Spatial analysis with the R Project for Statistical Computing

D G Rossiter
Cornell University
ISRIC-World Soil Information
Nanjing Normal University 南京师范大学地理学学院

January 14, 2019

Topics

1. Spatial analysis in R
2. The `sp` package: spatial classes
3. External file formats
4. Interfaces with other spatial analysis tools; Coördinate Reference Systems with `rgdal`
5. The `gstat` package: geostatistical modelling, prediction and simulation

Spatial analysis in R

1. Spatial data

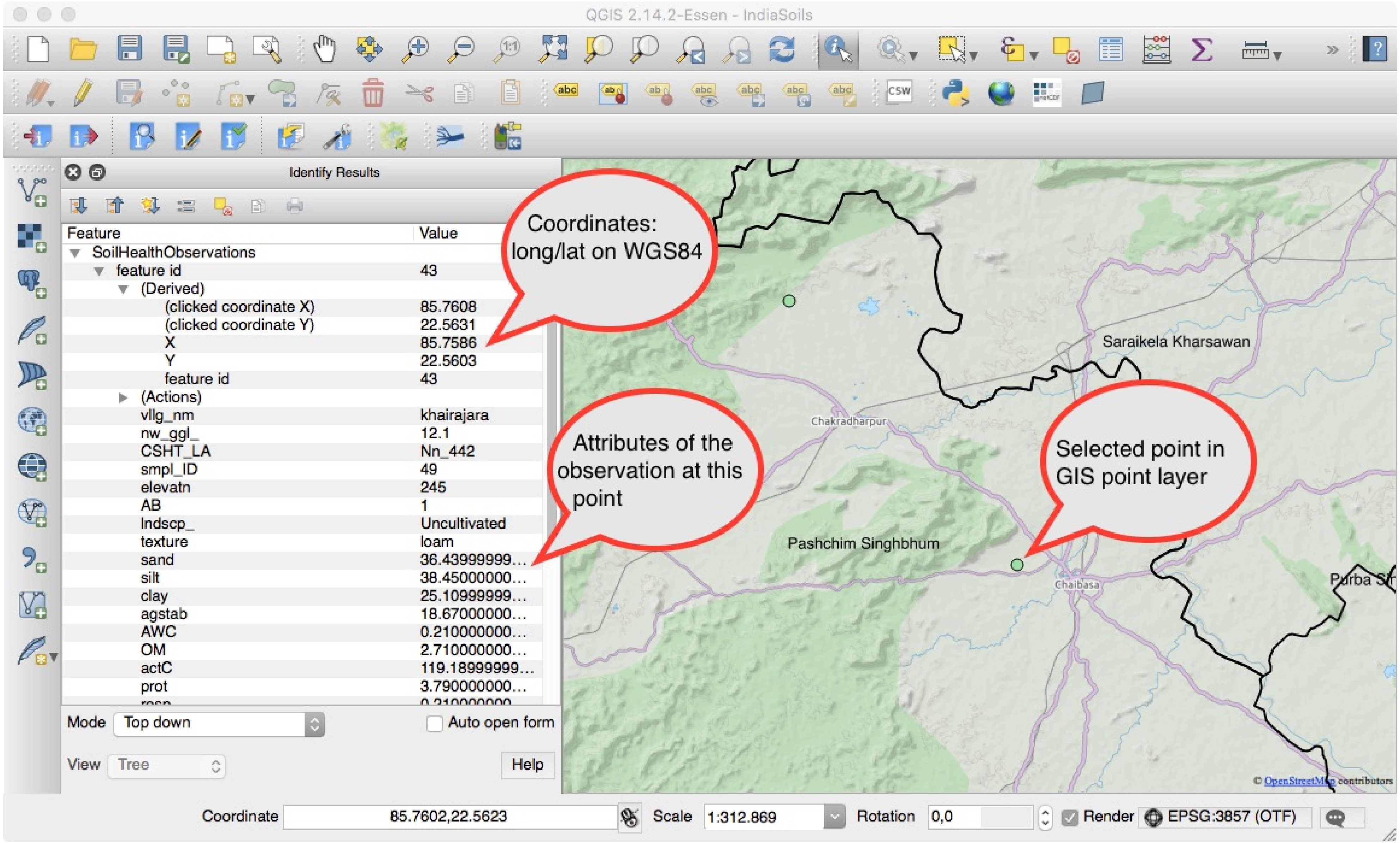
2. R approaches to spatial data

Spatial data – definition

- Spatial data have **coördinates**, i.e. known locations
  - The location itself is **spatial information**
  - Often have **attributes** – other information about the spatial object

- Coördinates are **absolute locations** in one, two or three dimensions ...
  - ... in some defined **coördinate reference system** (CRS)
  - If referring to real objects referenced to the Earth, must have a defined **projection** and **datum**

Spatial objects: location and attributes



Types of spatial objects

Points 0-dimensional

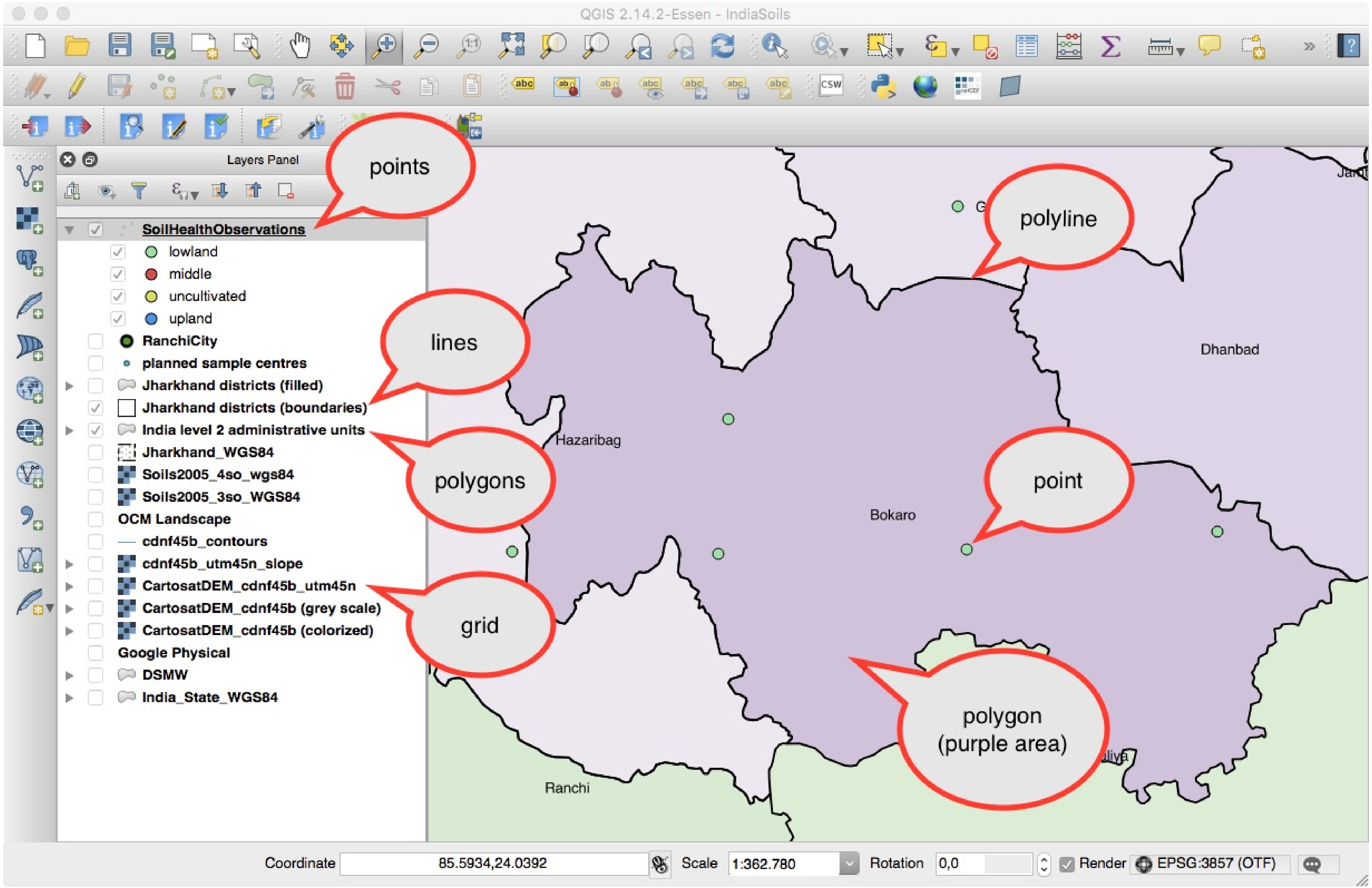
Lines 1-dimensional, defined by points, linked as **polylines**

Polygons area enclosed by connected polylines

Curves defined by control points and piecewise polynomials of the coördinates

Grid cells ; “rasters”; (usually regular) tesselation of space

Types of spatial objects



Spatial data – what is special?

- Points are implicitly related by **distance** and **direction** of **separation**
- Polygons are implicitly related by **adjacency**, **containment**, distance between centroids
- Polygons have **shape**, **perimeter**, **compactness** ...
- Such data requires **different analysis** than non-spatial data
  - e.g. can't assume independence of observations (**spatial dependence/correlation**)
- All objects with the same CRS are implicitly related, can be **overlayed**
- Some analysis is **purely spatial** (e.g., point-pattern analysis)
- They need special **data structures** which recognize the special status of coördinates

R approaches

- No native S classes for these
- S is **extensible** with new classes, methods and packages
- Add-in package which defines **spatial classes and methods**: `sp` (Bivand, Pebesma)
- Add-in packages to manipulate external files: `rgdal` (Geographic Data Abstraction Language), `sf` (Simple Features)
- Add-in packages for **spatial analysis**:
  - `spatial` (Ripley)
  - `geoR`, `geoRglm` (Ribeiro & Diggle Model-based geostatistics)
  - `gstat` (Pebesma)
  - `spatstat` (Baddeley & Turner): point patterns
  - `RandomFields` (Schlather)
  - `circular`: directional statistics

Interface to GIS

- Add-in packages:
  - `rgdal`: interface to Geospatial Data Abstraction Library (GDAL)
    * Coördinate Reference Systems, transformations
    * Read/write to foreign files
  - `sf`: interface to Simple Features specification
  - `raster` read, write, manipulate grid ("raster") data structures
  - `maptools`: interface to external spatial data structures e.g. shapefiles

(See later topic)

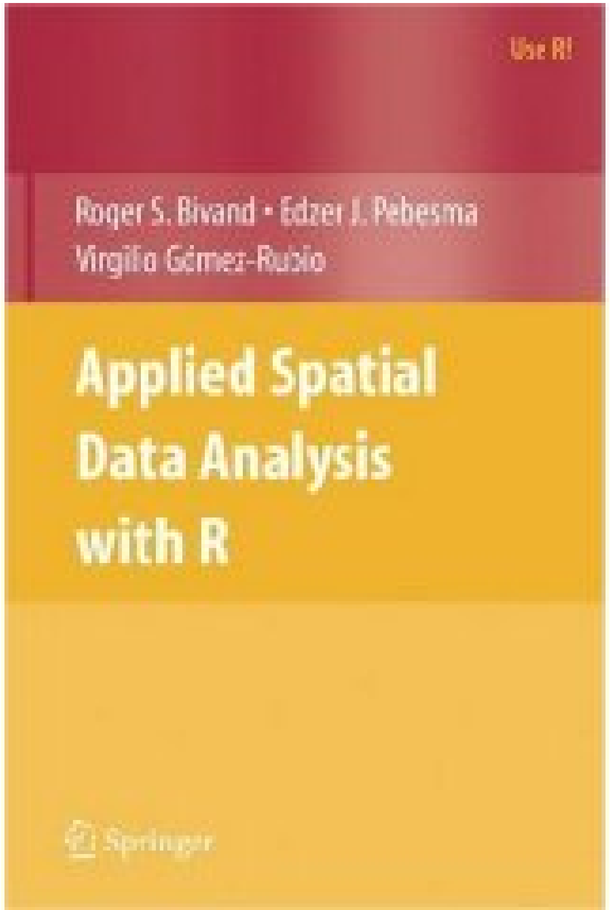
CRAN Task View: Analysis of Spatial Data

`http://cran.r-project.org/web/views/Spatial.html` explains what packages are available for:

- Classes for spatial data
- Handling spatial data
- Reading and writing spatial data
- Point pattern analysis
- Geostatistics
- Disease mapping and areal data analysis
- Spatial regression
- Ecological analysis

Advanced textbook

Bivand, R. S., Pebesma, E. J., & Gómez-Rubio, V. (2013). *Applied Spatial Data Analysis with R*, 2nd ed.: Springer. `http://www.asdar-book.org/`; ISBN 978-1-4614-7617-7; 978-1-4614-7618-4 (e-book)



The sp package

- This package provides **classes** and **methods** for dealing with **spatial data** in S

- It does not provide methods for spatial analysis

- Other packages can use these classes for spatial analysis

sp Spatial data structures

- Spatial **data structures** (S4 classes):
  - **points**
  - **lines**
  - **polygons**
  - **grids** (rasters)

- These may all have **attributes** (dataframes)

- S4 class names like `SpatialPointsDataFrame`

- Generic methods with appropriate behaviour for each class

Representation in sp – vector data structures – points

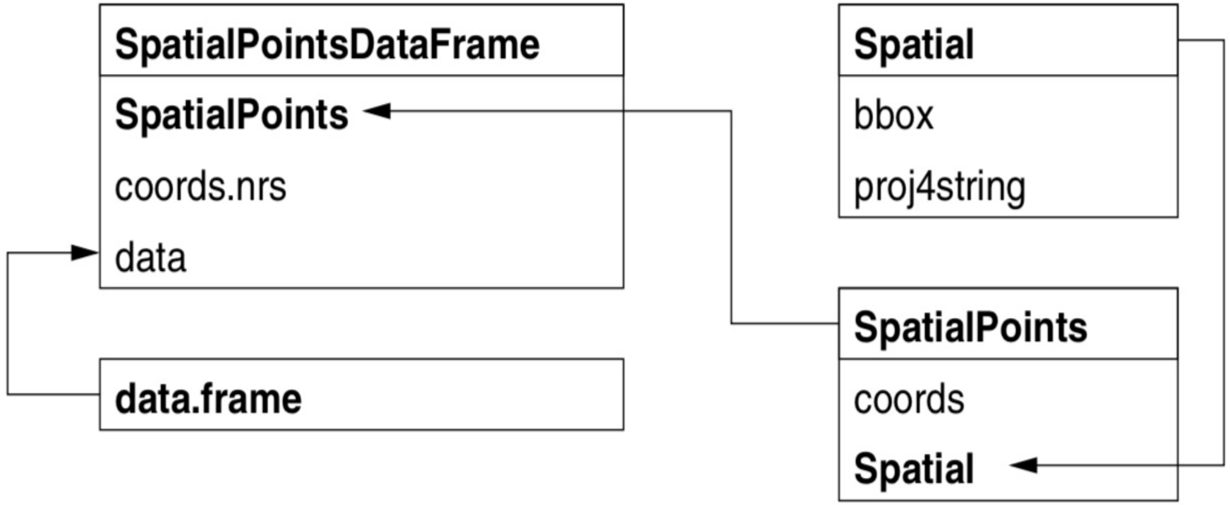


Fig. 2.2 Spatial points classes and their slots; *arrows* show subclass extensions

Source: Bivand, R. S., Pebesma, E. J., & Gómez-Rubio, V. (2008). Applied Spatial Data Analysis with R. Springer. (ASDAR)

Representation in sp – vector data structures – lines & polygons

40 2 Classes for Spatial Data in R

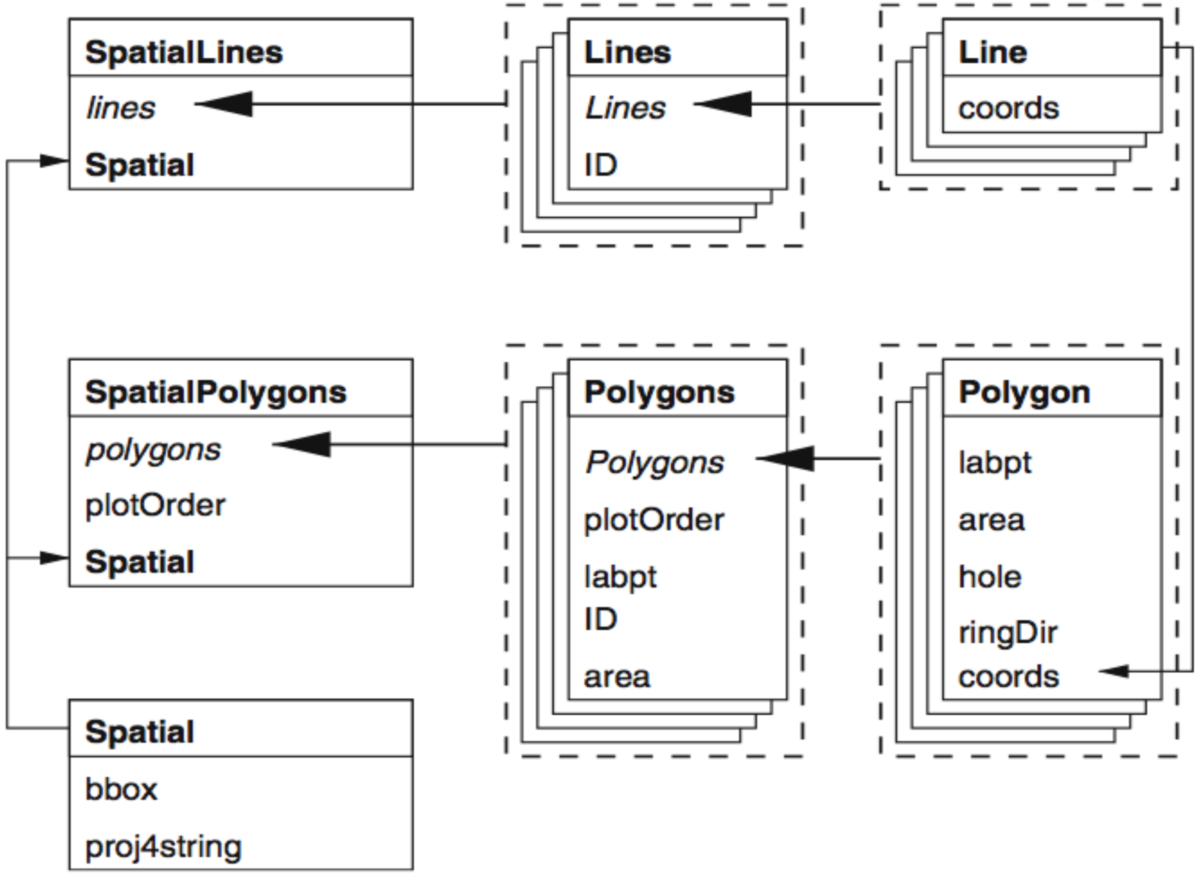


Fig. 2.4. SpatialLines and SpatialPolygons classes and slots; thin arrows show sub-class extensions, thick arrows the inclusion of lists of objects

Source: ASDAR

Representation in sp – raster data structures

52 2 Classes for Spatial Data in R

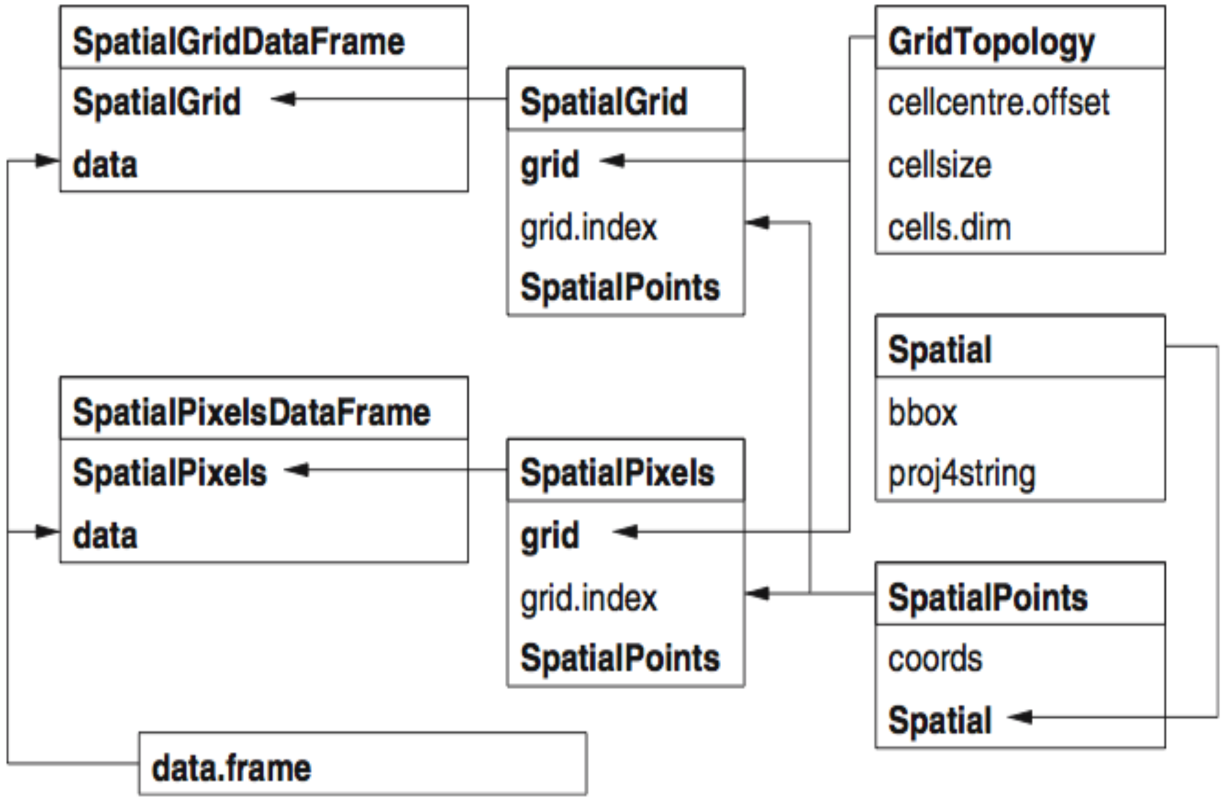


Fig. 2.8. `SpatialGrid` and `SpatialPixel` classes and their slots; arrows show subclass extensions

Source: ASDAR

sp Methods for handling spatial data

Some standard methods:

- `bbox`: bounding box: extreme coördinates enclosing object
- `dimensions`: number of spatial dimensions
- `coordinates`: set or extract coördinates
- `over`: combine two spatial layers of different type
  - e.g., retrieve the polygon or grid values on a set of points
  - e.g., retrieve the points or their attributes within (sets of) polygons
- `spsample`: point sampling schemes within a geographic context (grids or polygons)
- `spplot`: visualize spatial data

Converting data to sp classes

Simple rule: data is spatial if it has **coördinates**.

The most common way to assign coördinates is to use the `coordinates` method as the left-hand side of an assignment.

First we see how spatial data looks in a `data.frame` (base S class):

```
> library(gstat)   # meuse is an example dataset in this package
> data(meuse); str(meuse); spsample(meuse, 5, "random")

'data.frame':        155 obs. of  14 variables:
 $ x      : num  181072 181025 181165 181298 181307 ...
 $ y      : num  333611 333558 333537 333484 333330 ...
 $ cadmium: num  11.7 8.6 6.5 2.6 2.8 3 3.2 2.8 2.4 1.6 ...
 $ copper : num  85 81 68 81 48 61 31 29 37 24 ...

Error in function (classes, fdef, mtable)  :
        unable to find an inherited method for function "spsample",
        for signature "data.frame"
```

Fields `x` and `y` are coördinates but they have no special status in the `data.frame`. The method `spsample` expects a spatial object but does not find it.

Converting a data frame to spatial data

We explicitly identify the fields that are coördinates:

```
> coordinates(meuse) <- ~ x + y # or coordinates(meuse) <- c("x", "y")
> str(meuse)

Formal class 'SpatialPointsDataFrame' [package "sp"] with 5 slots
  ..@ data       :'data.frame':       155 obs. of  12 variables:
  .. ..$ cadmium: num [1:155] 11.7 8.6 6.5 2.6 2.8 3 3.2 2.8 2.4 1.6 ...
  .. ..$ copper : num [1:155] 85 81 68 81 48 61 31 29 37 24 ...
  ..@ coords.nrs : int [1:2] 1 2
  ..@ coords     : num [1:155, 1:2] 181072 181025 181165 181298 181307 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : NULL
  .. .. ..$ : chr [1:2] "x" "y"
  ..@ bbox       : num [1:2, 1:2] 178605 329714 181390 333611
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2] "x" "y"
  .. .. ..$ : chr [1:2] "min" "max"
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slots
  .. .. ..@ projargs: chr NA
```

The S4 class has **slots** (@) for coördinates, dimensions, data, bounding box, CRS

- Now the `spsample` method works

- It returns an object of class `SpatialPoints` (i.e. just the locations, no attributes)

```
> spsample(meuse, 5, "random")

SpatialPoints:
          x      y
[1,] 179246 330766
[2,] 180329 330055
[3,] 181280 329798
[4,] 180391 330627
[5,] 178725 330202
```

Assigning a CRS to a newly-converted object

When a `data.frame` is converted to an `sp` object, there is no way for the `coordinates` method to determine the CRS, because the `data.frame` only has a list of attributes for each object.

Therefore by default the `@proj4string` slot is set to `NA` = "not available".

If the analysis does not depend on knowing the CRS, it can be left as-is.

Otherwise, assign after conversion, with the `CRS` method, e.g.,

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

See details in next section.

External file formats

R can deal with geographic data in "all" file formats.

- vector
- gridded ("raster")

Vector data structure: ESRI shapefiles

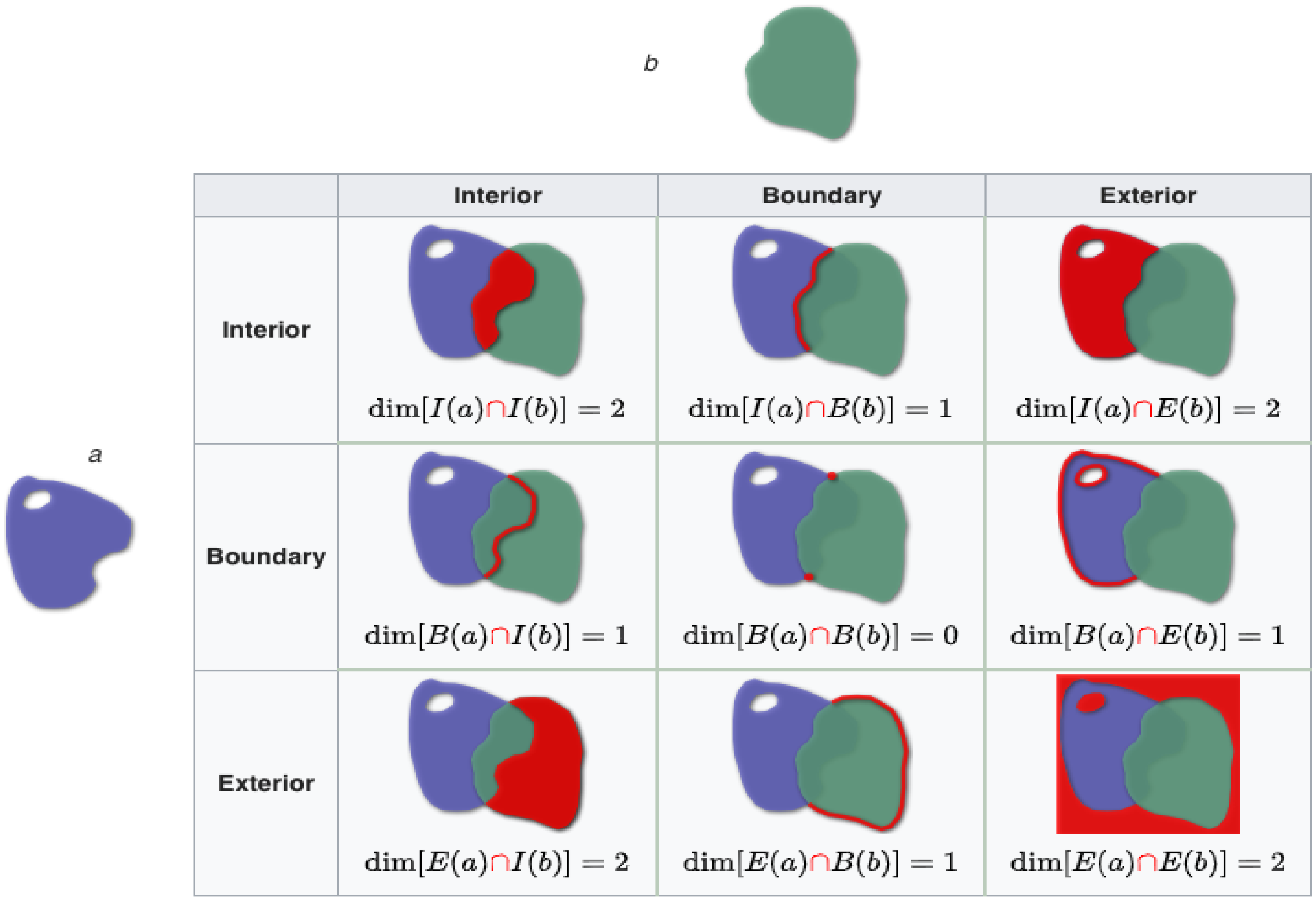
- Proprietory format, but reasonably well-documented[1]
- Geometry + attributes of 2D geometries
- Does not explictly store topology, must be built on-the-fly

[1] https://en.wikipedia.org/wiki/Shapefile

Vector data structure: Simple features

- open standard ISO 19125
- specifies a common **storage and access model** of geometries used by GIS
  - e.g., 2D point, line, polygon, multi-point, multi-lines; also 3D
- encoded as text, easy to interpret
- implemented within R as data tables with geometry
- supports the Dimensionally Extended nine-Intersection Model (DE-9IM), which specifies topological relations

Dimensionally Extended nine-Intersection Model (DE-9IM)



source: `https://en.wikipedia.org/wiki/DE-9IM`

Grid data structures

- Many standards: GeoTIFF, NetCDF (Network Common Data Form, from UCAR (University Corporation for Atmospheric Research) . . .

- represented in R as (sparse) matrices, sometimes implicit by the grid topology (e.g., `sp` package)

- `raster` package; analysis in `sp`, `spatial`, `fields` . . .

Interfaces with other spatial analysis tools

Most spatial data is prepared outside of R, usually in a GIS or spreadsheet.

Results of R analyses are often presented outside of R, e.g., as GIS maps. How is information exchanged?

- Import and export

- Projections and Datums

- Transformations

The rgdal package

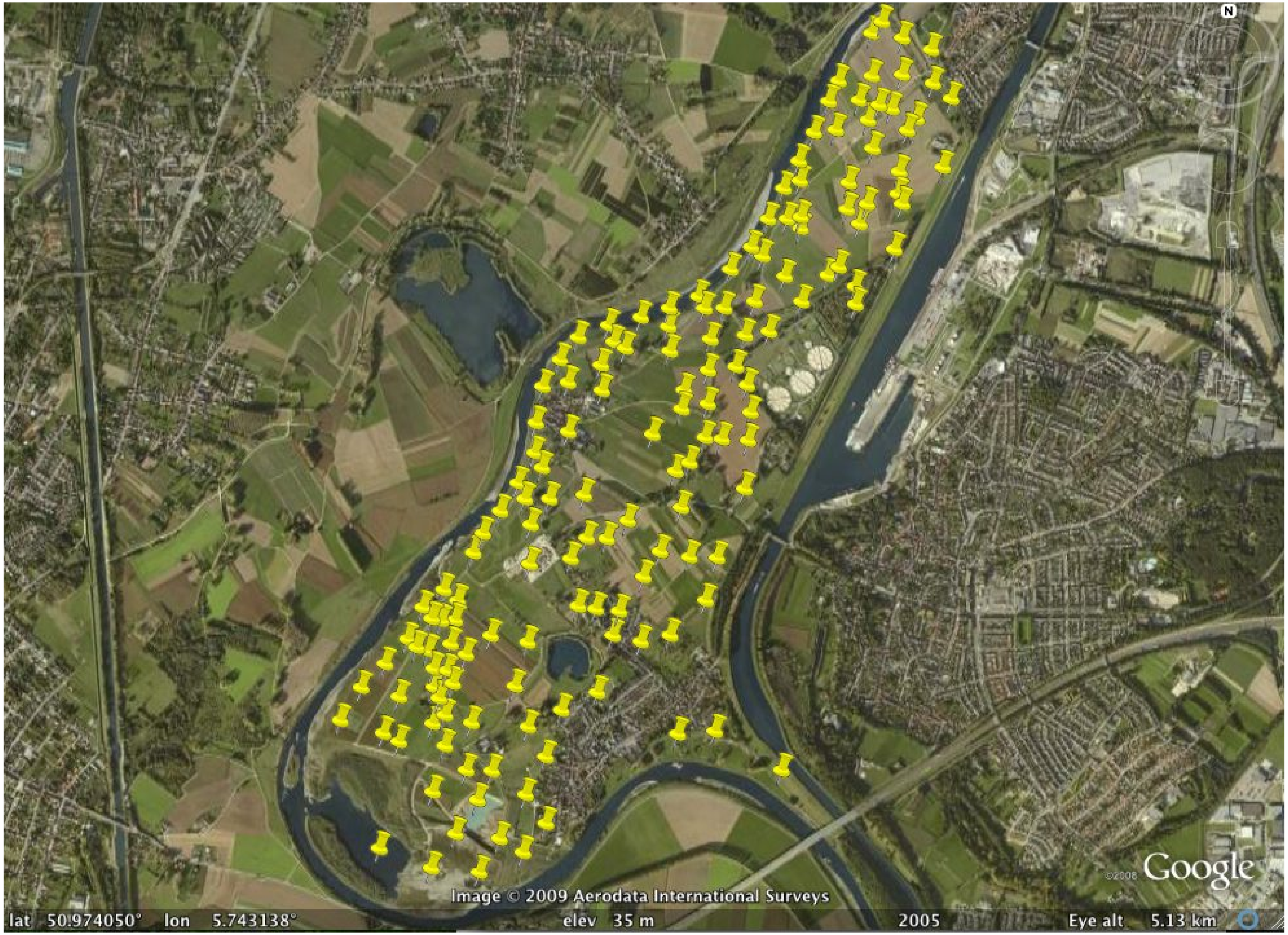
This package provides **bindings** for the **Geospatial Data Abstraction Library** (GDAL)[2], which is an open-source **translator** library for geospatial data formats.

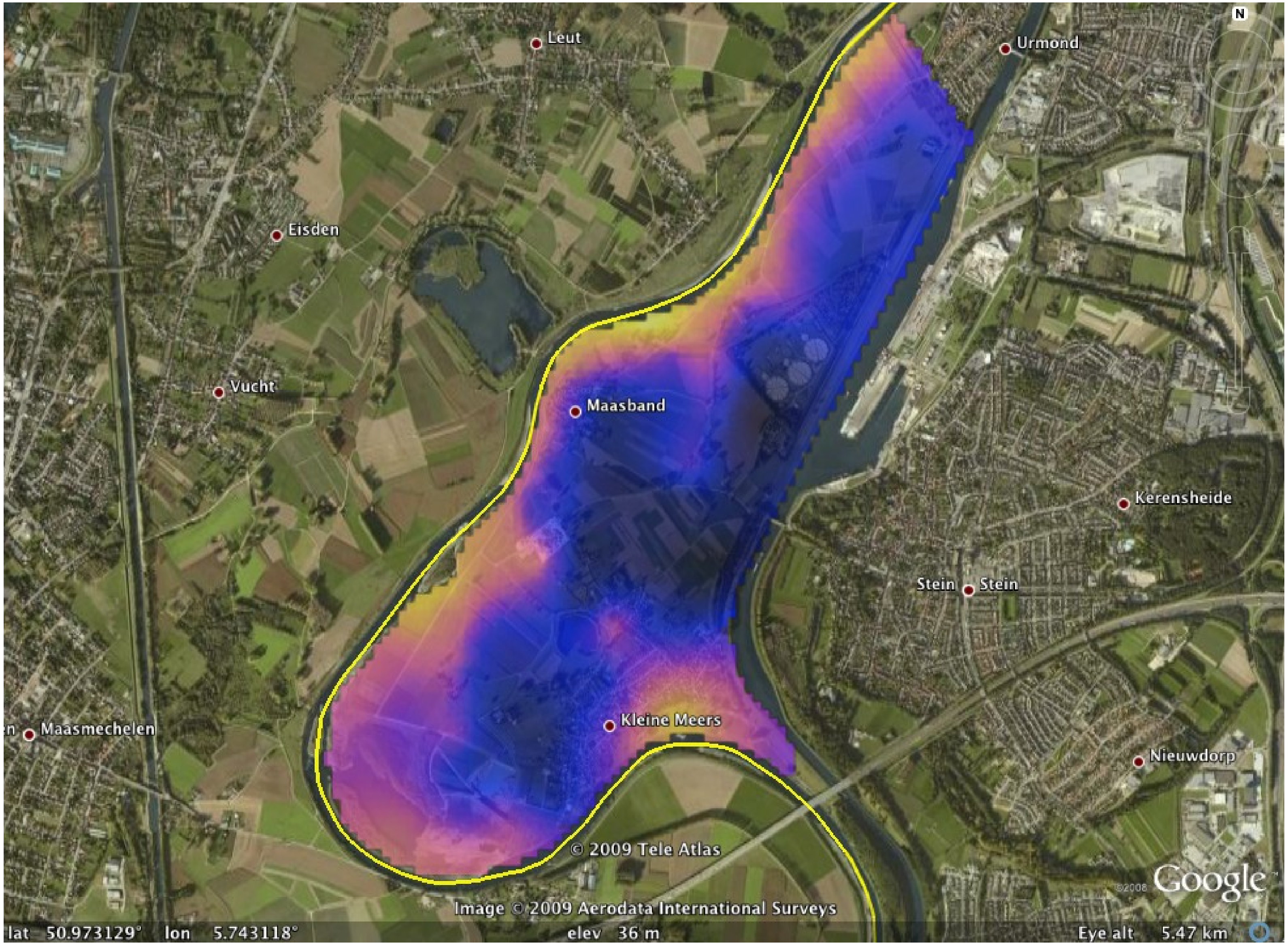
`rgdal` uses the `sp` classes.

- `readGDAL`; `writeGDAL`: Read/write between GDAL grid maps and Spatial objects
- `readOGR`, `writeOGR`: Read/write spatial vector data using OGR (including **KML** for Google Earth)
  - OGR: C++ open source library providing read (and sometimes write) access to a variety of **vector file formats** including ESRI Shapefiles, S-57, SDTS, PostGIS, Oracle Spatial, and Mapinfo mid/mif and TAB formats

[2] `http://www.gdal.org/`

Example Google Earth layers created with `writeOGR`




Leut
Urmond
Eisden
Vucht
Maasband
Kerensheide
Stein
Stein
Maasmechelen
Kleine Meers
Nieuwdorp
© 2009 Tele Atlas
Image © 2009 Aerodata International Surveys
Google
lat 50.973129° lon 5.743118°
elev 36 m
Eye alt 5.47 km
N

The CRS method

sp uses the CRS (Coördinate Reference System) method of the rgdal package to interface with the **proj.4** cartographic projection library from the USGS[3].

For example, to specify the Dutch Rijksdriehoek (RDH) coördinate system:

```
> proj4string(meuse) <- CRS("+proj=stere
+     +lat_0=52.15616055555555 +lon_0=5.38763888888889
+     +k=0.999908 +x_0=155000 +y_0=463000
+     +ellps=bessel +units=m +no_defs
+     +towgs84=565.2369,50.0087,465.658,
+       -0.406857330322398,0.350732676542563,-1.8703473836068,
+       4.0812")
```

ellps: ellipsoid; towgs84: relative datum;
proj: projection; lat, long: origin of projection;
k: scale factor at the origin; units of distance;
x_0, y_0: values of coördinates at origin (false E, N)

[3] http://trac.osgeo.org/proj/

The EPSG database

Most systems are included in the European Petroleum Survey Group (**EPSG**) database[4], in which case just the system's numbrer in that database is enough to specify it:

```
> library(rgdal)
> proj4string(meuse) <- CRS("+init=epsg:28992")  # assign a CRS by EPSG code
> proj4string(meuse)
[1] "+init=epsg:28992 +proj=sterea
      +lat_0=52.15616055555555 +lon_0=5.38763888888889
      +k=0.9999079 +x_0=155000 +y_0=463000 +ellps=bessel
      +towgs84=565.4171,50.3319,465.5524,
         -0.398957388243134,0.343987817378283,
         -1.87740163998045,4.0725
      +units=m +no_defs"
```

[4] http://www.epsg.org/

Finding EPSG codes in the database

```
> library(rgdal)
> epsg <- make_EPSG()              # Make a data frame of EPSG projection codes
> (epsg.zh <- epsg[grep("China", epsg$note), 1:2])  # search data frame
     code                                        note
254  4490 # China Geodetic Coordinate System 2000
1882 3415     # WGS 72BE / South China Sea Lambert
4836 4479 # China Geodetic Coordinate System 2000
> epsg[epsg.zh[1,"code"],"prj4"]     # display CRS information for one system
      code                           note
[1] +proj=utm +zone=11 +south +ellps=WGS72 +towgs84=0,0,4.5,0,0,0.554,0.2263
     +units=m +no_defs
```

Information on PROJ.4 in rgdal

```
> getPROJ4VersionInfo()
[1] "Rel. 4.9.1, 04 March 2015, [PJ_VERSION: 491]"

> str(proj <- projInfo("proj"))
'data.frame':        137 obs. of  2 variables:
 $ name       : Factor w/ 137 levels "aea","aeqd","airy",..
 $ description: Factor w/ 135 levels "Airy","Aitoff",..

> str(ellps <- projInfo("ellps"))
'data.frame':        43 obs. of  4 variables:
 $ name       : Factor w/ 43 levels "airy","andrae",..
 $ major      : Factor w/ 40 levels "a=6370997.0",..
 $ ell        : Factor w/ 26 levels "b=6355834.8467",..:
 $ description: Factor w/ 43 levels "Airy 1830","Andrae 1876 (Den., Iclnd.)",..

> ellps[ellps$name == "krass", ]
    name       major      ell      description
31 krass a=6378245.0 rf=298.3 Krassovsky, 1942
```

continued ...

```
> str(datum <- projInfo("datum"))
'data.frame':       10 obs. of  4 variables:
 $ name       : Factor w/ 10 levels "carthage","GGRS87",..
 $ ellipse    : Factor w/ 8 levels "airy","bessel",..
 $ definition : Factor w/ 9 levels "nadgrids=@conus,@alaska,@ntv2_0.gsb,@ntv1_can.dat",
 $ description: Factor w/ 10 levels "","Airy 1830",..

> datum[datum$name == "OSGB36", 2:3]
   ellipse                                                     definition
10  airy towgs84=446.448,-125.157,542.060,0.1502,0.2470,0.8421,-20.4894
```

Converting between CRS

The spTransform method of the rgdal R package:

- implements the PROJ.4 system[5] also found in the GDAL “Geospatial Data Abstraction Library” program[6]
- can convert between projections, backwards and forwards to/from geodetic coördinates
- with or without a datum transformtation

[5] http://trac.osgeo.org/proj/

[6] http://www.gdal.org/

Example – Meuse River soil pollution dataset (NL)

```
> require(sp)
> ## load an example dataset from the sp package
> data(meuse)
> ## convert to a spatial object; we must know which fields represent coordinates
> coordinates(meuse) <- ~x + y
> proj4string(meuse)

[1] NA

>  ## no CRS yet; define the CRS from metadata; first load GDAL
> require(rgdal)
> proj4string(meuse) <- CRS("+init=epsg:28992")
> proj4string(meuse)

    [1] "+init=epsg:28992 +proj=sterea
    +lat_0=52.15616055555555 +lon_0=5.38763888888889 +k=0.9999079
    +x_0=155000 +y_0=463000 +ellps=bessel
    +towgs84=565.417,50.3319,465.552,-0.398957,0.343988,-1.8774,4.0725
    +units=m +no_defs"

> ## EPSG database contained all required parameters
```

continued . . .

. . . continued

```
> ## un-project to geodetic system, also changing the datum
> meuse.wgs84 <- spTransform(meuse, CRS("+proj=longlat +datum=WGS84"))
> proj4string(meuse.wgs84)

 [1] "+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0"

> ## now this could be forward-projected into, e.g., UTM 31N on the WGS84 datum
> ## we find the appropriate initializtion in the EPSG database
> meuse.wgs84.utm31 <- spTransform(meuse.wgs84, CRS("+init=epsg:32631"))
> proj4string(meuse.wgs84.utm31)

[1] "+init=epsg:32619 +proj=utm +zone=31 +datum=WGS84 +units=m
      +no_defs +ellps=WGS84 +towgs84=0,0,0"

> ## note the origin is implicit in the definition of UTM31N
> ## now go directly from RDH to geographic coords, keeping the ellipsoid
> meuse.rdh.geo <- spTransform(meuse, CRS("+proj=longlat"))
> ## and to UTM31N on that ellipsoid
> meuse.rdh.utm31 <- spTransform(meuse, CRS("+proj=utm +zone=31"))
```

Compare coördinates of the first point in different CRS:

```
> coordinates(meuse)[1,]                  # RDH
 181072 333611
> coordinates(meuse.wgs84)[1,]            # WGS84 datum long/lat
 5.758536 50.991562
> coordinates(meuse.rdh.geo)[1,]          # RDH datum long/lat
 5.759029 50.992415
> coordinates(meuse.wgs84.utm31)[1,]      # UTM31N on WGS84 datum
 693585 5652509
> coordinates(meuse.rdh.utm31)[1,]        # UTM31N on RDH datum
 693616 5652605
```

Discrepency in UTM coördinates $\sqrt{(585-616)^2+(509-605)^2} = 100.9$ m

The gstat package

- R implementation of the stand-alone **gstat** package for geostatistics
- Author and maintainer Edzer Pebesma
  - mostly developed at Physical Geography, University of Utrecht (NL)
  - since Oct 2007 at Institute for Geoinformatics, University of Münster (D)
- 1992 – ???
- Purpose: "**modelling**, **prediction** and **simulation** of geostatistical data in one, two or three dimensions"
- Uses the `sp` spatial data structures

There are other R packages with overlapping aims but different methods and interfaces (e.g, `geoR`, `spatial`, `RandomFields`).

Modelling with gstat

- `variogram`: Compute **experimental variograms** (also directional, residual)
  - User-specifiable cutoff, bins, anisotropy angles and tolerances
  - Can use Matheron or robust estimators
  - Optional argument to produce a **variogram cloud** (all point-pairs)
  - Optional argument to produce a **directional variogram surface** ("map")

- `vgm`: specify a **theoretical variogram model** for an empirical variogram
  - Many authorized models
  - Can specify models with **multiple structures**

- `fit.variogram`: least-squares adjustment of a variogram model to the empirical model
  - User-selectable fitting criteria
  - Can also use restricted maximum likelihood `fit.variogram.reml`

- `fit.lmc`: fit a linear model of **coregionaliztion** (for cokriging)

- `gstat`: complicated procedures, e.g., co-kriging.

Conclusion

- A **disadvantage** of working in R is the lack of interactive graphical analysis (e.g. in ArcGIS Geostatistical Analyst)

- The main **advantage** of doing spatial analysis in R is that the full power of the R environment (data manipulation, non-spatial modelling, user-defined functions, graphics . . .) can be brought to bear on spatial analyses

- The advantages of R (**open-source**, **open environment**, packages contributed and vetted by statisticians) apply also to spatial analysis