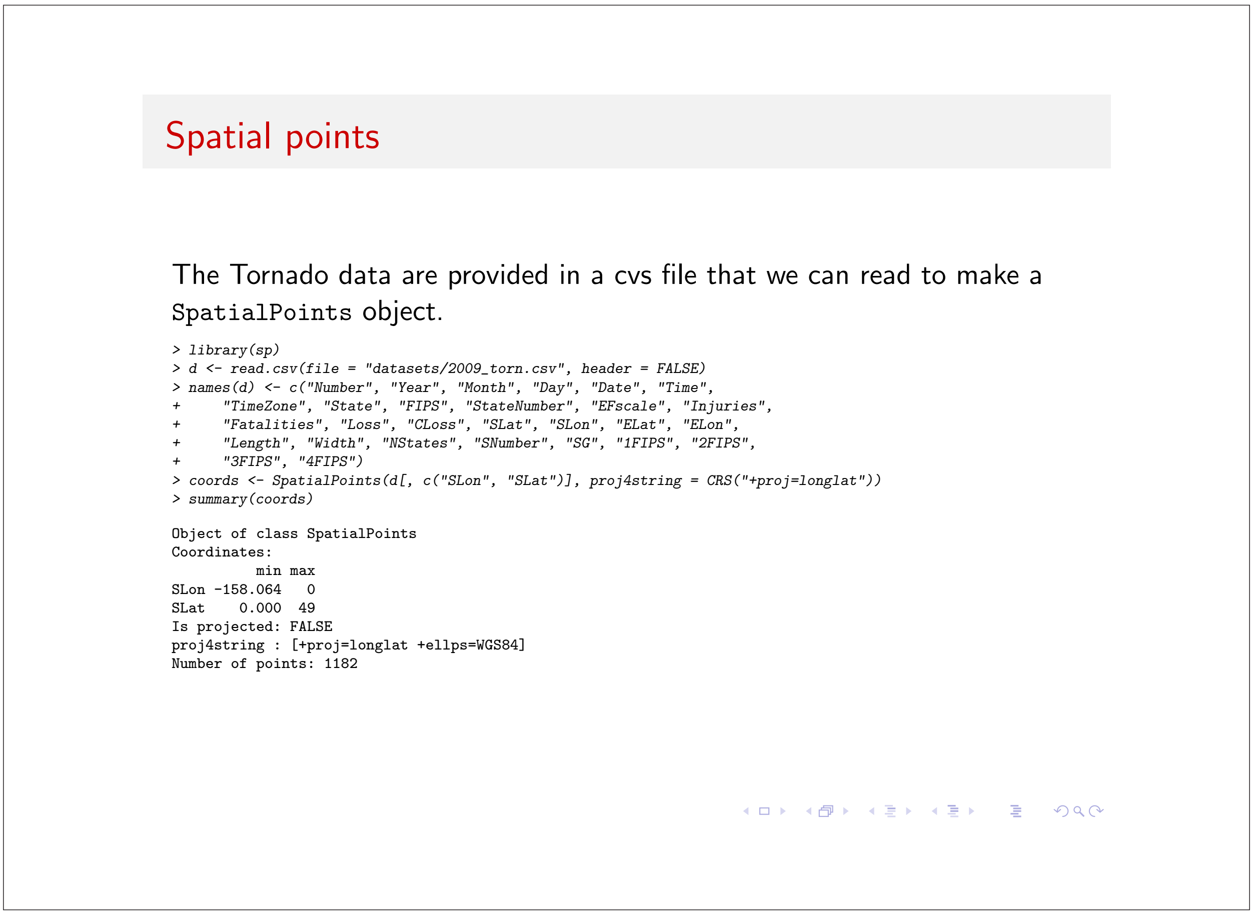
Spatial points

- The most basic spatial data object is a point, which may have 2 or 3 dimensions
- A single coordinate, or a set of such coordinates, may be used to define a `SpatialPoints` object; coordinates should be of mode `double` and will be promoted if not
- The points in a `SpatialPoints` object may be associated with a row of attributes to create a `SpatialPointsDataFrame` object
- The coordinates and attributes may, but do not have to be keyed to each other using ID values

Tornado Data 2009

- We will use some Tornado data to show the analysis of point patterns
- These data have been obtained from the *Storm Prediction Center*[a]
- Tornado data from 1955 until 2009 are available
- In addition to the coordinates, we have a wealth of related information for each tornado

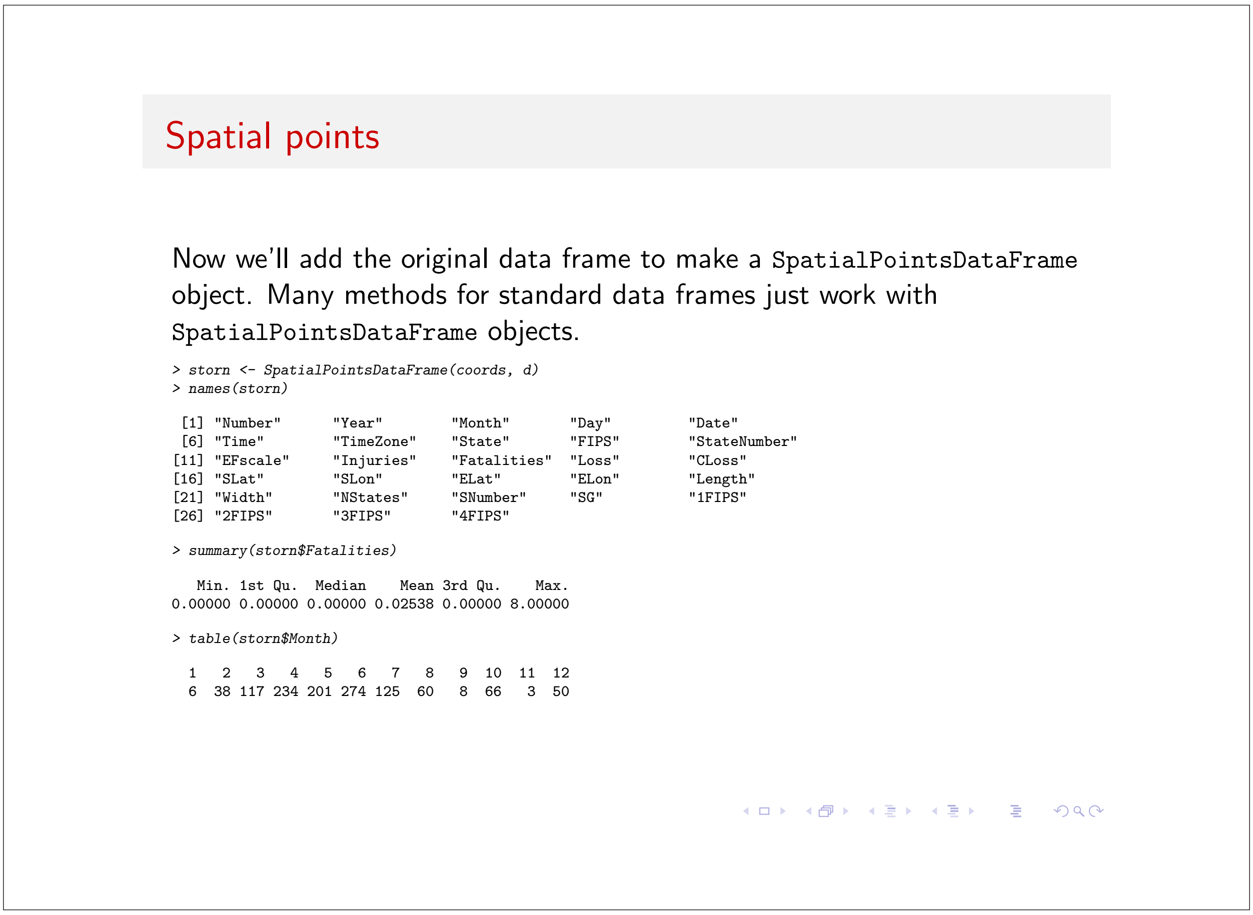
[a] http://www.spc.noaa.gov/wcm/index.html#data



Spatial points

The Tornado data are provided in a cvs file that we can read to make a `SpatialPoints` object.

```
> library(sp)
> d <- read.csv(file = "datasets/2009_torn.csv", header = FALSE)
> names(d) <- c("Number", "Year", "Month", "Day", "Date", "Time",
+      "TimeZone", "State", "FIPS", "StateNumber", "EFscale", "Injuries",
+      "Fatalities", "Loss", "CLoss", "SLat", "SLon", "ELat", "ELon",
+      "Length", "Width", "NStates", "SNumber", "SG", "1FIPS", "2FIPS",
+      "3FIPS", "4FIPS")
> coords <- SpatialPoints(d[, c("SLon", "SLat")], proj4string = CRS("+proj=longlat"))
> summary(coords)

Object of class SpatialPoints
Coordinates:
          min max
SLon -158.064   0
SLat    0.000  49
Is projected: FALSE
proj4string : [+proj=longlat +ellps=WGS84]
Number of points: 1182
```

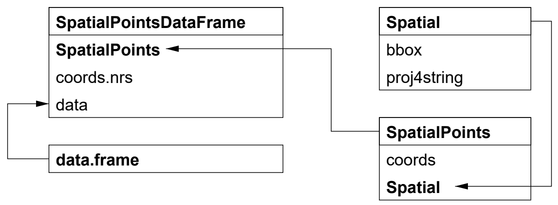


Spatial points

Now we'll add the original data frame to make a `SpatialPointsDataFrame` object. Many methods for standard data frames just work with `SpatialPointsDataFrame` objects.

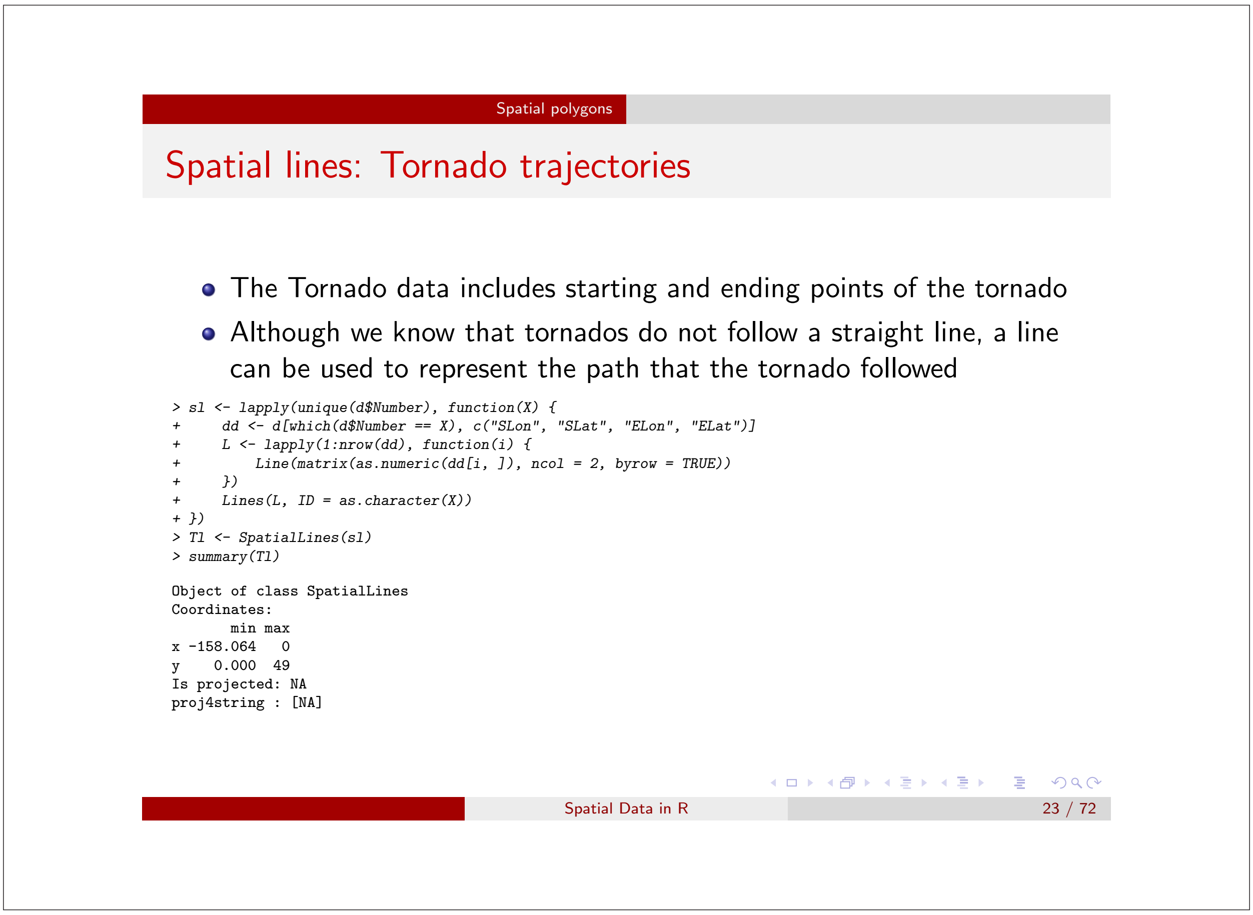
```
> storn <- SpatialPointsDataFrame(coords, d)
> names(storn)

 [1] "Number"      "Year"        "Month"       "Day"         "Date"
 [6] "Time"        "TimeZone"    "State"       "FIPS"        "StateNumber"
[11] "EFscale"     "Injuries"    "Fatalities"  "Loss"        "CLoss"
[16] "SLat"        "SLon"        "ELat"        "ELon"        "Length"
[21] "Width"       "NStates"     "SNumber"     "SG"          "1FIPS"
[26] "2FIPS"       "3FIPS"       "4FIPS"

> summary(storn$Fatalities)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.00000 0.00000 0.00000 0.02538 0.00000 8.00000

> table(storn$Month)

  1   2   3   4   5   6   7   8   9  10  11  12
  6  38 117 234 201 274 125  60   8  66   3  50
```

Spatial points classes and their slots



Spatial lines and polygons

- A `Line` object is just a spaghetti collection of 2D coordinates; a `Polygon` object is a `Line` object with equal first and last coordinates
- A `Lines` object is a list of `Line` objects, such as all the contours at a single elevation; the same relationship holds between a `Polygons` object and a list of `Polygon` objects, such as islands belonging to the same county
- `SpatialLines` and `SpatialPolygons` objects are made using lists of `Lines` or `Polygons` objects respectively
- `SpatialLinesDataFrame` and `SpatialPolygonsDataFrame` objects are defined using `SpatialLines` and `SpatialPolygons` objects and standard data frames, and the `ID` fields are here required to match the data frame row names



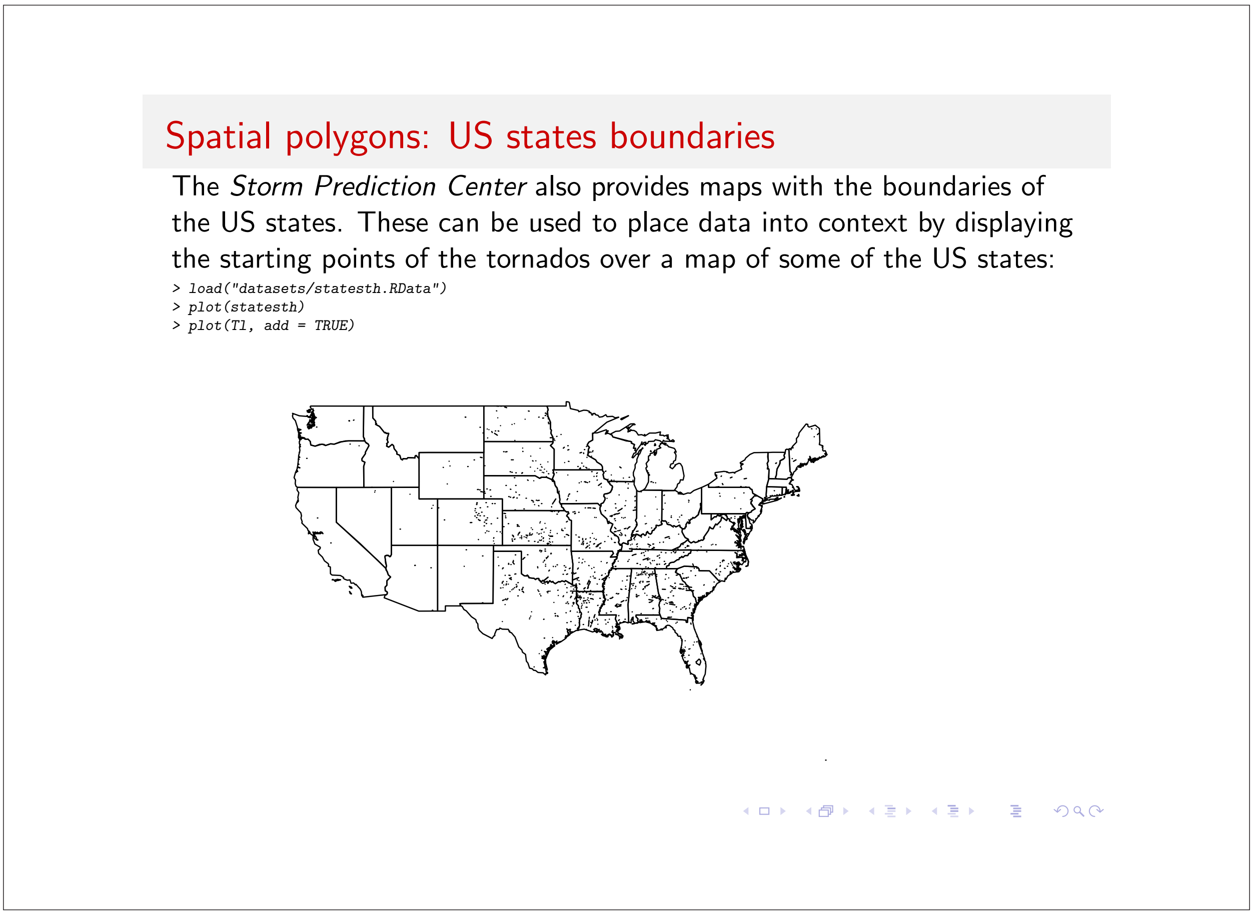
Spatial polygons

Spatial lines: Tornado trajectories

- The Tornado data includes starting and ending points of the tornado
- Although we know that tornados do not follow a straight line, a line can be used to represent the path that the tornado followed

```
> sl <- lapply(unique(d$Number), function(X) {
+     dd <- d[which(d$Number == X), c("SLon", "SLat", "ELon", "ELat")]
+     L <- lapply(1:nrow(dd), function(i) {
+         Line(matrix(as.numeric(dd[i, ]), ncol = 2, byrow = TRUE))
+     })
+     Lines(L, ID = as.character(X))
+ })
> Tl <- SpatialLines(sl)
> summary(Tl)

Object of class SpatialLines
Coordinates:
       min max
x -158.064   0
y    0.000  49
Is projected: NA
proj4string : [NA]
```

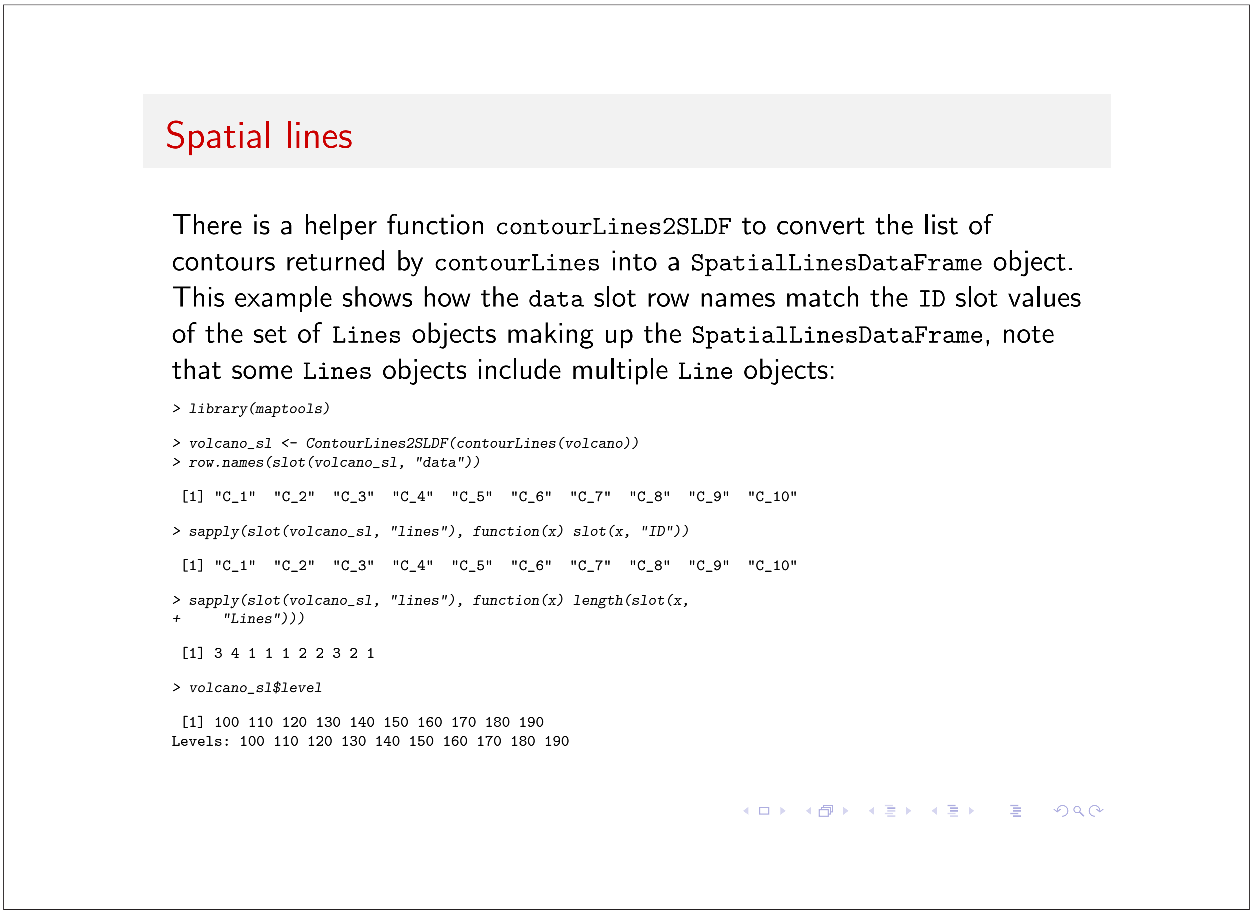
Spatial Data in R 23 / 72



Spatial polygons: US states boundaries

The *Storm Prediction Center* also provides maps with the boundaries of the US states. These can be used to place data into context by displaying the starting points of the tornados over a map of some of the US states:

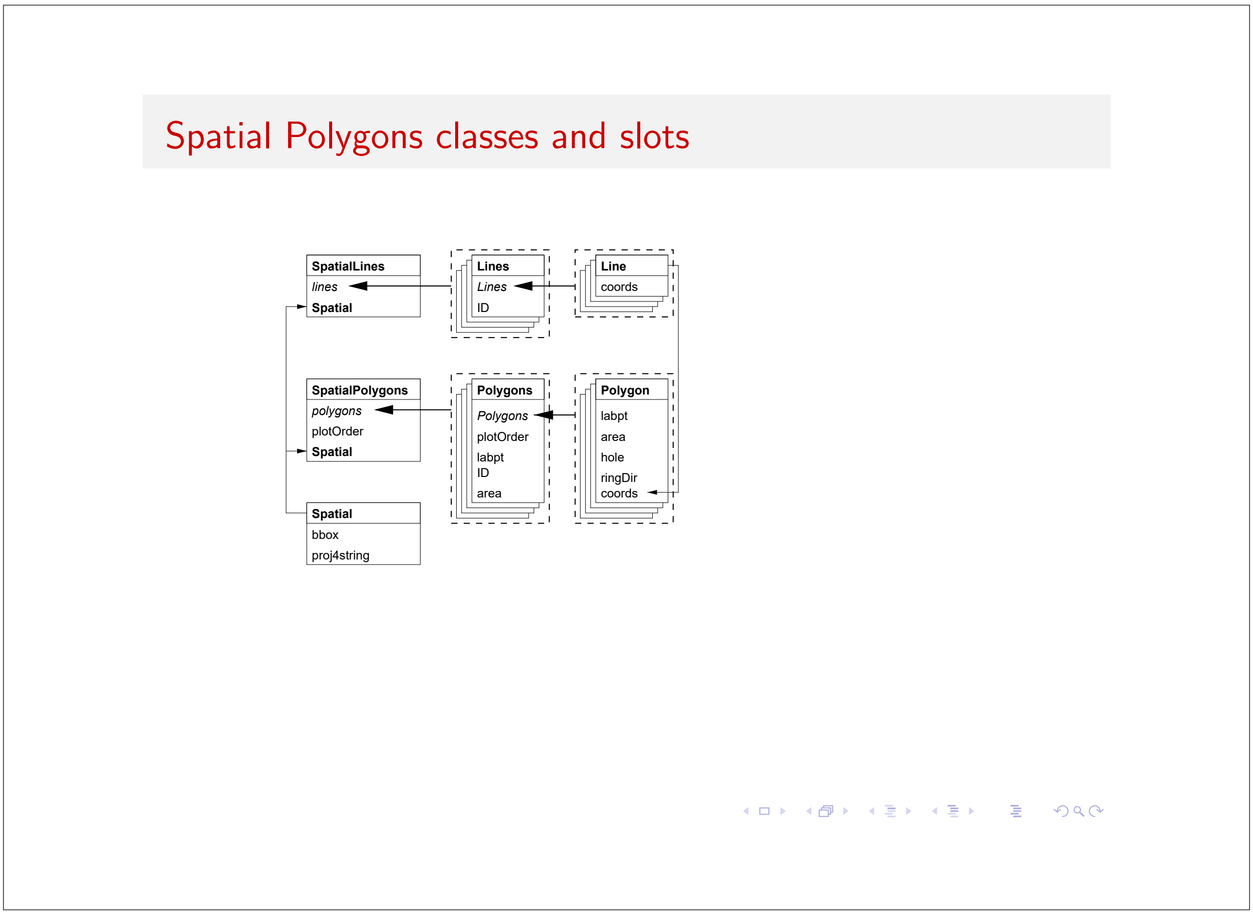
```
> load("datasets/statesth.RData")
> plot(statesth)
> plot(Tl, add = TRUE)
```



Spatial lines

There is a helper function `contourLines2SLDF` to convert the list of contours returned by `contourLines` into a `SpatialLinesDataFrame` object. This example shows how the `data` slot row names match the `ID` slot values of the set of `Lines` objects making up the `SpatialLinesDataFrame`, note that some `Lines` objects include multiple `Line` objects:

```
> library(maptools)

> volcano_sl <- ContourLines2SLDF(contourLines(volcano))
> row.names(slot(volcano_sl, "data"))

 [1] "C_1"  "C_2"  "C_3"  "C_4"  "C_5"  "C_6"  "C_7"  "C_8"  "C_9"  "C_10"

> sapply(slot(volcano_sl, "lines"), function(x) slot(x, "ID"))

 [1] "C_1"  "C_2"  "C_3"  "C_4"  "C_5"  "C_6"  "C_7"  "C_8"  "C_9"  "C_10"

> sapply(slot(volcano_sl, "lines"), function(x) length(slot(x,
+     "Lines")))

 [1] 3 4 1 1 1 2 2 3 2 1

> volcano_sl$level

 [1] 100 110 120 130 140 150 160 170 180 190
Levels: 100 110 120 130 140 150 160 170 180 190
```



Spatial Polygons classes and slots

SpatialLines
lines
Spatial

Lines
Lines
ID

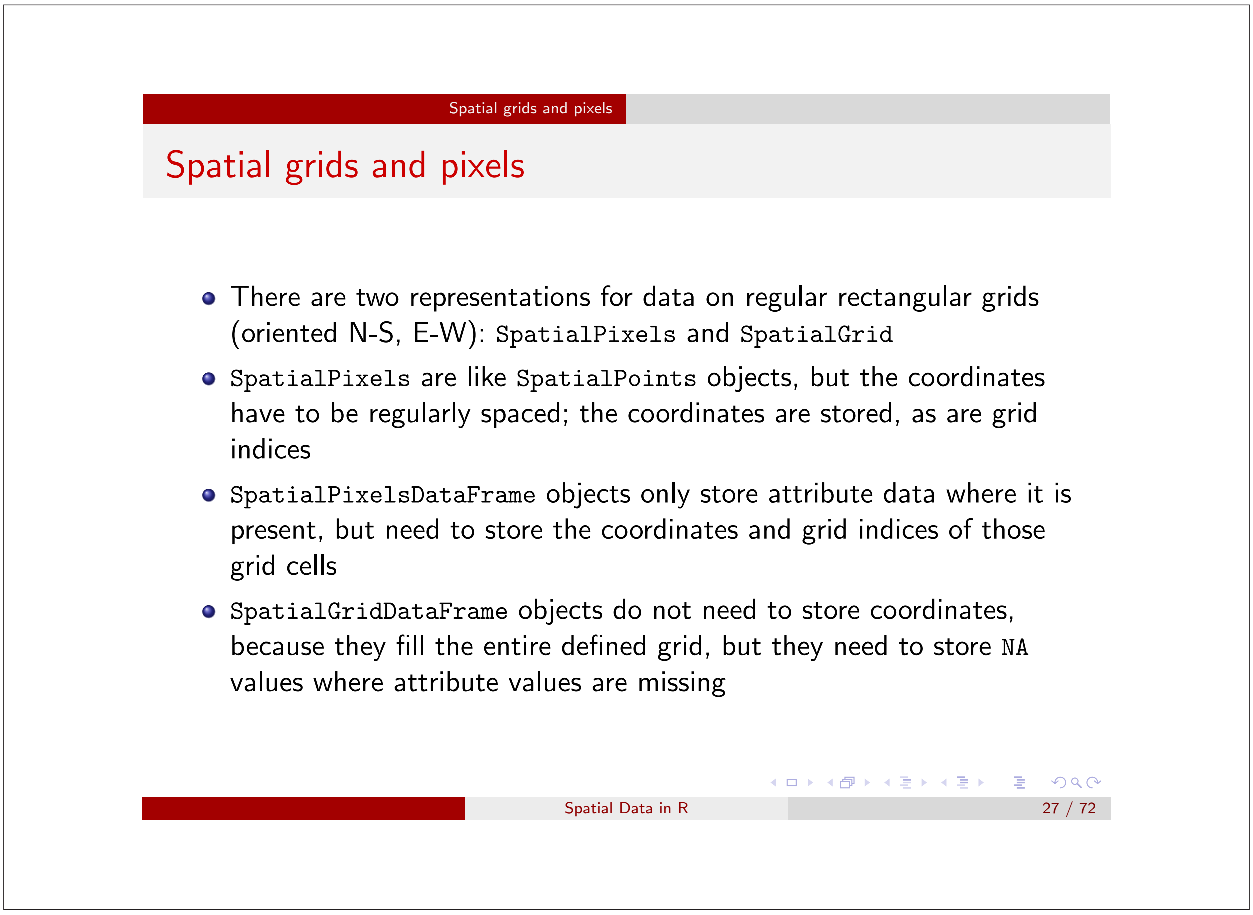
Line
coords

SpatialPolygons
polygons
plotOrder
Spatial

Polygons
Polygons
plotOrder
labpt
ID
area

Polygon
labpt
area
hole
ringDir
coords

Spatial
bbox
proj4string

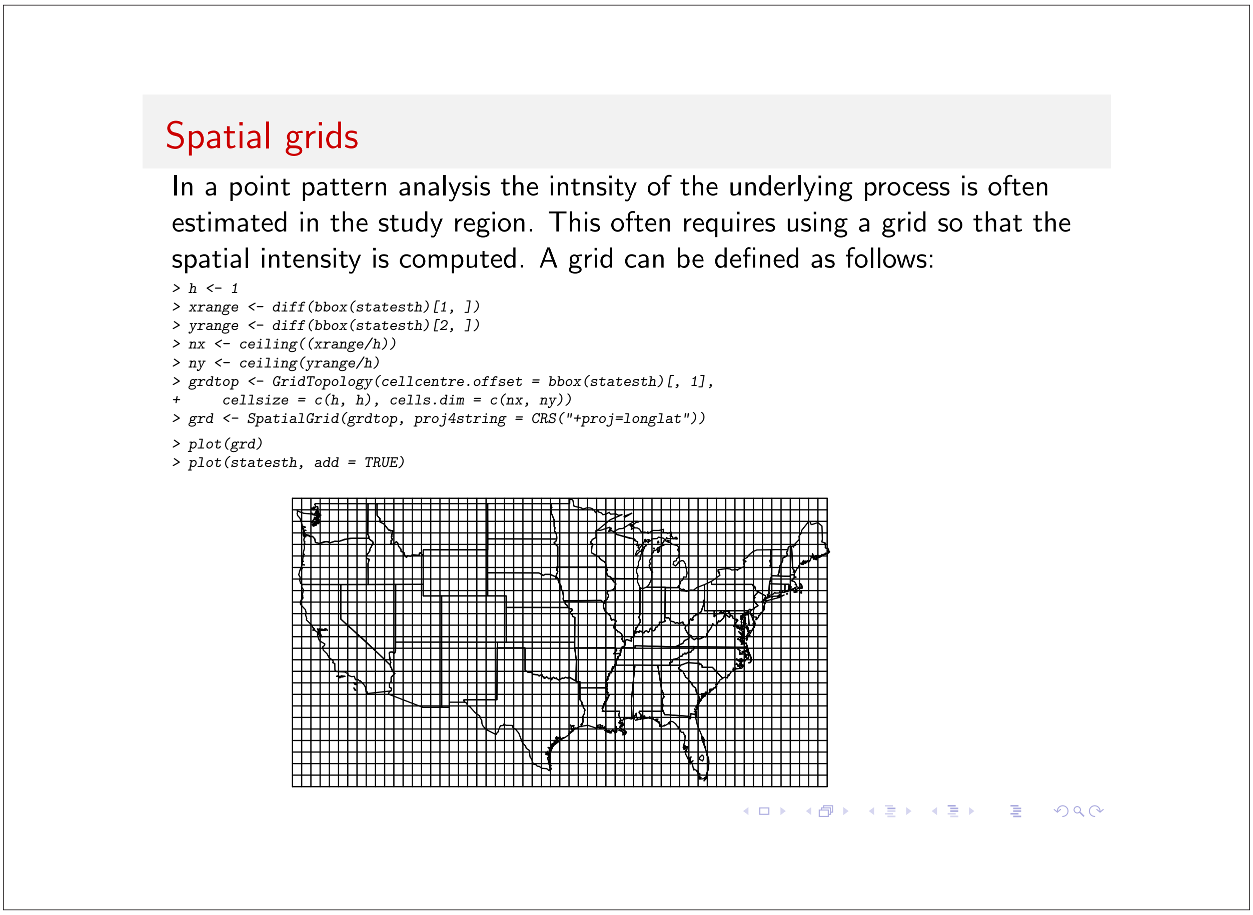


Spatial grids and pixels

Spatial grids and pixels

- There are two representations for data on regular rectangular grids (oriented N-S, E-W): `SpatialPixels` and `SpatialGrid`
- `SpatialPixels` are like `SpatialPoints` objects, but the coordinates have to be regularly spaced; the coordinates are stored, as are grid indices
- `SpatialPixelsDataFrame` objects only store attribute data where it is present, but need to store the coordinates and grid indices of those grid cells
- `SpatialGridDataFrame` objects do not need to store coordinates, because they fill the entire defined grid, but they need to store `NA` values where attribute values are missing

Spatial Data in R 27 / 72



Spatial grids

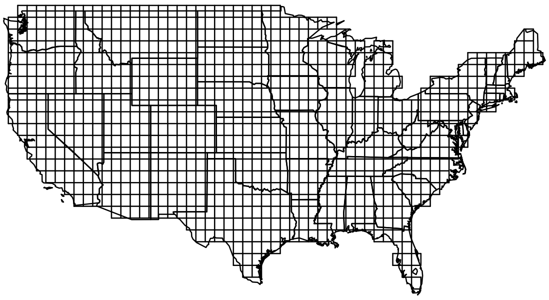
In a point pattern analysis the intnsity of the underlying process is often estimated in the study region. This often requires using a grid so that the spatial intensity is computed. A grid can be defined as follows:

```
> h <- 1
> xrange <- diff(bbox(statesth)[1, ])
> yrange <- diff(bbox(statesth)[2, ])
> nx <- ceiling((xrange/h))
> ny <- ceiling(yrange/h)
> grdtop <- GridTopology(cellcentre.offset = bbox(statesth)[, 1],
+     cellsize = c(h, h), cells.dim = c(nx, ny))
> grd <- SpatialGrid(grdtop, proj4string = CRS("+proj=longlat"))
> plot(grd)
> plot(statesth, add = TRUE)
```

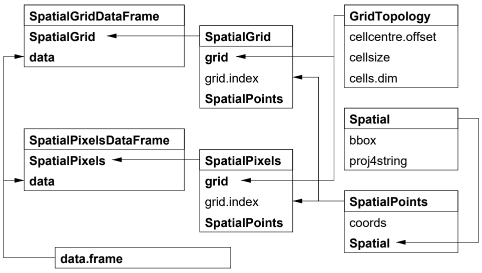
Spatial pixels

Although this is a convenient way of creating and using grids, some of the points fall in the middle of the ocean, far from any US state. We could combine the US boundaries and the grid to keep only the points that are inside the US boundaries in a `SpatialPixels` object:

```
> grdidx <- over(grd, statesth)[, 1]
> grd2 <- SpatialPixels(SpatialPoints(coordinates(grd))[!is.na(grdidx),
+     ])
> proj4string(grd2) <- CRS("+proj=longlat")
> plot(grd2)
> plot(statesth, add = TRUE)
```



Spatial grid and pixels classes and their slots



SpatialGridDataFrame
SpatialGrid
data

SpatialGrid
grid
grid.index
SpatialPoints

GridTopology
cellcentre.offset
cellsize
cells.dim

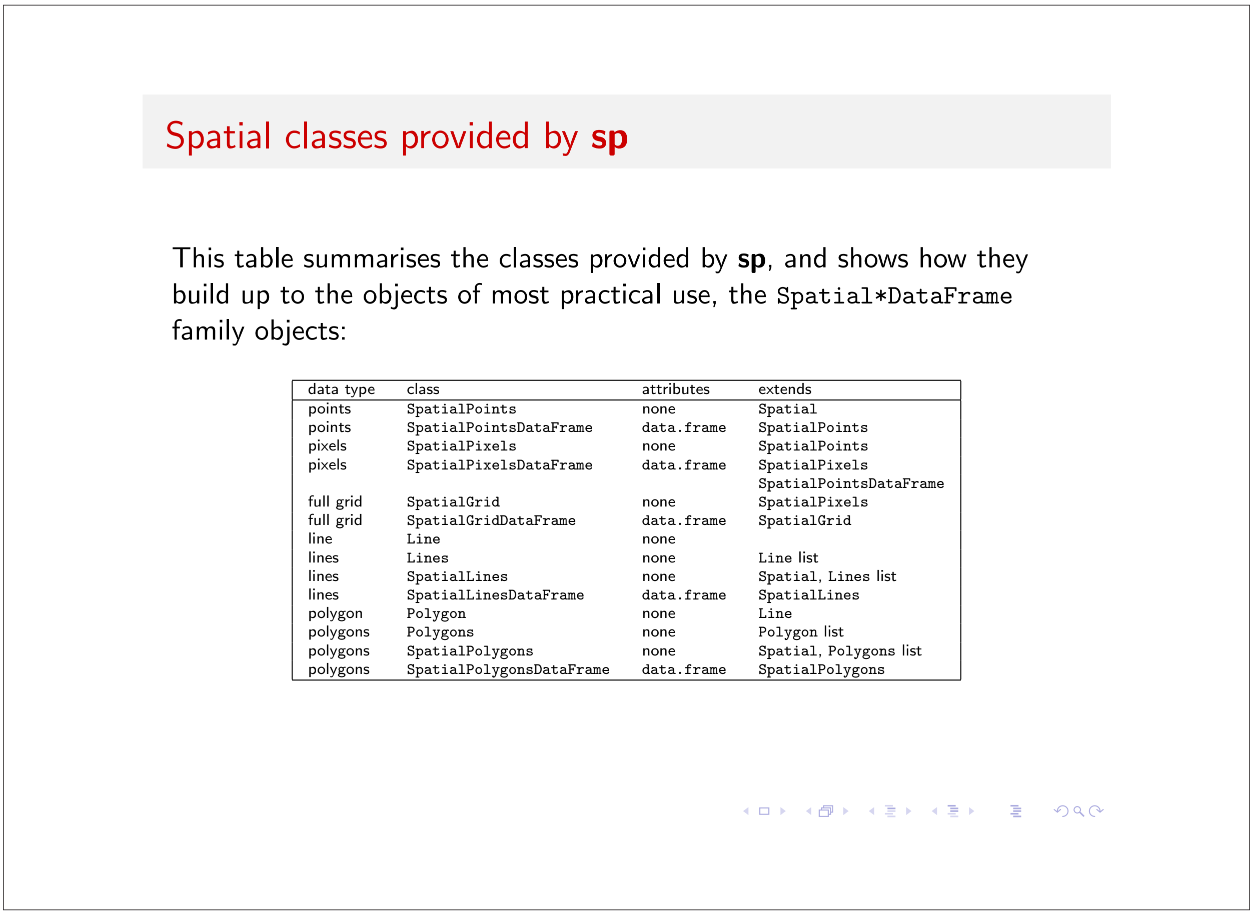
Spatial
bbox
proj4string

SpatialPixelsDataFrame
SpatialPixels
data

SpatialPixels
grid
grid.index
SpatialPoints

SpatialPoints
coords
Spatial

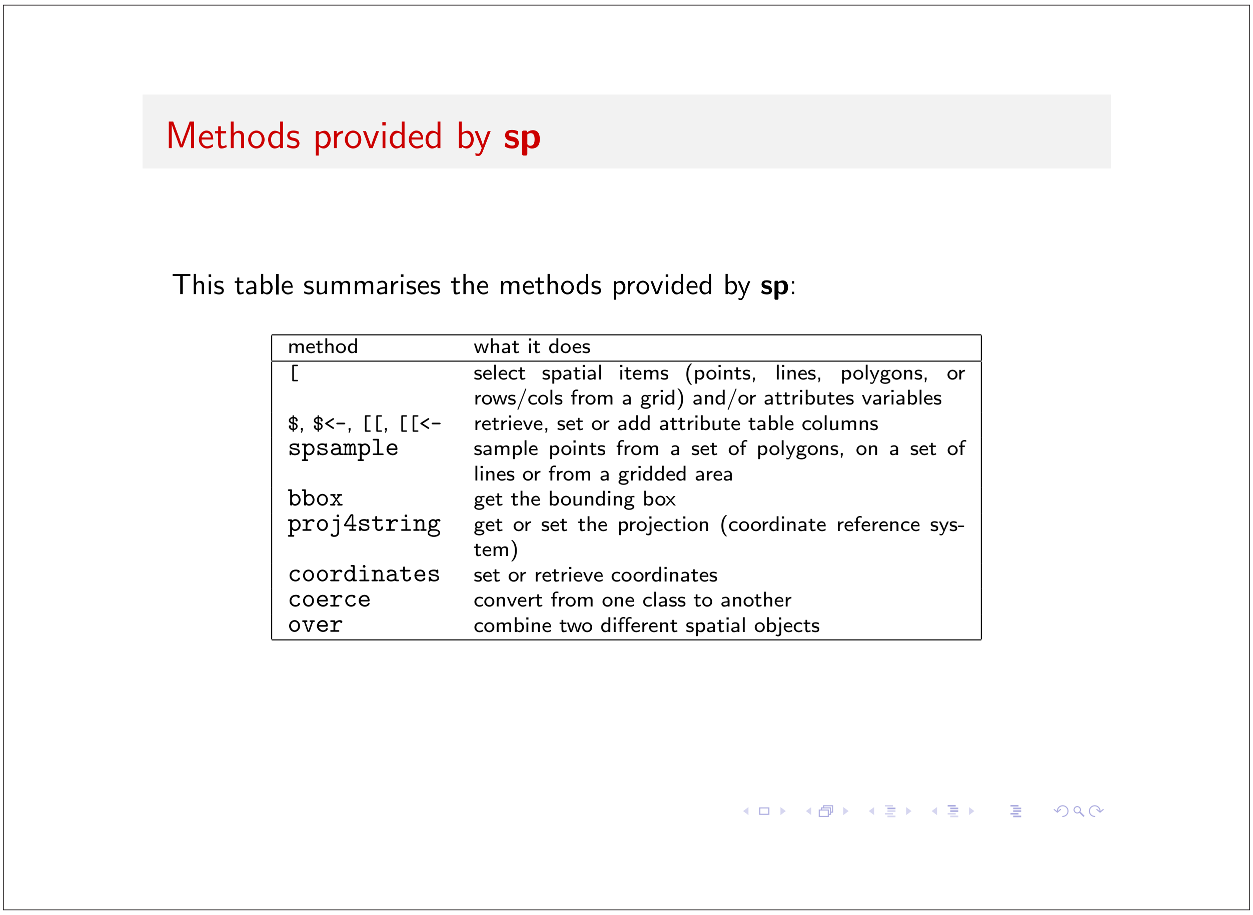
data.frame



Spatial classes provided by **sp**

This table summarises the classes provided by **sp**, and shows how they build up to the objects of most practical use, the `Spatial*DataFrame` family objects:

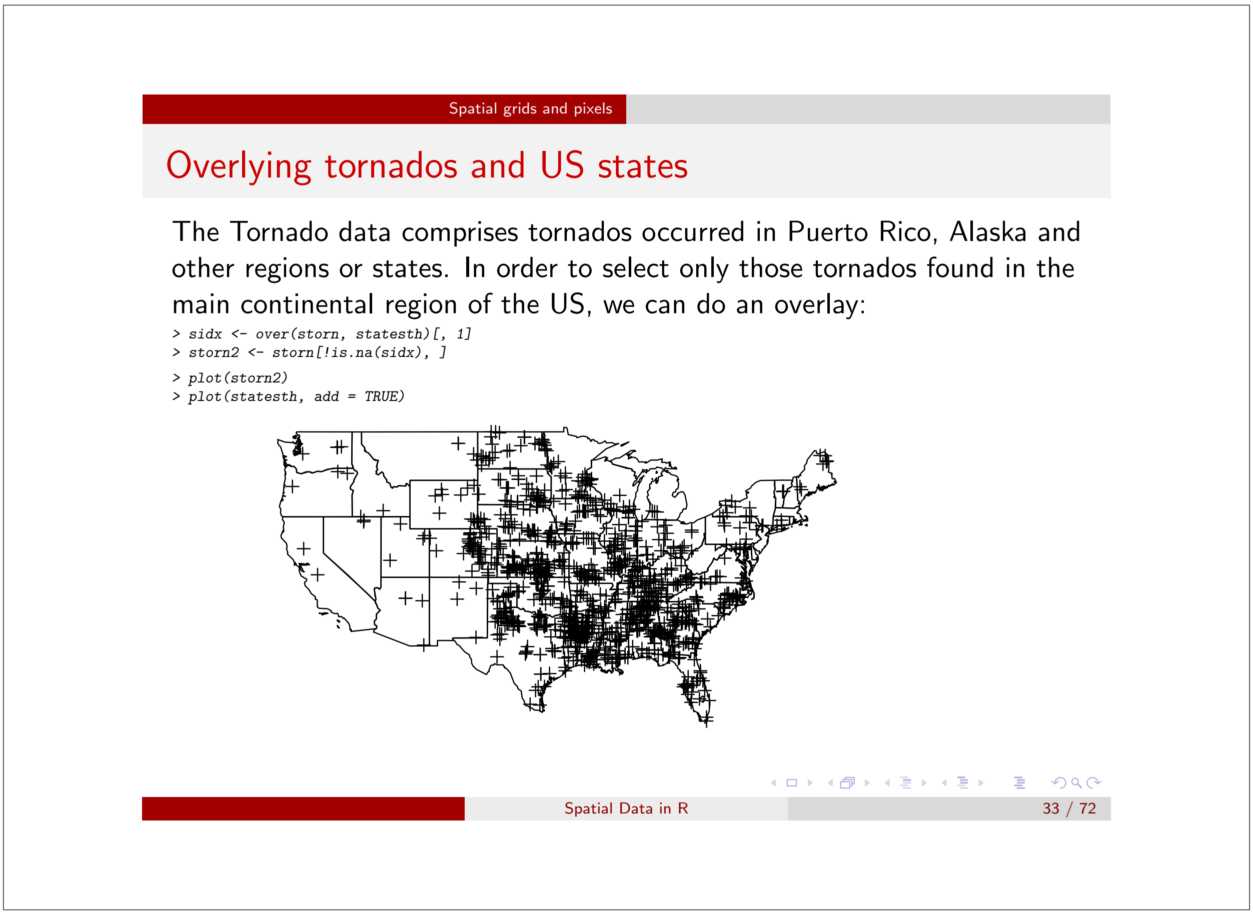
data type	class	attributes	extends
points	`SpatialPoints`	none	`Spatial`
points	`SpatialPointsDataFrame`	`data.frame`	`SpatialPoints`
pixels	`SpatialPixels`	none	`SpatialPoints`
pixels	`SpatialPixelsDataFrame`	`data.frame`	`SpatialPixels` `SpatialPointsDataFrame`
full grid	`SpatialGrid`	none	`SpatialPixels`
full grid	`SpatialGridDataFrame`	`data.frame`	`SpatialGrid`
line	`Line`	none	
lines	`Lines`	none	`Line` list
lines	`SpatialLines`	none	`Spatial`, `Lines` list
lines	`SpatialLinesDataFrame`	`data.frame`	`SpatialLines`
polygon	`Polygon`	none	`Line`
polygons	`Polygons`	none	`Polygon` list
polygons	`SpatialPolygons`	none	`Spatial`, `Polygons` list
polygons	`SpatialPolygonsDataFrame`	`data.frame`	`SpatialPolygons`



Methods provided by **sp**

This table summarises the methods provided by **sp**:

method	what it does
[	select spatial items (points, lines, polygons, or rows/cols from a grid) and/or attributes variables
$, $<-, [[, [[<-	retrieve, set or add attribute table columns
`spsample`	sample points from a set of polygons, on a set of lines or from a gridded area
`bbox`	get the bounding box
`proj4string`	get or set the projection (coordinate reference system)
`coordinates`	set or retrieve coordinates
`coerce`	convert from one class to another
`over`	combine two different spatial objects

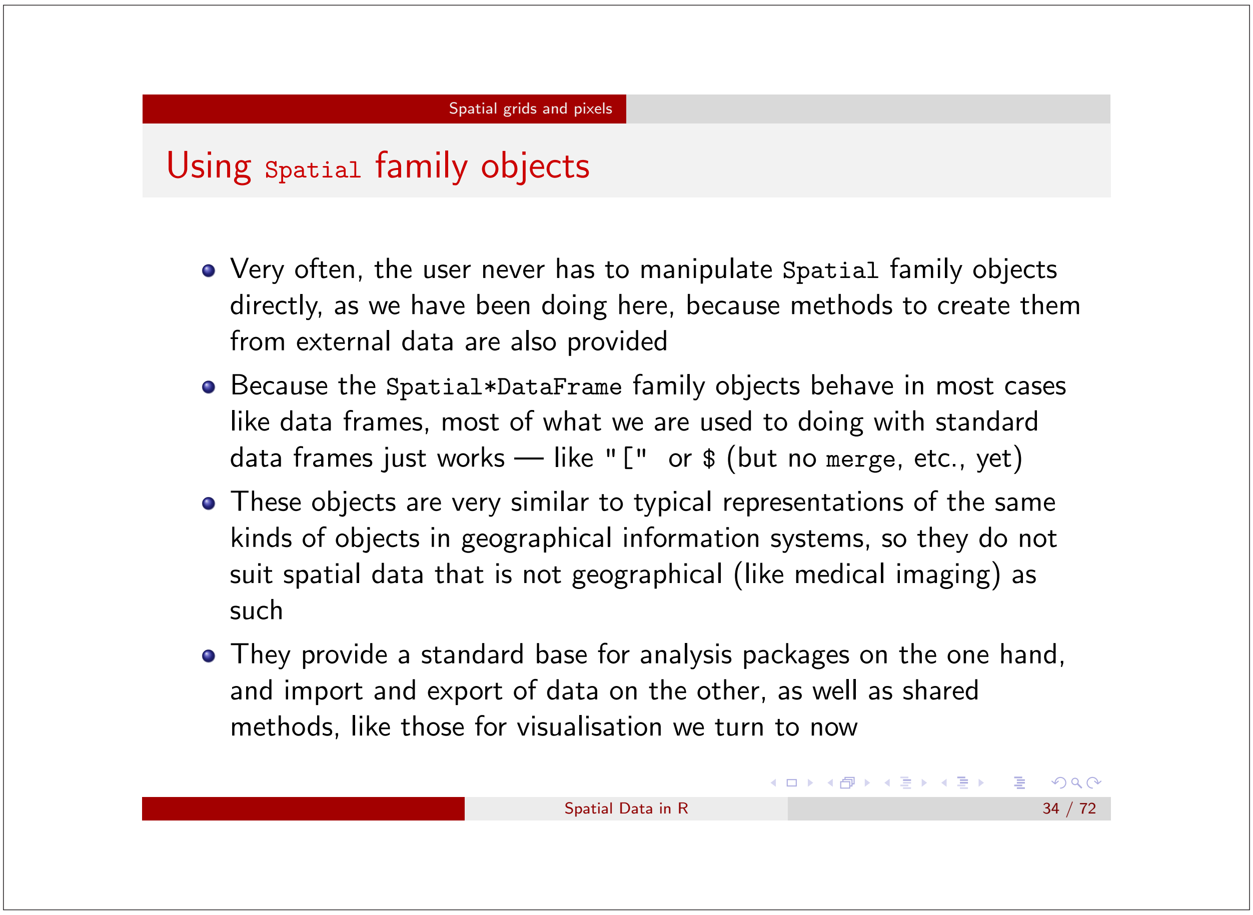


Overlying tornados and US states

The Tornado data comprises tornados occurred in Puerto Rico, Alaska and other regions or states. In order to select only those tornados found in the main continental region of the US, we can do an overlay:

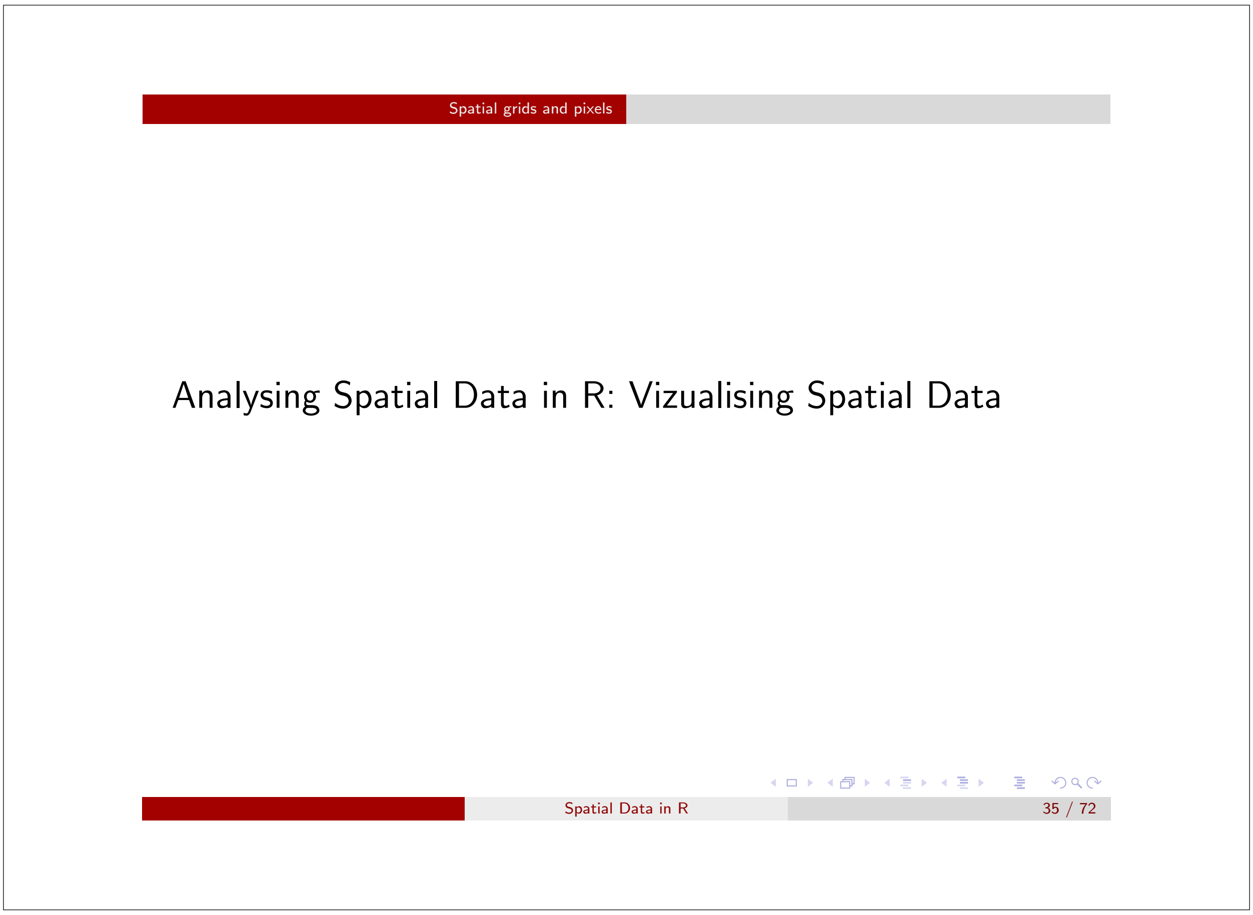
```
> sidx <- over(storn, statesth)[, 1]
> storn2 <- storn[!is.na(sidx), ]
```

```
> plot(storn2)
> plot(statesth, add = TRUE)
```



Using `Spatial` family objects

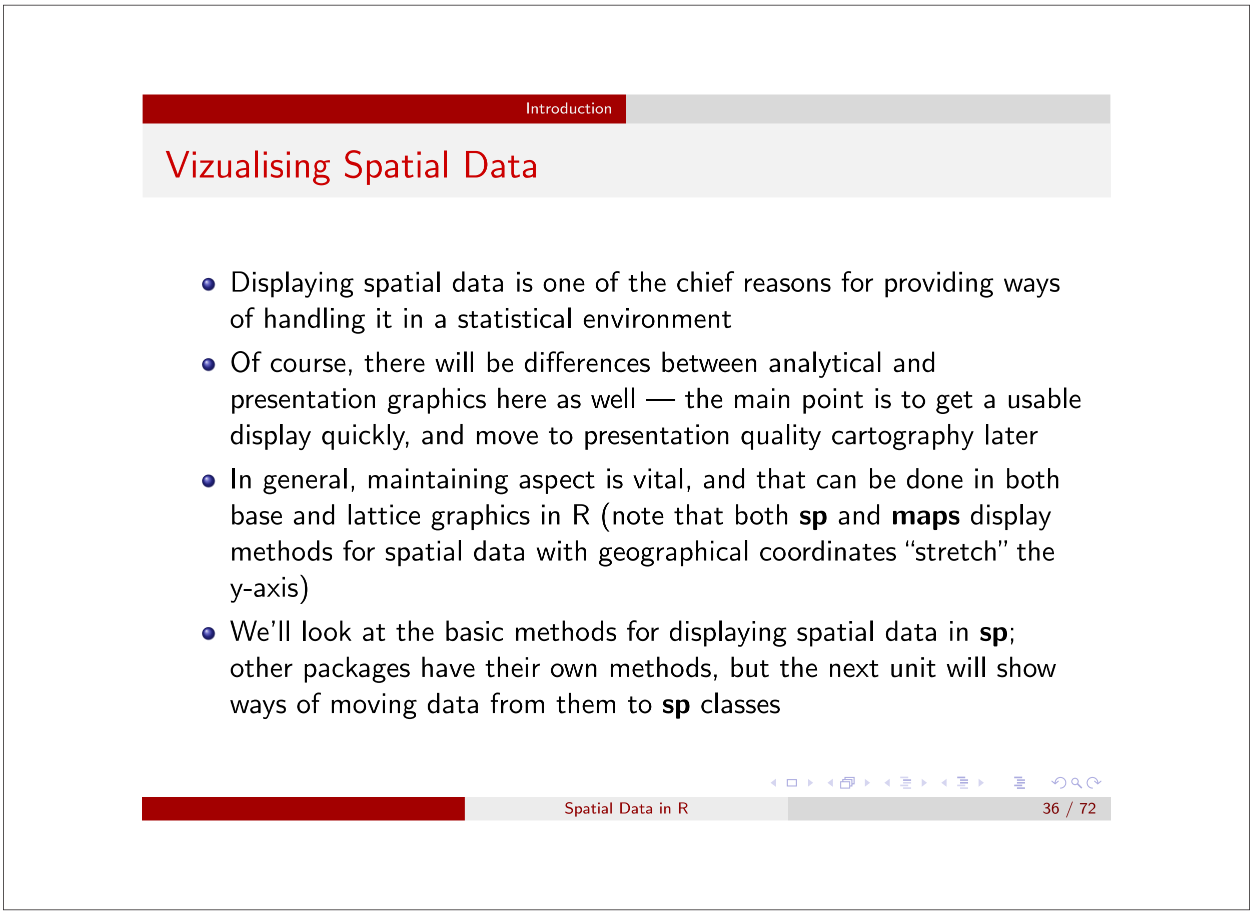
- Very often, the user never has to manipulate `Spatial` family objects directly, as we have been doing here, because methods to create them from external data are also provided
- Because the `Spatial*DataFrame` family objects behave in most cases like data frames, most of what we are used to doing with standard data frames just works — like `"["` or `$` (but no `merge`, etc., yet)
- These objects are very similar to typical representations of the same kinds of objects in geographical information systems, so they do not suit spatial data that is not geographical (like medical imaging) as such
- They provide a standard base for analysis packages on the one hand, and import and export of data on the other, as well as shared methods, like those for visualisation we turn to now



Spatial grids and pixels

Analysing Spatial Data in R: Vizualising Spatial Data

Spatial Data in R 35 / 72

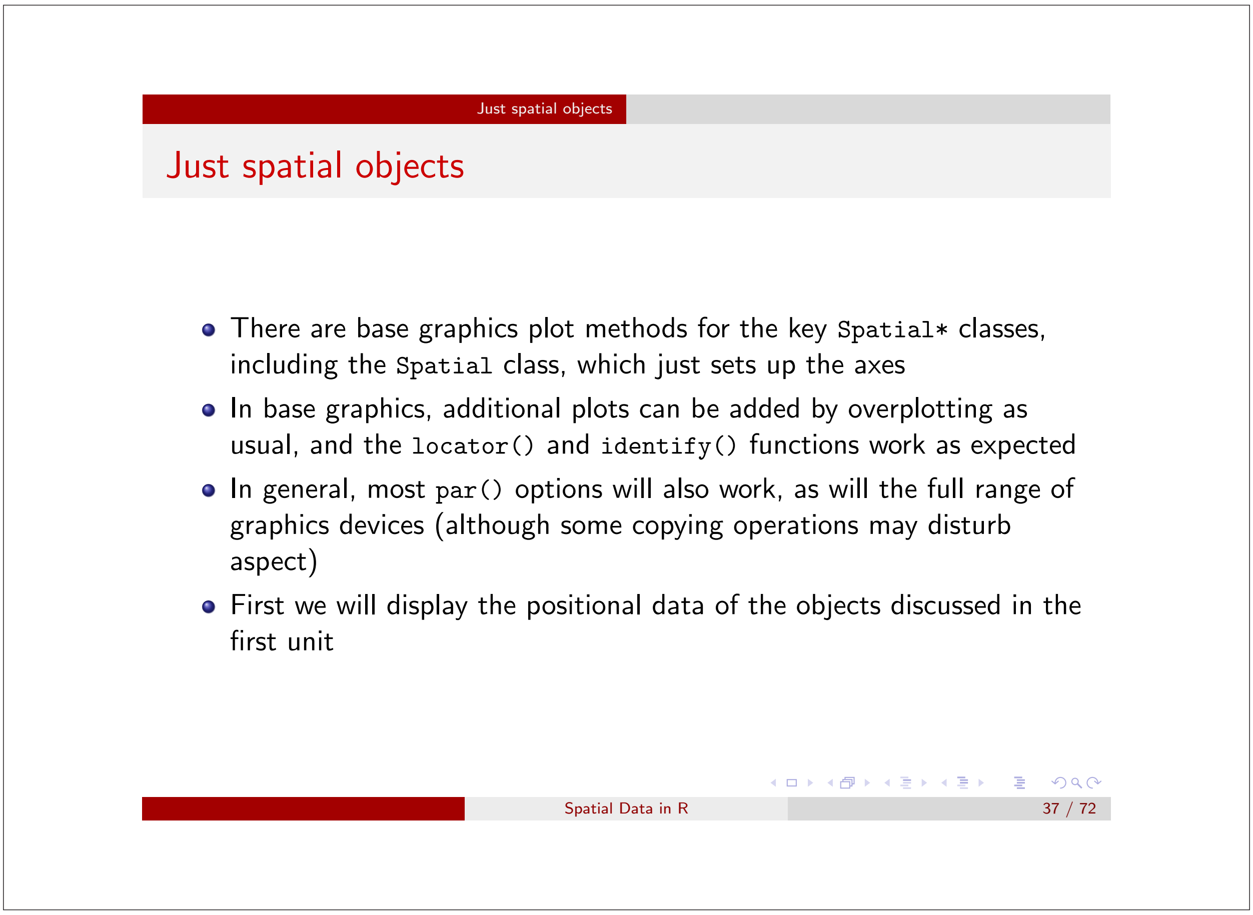


Introduction

Vizualising Spatial Data

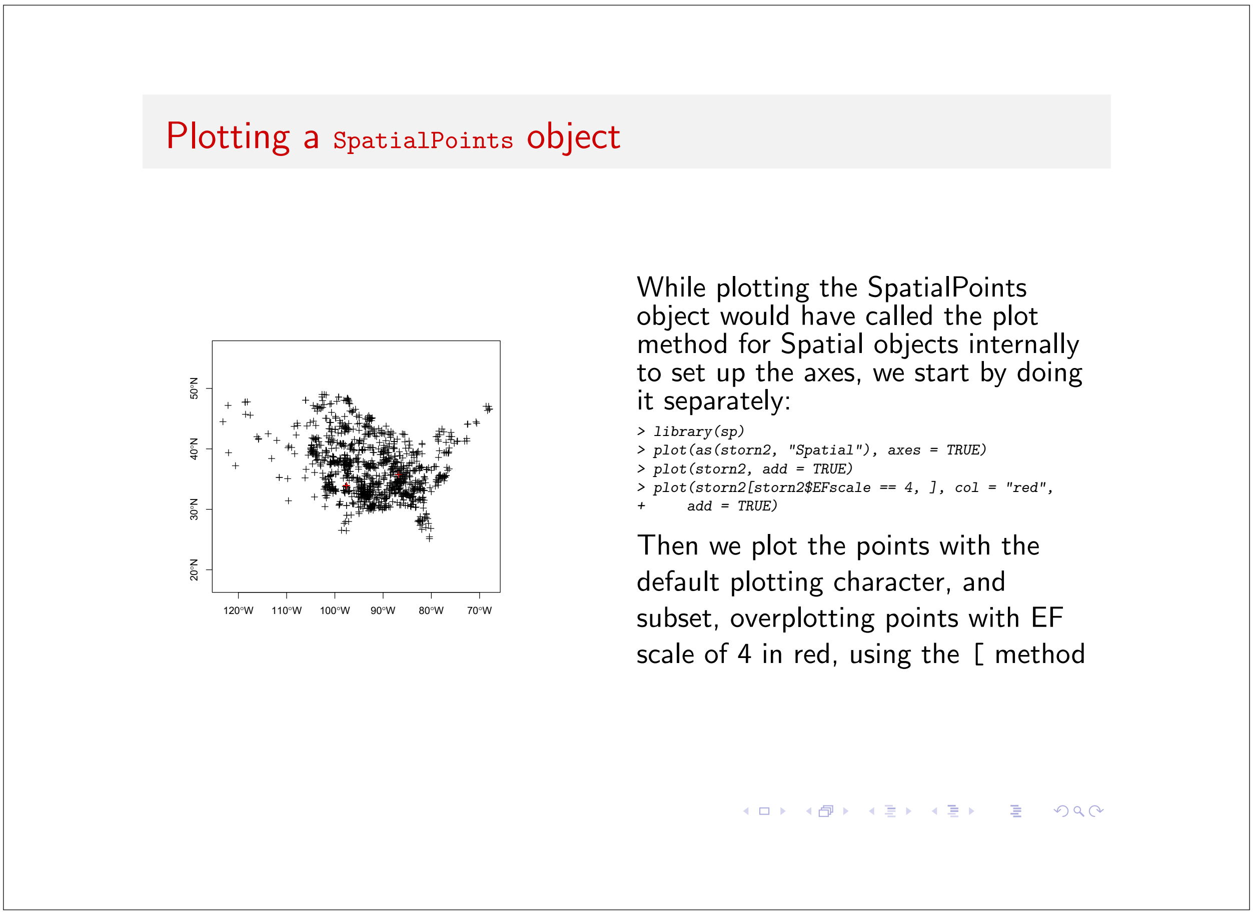
- Displaying spatial data is one of the chief reasons for providing ways of handling it in a statistical environment
- Of course, there will be differences between analytical and presentation graphics here as well — the main point is to get a usable display quickly, and move to presentation quality cartography later
- In general, maintaining aspect is vital, and that can be done in both base and lattice graphics in R (note that both **sp** and **maps** display methods for spatial data with geographical coordinates "stretch" the y-axis)
- We'll look at the basic methods for displaying spatial data in **sp**; other packages have their own methods, but the next unit will show ways of moving data from them to **sp** classes

Spatial Data in R 36 / 72



Just spatial objects

- There are base graphics plot methods for the key `Spatial*` classes, including the `Spatial` class, which just sets up the axes
- In base graphics, additional plots can be added by overplotting as usual, and the `locator()` and `identify()` functions work as expected
- In general, most `par()` options will also work, as will the full range of graphics devices (although some copying operations may disturb aspect)
- First we will display the positional data of the objects discussed in the first unit

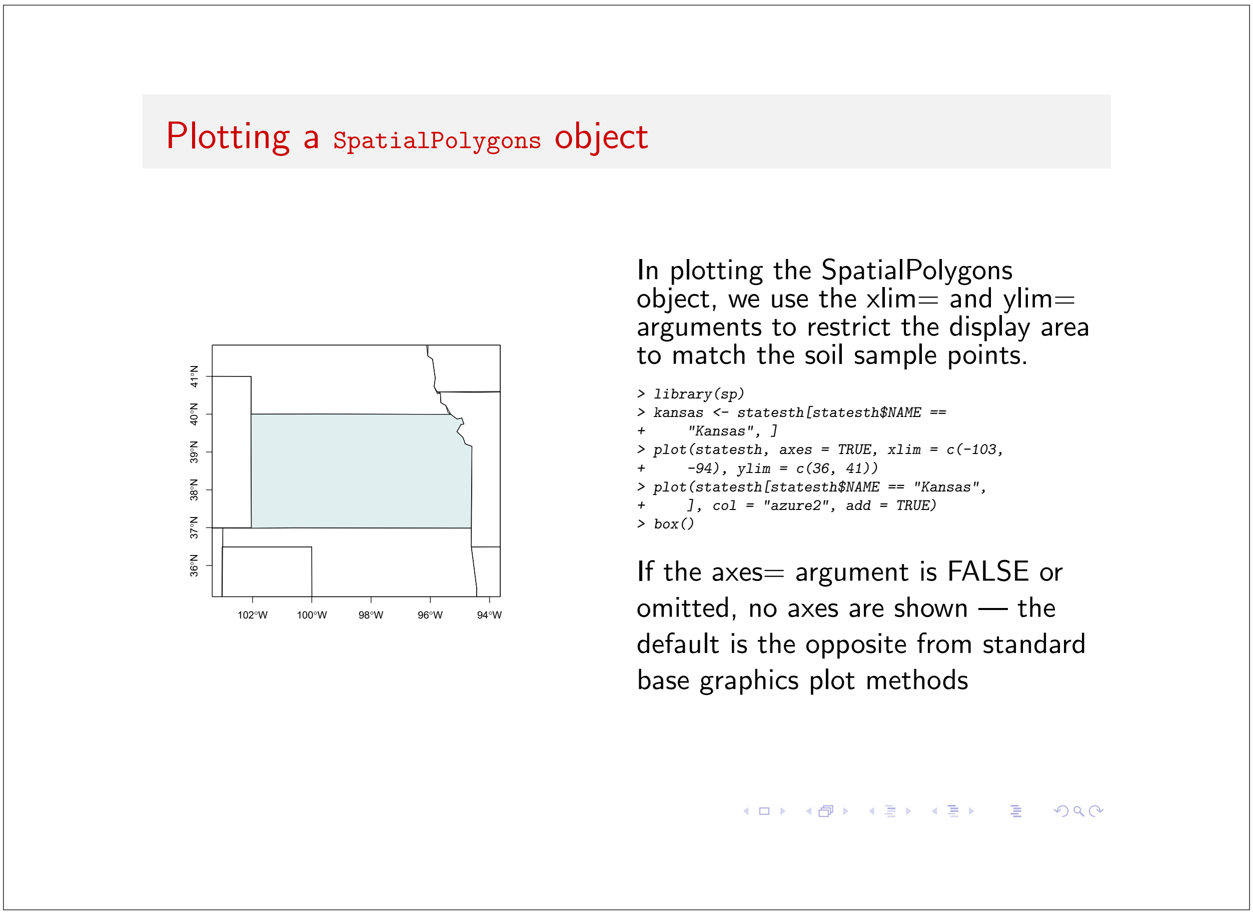


Plotting a `SpatialPoints` object

While plotting the SpatialPoints object would have called the plot method for Spatial objects internally to set up the axes, we start by doing it separately:

```
> library(sp)
> plot(as(storn2, "Spatial"), axes = TRUE)
> plot(storn2, add = TRUE)
> plot(storn2[storn2$EFscale == 4, ], col = "red",
+     add = TRUE)
```

Then we plot the points with the default plotting character, and subset, overplotting points with EF scale of 4 in red, using the [method

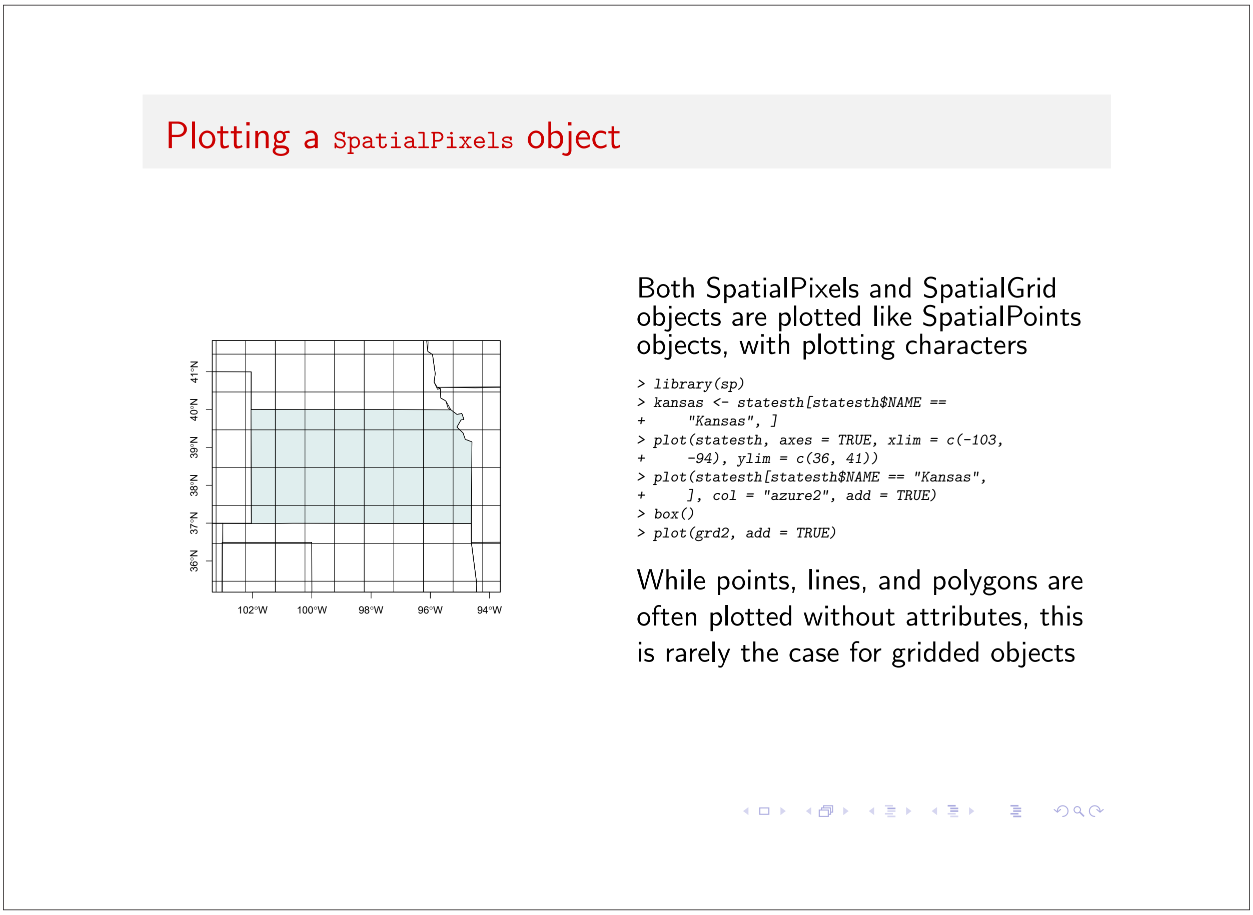


Plotting a `SpatialPolygons` object

In plotting the SpatialPolygons object, we use the xlim= and ylim= arguments to restrict the display area to match the soil sample points.

```
> library(sp)
> kansas <- statesth[statesth$NAME ==
+     "Kansas", ]
> plot(statesth, axes = TRUE, xlim = c(-103,
+     -94), ylim = c(36, 41))
> plot(statesth[statesth$NAME == "Kansas",
+     ], col = "azure2", add = TRUE)
> box()
```

If the axes= argument is FALSE or omitted, no axes are shown — the default is the opposite from standard base graphics plot methods

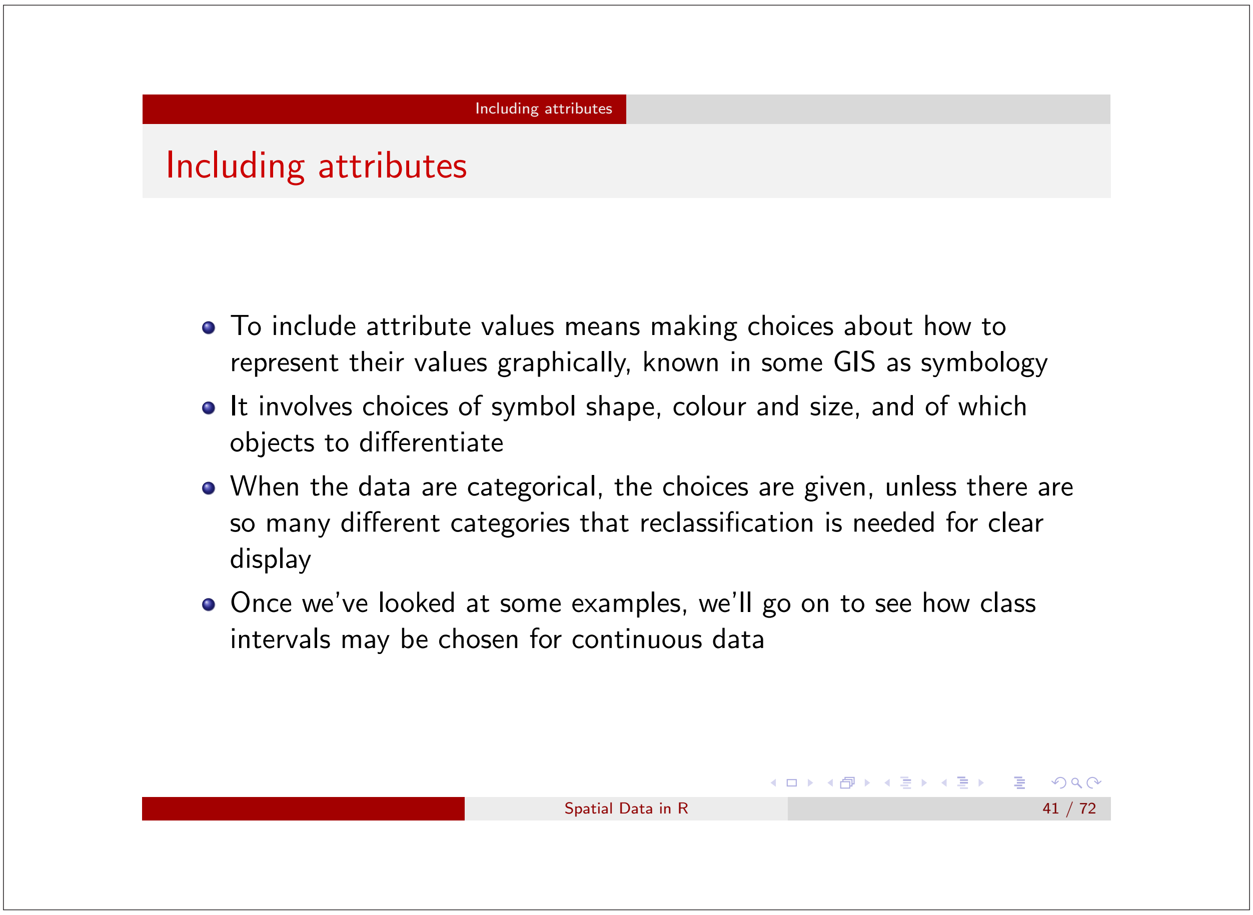


Plotting a `SpatialPixels` object

Both SpatialPixels and SpatialGrid objects are plotted like SpatialPoints objects, with plotting characters

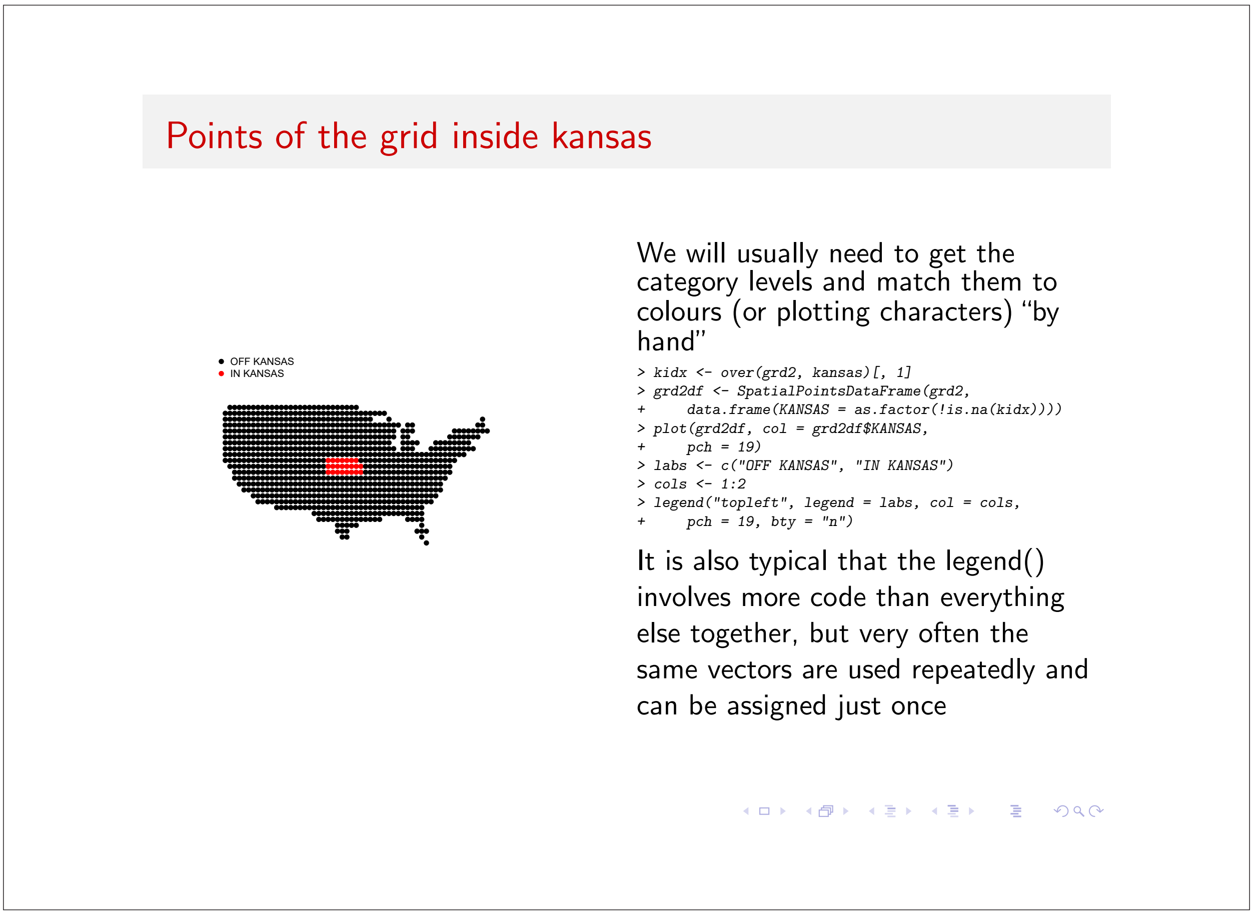
```
> library(sp)
> kansas <- statesth[statesth$NAME ==
+     "Kansas", ]
> plot(statesth, axes = TRUE, xlim = c(-103,
+     -94), ylim = c(36, 41))
> plot(statesth[statesth$NAME == "Kansas",
+     ], col = "azure2", add = TRUE)
> box()
> plot(grd2, add = TRUE)
```

While points, lines, and polygons are often plotted without attributes, this is rarely the case for gridded objects



Including attributes

- To include attribute values means making choices about how to represent their values graphically, known in some GIS as symbology
- It involves choices of symbol shape, colour and size, and of which objects to differentiate
- When the data are categorical, the choices are given, unless there are so many different categories that reclassification is needed for clear display
- Once we've looked at some examples, we'll go on to see how class intervals may be chosen for continuous data



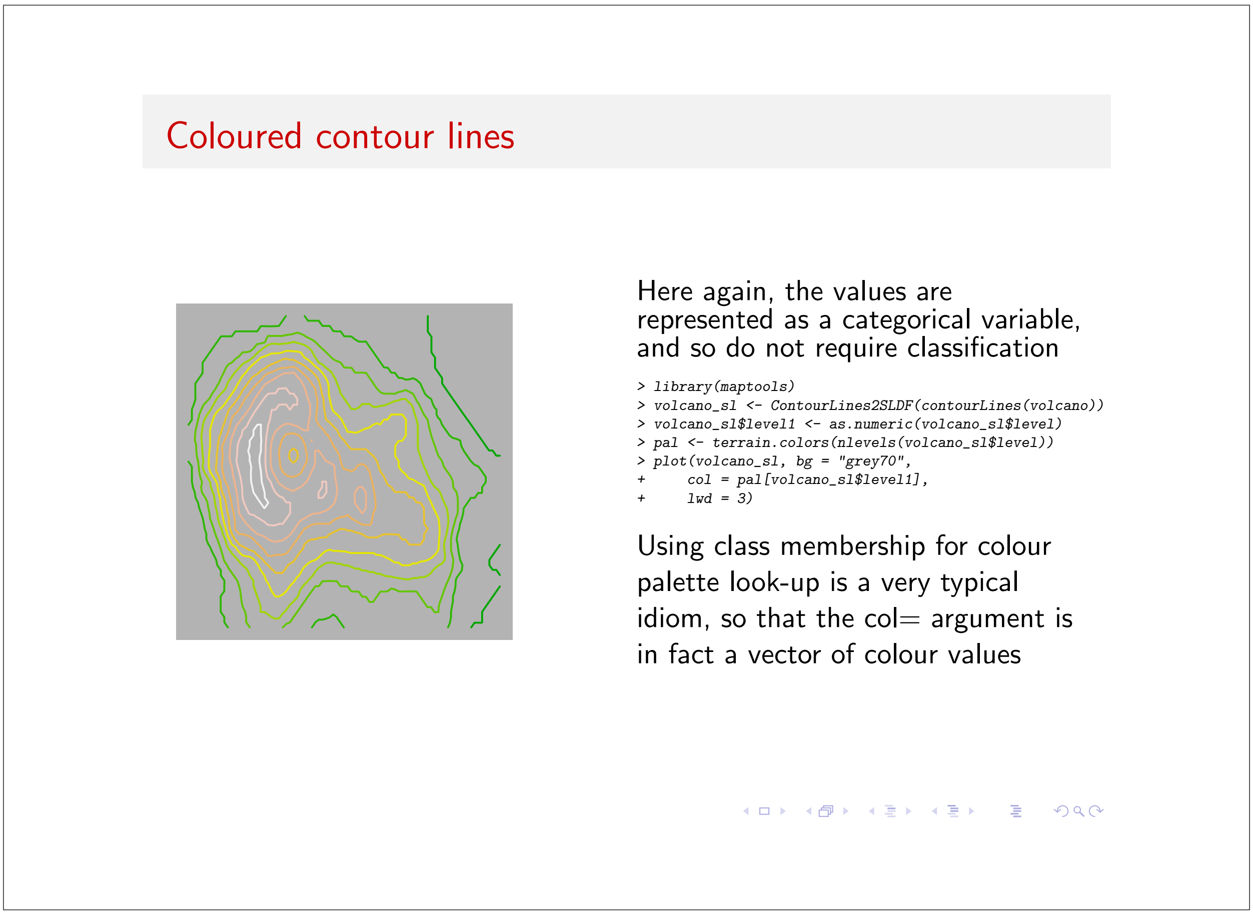
Points of the grid inside kansas

● OFF KANSAS
● IN KANSAS

We will usually need to get the category levels and match them to colours (or plotting characters) "by hand"

```
> kidx <- over(grd2, kansas)[, 1]
> grd2df <- SpatialPointsDataFrame(grd2,
+     data.frame(KANSAS = as.factor(!is.na(kidx))))
> plot(grd2df, col = grd2df$KANSAS,
+     pch = 19)
> labs <- c("OFF KANSAS", "IN KANSAS")
> cols <- 1:2
> legend("topleft", legend = labs, col = cols,
+     pch = 19, bty = "n")
```

It is also typical that the legend() involves more code than everything else together, but very often the same vectors are used repeatedly and can be assigned just once

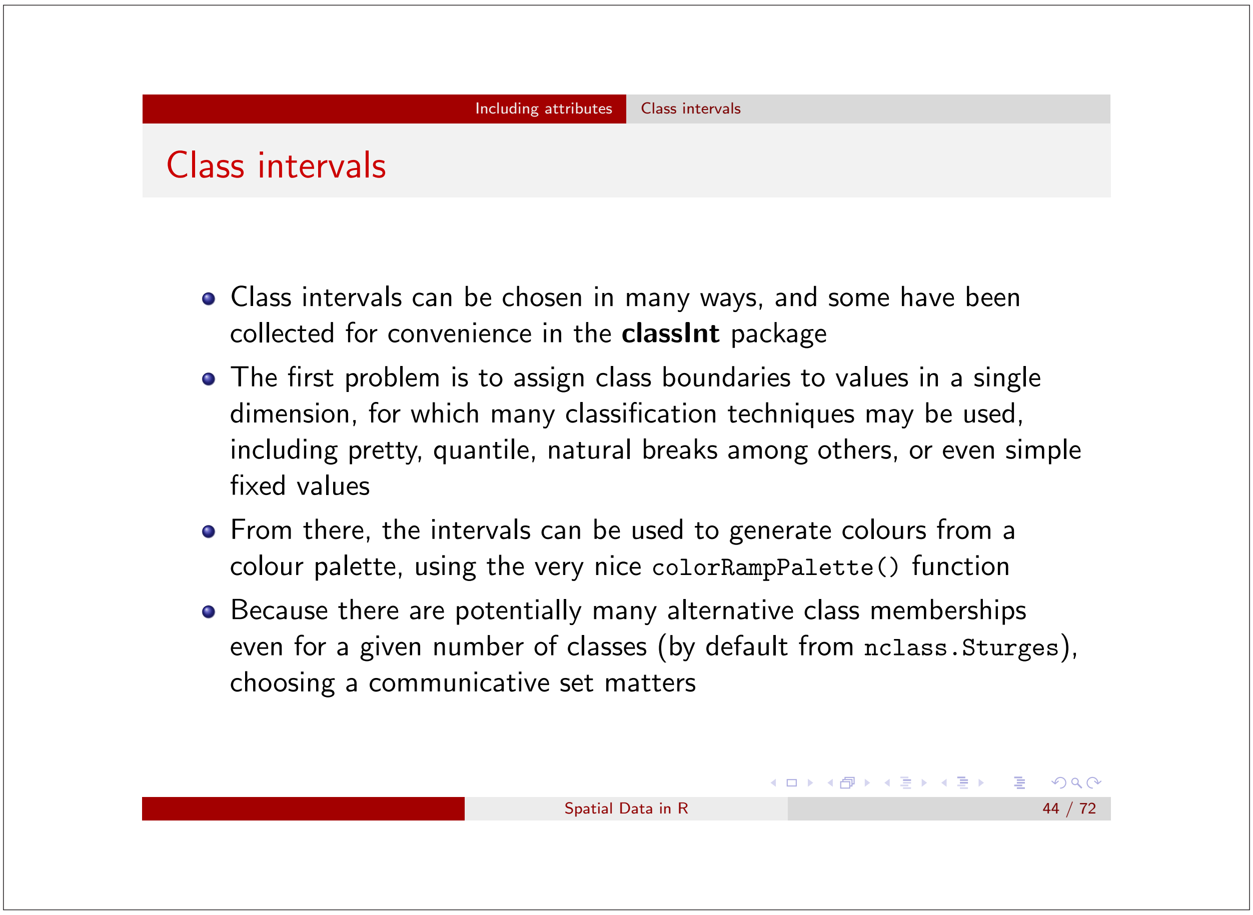


Coloured contour lines

Here again, the values are represented as a categorical variable, and so do not require classification

```
> library(maptools)
> volcano_sl <- ContourLines2SLDF(contourLines(volcano))
> volcano_sl$level1 <- as.numeric(volcano_sl$level)
> pal <- terrain.colors(nlevels(volcano_sl$level))
> plot(volcano_sl, bg = "grey70",
+     col = pal[volcano_sl$level1],
+     lwd = 3)
```

Using class membership for colour palette look-up is a very typical idiom, so that the col= argument is in fact a vector of colour values

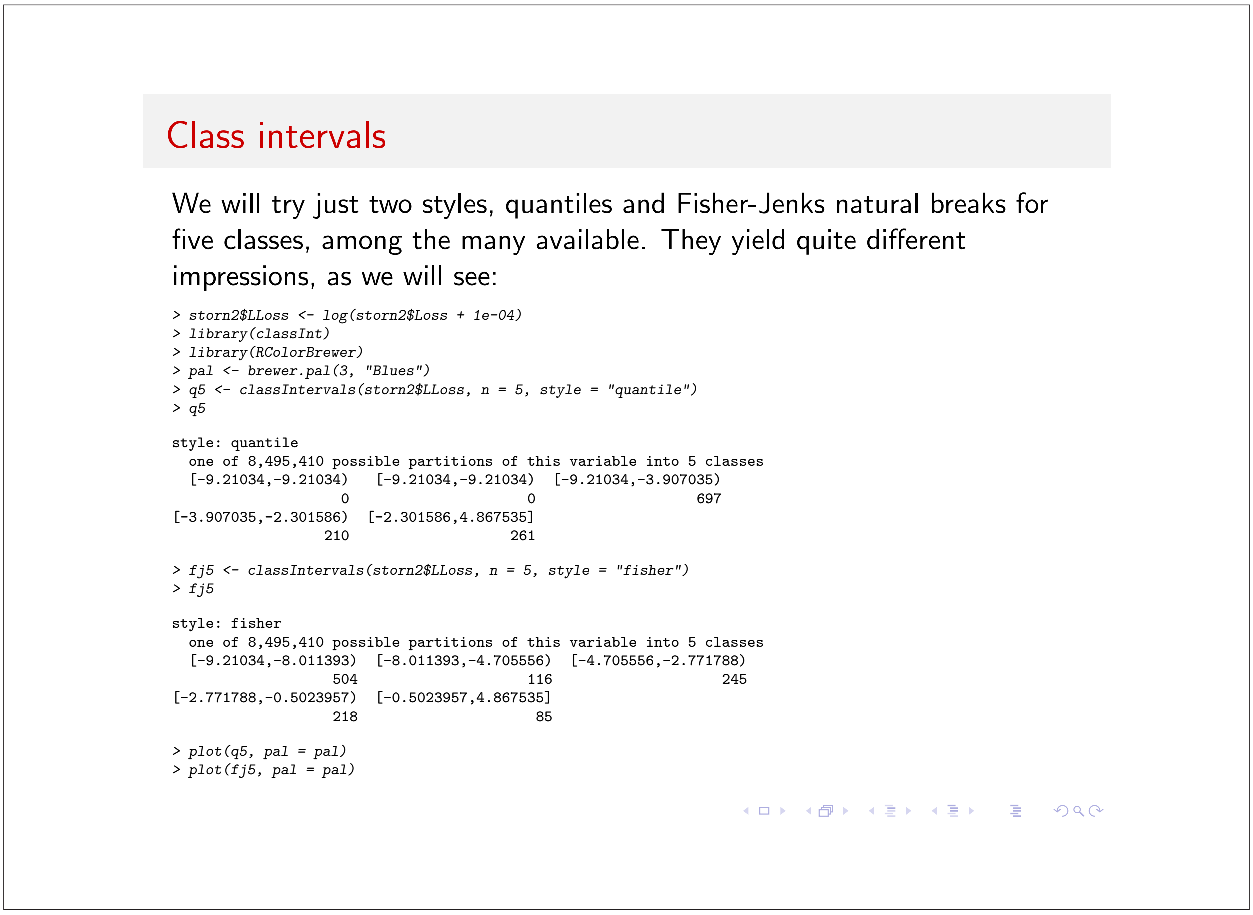


Including attributes Class intervals

Class intervals

- Class intervals can be chosen in many ways, and some have been collected for convenience in the **classInt** package
- The first problem is to assign class boundaries to values in a single dimension, for which many classification techniques may be used, including pretty, quantile, natural breaks among others, or even simple fixed values
- From there, the intervals can be used to generate colours from a colour palette, using the very nice `colorRampPalette()` function
- Because there are potentially many alternative class memberships even for a given number of classes (by default from `nclass.Sturges`), choosing a communicative set matters

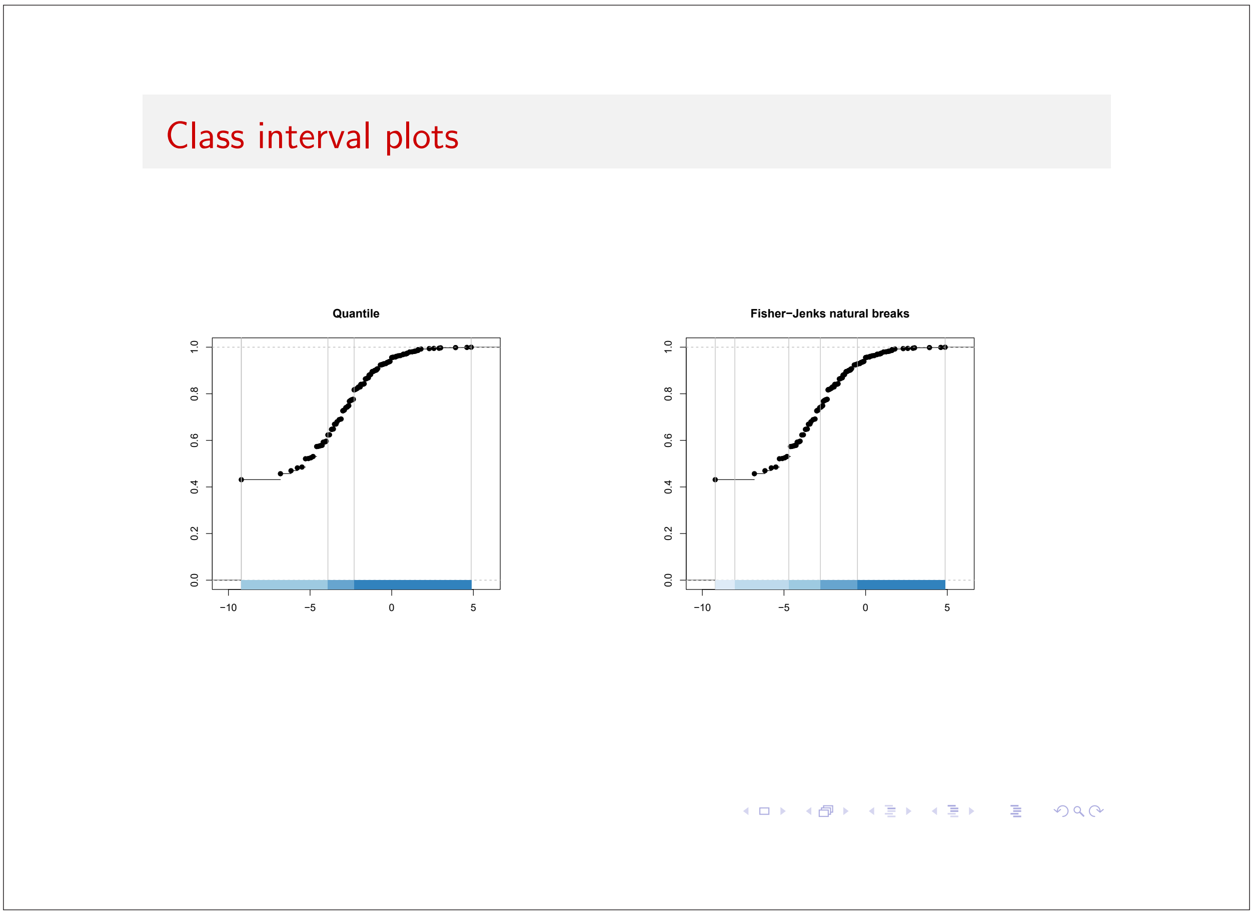
Spatial Data in R 44 / 72



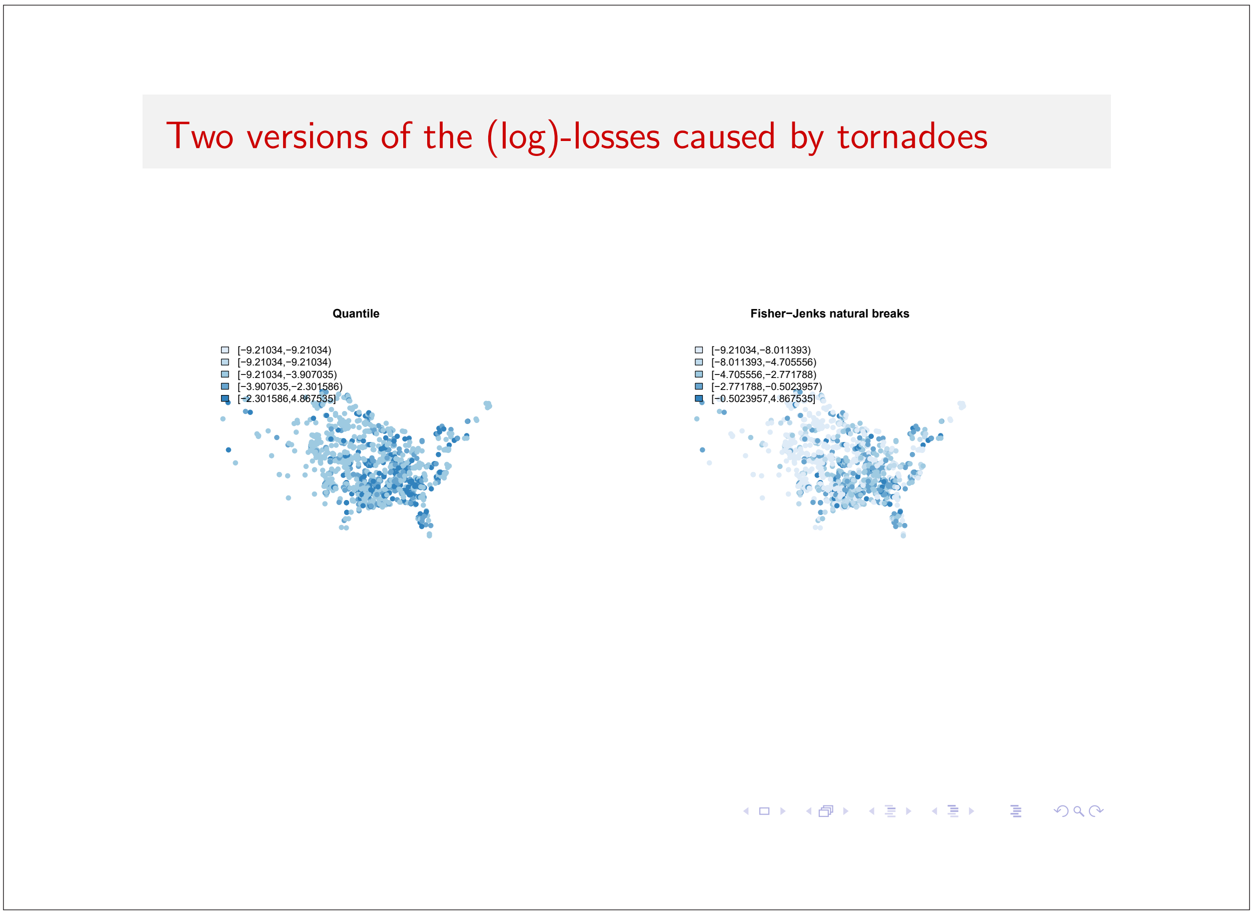
Class intervals

We will try just two styles, quantiles and Fisher-Jenks natural breaks for five classes, among the many available. They yield quite different impressions, as we will see:

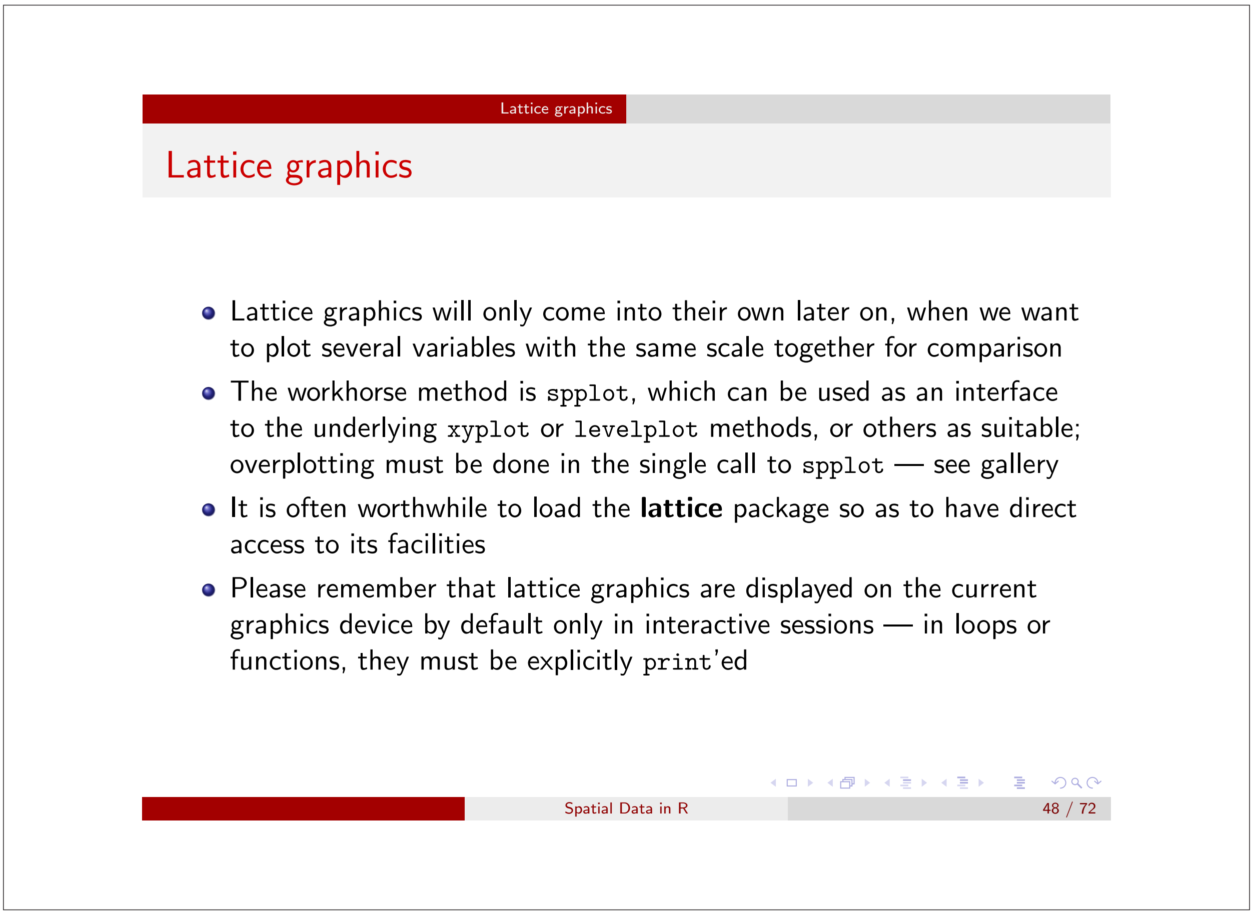
```
> storn2$LLoss <- log(storn2$Loss + 1e-04)
> library(classInt)
> library(RColorBrewer)
> pal <- brewer.pal(3, "Blues")
> q5 <- classIntervals(storn2$LLoss, n = 5, style = "quantile")
> q5

style: quantile
  one of 8,495,410 possible partitions of this variable into 5 classes
  [-9.21034,-9.21034)   [-9.21034,-9.21034)  [-9.21034,-3.907035)
                    0                     0                   697
[-3.907035,-2.301586)  [-2.301586,4.867535]
                  210                   261

> fj5 <- classIntervals(storn2$LLoss, n = 5, style = "fisher")
> fj5

style: fisher
  one of 8,495,410 possible partitions of this variable into 5 classes
  [-9.21034,-8.011393)  [-8.011393,-4.705556)  [-4.705556,-2.771788)
                    504                    116                    245
[-2.771788,-0.5023957)  [-0.5023957,4.867535]
                    218                     85

> plot(q5, pal = pal)
> plot(fj5, pal = pal)
```



Class interval plots



Two versions of the (log)-losses caused by tornadoes



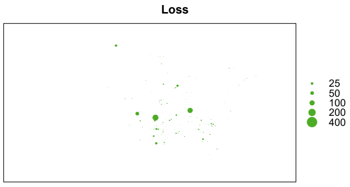
Lattice graphics

Lattice graphics

- Lattice graphics will only come into their own later on, when we want to plot several variables with the same scale together for comparison
- The workhorse method is `spplot`, which can be used as an interface to the underlying `xyplot` or `levelplot` methods, or others as suitable; overplotting must be done in the single call to `spplot` — see gallery
- It is often worthwhile to load the **lattice** package so as to have direct access to its facilities
- Please remember that lattice graphics are displayed on the current graphics device by default only in interactive sessions — in loops or functions, they must be explicitly `print`'ed

Spatial Data in R 48 / 72

Bubble plots

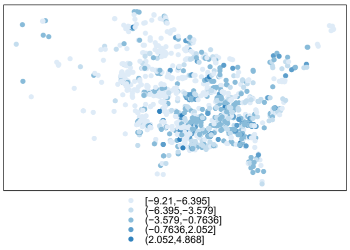


Bubble plots are a convenient way of representing the attribute values by the size of a symbol

```
> library(lattice)
> print(bubble(storn2, "Loss",
+     maxsize = 2, key.entries = 25 *
+         2^(0:4)))
```

As with all lattice graphics objects, the function can return an object from which symbol sizes can be recovered

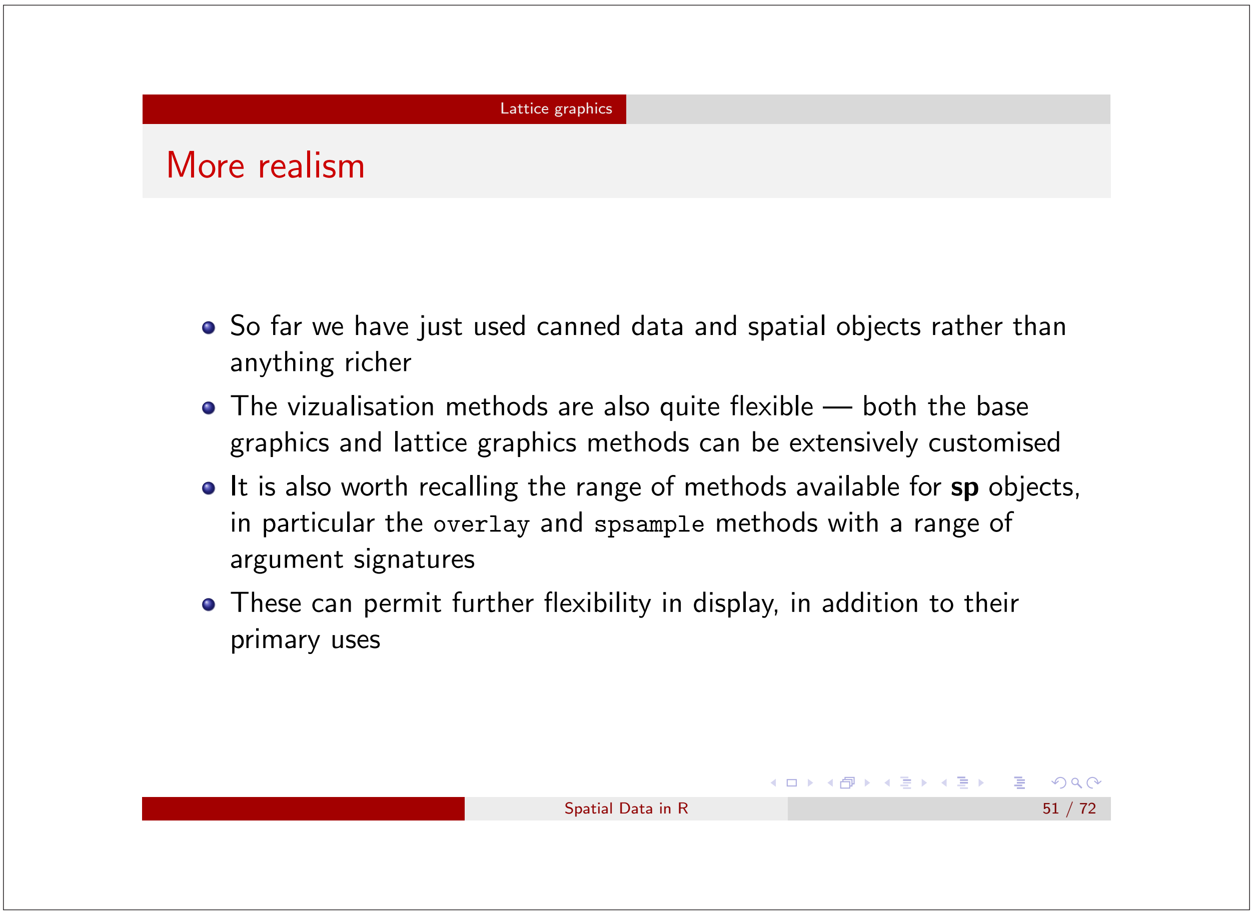
Level plots



The use of lattice plotting methods yields easy legend generation, which is another attraction

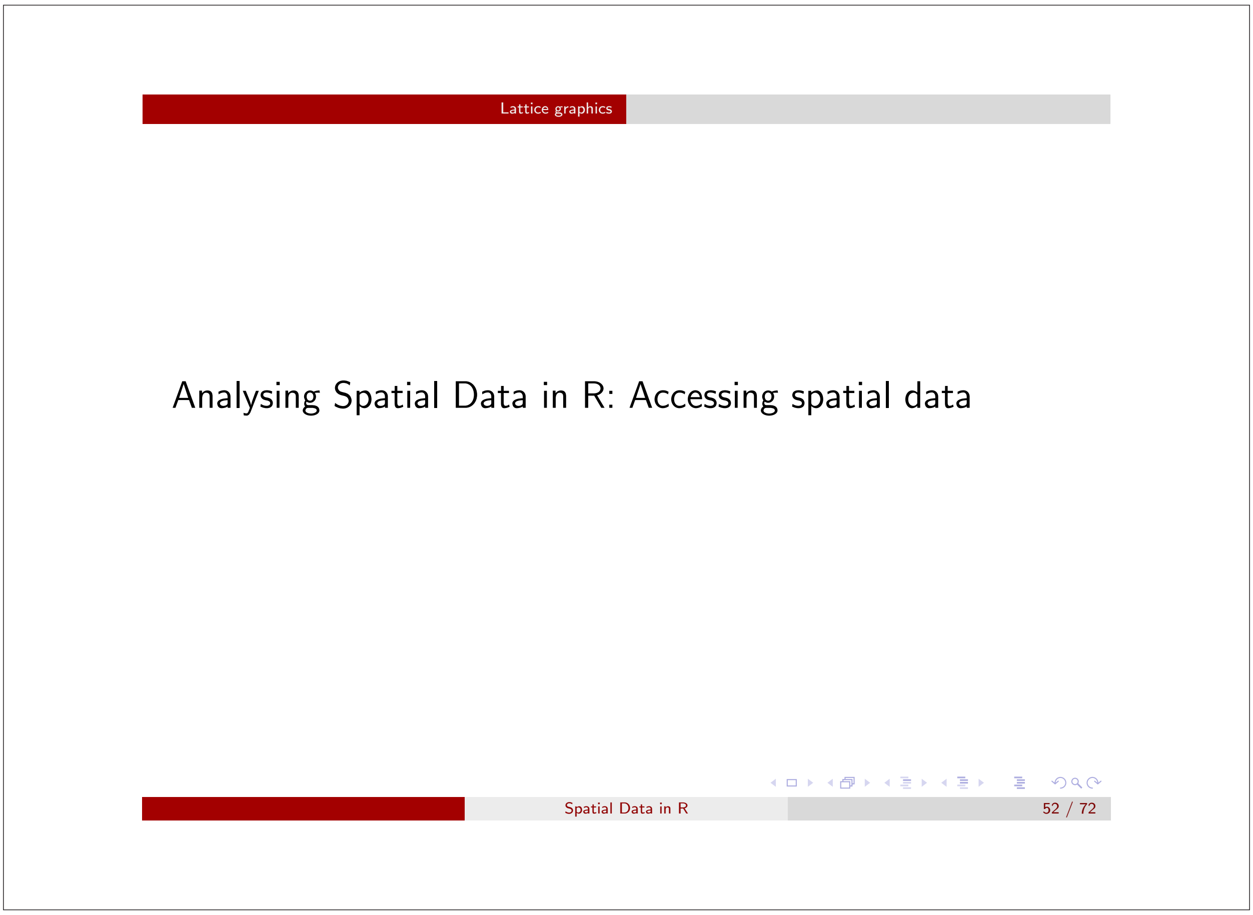
```
> bpal <- colorRampPalette(pal)(6)
> print(spplot(storn2, "LLoss",
+     col.regions = bpal, cuts = 5))
```

Here we are showing the distances from the river of grid points in the study area; we can also pass in intervals chosen previously

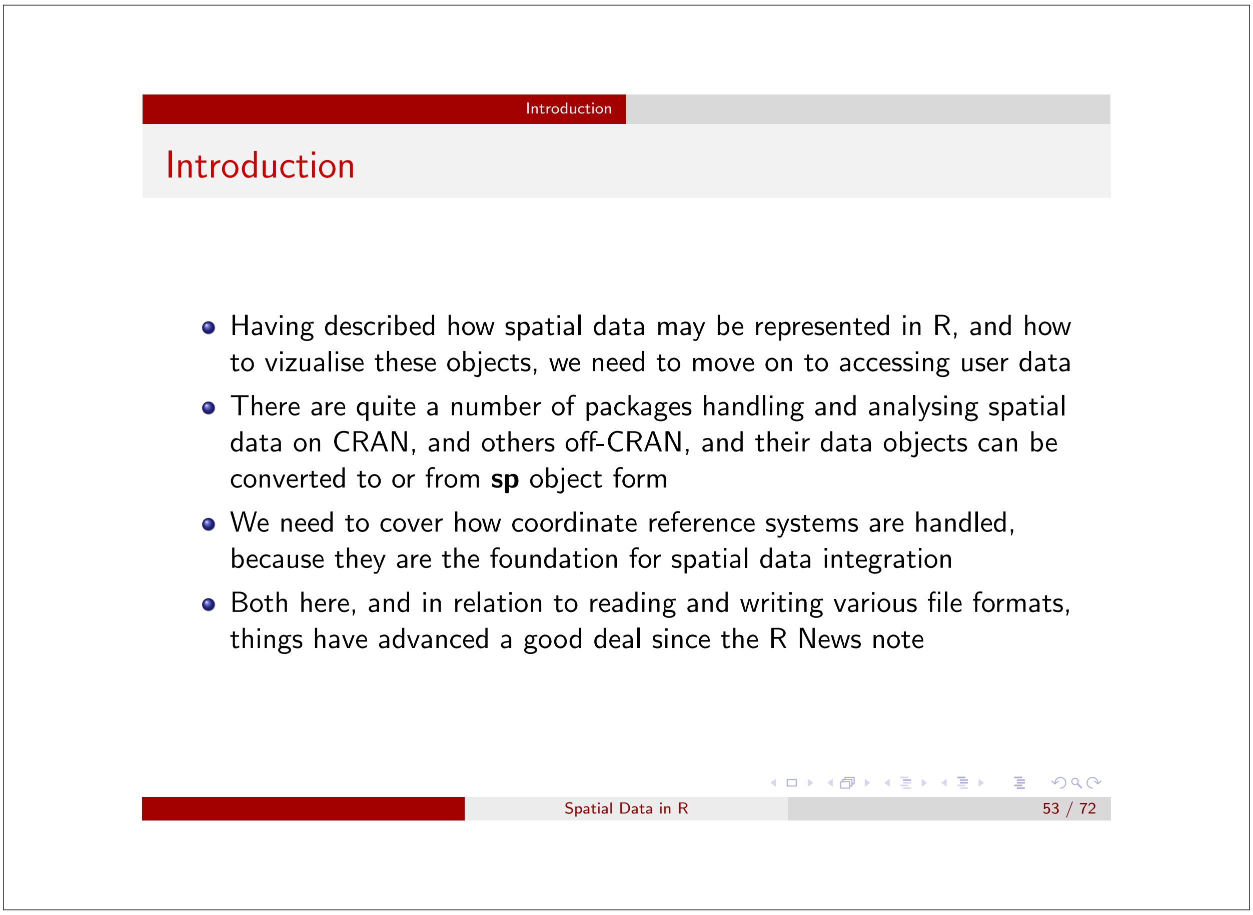


More realism

- So far we have just used canned data and spatial objects rather than anything richer
- The vizualisation methods are also quite flexible — both the base graphics and lattice graphics methods can be extensively customised
- It is also worth recalling the range of methods available for **sp** objects, in particular the `overlay` and `spsample` methods with a range of argument signatures
- These can permit further flexibility in display, in addition to their primary uses

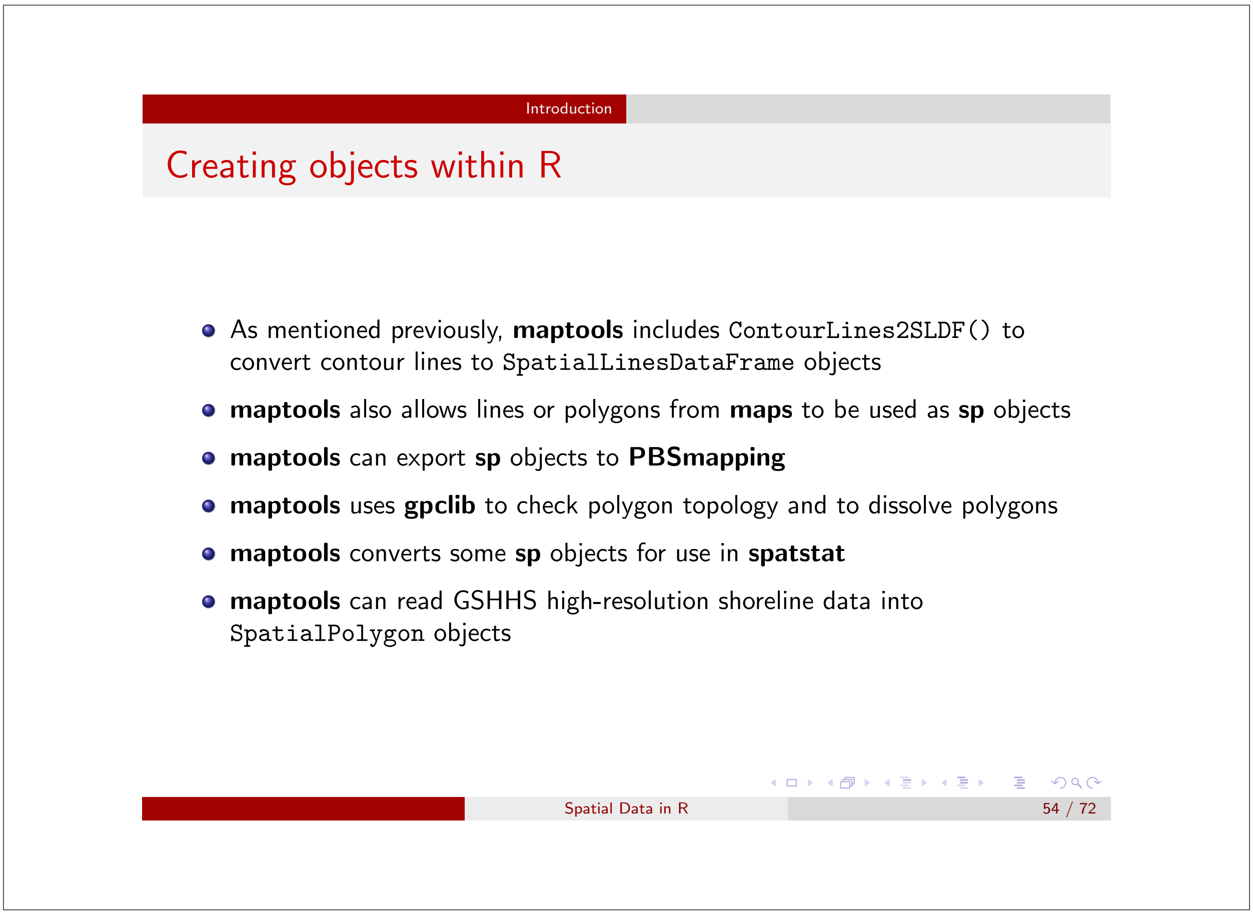


Analysing Spatial Data in R: Accessing spatial data



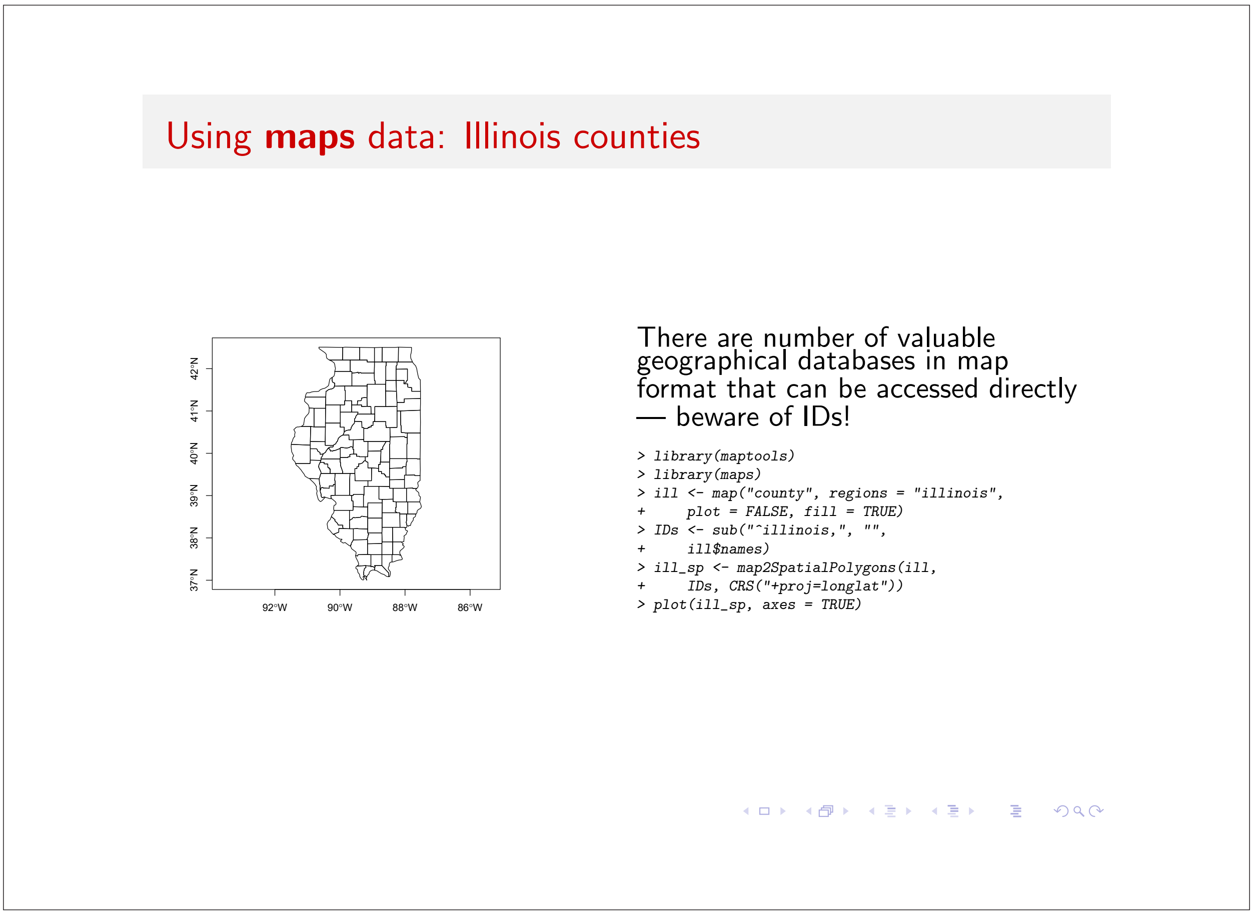
Introduction

- Having described how spatial data may be represented in R, and how to vizualise these objects, we need to move on to accessing user data
- There are quite a number of packages handling and analysing spatial data on CRAN, and others off-CRAN, and their data objects can be converted to or from **sp** object form
- We need to cover how coordinate reference systems are handled, because they are the foundation for spatial data integration
- Both here, and in relation to reading and writing various file formats, things have advanced a good deal since the R News note



Creating objects within R

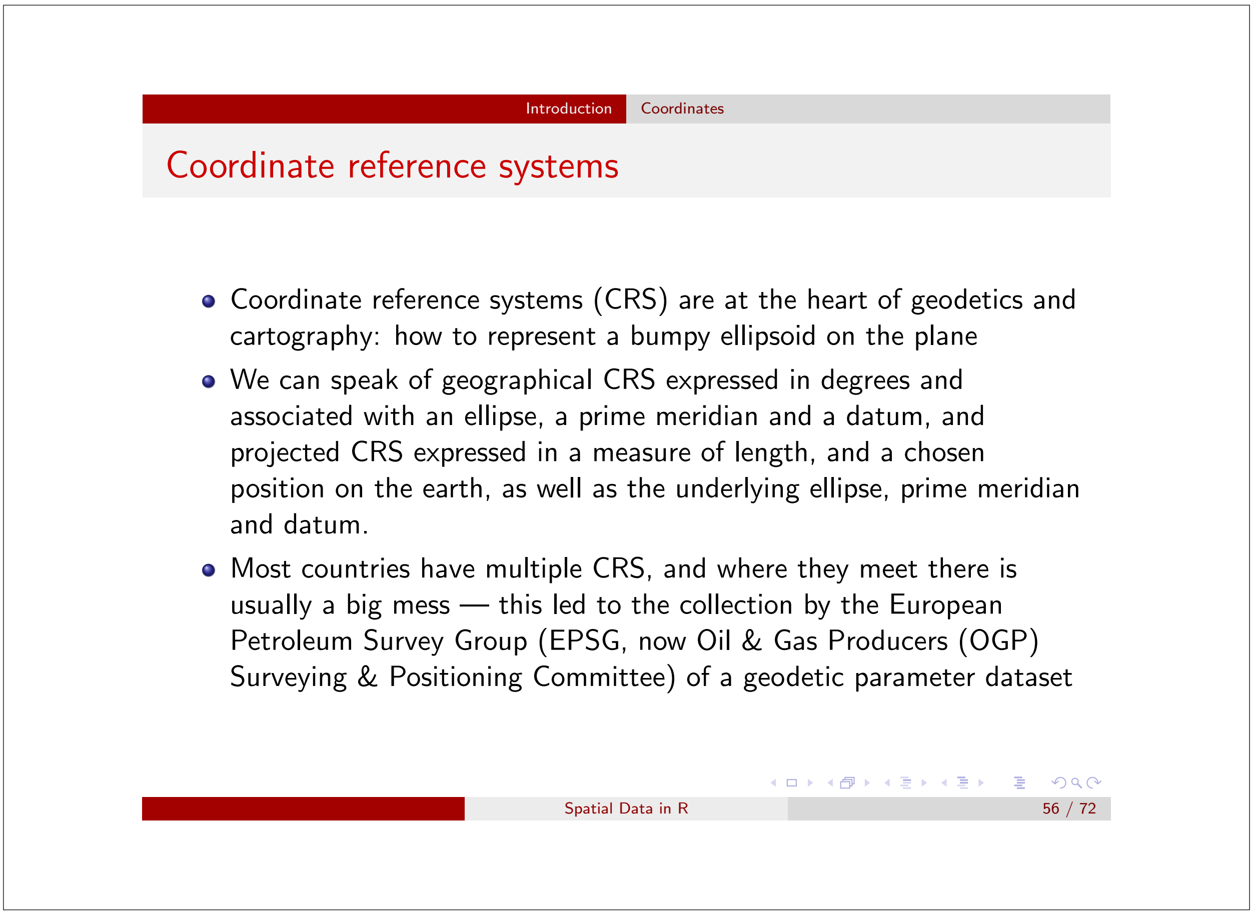
- As mentioned previously, **maptools** includes `ContourLines2SLDF()` to convert contour lines to `SpatialLinesDataFrame` objects
- **maptools** also allows lines or polygons from **maps** to be used as **sp** objects
- **maptools** can export **sp** objects to **PBSmapping**
- **maptools** uses **gpclib** to check polygon topology and to dissolve polygons
- **maptools** converts some **sp** objects for use in **spatstat**
- **maptools** can read GSHHS high-resolution shoreline data into `SpatialPolygon` objects



Using **maps** data: Illinois counties

There are number of valuable geographical databases in map format that can be accessed directly — beware of IDs!

```
> library(maptools)
> library(maps)
> ill <- map("county", regions = "illinois",
+     plot = FALSE, fill = TRUE)
> IDs <- sub("^illinois,", "",
+     ill$names)
> ill_sp <- map2SpatialPolygons(ill,
+     IDs, CRS("+proj=longlat"))
> plot(ill_sp, axes = TRUE)
```

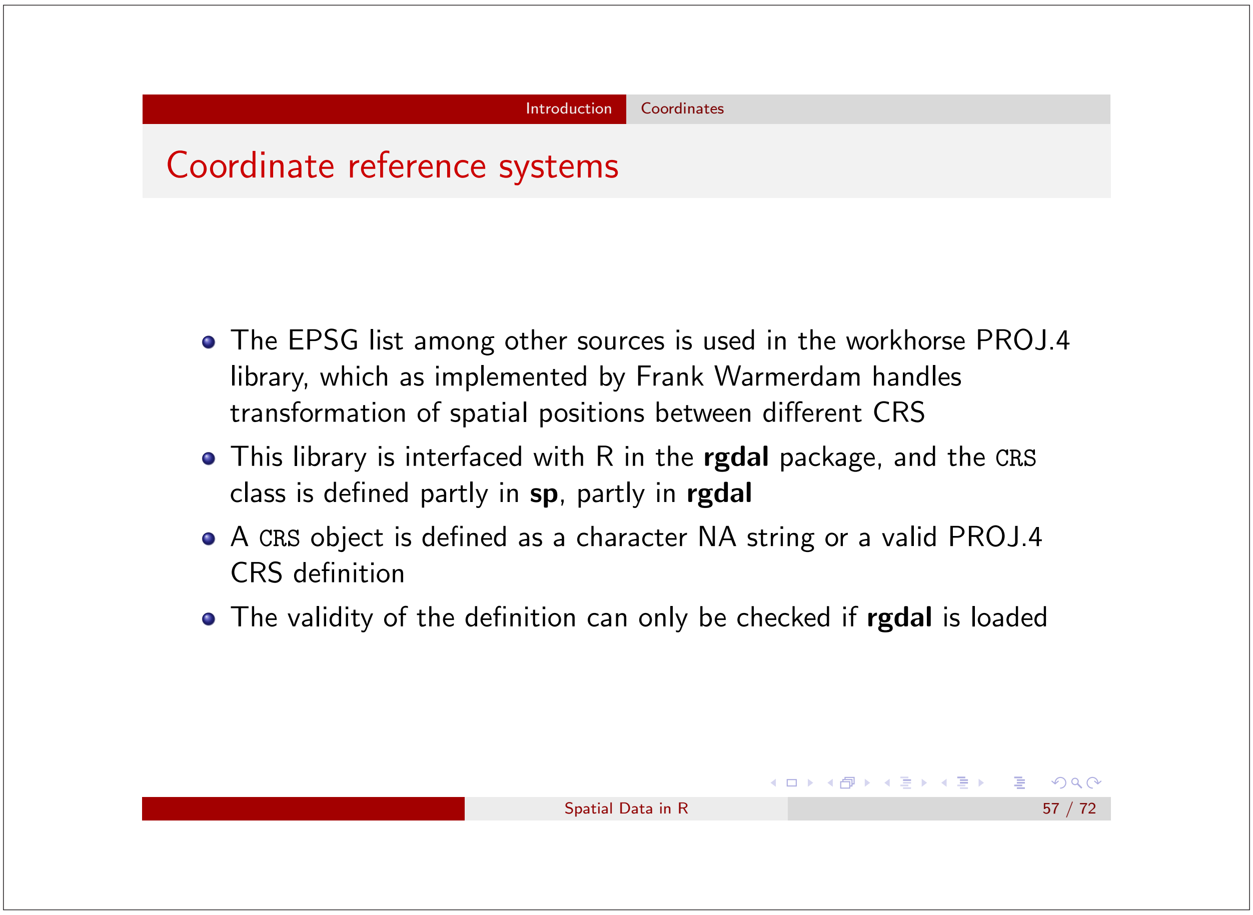


Introduction Coordinates

Coordinate reference systems

- Coordinate reference systems (CRS) are at the heart of geodetics and cartography: how to represent a bumpy ellipsoid on the plane
- We can speak of geographical CRS expressed in degrees and associated with an ellipse, a prime meridian and a datum, and projected CRS expressed in a measure of length, and a chosen position on the earth, as well as the underlying ellipse, prime meridian and datum.
- Most countries have multiple CRS, and where they meet there is usually a big mess — this led to the collection by the European Petroleum Survey Group (EPSG, now Oil & Gas Producers (OGP) Surveying & Positioning Committee) of a geodetic parameter dataset

Spatial Data in R 56 / 72

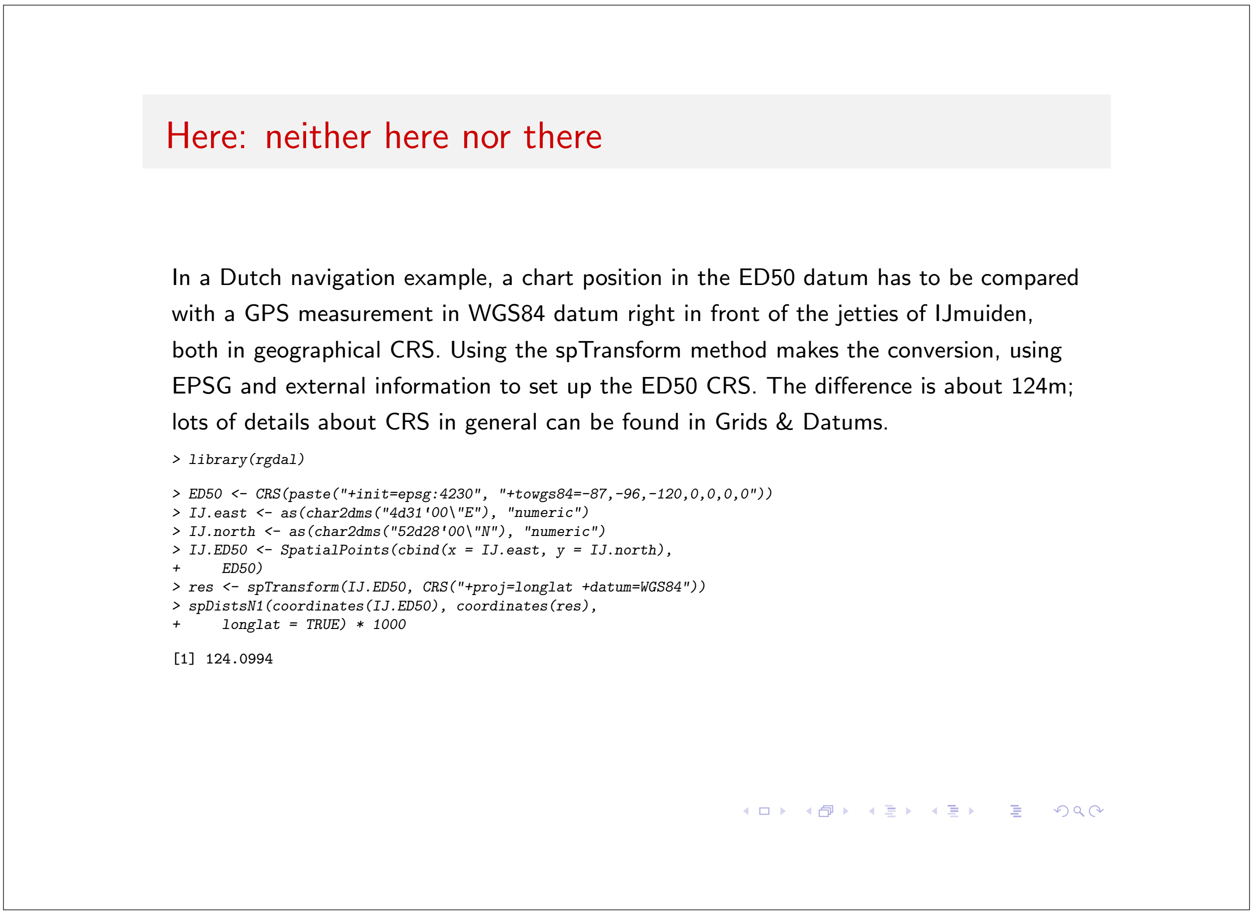


Introduction Coordinates

Coordinate reference systems

- The EPSG list among other sources is used in the workhorse PROJ.4 library, which as implemented by Frank Warmerdam handles transformation of spatial positions between different CRS
- This library is interfaced with R in the **rgdal** package, and the `CRS` class is defined partly in **sp**, partly in **rgdal**
- A `CRS` object is defined as a character NA string or a valid PROJ.4 CRS definition
- The validity of the definition can only be checked if **rgdal** is loaded

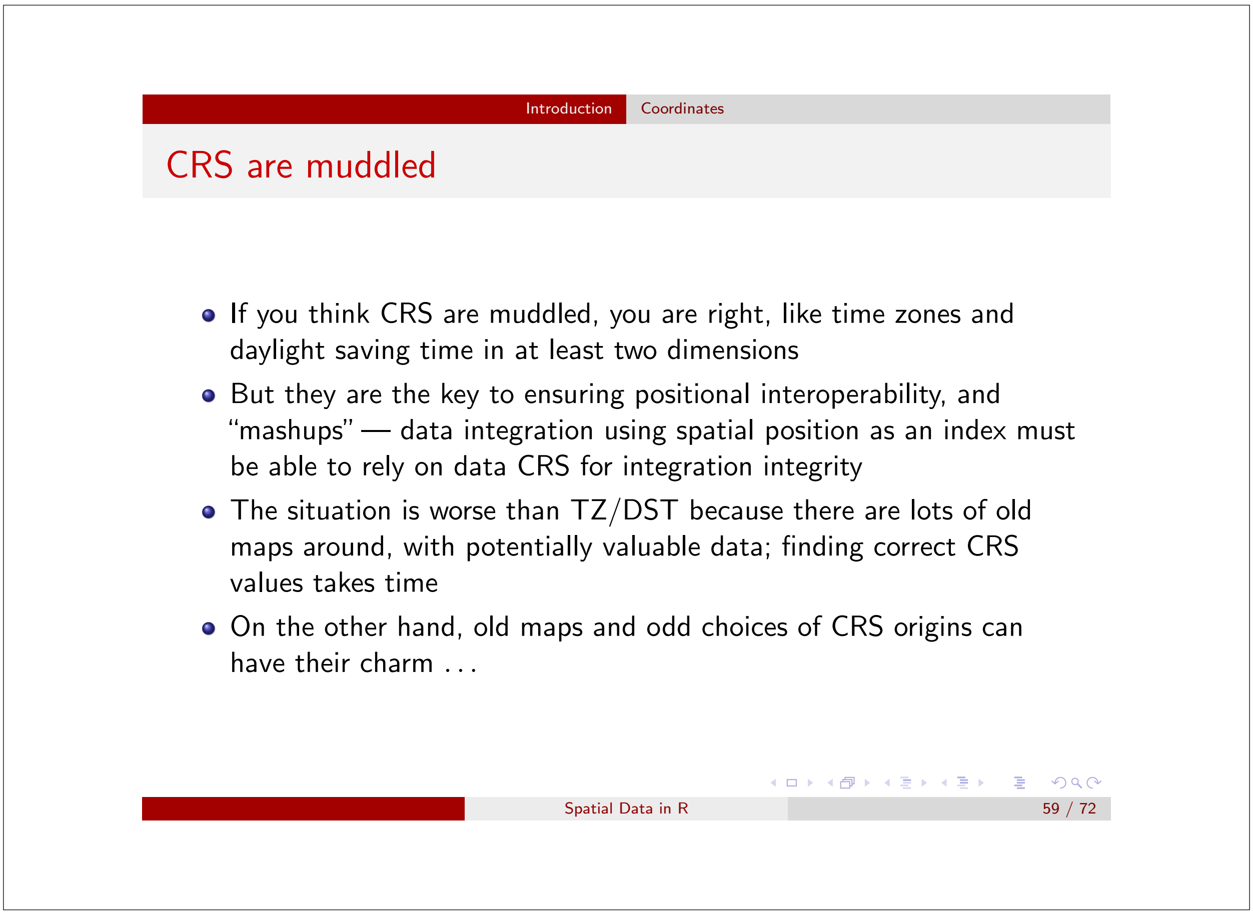
Spatial Data in R 57 / 72



Here: neither here nor there

In a Dutch navigation example, a chart position in the ED50 datum has to be compared with a GPS measurement in WGS84 datum right in front of the jetties of IJmuiden, both in geographical CRS. Using the spTransform method makes the conversion, using EPSG and external information to set up the ED50 CRS. The difference is about 124m; lots of details about CRS in general can be found in Grids & Datums.

```
> library(rgdal)

> ED50 <- CRS(paste("+init=epsg:4230", "+towgs84=-87,-96,-120,0,0,0,0"))
> IJ.east <- as(char2dms("4d31'00\"E"), "numeric")
> IJ.north <- as(char2dms("52d28'00\"N"), "numeric")
> IJ.ED50 <- SpatialPoints(cbind(x = IJ.east, y = IJ.north),
+     ED50)
> res <- spTransform(IJ.ED50, CRS("+proj=longlat +datum=WGS84"))
> spDistsN1(coordinates(IJ.ED50), coordinates(res),
+     longlat = TRUE) * 1000

[1] 124.0994
```

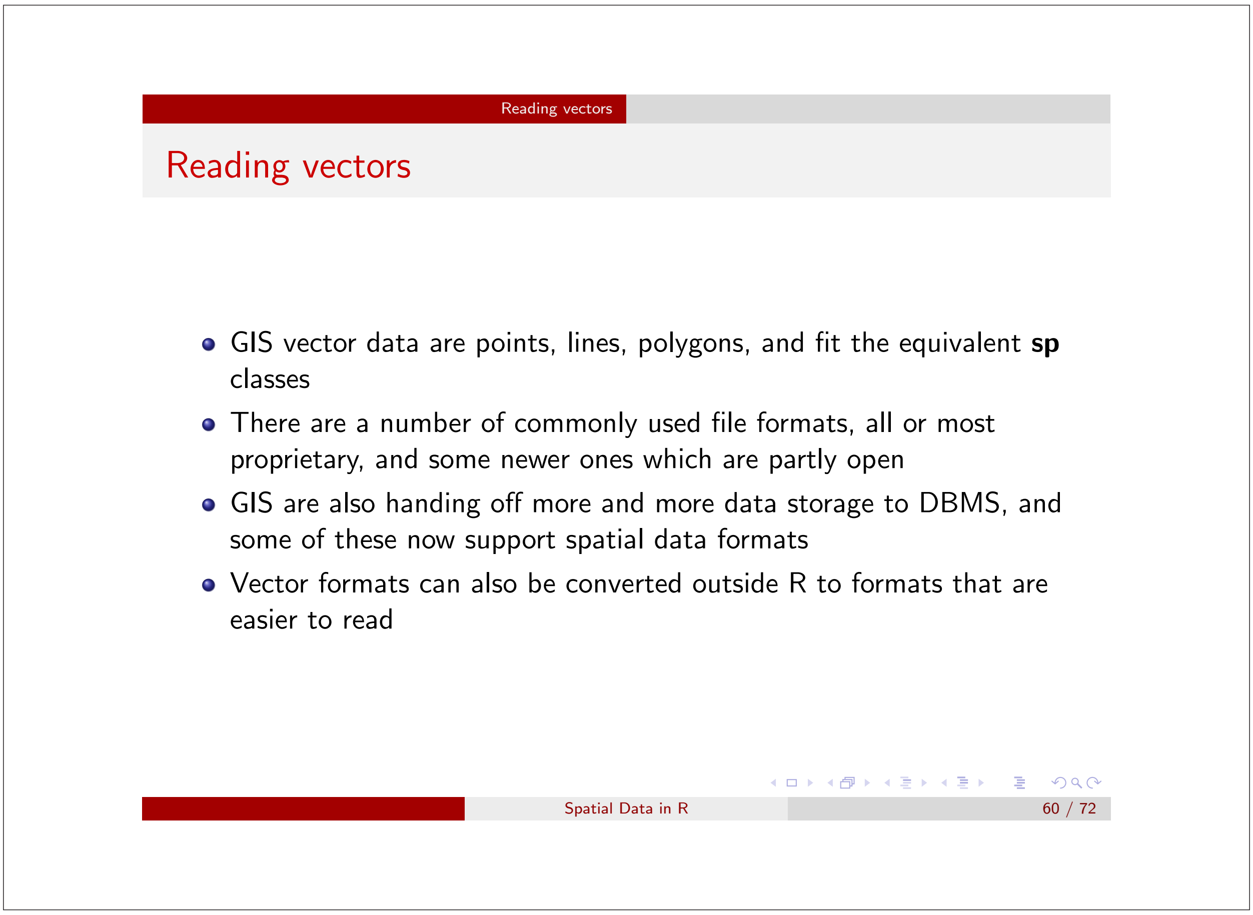


Introduction Coordinates

CRS are muddled

- If you think CRS are muddled, you are right, like time zones and daylight saving time in at least two dimensions
- But they are the key to ensuring positional interoperability, and "mashups" — data integration using spatial position as an index must be able to rely on data CRS for integration integrity
- The situation is worse than TZ/DST because there are lots of old maps around, with potentially valuable data; finding correct CRS values takes time
- On the other hand, old maps and odd choices of CRS origins can have their charm . . .

Spatial Data in R 59 / 72

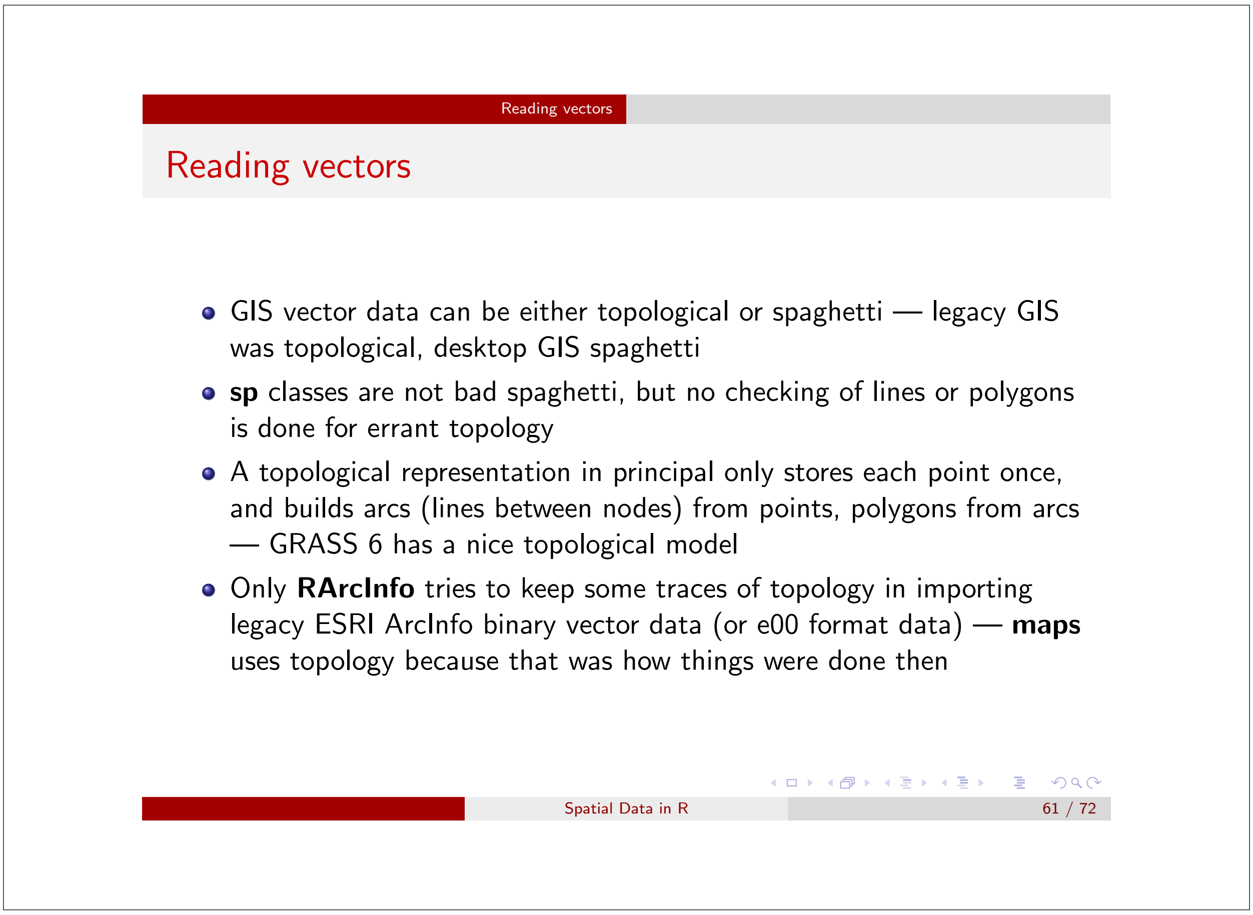


Reading vectors

Reading vectors

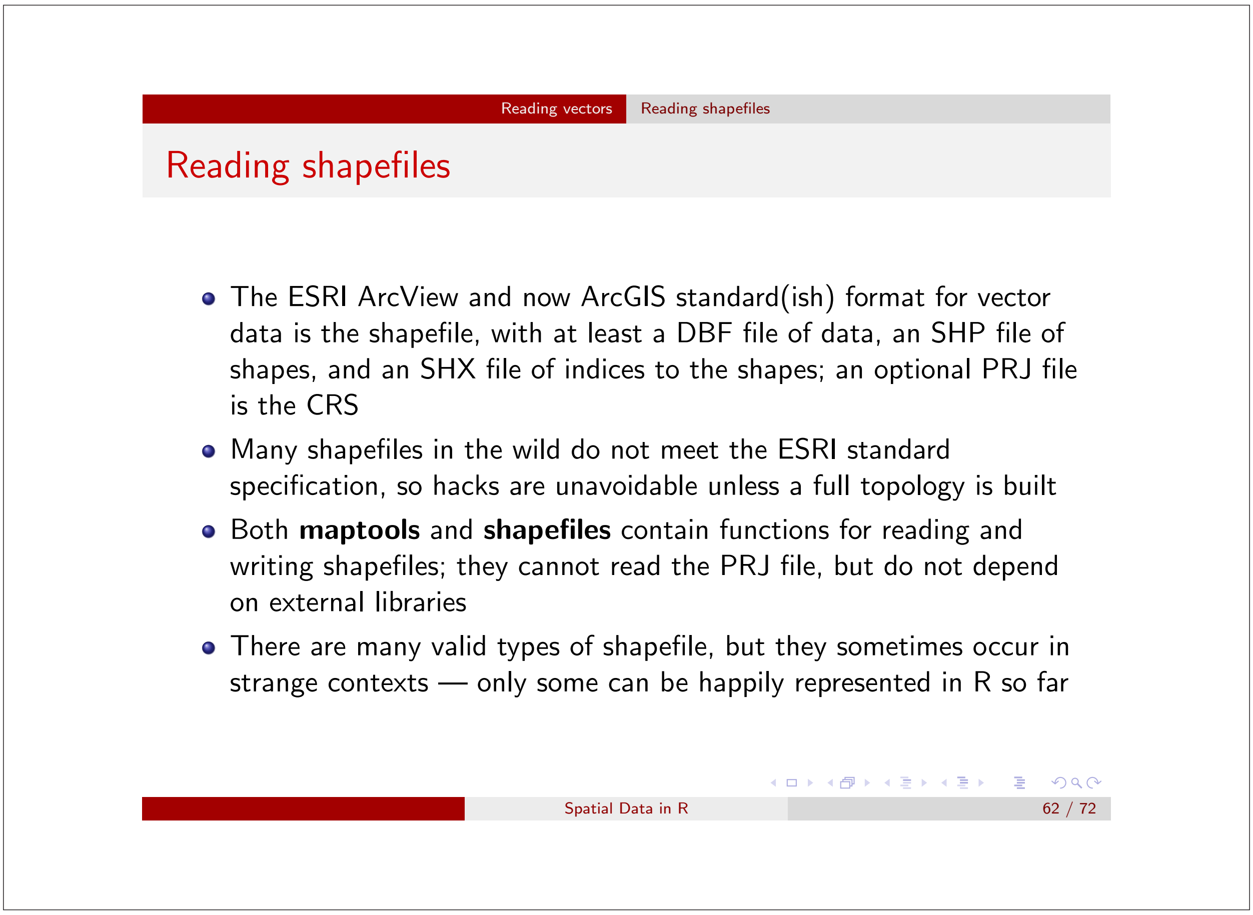
- GIS vector data are points, lines, polygons, and fit the equivalent **sp** classes
- There are a number of commonly used file formats, all or most proprietary, and some newer ones which are partly open
- GIS are also handing off more and more data storage to DBMS, and some of these now support spatial data formats
- Vector formats can also be converted outside R to formats that are easier to read

Spatial Data in R 60 / 72



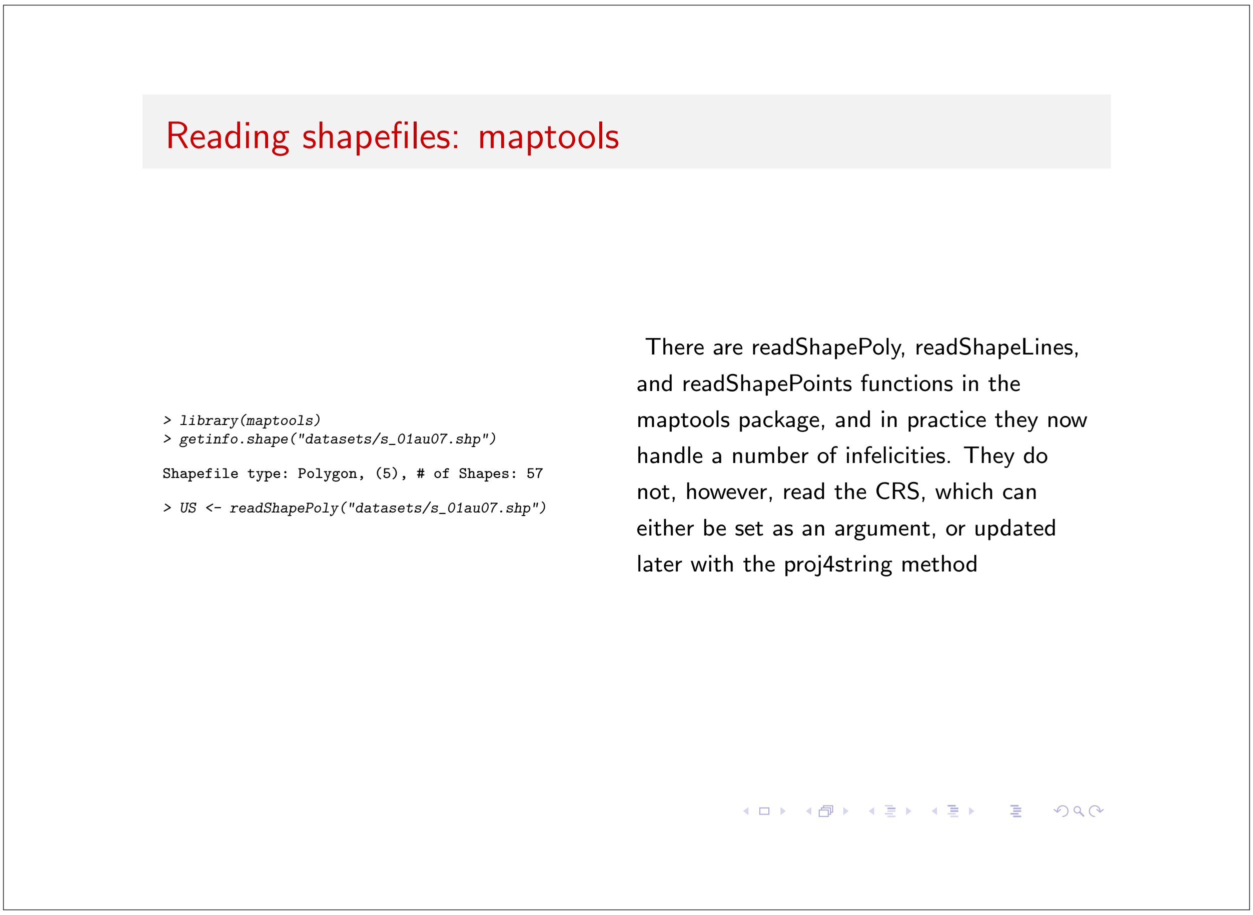
Reading vectors

- GIS vector data can be either topological or spaghetti — legacy GIS was topological, desktop GIS spaghetti
- **sp** classes are not bad spaghetti, but no checking of lines or polygons is done for errant topology
- A topological representation in principal only stores each point once, and builds arcs (lines between nodes) from points, polygons from arcs — GRASS 6 has a nice topological model
- Only **RArcInfo** tries to keep some traces of topology in importing legacy ESRI ArcInfo binary vector data (or e00 format data) — **maps** uses topology because that was how things were done then



Reading shapefiles

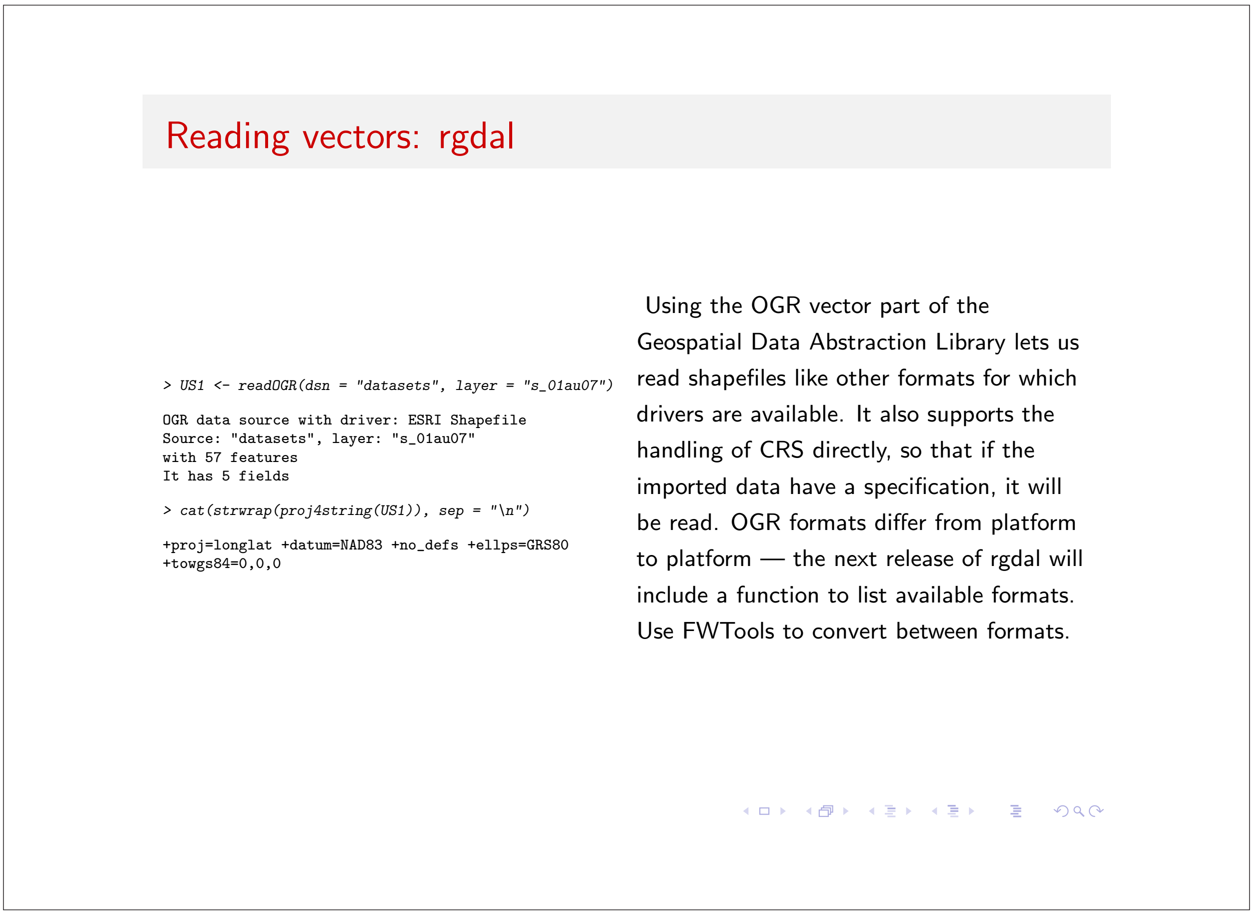
- The ESRI ArcView and now ArcGIS standard(ish) format for vector data is the shapefile, with at least a DBF file of data, an SHP file of shapes, and an SHX file of indices to the shapes; an optional PRJ file is the CRS
- Many shapefiles in the wild do not meet the ESRI standard specification, so hacks are unavoidable unless a full topology is built
- Both **maptools** and **shapefiles** contain functions for reading and writing shapefiles; they cannot read the PRJ file, but do not depend on external libraries
- There are many valid types of shapefile, but they sometimes occur in strange contexts — only some can be happily represented in R so far



Reading shapefiles: maptools

```
> library(maptools)
> getinfo.shape("datasets/s_01au07.shp")

Shapefile type: Polygon, (5), # of Shapes: 57

> US <- readShapePoly("datasets/s_01au07.shp")
```

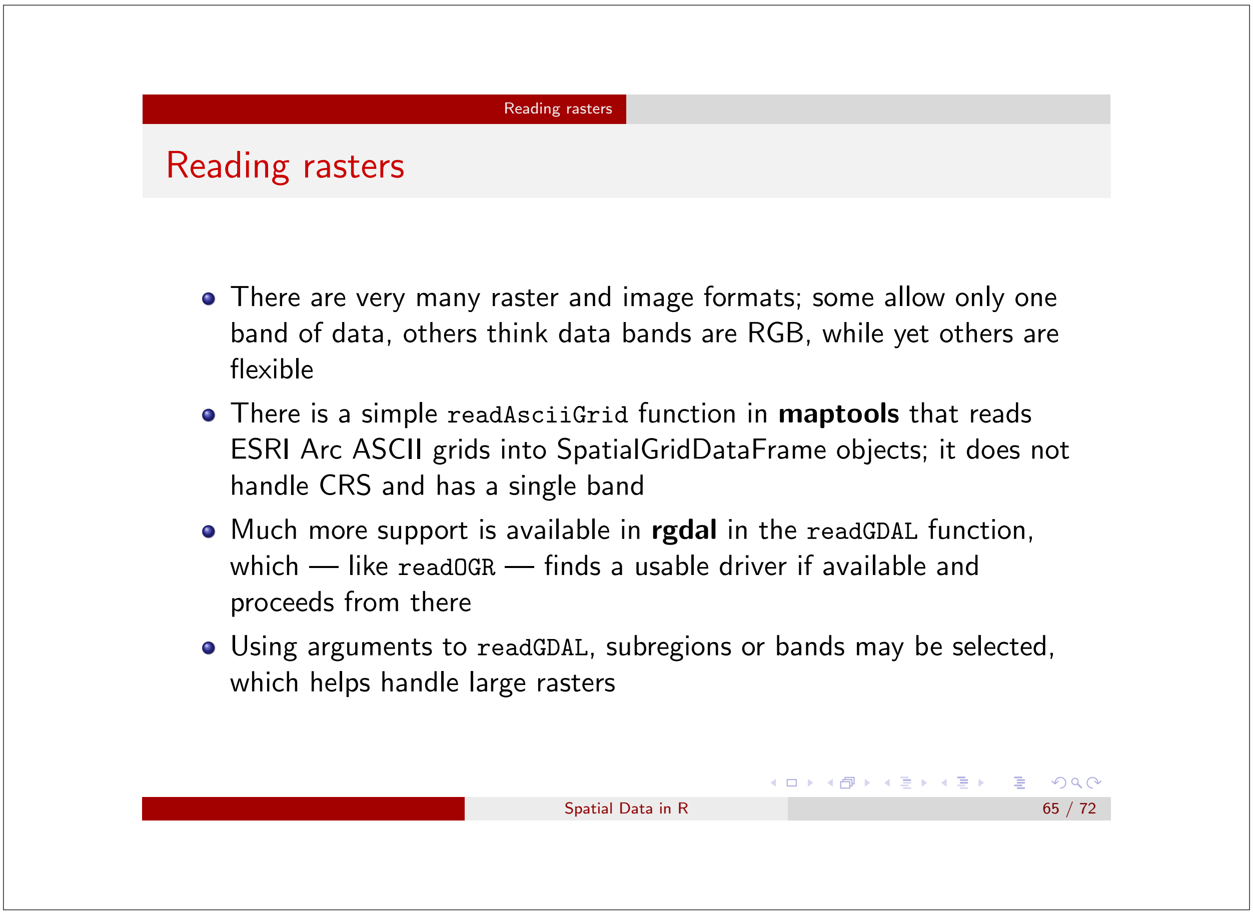
There are readShapePoly, readShapeLines, and readShapePoints functions in the maptools package, and in practice they now handle a number of infelicities. They do not, however, read the CRS, which can either be set as an argument, or updated later with the proj4string method



Reading vectors: rgdal

```
> US1 <- readOGR(dsn = "datasets", layer = "s_01au07")

OGR data source with driver: ESRI Shapefile
Source: "datasets", layer: "s_01au07"
with 57 features
It has 5 fields

> cat(strwrap(proj4string(US1)), sep = "\n")

+proj=longlat +datum=NAD83 +no_defs +ellps=GRS80
+towgs84=0,0,0
```

Using the OGR vector part of the Geospatial Data Abstraction Library lets us read shapefiles like other formats for which drivers are available. It also supports the handling of CRS directly, so that if the imported data have a specification, it will be read. OGR formats differ from platform to platform — the next release of rgdal will include a function to list available formats. Use FWTools to convert between formats.

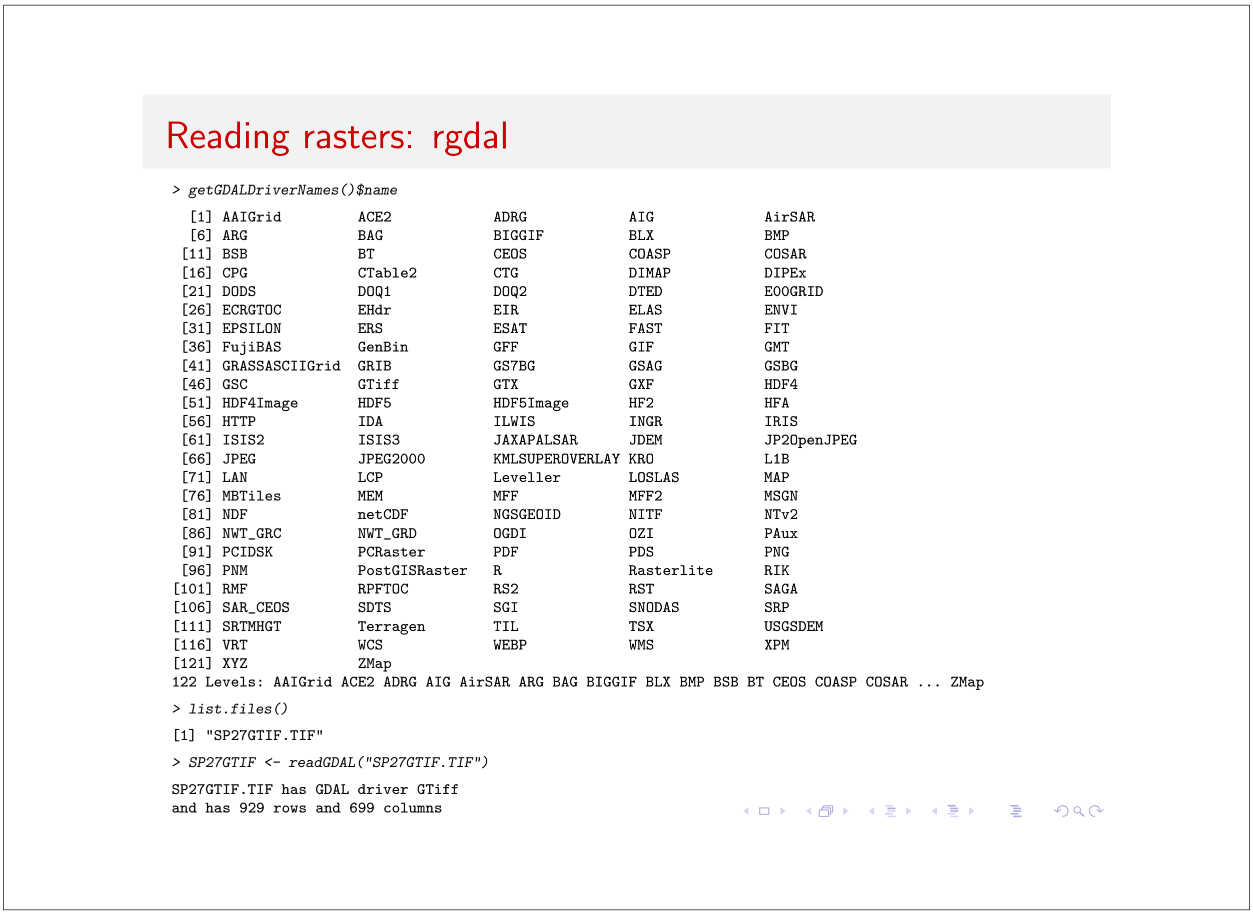


Reading rasters

Reading rasters

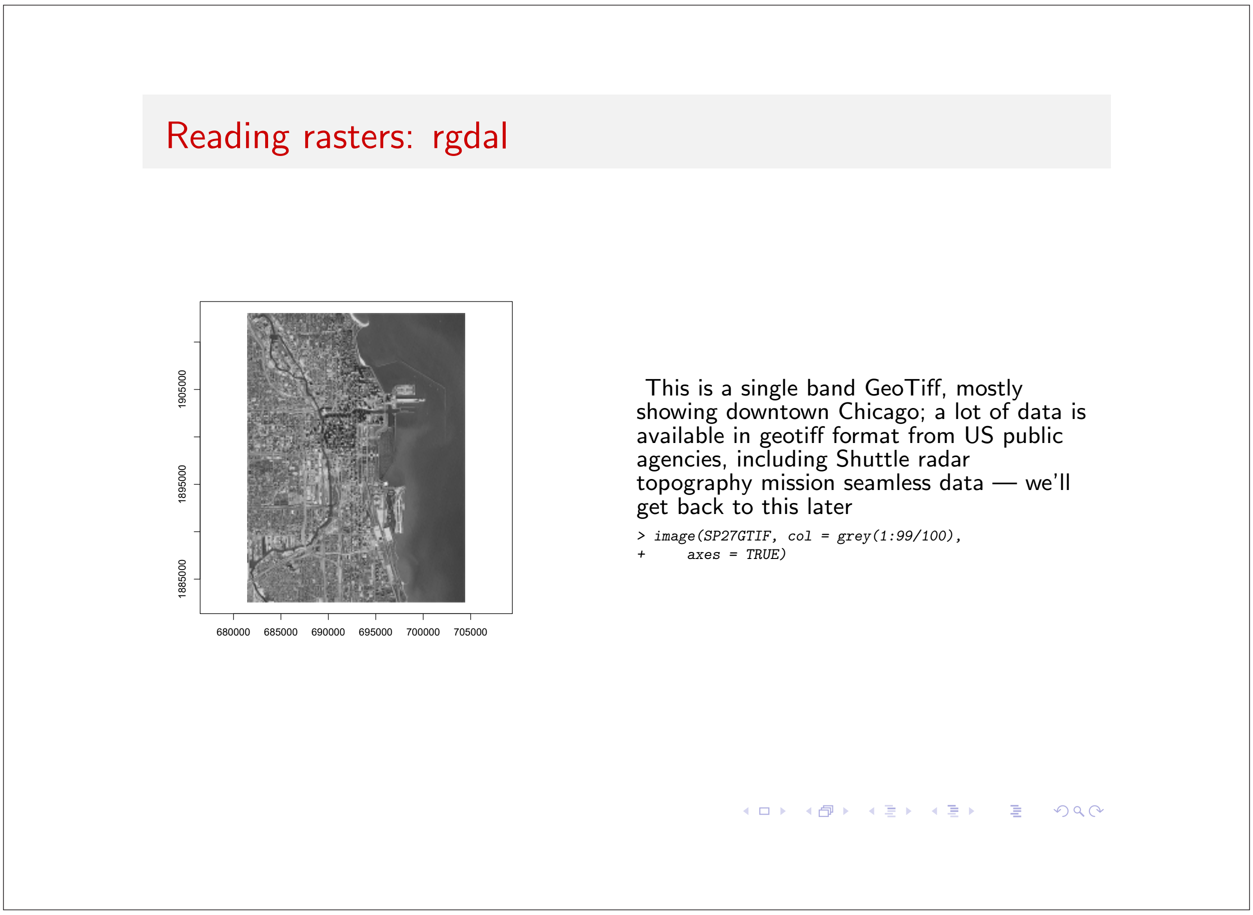
- There are very many raster and image formats; some allow only one band of data, others think data bands are RGB, while yet others are flexible
- There is a simple `readAsciiGrid` function in **maptools** that reads ESRI Arc ASCII grids into SpatialGridDataFrame objects; it does not handle CRS and has a single band
- Much more support is available in **rgdal** in the `readGDAL` function, which — like `readOGR` — finds a usable driver if available and proceeds from there
- Using arguments to `readGDAL`, subregions or bands may be selected, which helps handle large rasters

Spatial Data in R 65 / 72



Reading rasters: rgdal

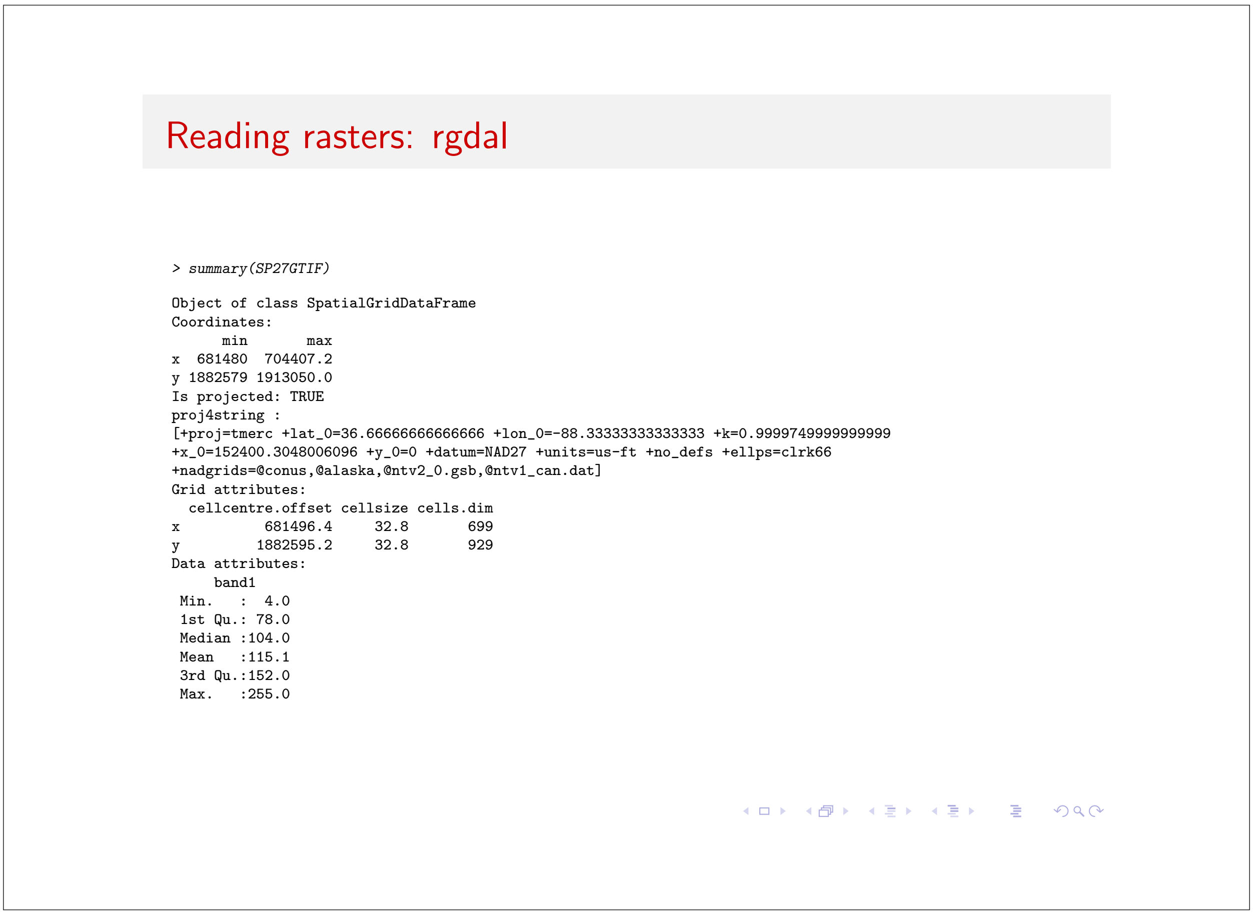
```
> getGDALDriverNames()$name
  [1] AAIGrid         ACE2            ADRG            AIG             AirSAR
  [6] ARG             BAG             BIGGIF          BLX             BMP
 [11] BSB             BT              CEOS            COASP           COSAR
 [16] CPG             CTable2         CTG             DIMAP           DIPEx
 [21] DODS            DOQ1            DOQ2            DTED            E00GRID
 [26] ECRGTOC         EHdr            EIR             ELAS            ENVI
 [31] EPSILON         ERS             ESAT            FAST            FIT
 [36] FujiBAS         GenBin          GFF             GIF             GMT
 [41] GRASSASCIIGrid  GRIB            GS7BG           GSAG            GSBG
 [46] GSC             GTiff           GTX             GXF             HDF4
 [51] HDF4Image       HDF5            HDF5Image       HF2             HFA
 [56] HTTP            IDA             ILWIS           INGR            IRIS
 [61] ISIS2           ISIS3           JAXAPALSAR      JDEM            JP2OpenJPEG
 [66] JPEG            JPEG2000        KMLSUPEROVERLAY KRO             L1B
 [71] LAN             LCP             Leveller        LOSLAS          MAP
 [76] MBTiles         MEM             MFF             MFF2            MSGN
 [81] NDF             netCDF          NGSGEOID        NITF            NTv2
 [86] NWT_GRC         NWT_GRD         OGDI            OZI             PAux
 [91] PCIDSK          PCRaster        PDF             PDS             PNG
 [96] PNM             PostGISRaster   R               Rasterlite      RIK
[101] RMF             RPFTOC          RS2             RST             SAGA
[106] SAR_CEOS        SDTS            SGI             SNODAS          SRP
[111] SRTMHGT         Terragen        TIL             TSX             USGSDEM
[116] VRT             WCS             WEBP            WMS             XPM
[121] XYZ             ZMap
122 Levels: AAIGrid ACE2 ADRG AIG AirSAR ARG BAG BIGGIF BLX BMP BSB BT CEOS COASP COSAR ... ZMap

> list.files()

[1] "SP27GTIF.TIF"

> SP27GTIF <- readGDAL("SP27GTIF.TIF")

SP27GTIF.TIF has GDAL driver GTiff
and has 929 rows and 699 columns
```



Reading rasters: rgdal

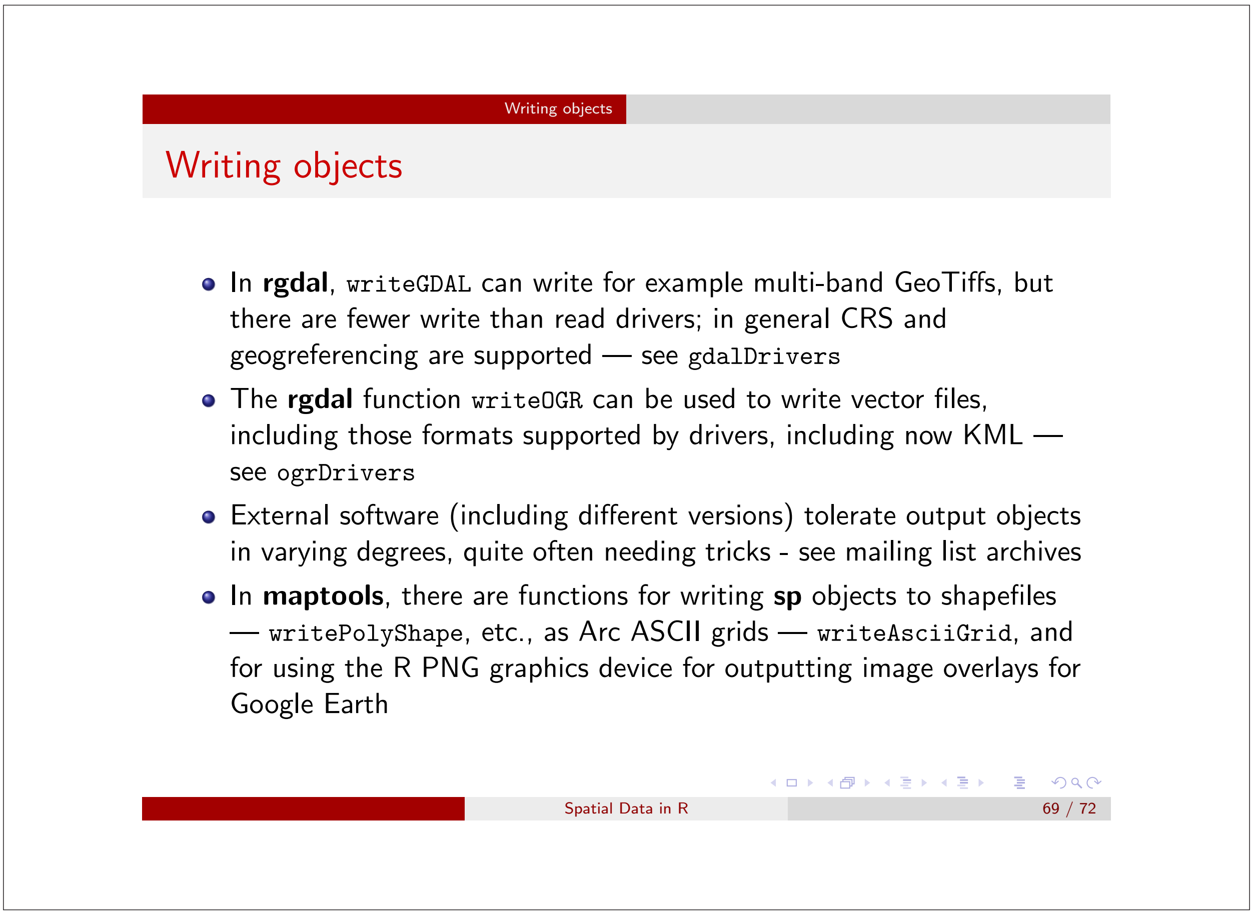
This is a single band GeoTiff, mostly showing downtown Chicago; a lot of data is available in geotiff format from US public agencies, including Shuttle radar topography mission seamless data — we'll get back to this later

```
> image(SP27GTIF, col = grey(1:99/100),
+     axes = TRUE)
```



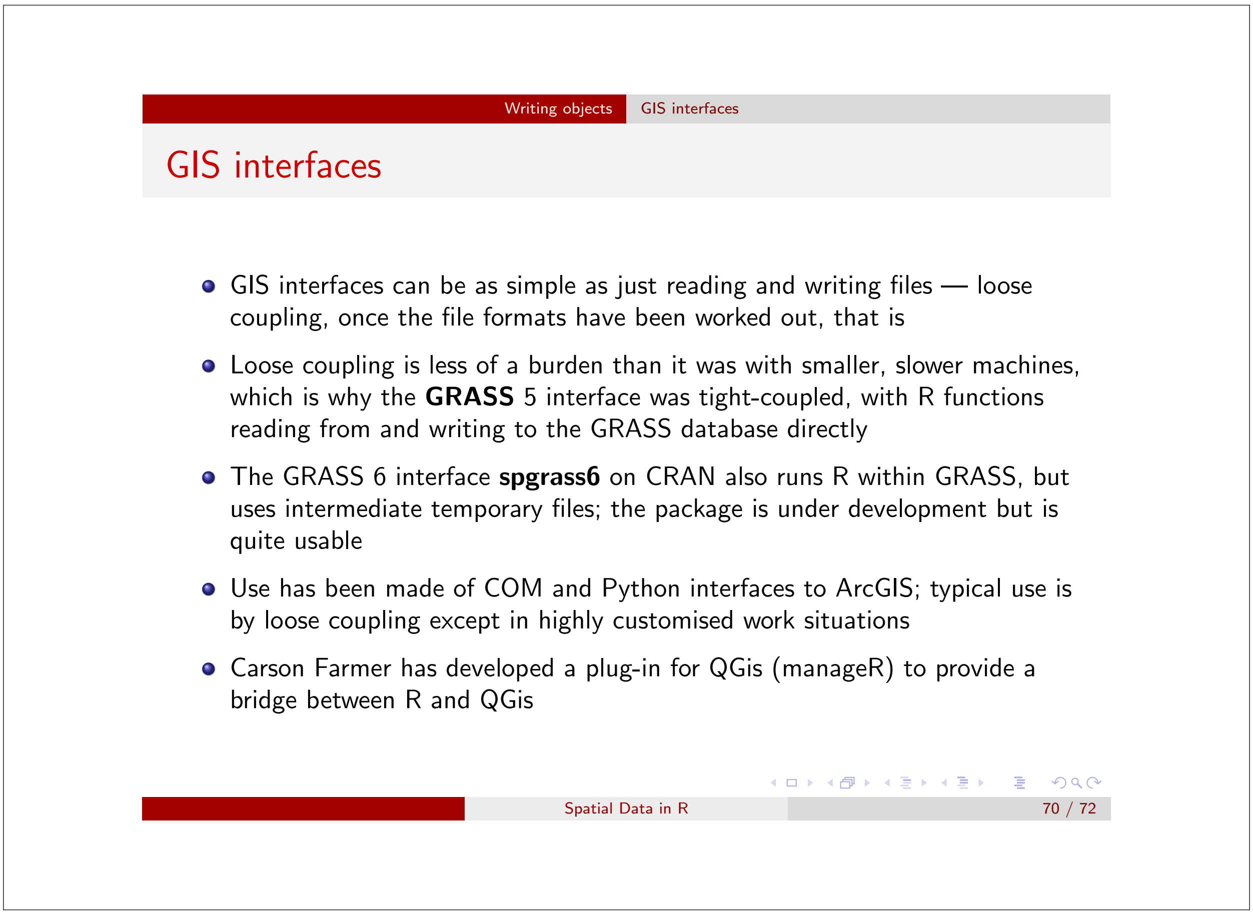
Reading rasters: rgdal

```
> summary(SP27GTIF)

Object of class SpatialGridDataFrame
Coordinates:
      min       max
x  681480  704407.2
y 1882579 1913050.0
Is projected: TRUE
proj4string :
[+proj=tmerc +lat_0=36.66666666666666 +lon_0=-88.33333333333333 +k=0.9999749999999999
+x_0=152400.3048006096 +y_0=0 +datum=NAD27 +units=us-ft +no_defs +ellps=clrk66
+nadgrids=@conus,@alaska,@ntv2_0.gsb,@ntv1_can.dat]
Grid attributes:
  cellcentre.offset cellsize cells.dim
x          681496.4     32.8       699
y         1882595.2     32.8       929
Data attributes:
     band1
 Min.   :  4.0
 1st Qu.: 78.0
 Median :104.0
 Mean   :115.1
 3rd Qu.:152.0
 Max.   :255.0
```



Writing objects

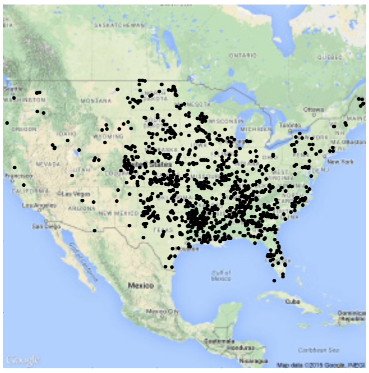
- In **rgdal**, `writeGDAL` can write for example multi-band GeoTiffs, but there are fewer write than read drivers; in general CRS and geogreferencing are supported — see `gdalDrivers`
- The **rgdal** function `writeOGR` can be used to write vector files, including those formats supported by drivers, including now KML — see `ogrDrivers`
- External software (including different versions) tolerate output objects in varying degrees, quite often needing tricks - see mailing list archives
- In **maptools**, there are functions for writing **sp** objects to shapefiles — `writePolyShape`, etc., as Arc ASCII grids — `writeAsciiGrid`, and for using the R PNG graphics device for outputting image overlays for Google Earth



GIS interfaces

- GIS interfaces can be as simple as just reading and writing files — loose coupling, once the file formats have been worked out, that is
- Loose coupling is less of a burden than it was with smaller, slower machines, which is why the **GRASS** 5 interface was tight-coupled, with R functions reading from and writing to the GRASS database directly
- The GRASS 6 interface **spgrass6** on CRAN also runs R within GRASS, but uses intermediate temporary files; the package is under development but is quite usable
- Use has been made of COM and Python interfaces to ArcGIS; typical use is by loose coupling except in highly customised work situations
- Carson Farmer has developed a plug-in for QGis (manageR) to provide a bridge between R and QGis

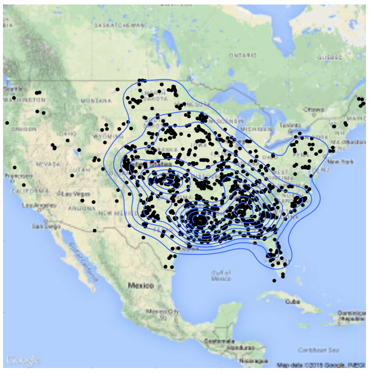
Using Google Maps



This is a simple example on how to use **ggmap** to display the tornado dataset using a background taken from Google Maps:

```
> library(ggmap)
> load("results/unit1.RData")
> pts <- as.data.frame(coordinates(storn))
> names(pts) <- c("lon", "lat")
> qmap("usa", zoom = 4) + geom_point(data = pts)
```

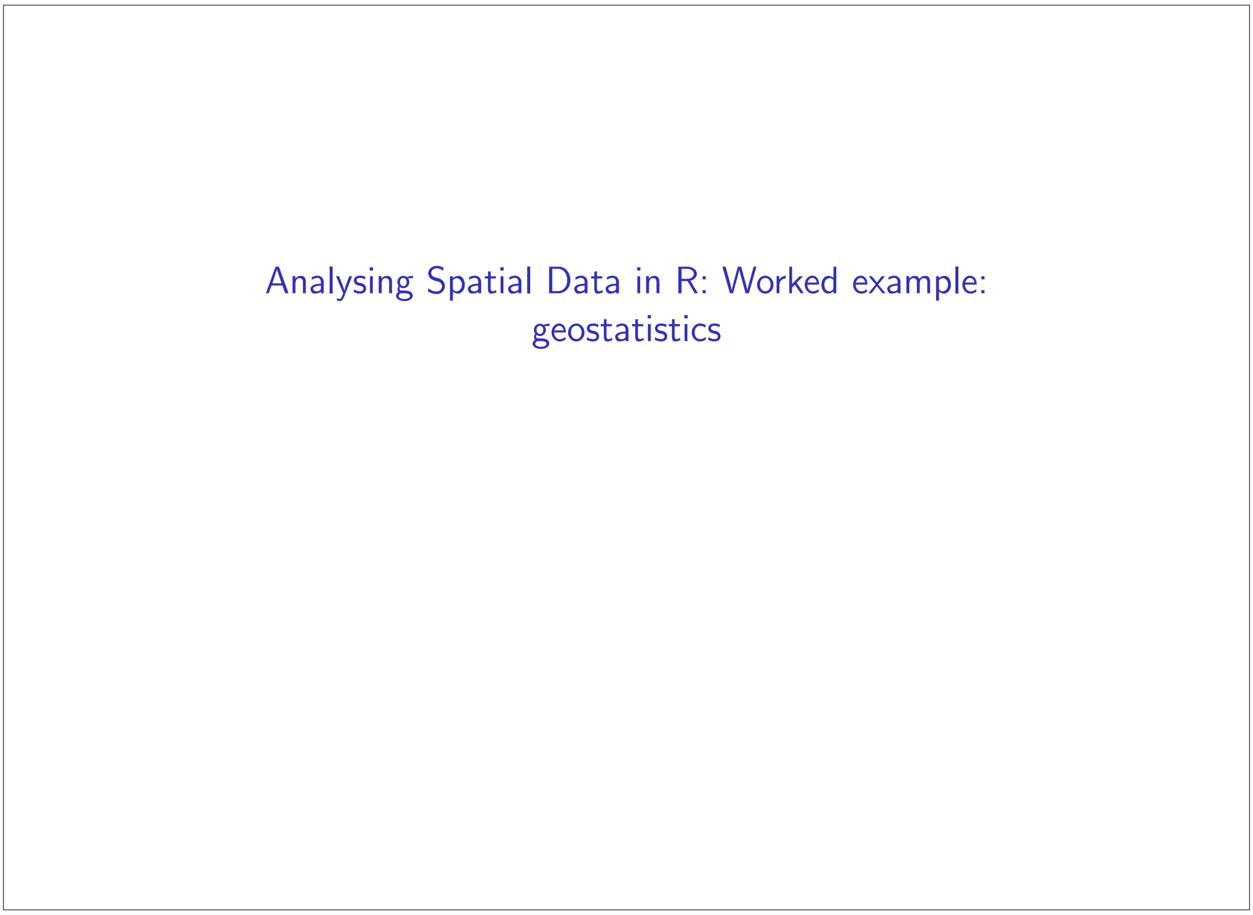
Using Google Maps



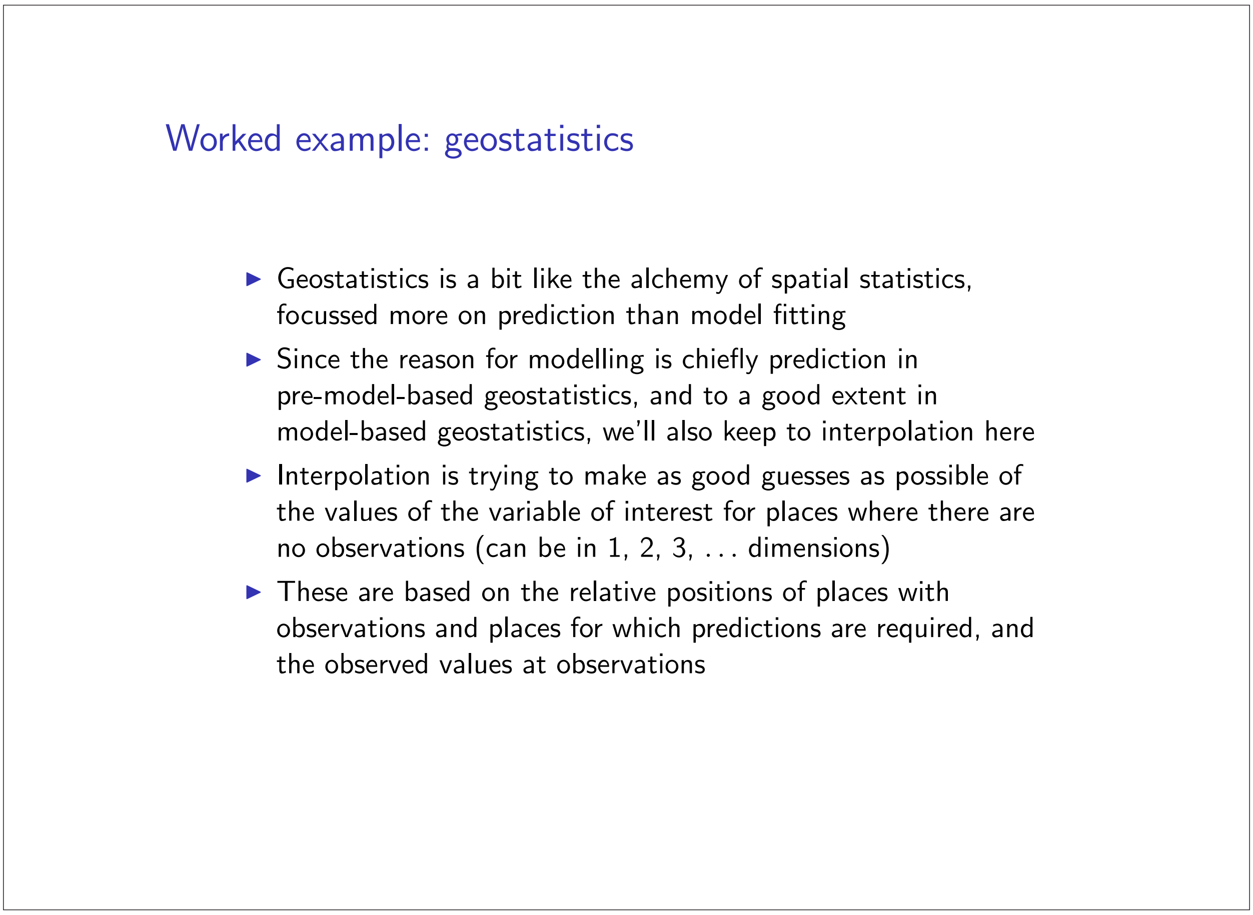
This is a simple example on how to use **ggmap** to display the tornado dataset using a background taken from Google Maps. Now we have also added a kernel density smoothing:

```
> qmap("usa", zoom = 4) + geom_point(data = pts) +
+     geom_density2d(data = pts)
```

Analysing Spatial Data in R: Worked example: geostatistics

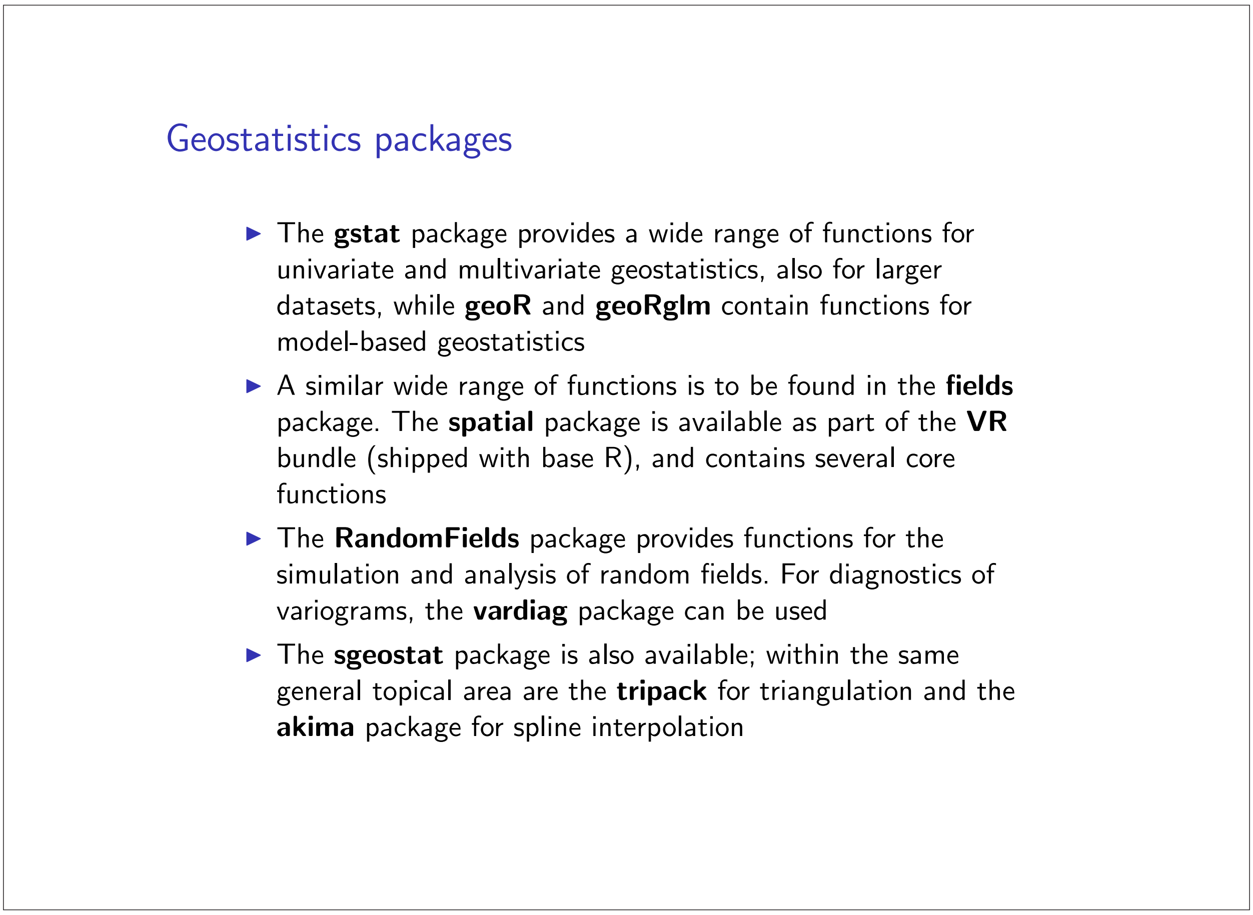


Analysing Spatial Data in R: Worked example: geostatistics



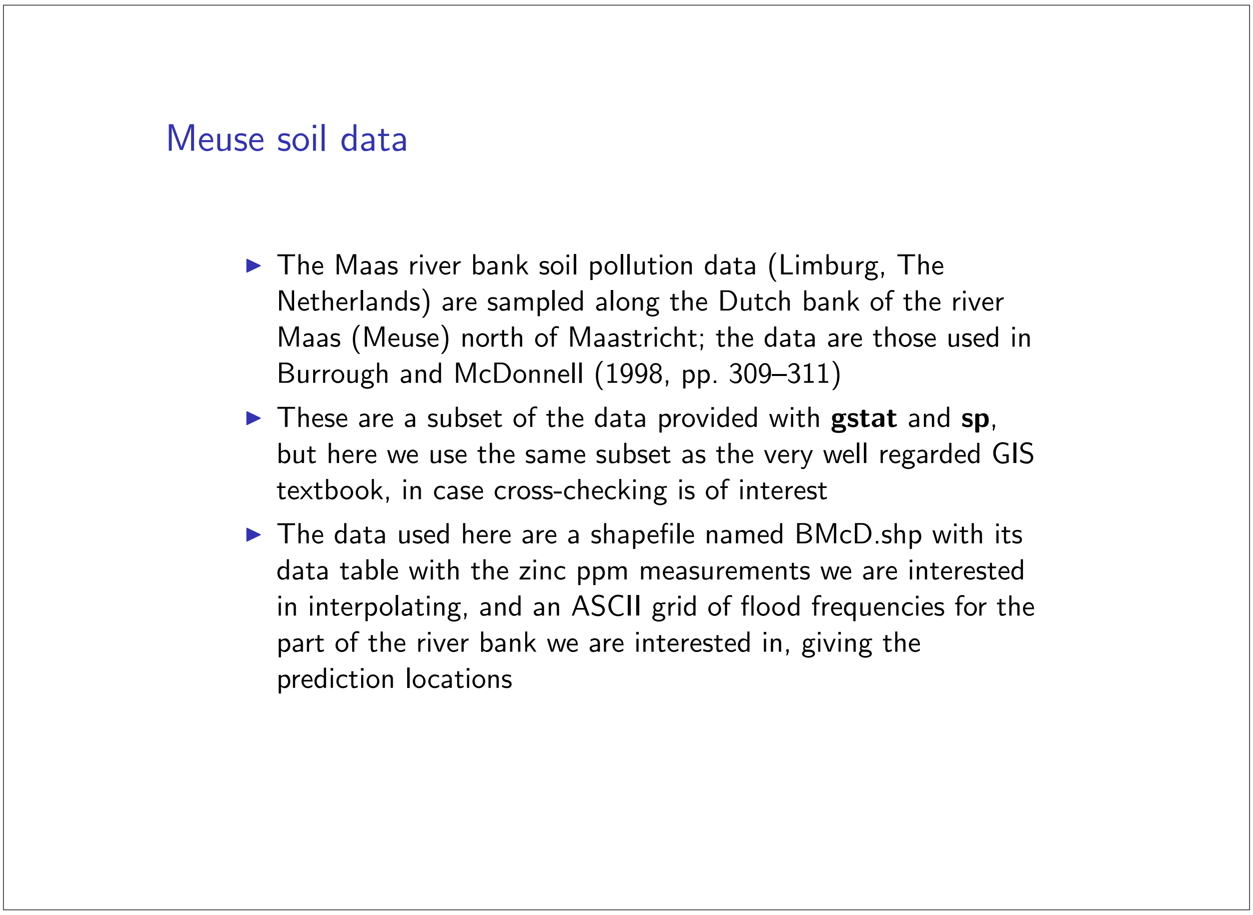
Worked example: geostatistics

- 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

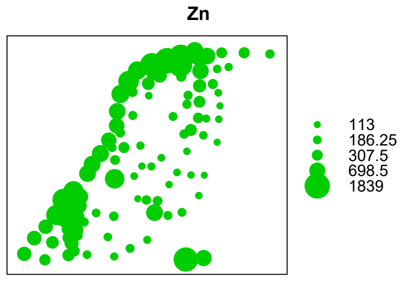
- The Maas river bank soil pollution data (Limburg, The Netherlands) are sampled along the Dutch bank of the river Maas (Meuse) north of Maastricht; the data are those used in Burrough and McDonnell (1998, pp. 309–311)
- These are a subset of the data provided with **gstat** and **sp**, but here we use 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

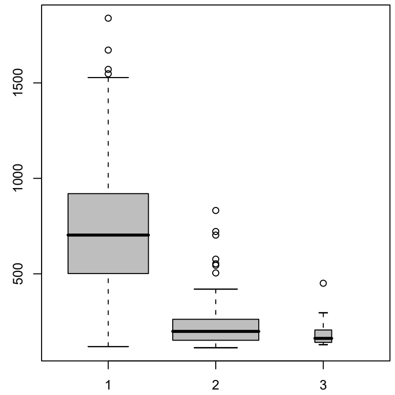
Observed zinc ppm levels



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")
```

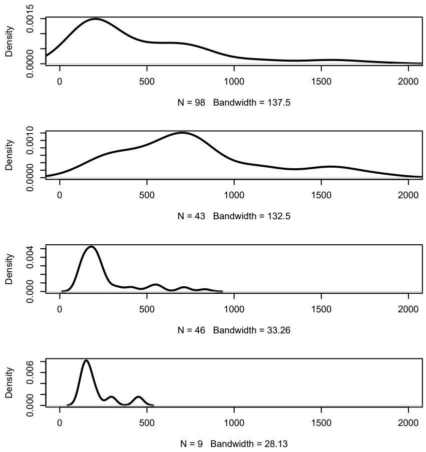
Flood frequency boxplots



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



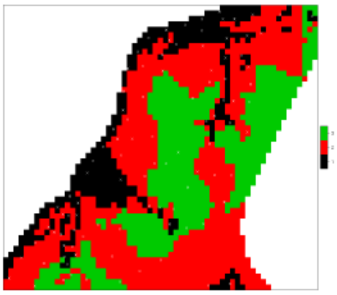
This impression is supported by dividing density plots up into one pooled, and three separate flood frequency classes — the at least annual flooding class has higher values than the others

```
> 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:

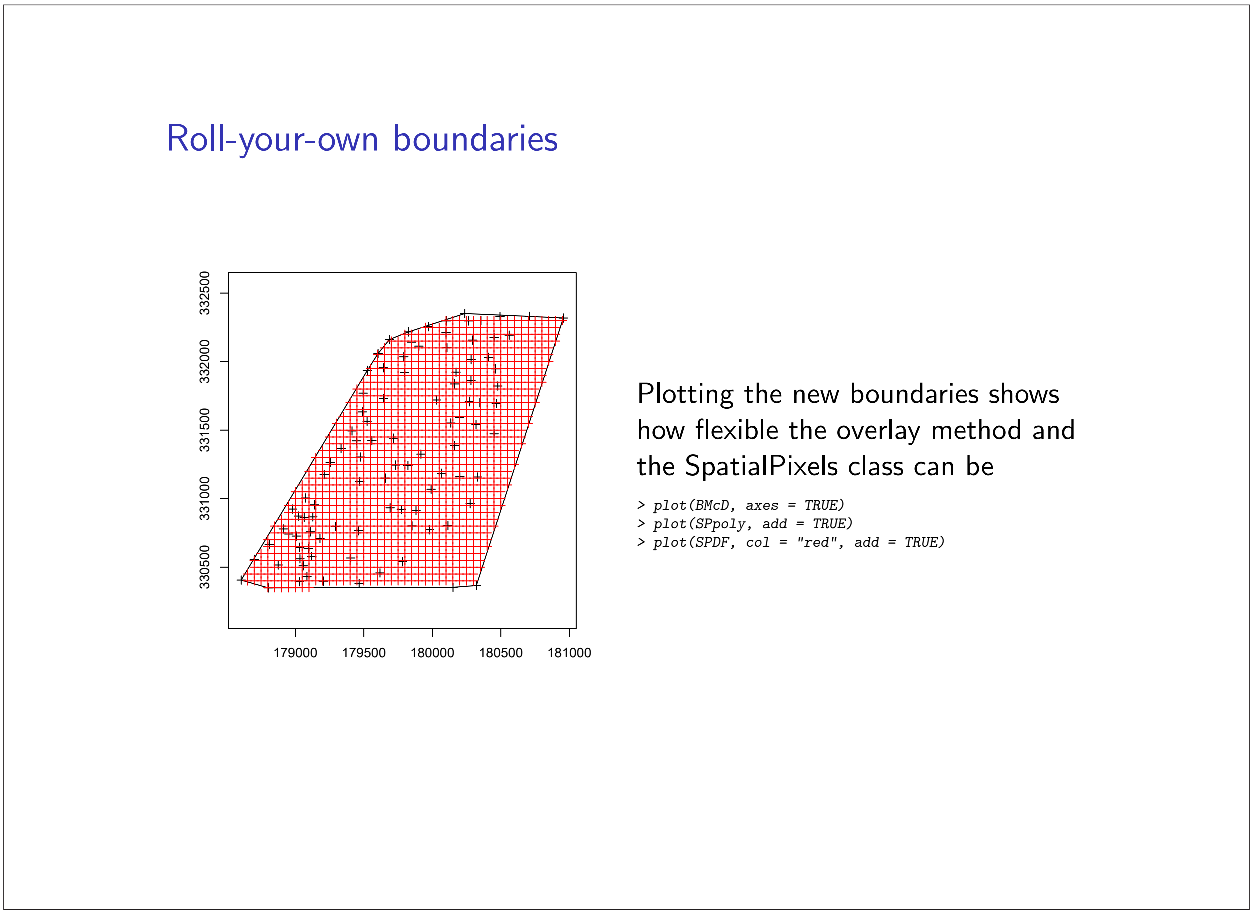
```
> BMcD_grid <- as(readGDAL("BMcD_fldf.txt"),
+     "SpatialPixelsDataFrame")

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:

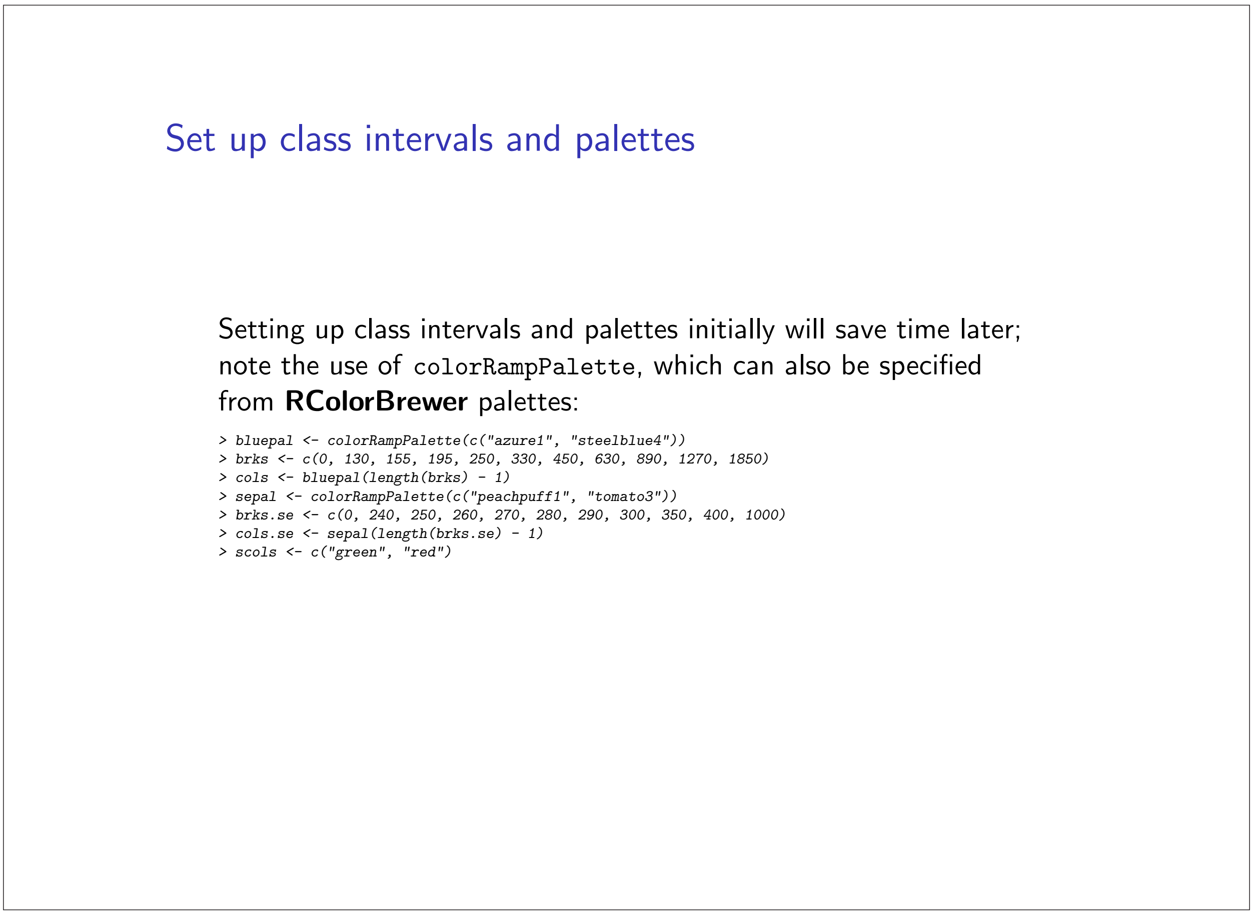
```
> crds <- coordinates(BMcD)
> poly <- crds[chull(crds), ]
> poly <- rbind(poly, poly[1, ])
> SPpoly <- SpatialPolygons(list(Polygons(list(Polygon(poly)), ID = "poly")))
> bbox(BMcD)

              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:

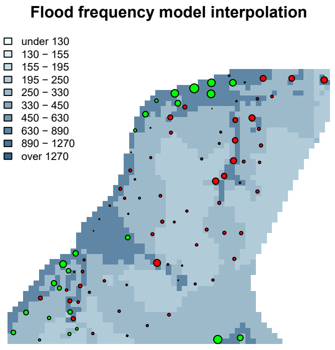
```
> bluepal <- colorRampPalette(c("azure1", "steelblue4"))
> brks <- c(0, 130, 155, 195, 250, 330, 450, 630, 890, 1270, 1850)
> cols <- bluepal(length(brks) - 1)
> sepal <- colorRampPalette(c("peachpuff1", "tomato3"))
> brks.se <- c(0, 240, 250, 260, 270, 280, 290, 300, 350, 400, 1000)
> cols.se <-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

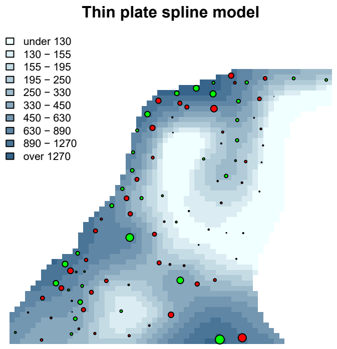
```
> library(maptools)
> BMcD_grid$lm_pred <- predict(fres,
+     newdata = BMcD_grid)
> image(BMcD_grid, "lm_pred",
+     breaks = brks, col = cols)
> title("Flood frequency model interpolation")
> pe <- BMcD$Zn - eres$predictions
> symbols(coordinates(BMcD), circles = sqrt(abs(pe)),
+     fg = "black", 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:

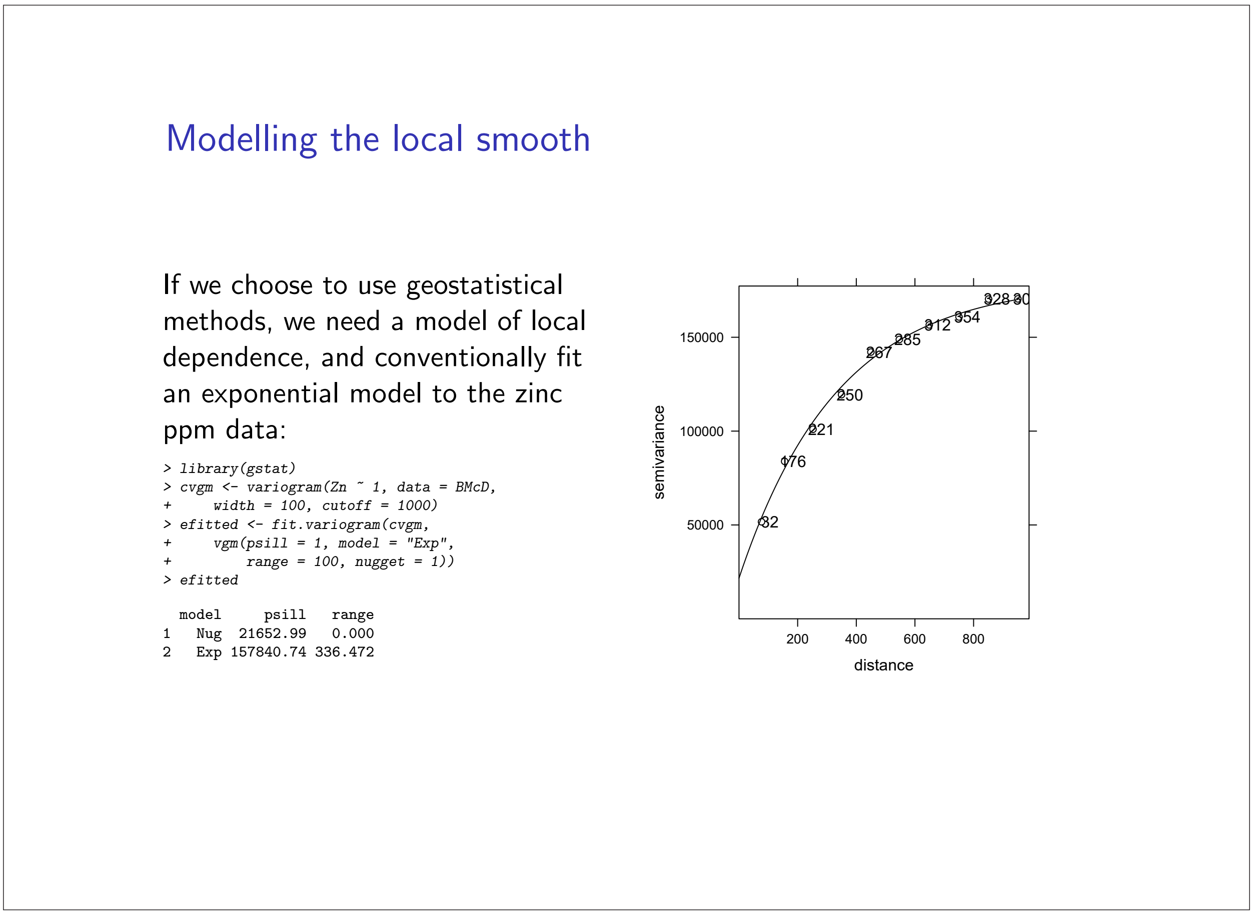
```
> library(fields)
> pe_tps <- numeric(nrow(BMcD))
> cBMcD <- coordinates(BMcD)
> for (i in seq(along = pe_tps)) {
+     tpsi <- Tps(cBMcD[-i, ], BMcD$Zn[-i])
+     pri <- predict(tpsi, cBMcD[i, , drop = FALSE])
+     pe_tps[i] <- BMcD$Zn[i] - pri
+ }
> round(sqrt(mean(pe_tps^2)), 2)

[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:

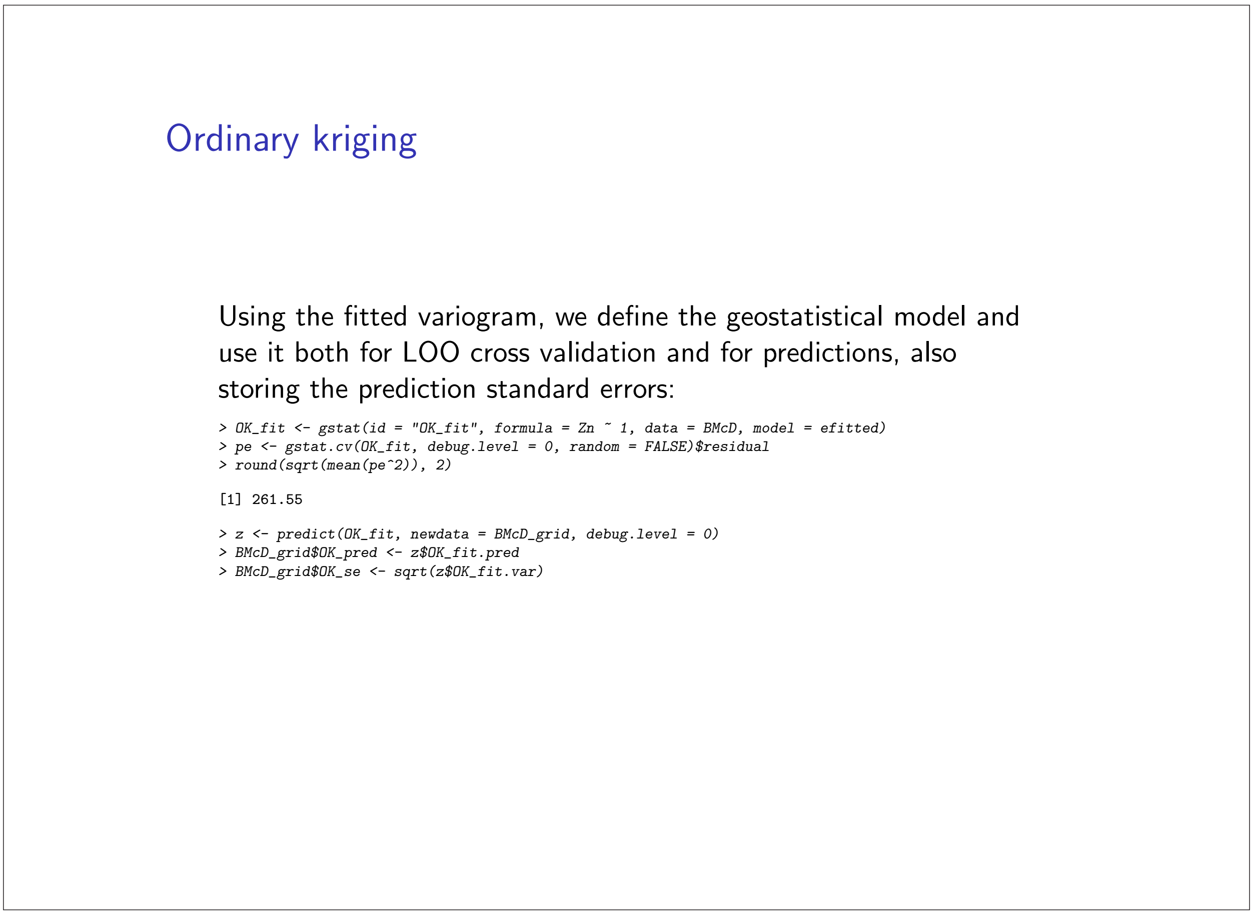
```
> BMcD_grid$spl_pred <- predict(tps,
+     coordinates(BMcD_grid))
> image(BMcD_grid, "spl_pred",
+     breaks = brks, col = cols)
```



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
```

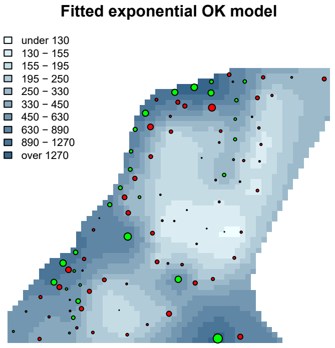


Ordinary kriging

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)
```

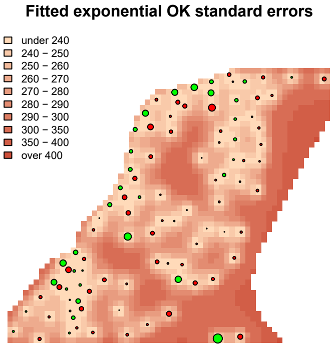
Ordinary kriging predictions



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:

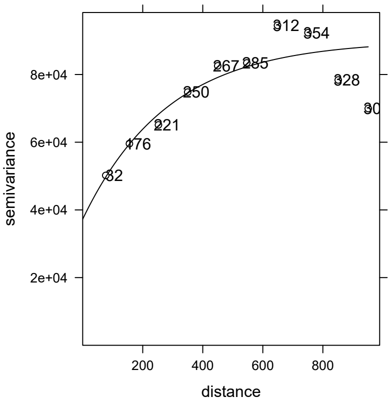
```
> image(BMcD_grid, "OK_se", breaks = brks.se,
+     col = cols.se)
```

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

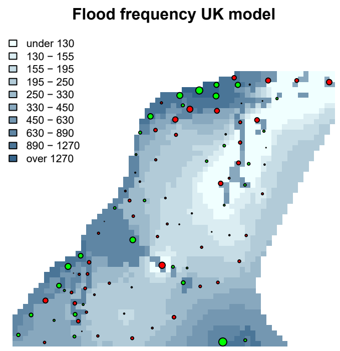
The geostatistical packages, like **gstat**, use formula objects in standard ways where possible, which allows for considerable flexibility, as in this case, where we do really quite well in terms of LOO CV — and reach the same conclusion as Burrough and McDonnell about the choice of model:

```
> 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)
```

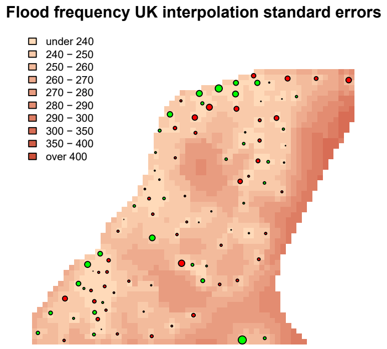
Universal kriging predictions



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)
```

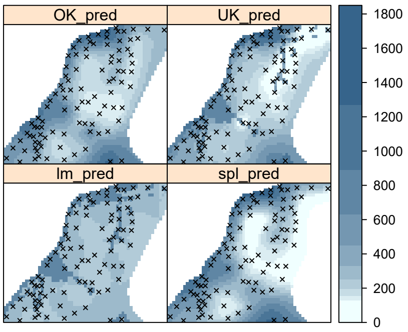
Universal kriging standard errors



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:

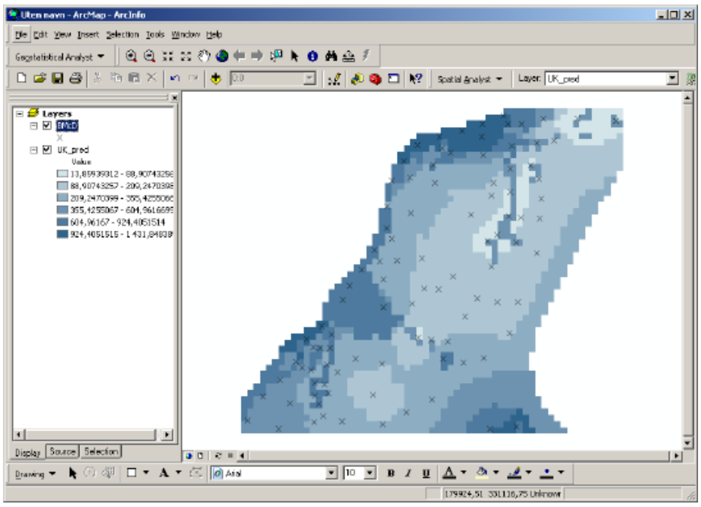
```
> pts = list("sp.points", BMcD,
+      pch = 4, col = "black",
+      cex = 0.5)
> spplot(BMcD_grid, c("lm_pred",
+      "spl_pred", "OK_pred", "UK_pred"),
+      at = brks, col.regions = cols,
+      sp.layout = list(pts))
```

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")
```

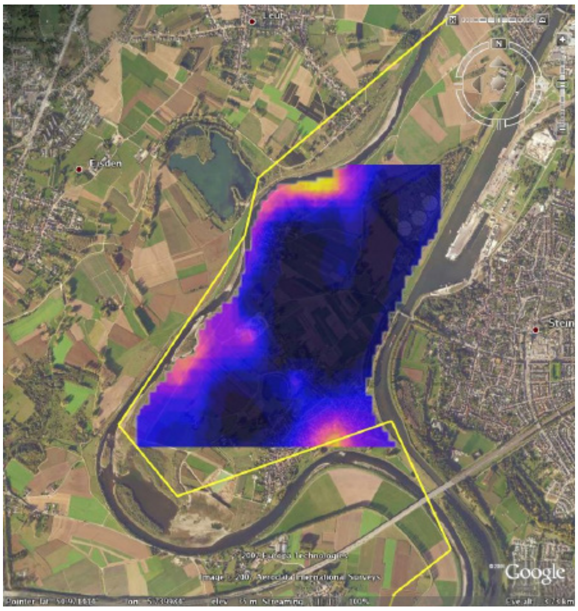
The exported raster viewed in ArcGIS



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")
```

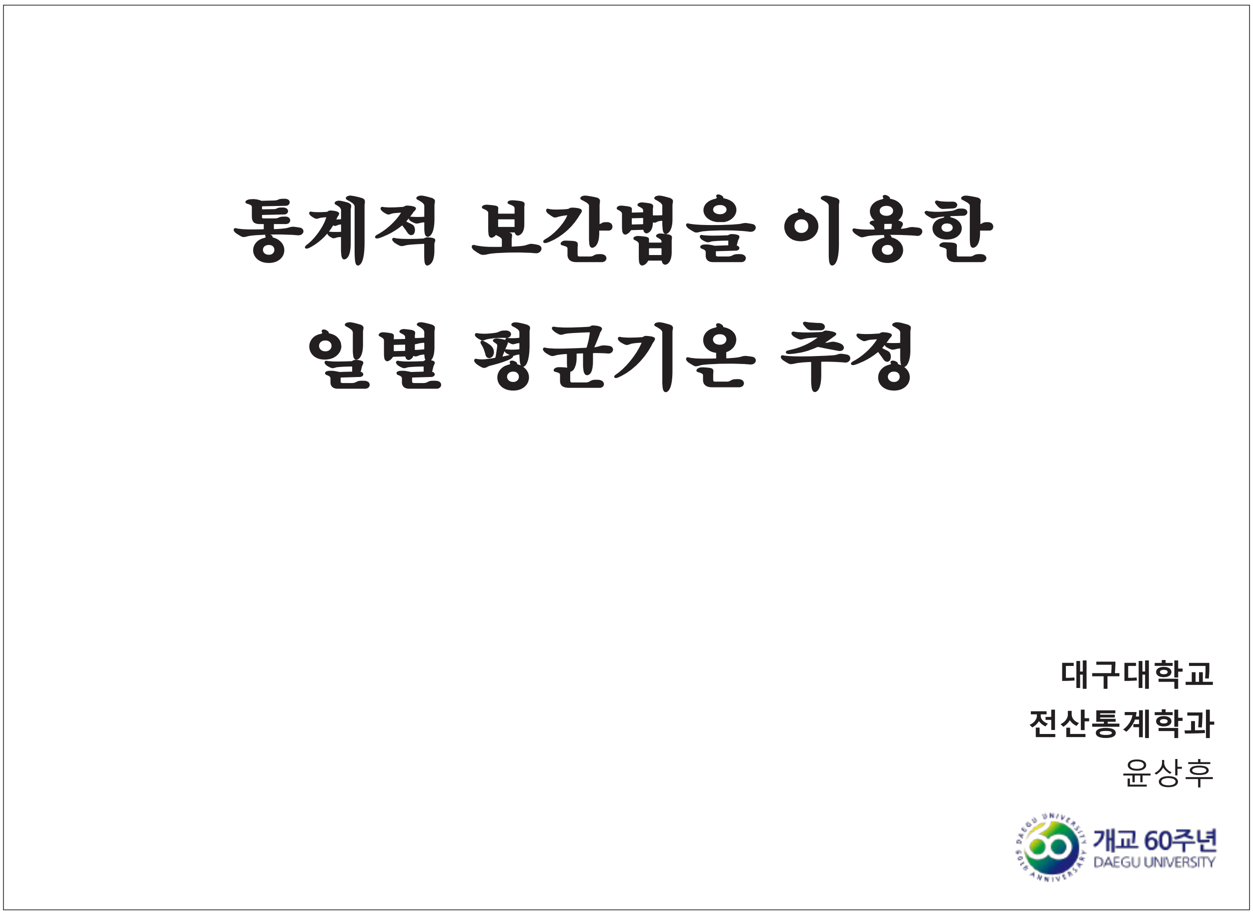
The image overlay viewed in GE



Conclusions

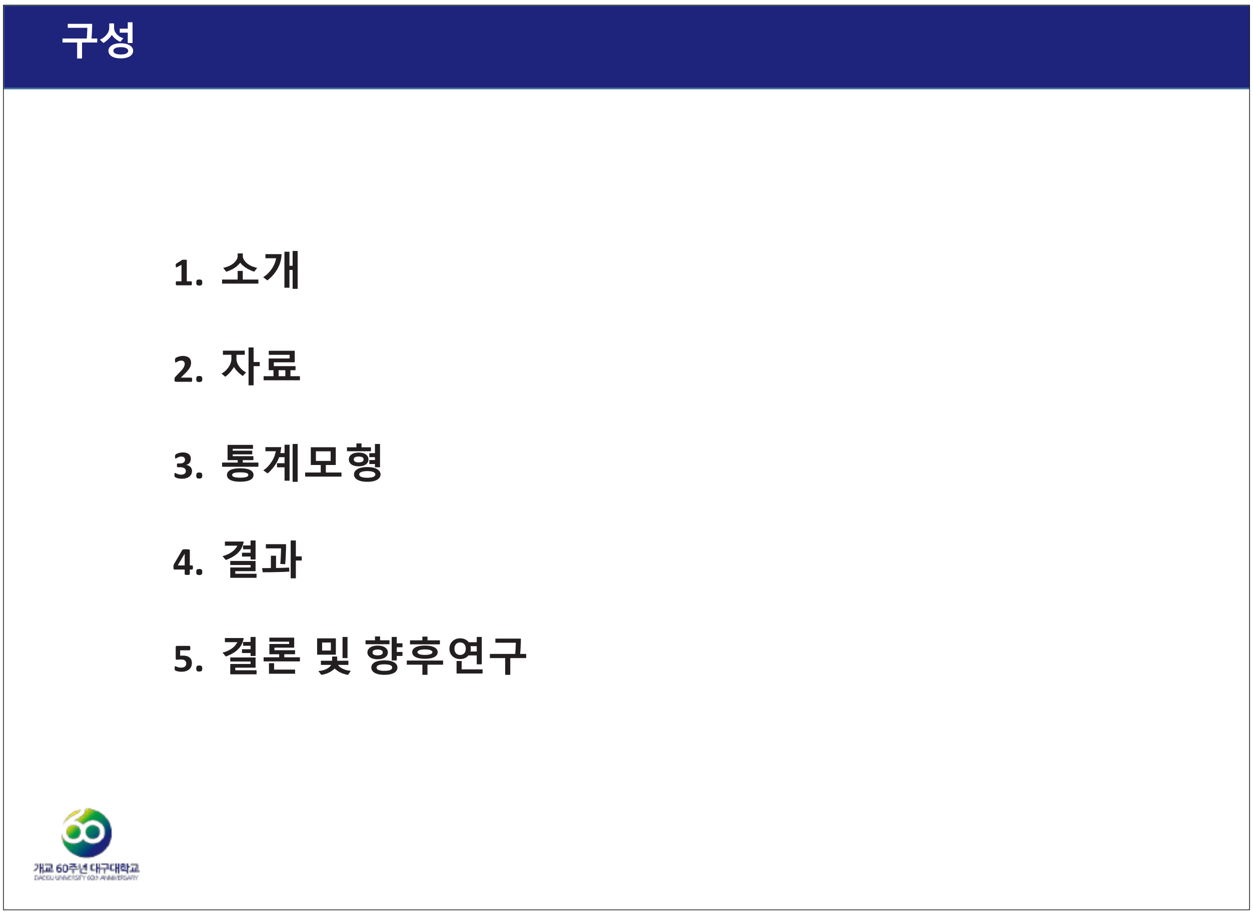
- The **sp** classes can be used (more or less) like data frames in many contexts
- The display methods on generated predictions and standard errors can be used directly, with spatial position being handled within the **sp** class objects
- Generating output for interfacing with other software is a bit 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

example of korean daily mean data



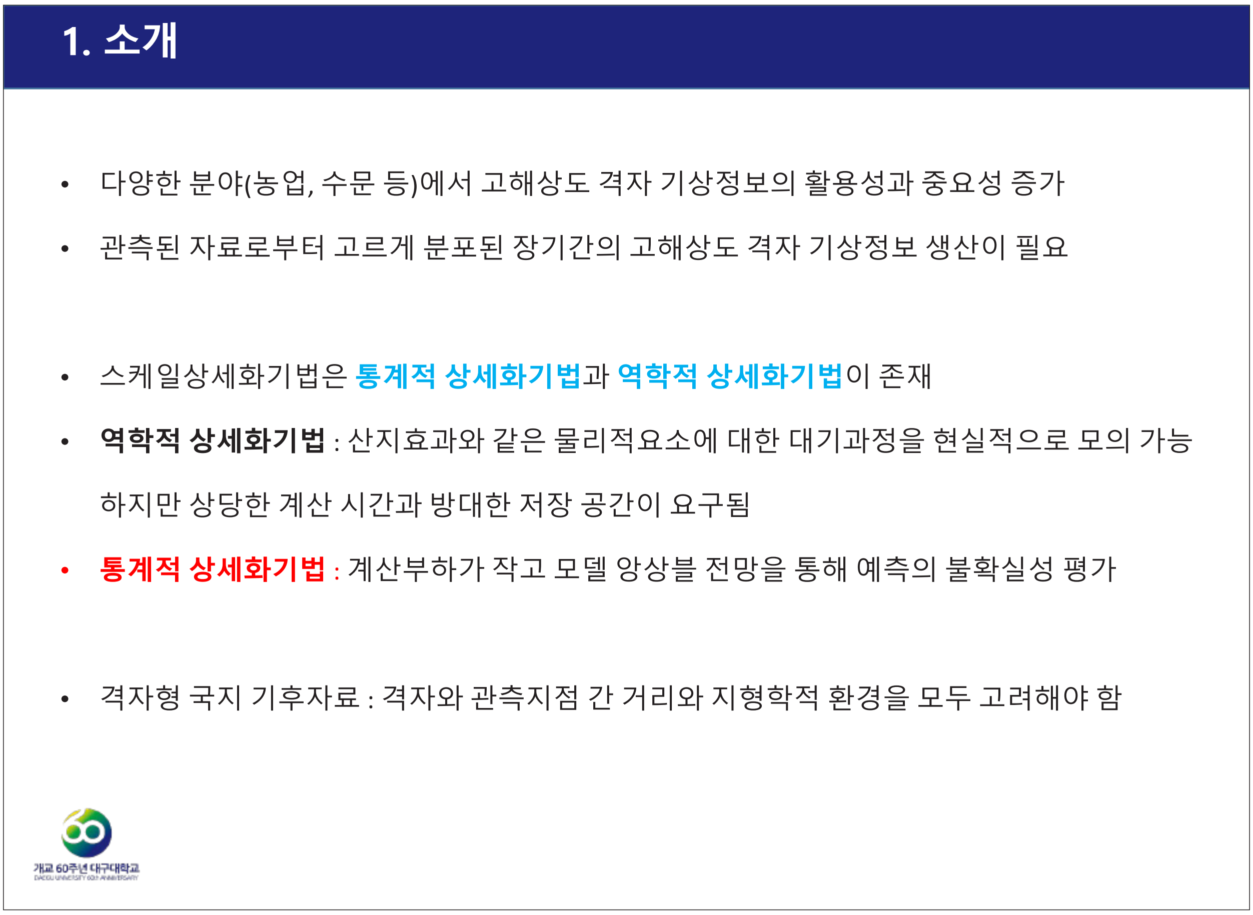
통계적 보간법을 이용한 일별 평균기온 추정

대구대학교
전산통계학과
윤상후



구성

1. 소개
2. 자료
3. 통계모형
4. 결과
5. 결론 및 향후연구

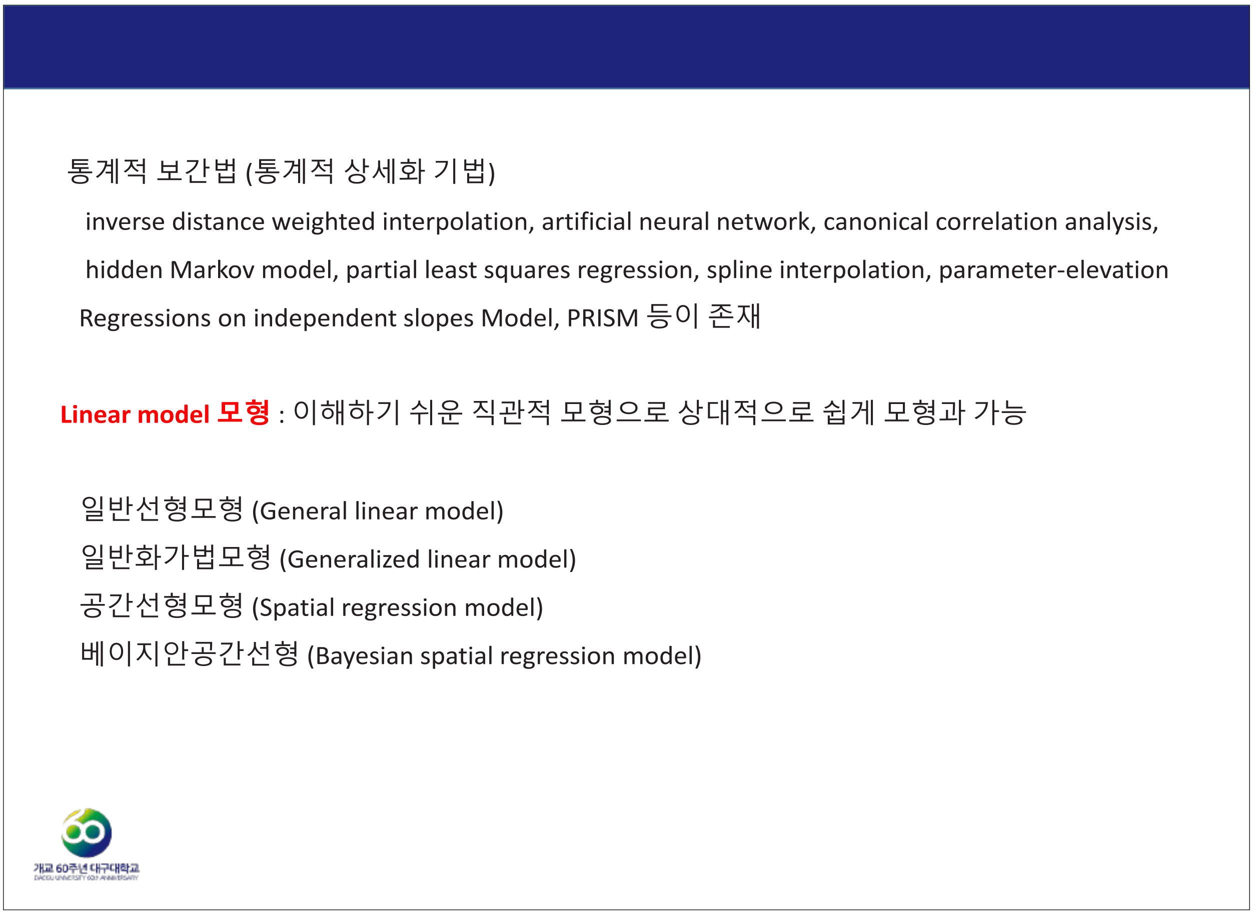


1. 소개

- 다양한 분야(농업, 수문 등)에서 고해상도 격자 기상정보의 활용성과 중요성 증가
- 관측된 자료로부터 고르게 분포된 장기간의 고해상도 격자 기상정보 생산이 필요

- 스케일상세화기법은 **통계적 상세화기법**과 **역학적 상세화기법**이 존재
- **역학적 상세화기법** : 산지효과와 같은 물리적요소에 대한 대기과정을 현실적으로 모의 가능하지만 상당한 계산 시간과 방대한 저장 공간이 요구됨
- **통계적 상세화기법** : 계산부하가 작고 모델 앙상블 전망을 통해 예측의 불확실성 평가

- 격자형 국지 기후자료 : 격자와 관측지점 간 거리와 지형학적 환경을 모두 고려해야 함



통계적 보간법 (통계적 상세화 기법)

inverse distance weighted interpolation, artificial neural network, canonical correlation analysis, hidden Markov model, partial least squares regression, spline interpolation, parameter-elevation Regressions on independent slopes Model, PRISM 등이 존재

Linear model 모형 : 이해하기 쉬운 직관적 모형으로 상대적으로 쉽게 모형과 가능

일반선형모형 (General linear model)
일반화가법모형 (Generalized linear model)
공간선형모형 (Spatial regression model)
베이지안공간선형 (Bayesian spatial regression model)

2. 자료

기간 : 2003년~2012년 (10년)

종속변수 : 1월 평균기온

독립변수 : **위도, 경도, 고도**

모델적합자료 : 종간기상관측소 (ASOS, 60)

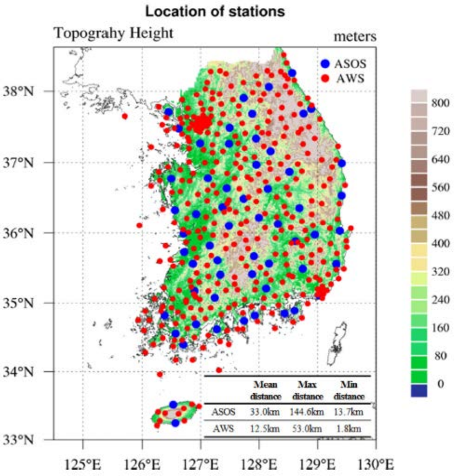
기압, 기온, 풍향, 풍속, 습도, 강수량, 시정, 구름 등 20여개 요소

검증자료 : 자동기상관측지점(AWS, 352)

풍향, 풍속, 기온, 강수량, 강수유무, 5개 요소

ASOS : 1973년부터 60개 지점에서 자료 수집

AWS : 1990년 후반부터 구축



	Mean distance	Max distance	Min distance
ASOS	33.0km	144.6km	13.7km
AWS	12.5km	53.0km	1.8km

3. 통계모형

독립변수 : 위도, 경도, 해발

General Linear Model(GLM) : OLS estimation

$$y(s) = X^t(s)\beta + \epsilon(s), \quad Var(\epsilon(s)) = I\sigma^2.$$

β : unknown parameters corresponding to each explanatory variables

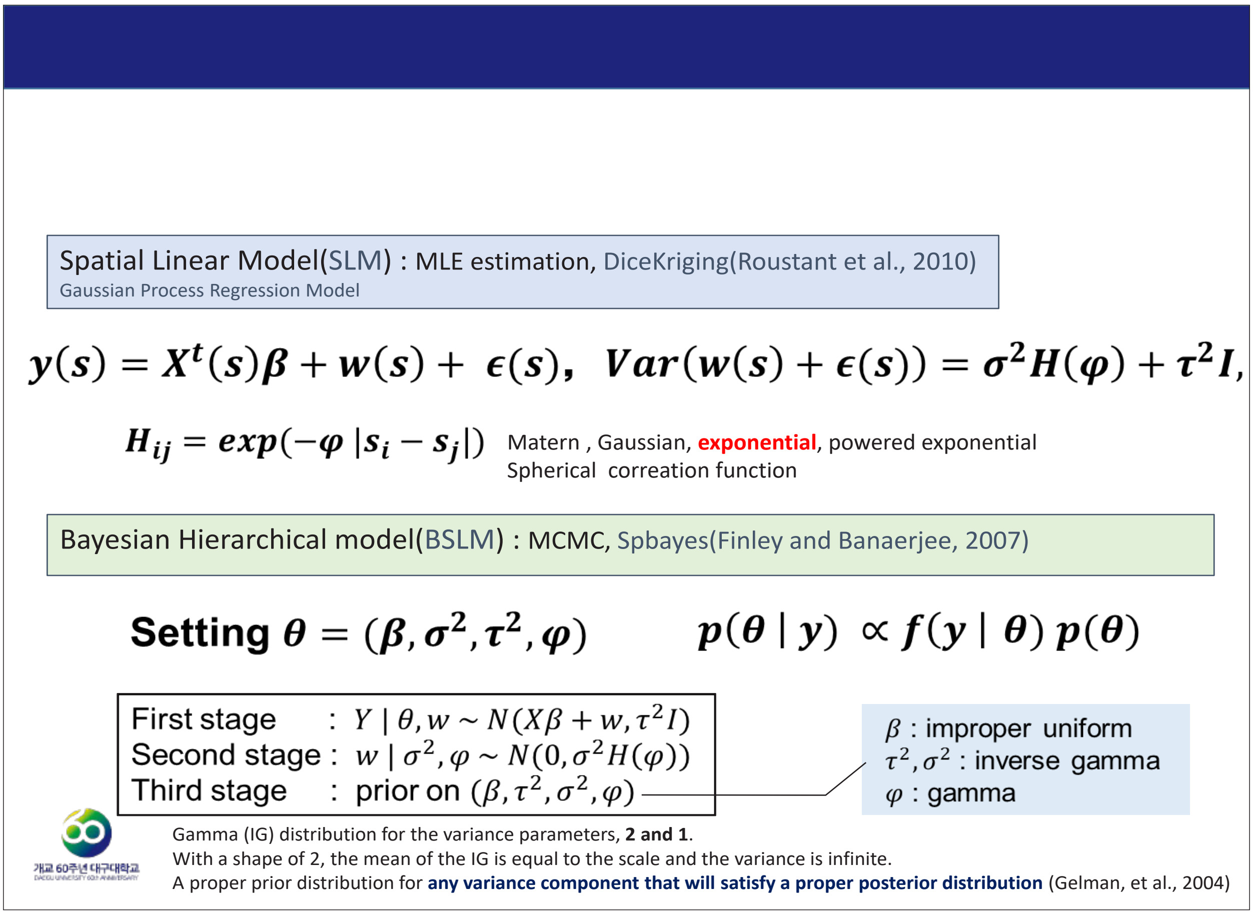
Generalized Additive Model(GAM): REML estimation, mgcv(Wood, 2012)

$$y(s) = f\big(X(s)\big) + \epsilon(s).$$ 지수분포족 (정규분포, 지수분포, 감마분포 등)

f : unspecified functions(Hastie and Tibshirani, 1990; Wood, 2006)

Suitable function : Regression spline (or Spline smoothing) Algorithm

Solution : 매듭(knot)을 Natural cubic spline



Spatial Linear Model(SLM) : MLE estimation, DiceKriging(Roustant et al., 2010)
Gaussian Process Regression Model

$$y(s) = X^t(s)\beta + w(s) + \epsilon(s), \quad Var(w(s)+\epsilon(s)) = \sigma^2 H(\varphi) + \tau^2 I,$$

$$H_{ij} = exp(-\varphi\,|s_i - s_j|)$$ Matern , Gaussian, **exponential**, powered exponential
Spherical correation function

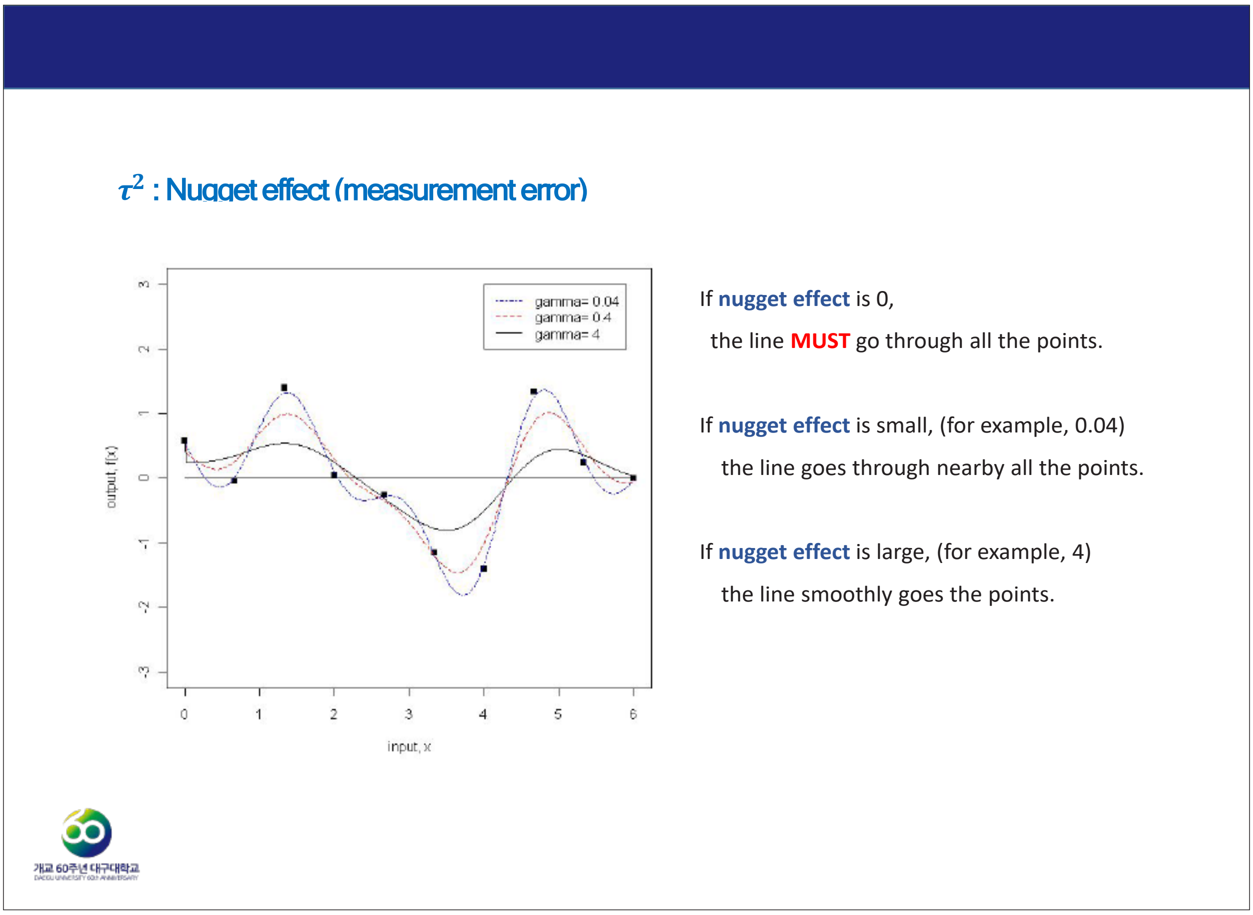
Bayesian Hierarchical model(BSLM) : MCMC, Spbayes(Finley and Banaerjee, 2007)

Setting $\theta = (\beta, \sigma^2, \tau^2, \varphi)$ $\qquad p(\theta \mid y) \propto f(y \mid \theta)\, p(\theta)$

First stage : $Y \mid \theta, w \sim N(X\beta + w, \tau^2 I)$
Second stage : $w \mid \sigma^2, \varphi \sim N(0, \sigma^2 H(\varphi))$
Third stage : prior on $(\beta, \tau^2, \sigma^2, \varphi)$

β : improper uniform
τ^2, σ^2 : inverse gamma
φ : gamma

Gamma (IG) distribution for the variance parameters, **2 and 1**.
With a shape of 2, the mean of the IG is equal to the scale and the variance is infinite.
A proper prior distribution for **any variance component that will satisfy a proper posterior distribution** (Gelman, et al., 2004)

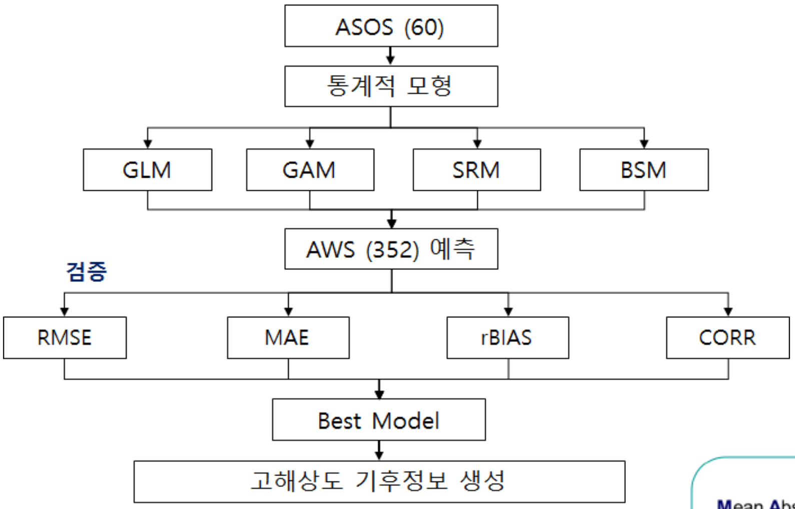


τ^2 : Nugget effect (measurement error)

If **nugget effect** is 0,
the line **MUST** go through all the points.

If **nugget effect** is small, (for example, 0.04)
the line goes through nearby all the points.

If **nugget effect** is large, (for example, 4)
the line smoothly goes the points.



ASOS (60)

통계적 모형

GLM

GAM

SRM

BSM

AWS (352) 예측

검증

RMSE

MAE

rBIAS

CORR

Best Model

고해상도 기후정보 생성

ASOS(60) + AWS (352) = 412 sites

Mean **A**bsolute **E**rror (MAE) = $\sqrt{\frac{1}{N}\sum_{i=1}^{N} |\,\hat{y}_i - y_i|}$

Root **M**ean **S**quare **E**rror (RMSE) = $\sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}$

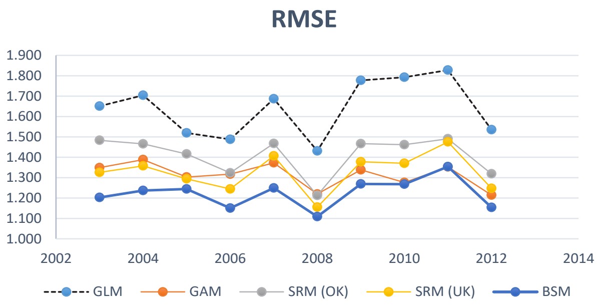
rBias = $\sum_{i=1}^{N}\frac{(y_i-\hat{y}_i)}{y_i}$

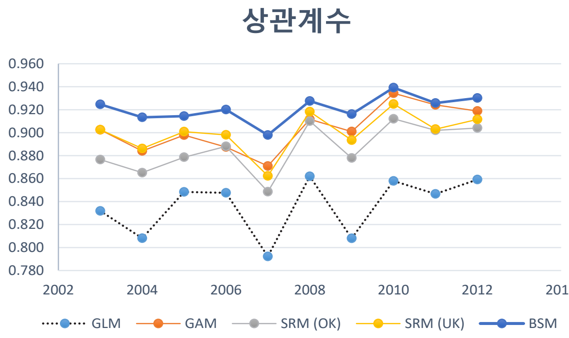
4. 결과

Year	GLM			GAM			SRM (OK)			SRM (UK)			BSM		
	RMSE	rBIAS	Corr	RMSE	rBIAS	Corr	RMSE	rBIAS	Corr	RMSE	rBIAS	Corr	RMSE	rBIAS	Corr
2003	1.651	0.616	0.832	1.349	0.838	0.903	1.483	0.503	0.877	1.326	0.608	0.903	**1.203**	0.548	**0.925**
2004	1.704	-0.027	0.808	1.388	0.122	0.884	1.466	0.046	0.865	1.358	0.026	0.886	**1.237**	0.069	**0.913**
2005	1.519	-0.101	0.848	1.303	-0.176	0.898	1.416	-0.329	0.879	1.294	-0.238	0.901	**1.244**	-0.166	**0.914**
2006	1.488	2.537	0.848	1.317	3.718	0.887	1.323	4.127	0.888	1.244	2.228	0.898	**1.150**	3.181	**0.920**
2007	1.687	0.059	0.792	1.373	-0.152	0.871	1.468	-0.024	0.849	1.407	-0.140	0.862	**1.249**	-0.058	**0.898**
2008	1.431	-0.181	0.862	1.220	-0.220	0.912	1.213	-0.194	0.910	1.155	-0.192	0.918	**1.109**	-0.231	**0.928**
2009	1.777	-0.046	0.808	1.338	-0.092	0.901	1.467	-0.078	0.878	1.378	-0.060	0.893	**1.269**	-0.109	**0.916**
2010	1.793	-4.803	0.858	1.276	-4.874	0.934	1.462	-5.647	0.912	1.370	-4.617	0.925	**1.268**	-4.121	**0.939**
2011	1.828	-0.038	0.847	1.353	-0.064	0.924	1.491	-0.039	0.902	1.475	-0.029	0.903	**1.354**	-0.063	**0.926**
2012	1.536	-0.085	0.859	1.214	-0.117	0.919	1.319	-0.126	0.904	1.247	-0.088	0.912	**1.154**	-0.116	**0.930**
Average	1.641	-0.207	0.836	1.313	-0.102	0.903	1.411	-0.176	0.886	1.325	-0.250	0.900	**1.224**	-0.106	**0.921**

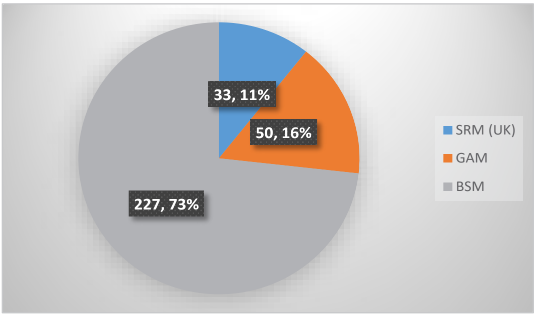
RMSE : BSM (1.224) < GAM (1.313) < SRM (UK, 1.325) < SRM (OK, 1.411) < GLM (1.641)

Corr : GLM (0.836) < SRM (OK, 0.886) < SRM (UK, 0.900) < GAM (0.903) < **BSM (0.921)**





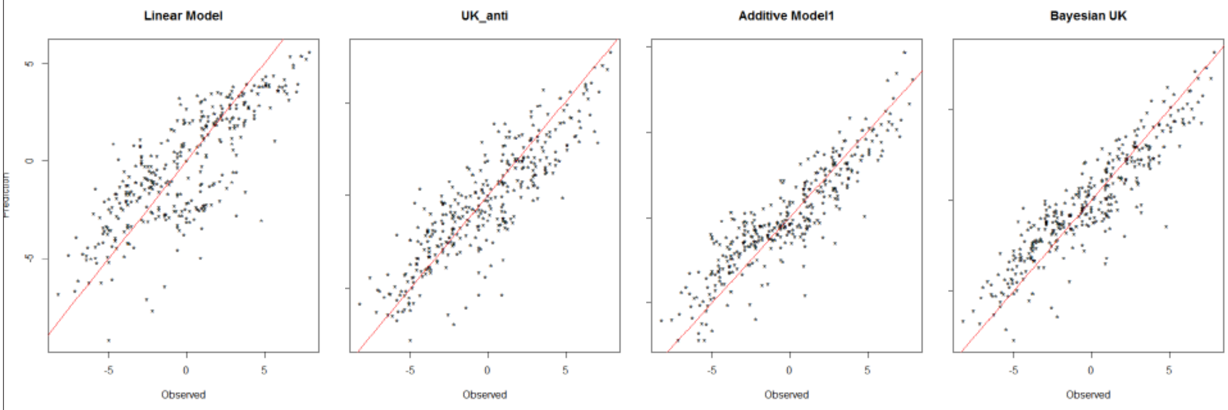
BSM 모형의 RMSE가 가장 낮고, 상관계수가 높음



총 310일 중 최적모형 (RMSE 기준)

SRM (UK)	GAM	BSM
33	50	227

일단위 분석 : 2007년 1월 15일 (예)



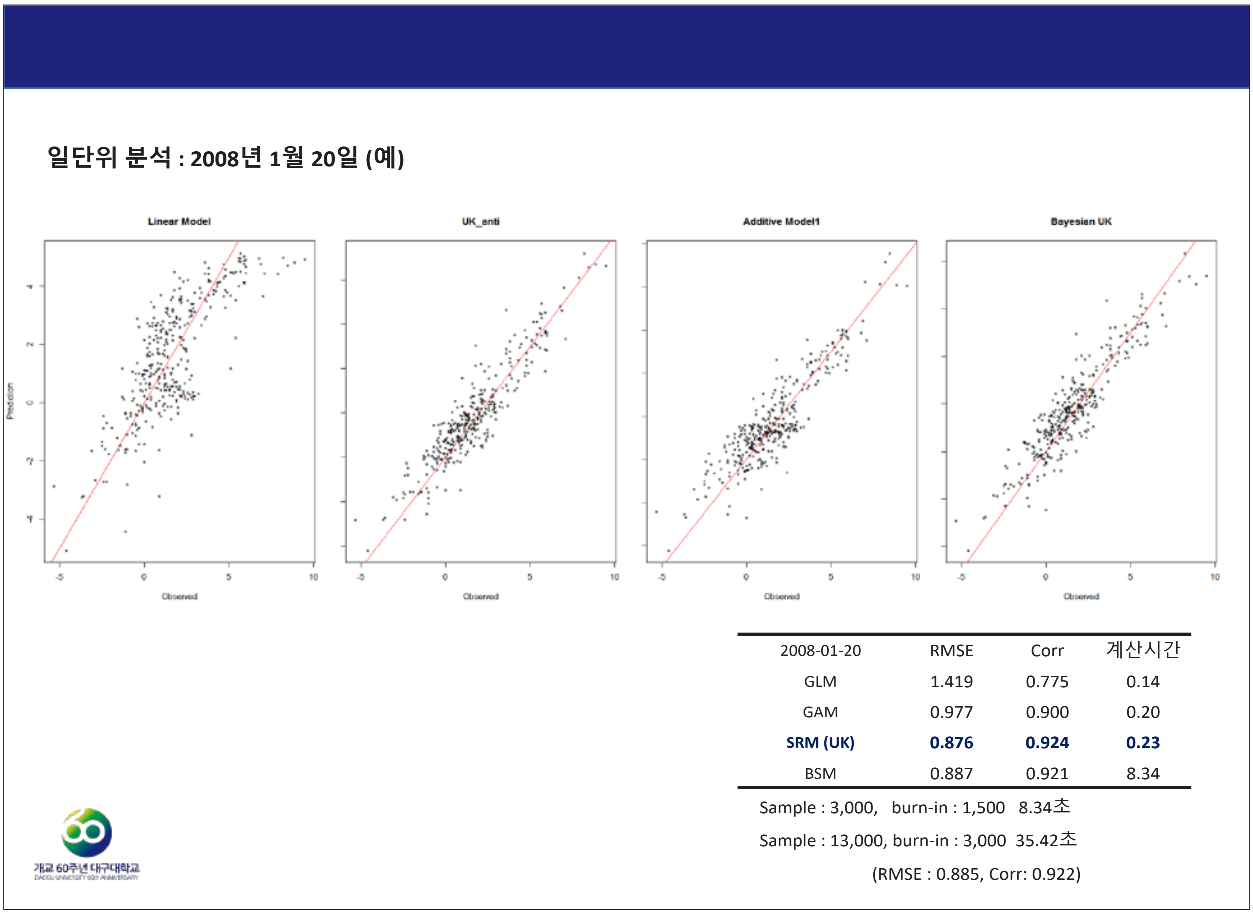
<table>
<tr><th rowspan="2">2007-01-15</th><th colspan="2">GLM</th><th colspan="2">GAM</th><th colspan="2">SRM (UK)</th><th colspan="2">BSM</th></tr>
<tr><th>est.</th><th>se</th><th>est.</th><th>se</th><th>est.</th><th>se</th><th>est.</th><th>se</th></tr>
<tr><td>(Intercept)</td><td>-35.187</td><td>39.331</td><td>1.204</td><td>0.233</td><td>-59.722</td><td></td><td>-63.471</td><td>55.790</td></tr>
<tr><td>lat</td><td>-1.910</td><td>0.242</td><td></td><td></td><td>-2.139</td><td></td><td>-1.982</td><td>0.353</td></tr>
<tr><td>lon</td><td>0.825</td><td>0.313</td><td></td><td></td><td>1.085</td><td></td><td>1.072</td><td>0.452</td></tr>
<tr><td>ele</td><td>-0.009</td><td>0.002</td><td>-0.009</td><td>0.002</td><td>-0.008</td><td></td><td>-0.010</td><td>0.002</td></tr>
<tr><td>$\emptyset_{lon}$</td><td></td><td></td><td></td><td></td><td>3.397</td><td></td><td rowspan="2">2.593</td><td rowspan="2">0.394</td></tr>
<tr><td>$\emptyset_{lat}$</td><td></td><td></td><td></td><td></td><td>1.302</td><td></td></tr>
<tr><td>$\emptyset_{ele}$</td><td></td><td></td><td></td><td></td><td>85.700</td><td></td><td></td><td></td></tr>
<tr><td>smooth (lon, lat)</td><td></td><td></td><td>15.74</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>σ^2</td><td>3.778</td><td>0.253</td><td>1.146</td><td>0.139</td><td>5.357</td><td></td><td>3.397</td><td>0.838</td></tr>
<tr><td>τ^2</td><td></td><td></td><td></td><td></td><td>0</td><td></td><td>0.366</td><td>0.291</td></tr>
</table>

2007-01-15	RMSE	Corr	계산시간
GLM	2.151	0.787	0.14
GAM	1.563	0.897	0.20
SRM (UK)	1.746	0.866	0.23
BSM	**1.455**	**0.910**	**8.34**

Sample : 3,000, burn-in : 1,500 8.34초

Sample : 13,000, burn-in : 3,000 35.42초

(RMSE : 1.452, Corr: 0.911)



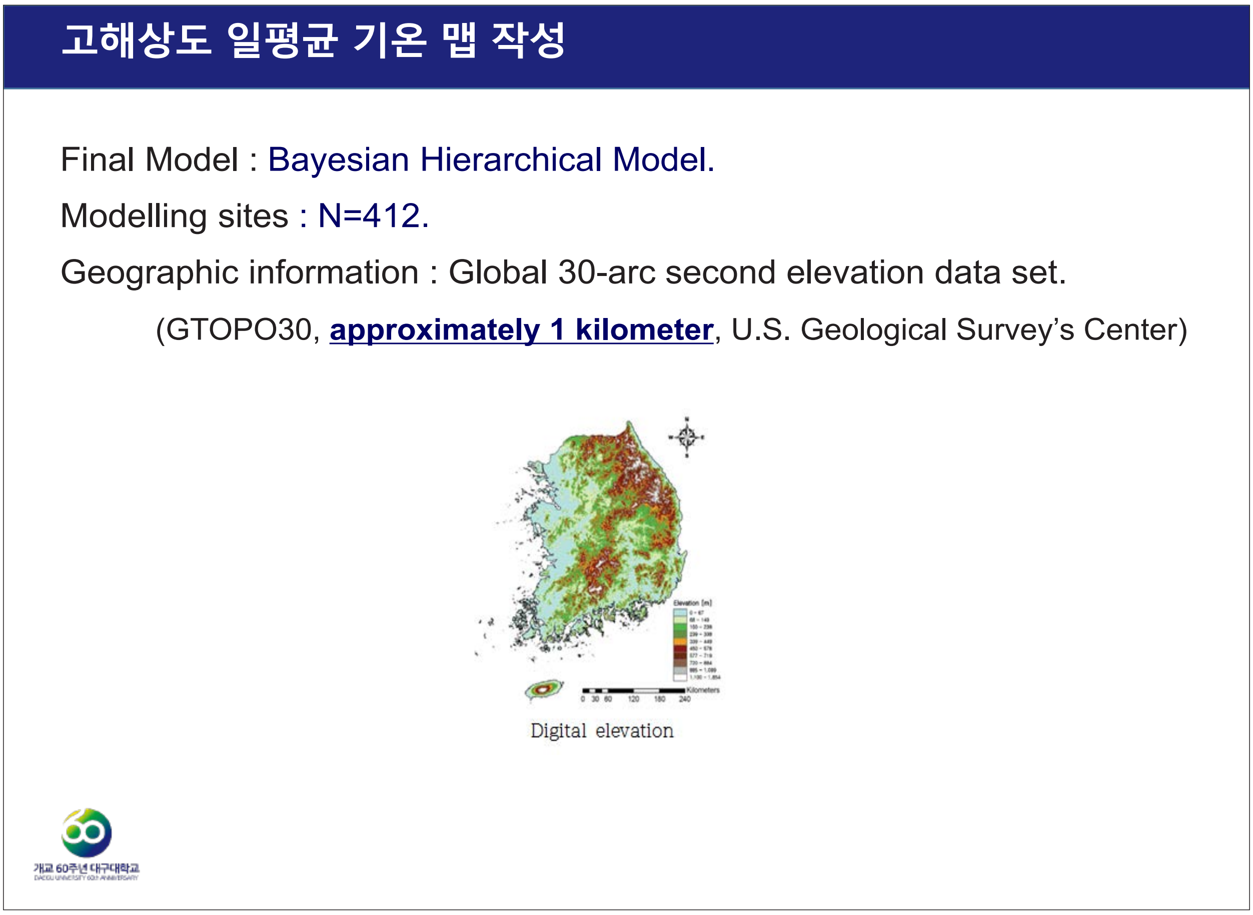
일단위 분석 : 2008년 1월 20일 (예)

2008-01-20	RMSE	Corr	계산시간
GLM	1.419	0.775	0.14
GAM	0.977	0.900	0.20
SRM (UK)	**0.876**	**0.924**	**0.23**
BSM	0.887	0.921	8.34

Sample : 3,000, burn-in : 1,500 8.34초

Sample : 13,000, burn-in : 3,000 35.42초

(RMSE : 0.885, Corr: 0.922)



고해상도 일평균 기온 맵 작성

Final Model : Bayesian Hierarchical Model.

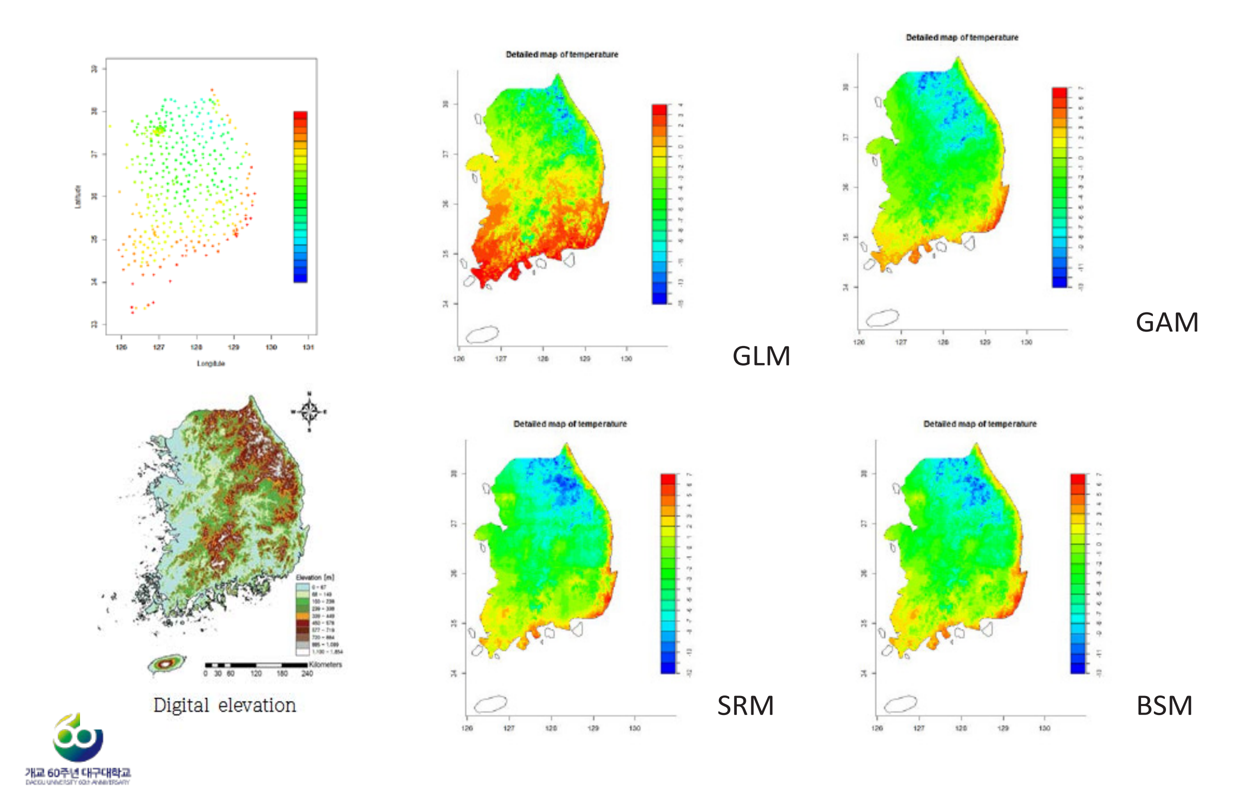
Modelling sites : N=412.

Geographic information : Global 30-arc second elevation data set.

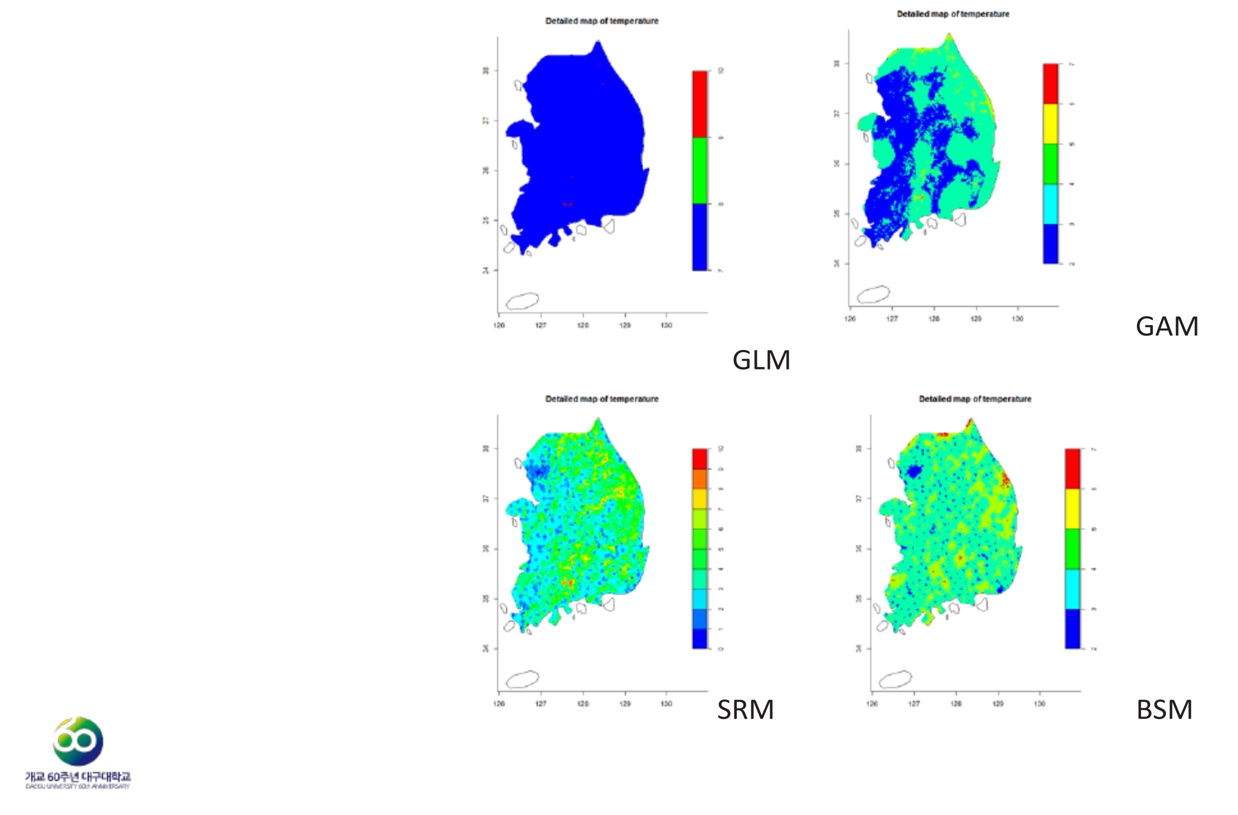
(GTOPO30, **approximately 1 kilometer**, U.S. Geological Survey's Center)

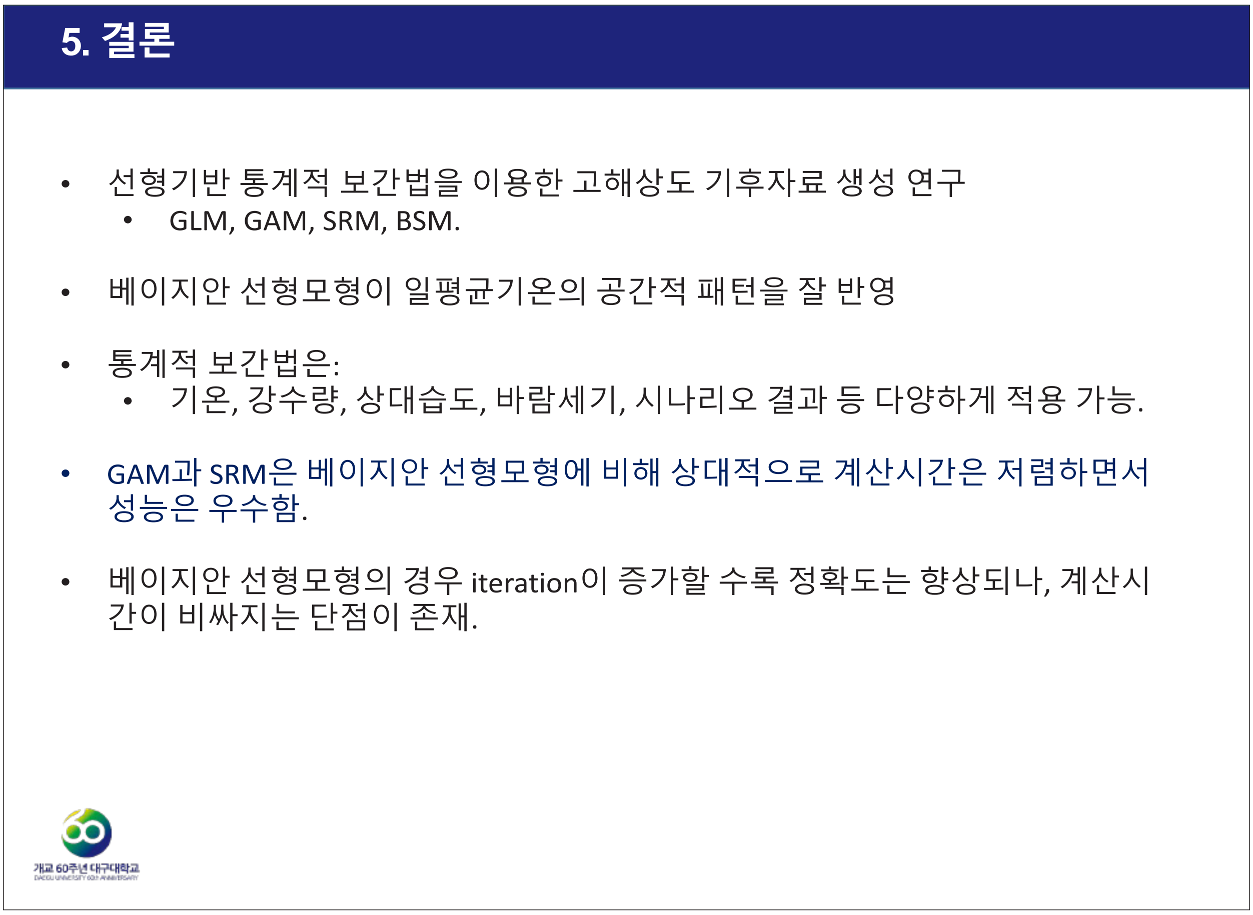
Digital elevation

1km 해상도 평균기온 그림(2007.1.15)


Latitude
Longitude
Detailed map of temperature
GLM
Detailed map of temperature
GAM
Elevation (m)
Kilometers
Digital elevation
Detailed map of temperature
SRM
Detailed map of temperature
BSM
개교 60주년 대구대학교
DAEGU UNIVERSITY 60th ANNIVERSARY

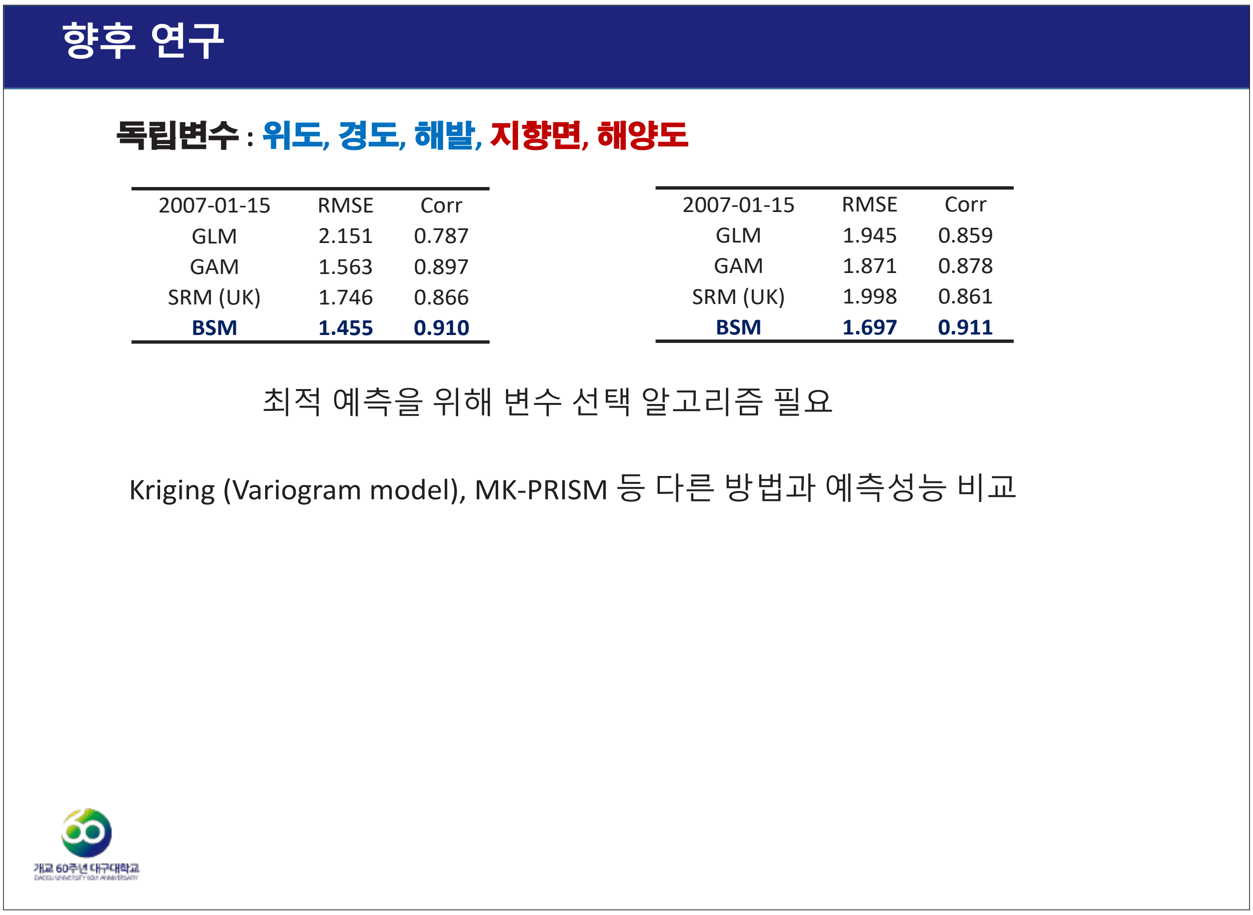
1km 해상도 95%신뢰구간


Detailed map of temperature
GLM
Detailed map of temperature
GAM
Detailed map of temperature
SRM
Detailed map of temperature
BSM
개교 60주년 대구대학교
DAEGU UNIVERSITY 60th ANNIVERSARY



5. 결론

- 선형기반 통계적 보간법을 이용한 고해상도 기후자료 생성 연구
  - GLM, GAM, SRM, BSM.
- 베이지안 선형모형이 일평균기온의 공간적 패턴을 잘 반영
- 통계적 보간법은:
  - 기온, 강수량, 상대습도, 바람세기, 시나리오 결과 등 다양하게 적용 가능.
- GAM과 SRM은 베이지안 선형모형에 비해 상대적으로 계산시간은 저렴하면서 성능은 우수함.
- 베이지안 선형모형의 경우 iteration이 증가할 수록 정확도는 향상되나, 계산시간이 비싸지는 단점이 존재.



향후 연구

독립변수 : 위도, 경도, 해발, 지향면, 해양도

2007-01-15	RMSE	Corr
GLM	2.151	0.787
GAM	1.563	0.897
SRM (UK)	1.746	0.866
BSM	**1.455**	**0.910**

2007-01-15	RMSE	Corr
GLM	1.945	0.859
GAM	1.871	0.878
SRM (UK)	1.998	0.861
BSM	**1.697**	**0.911**

최적 예측을 위해 변수 선택 알고리즘 필요

Kriging (Variogram model), MK-PRISM 등 다른 방법과 예측성능 비교

Thank You