Introduction to R and package sp

Edzer J. Pebesma
Institute for Geoinformatics, University of Münster
edzer.pebesma@uni-muenster.de

GIS Aufbaukurs, Feb 20, 2008

What is R?

- `www.r-project.org`: "R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To download R, please choose your preferred CRAN mirror."
- R implements the language S, an object-oriented language *designed for data analysis.*
- R is used mostly in academia, S-Plus more in corporate businesses
- everything in R is an object
- R uses a data base where it stores its objects; this is empty or loaded on start-up, and (possibly) saved on exit
- during run-time, R does everything in memory, unless you load or save data from/to disk or connection.

ifgi

R has functions

In

```
> library(foreign)
> control = read.dbf("points/control.dbf")
```

the function `library` returns nothing, but has a *side effect*.
`foreign` is the argument: it is the name of the library that needs to be loaded. The side effect is that the functions in `foreign` become available.
`read.dbf` is a function that reads an external DBF file and puts a `data.frame` with name `control` in the data base. Its argument is a file, here `control.dbf` in directory `points`, relative to the current working directory.

Loading data from a package

In

```
> library(sp)
> data(meuse)
```

the `data(meuse)` command has the side effect that it makes the `meuse` data set avaible to to current session: it is copied from the data section in package `sp`. Changes to `meuse` will be lost after

```
> data(meuse)
```

is repeated.

Assignment

Symbols `=` and `<-` assign, as in

```
> a = 3
> a <- 3
> a
[1] 3
```

when no assignment takes place, the result is shown (printed or plotted)

Classes – every object has a class

```
> a = 3
> class(a)
[1] "numeric"
> b = list(first = 3, second = "some text", 3:7)
> b
$first
[1] 3

$second
[1] "some text"

[[3]]
[1] 3 4 5 6 7
> class(b)
[1] "list"
> class(mean)
[1] "function"
```

Lists and subsetting

```
> b = list(first = 3, second = "some text", 3:7)
> b[1]

$first
[1] 3

> b["first"]

$first
[1] 3

> b[["first"]]

[1] 3

> b[-(2:3)]

$first
[1] 3
```

Replacement and removal

```
> b = list(first = 3, second = "some text", 3:7)
> b[[1]] = 4
> b[["second"]] = NULL
> b

$first
[1] 4

[[2]]
[1] 3 4 5 6 7
```

ifgi

vectors and factors

```
> a = c(1, 2, 10.5)
> a
[1]  1.0  2.0 10.5
> b = c("NL", "NL", "UK", "UK", "DE")
> b
[1] "NL" "NL" "UK" "UK" "DE"
> f = factor(b)
> f
[1] NL NL UK UK DE
Levels: DE NL UK
> as.numeric(f)
[1] 2 2 3 3 1
```

data.frame

`data.frame` is the standard structure for tabular data:

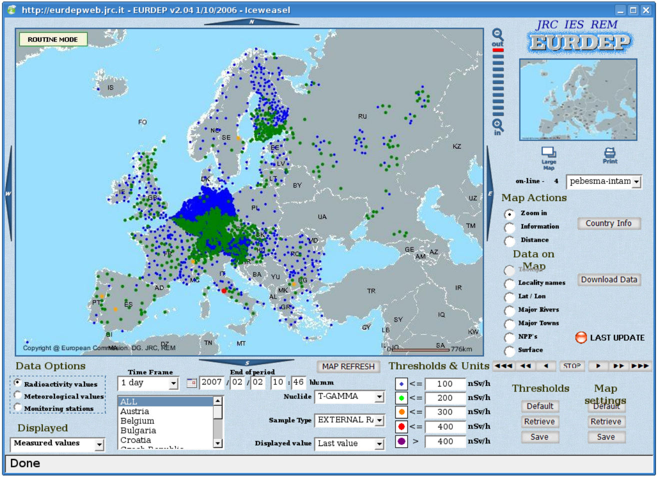
```
> f = as.factor(c("a", "a", "b"))
> a = data.frame(x1 = 1:3, x2 = rnorm(3), f = f)
> a

  x1         x2 f
1  1  0.1285906 a
2  2  2.3791273 a
3  3 -0.6756605 b

> a[1, ]

  x1        x2 f
1  1 0.1285906 a

> a[, 2]

[1]  0.1285906  2.3791273 -0.6756605

> a[1, 2]

[1] 0.1285906
```

The $ sign

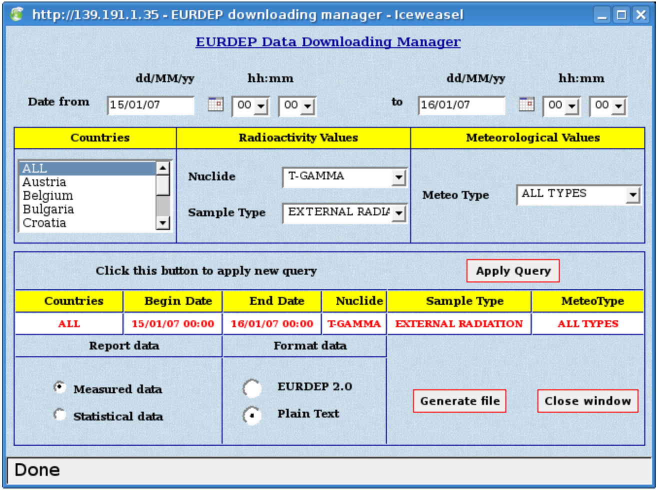
The $ sign is short for `[[` for named list elements or `data.frame` colums:

```
> b$first
NULL
> a$x1
[1] 1 2 3
> a$x1 = 3:1
> a
  x1         x2 f
1  3  0.1285906 a
2  2  2.3791273 a
3  1 -0.6756605 b
```

EURDEP data for 2007/02/02, downloaded 2007/02/26



EURDEP data for 2007/01/15, downloaded 2007/02/26



EURDEP data

```
> filename = "260207105826_eurdepdata_0.TXT"
> eurdep = read.delim(filename, na.string = "-")
> dim(eurdep)

[1] 100876     28

> tstart = strptime(eurdep$BEGIN, "%Y-%m-%dT%H:%M:%SZ")
> tend = strptime(eurdep$END, "%Y-%m-%dT%H:%M:%SZ")
> noon = ISOdate(2007, 1, 15, 12, 0, 0)
> eurdep = eurdep[tstart < noon & tend > noon, ]
> dim(eurdep)

[1] 2693   28
```

EURDEP data – exploration

```
> names(filename)
> table(eurdep$COUNTRY_CODE)
> lapply(eurdep, class)
> summary(eurdep)
```

formulae and methods

A `formula` is a syntactic form to express a model:

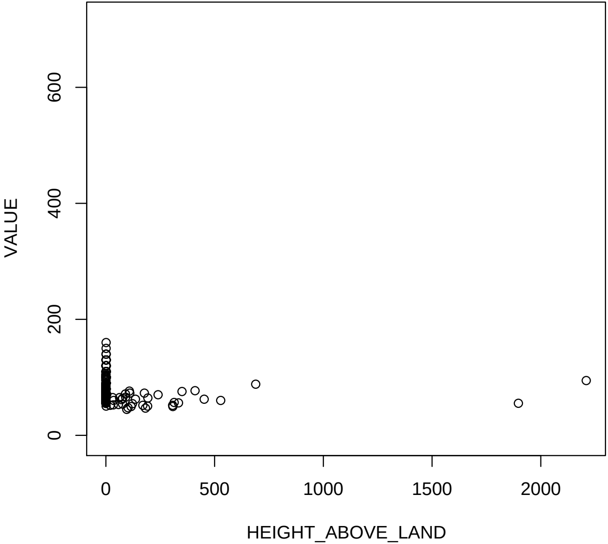
```
> VALUE ~ COUNTRY_CODE

VALUE ~ COUNTRY_CODE
```

and can be passed to the linear regression function `lm` along with the data where these names can be resolved, as in

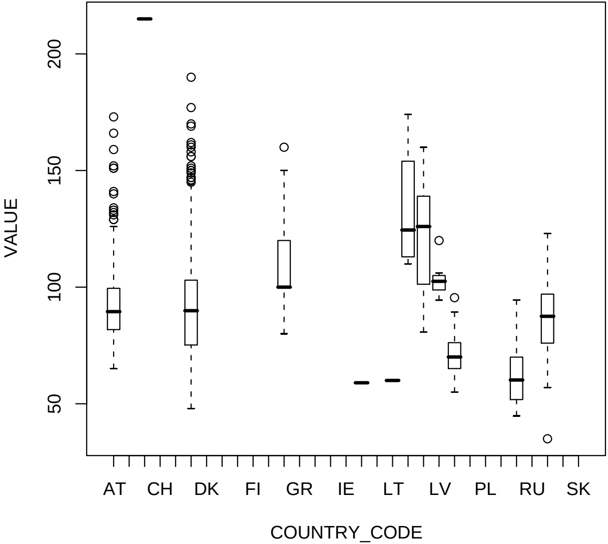
```
> lm(VALUE ~ HEIGHT_ABOVE_LAND, eurdep)

Call:
lm(formula = VALUE ~ HEIGHT_ABOVE_LAND, data = eurdep)

Coefficients:
      (Intercept)  HEIGHT_ABOVE_LAND
         81.73501           -0.01255
```

```
> height.lm = lm(VALUE ~ HEIGHT_ABOVE_LAND, eurdep)
> summary(height.lm)

Call:
lm(formula = VALUE ~ HEIGHT_ABOVE_LAND, data = eurdep)

Residuals:
    Min      1Q  Median      3Q     Max
-35.731 -15.235  -6.335  18.278  78.278

Coefficients:
                   Estimate Std. Error t value Pr(>|t|)
(Intercept)       81.735009   1.268384  64.440   <2e-16 ***
HEIGHT_ABOVE_LAND -0.012545   0.006682  -1.878   0.0615 .
---
Signif. codes:  0 "***" 0.001 "**" 0.01 "*" 0.05 "." 0.1 " " 1

Residual standard error: 21.24 on 289 degrees of freedom
  (2402 observations deleted due to missingness)
Multiple R-Squared: 0.01205,      Adjusted R-squared: 0.008632
F-statistic: 3.525 on 1 and 289 DF,  p-value: 0.06145
```

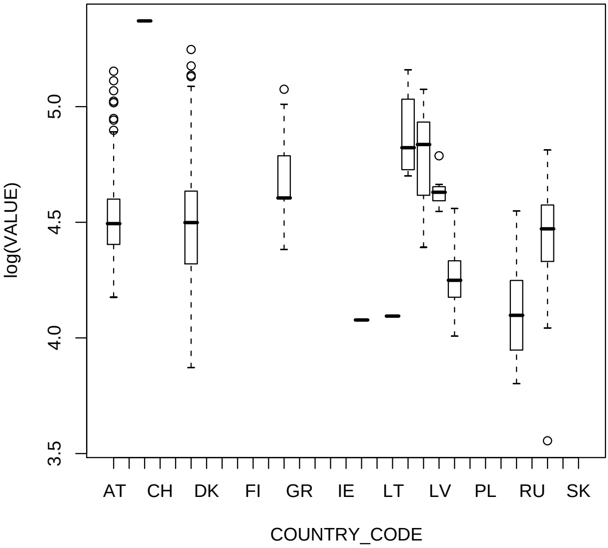
```
> plot(VALUE ~ HEIGHT_ABOVE_LAND, eurdep)
```



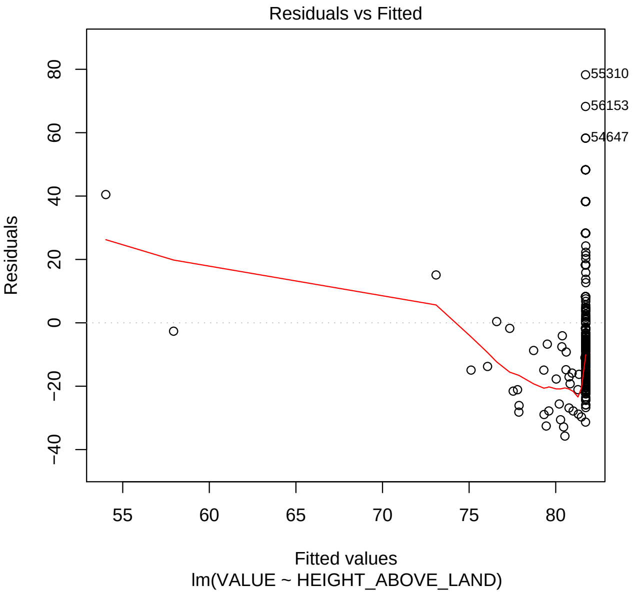
```
> plot(VALUE ~ COUNTRY_CODE, eurdep)
```



```
> plot(log(VALUE) ~ COUNTRY_CODE, eurdep)
```



```
> plot(height.lm, which = 1)
```



Methods in R

R provides methods that provide "expected" behaviour:

- plot: plots data, models, maps, ...
- summary: gives a summary in a few lines
- print: prints the full contents
- subsetting, selecting:

```
> library(rgdal)
> nuts1 = readOGR("GISCO/NUTS/NUTS_RG_10M_2007",
+       "NUTS_RG_10M_2007")
> nuts1[nuts1$CNTR_CODE == "DE", ]
```

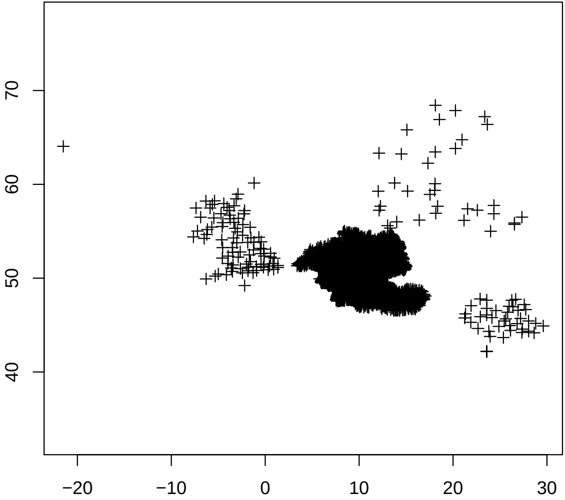
Spatial data – package sp

Package `sp` provides methods and classes for spatial data. `sp` objects

- behave as much as possible as `data.frame`s (subsetting, replacement etc)
- are recognized by the spatial analysis packages (gstat, splancs, spatstat, geoR, ...)
- are recognized by GIS I/O and coordinate transformation packages (maptools, rgdal, ...)
- have a bounding box and a CRS
- know which information refers to topology, and which to attributes
- include points, lines, polygons (rings, no topology), grids (pixel/grid)
- may or may not have attributes

```
> eurdep[1:3, c("LONGITUDE", "LATITUDE", "VALUE")]
  LONGITUDE LATITUDE VALUE
2 E016.6275 N47.6314  80.5
5 E016.4600 N47.1075 101.0
8 E016.5378 N47.8544  88.4
> class(eurdep)
[1] "data.frame"
> library(sp)
> eurdep$y = as.numeric(sub("N", "", as.character(eurdep$LA
> eurdep$x = as.numeric(sub("W", "-", sub("E", "",
+     as.character(eurdep$LONGITUDE))))
> coordinates(eurdep) = ~x + y
> eurdep[1:3, "VALUE"]
           coordinates VALUE
2 (16.6275, 47.6314)  80.5
5   (16.46, 47.1075) 101.0
8 (16.5378, 47.8544)  88.4
> class(eurdep)
[1] "SpatialPointsDataFrame"
```

```
> plot(eurdep, axes = TRUE)
```



rgdal: coordinate transformation, GE

convert coordinate system to ID ETRS-LAEA (the "INSPIRE" one)

```
> library(rgdal)
```

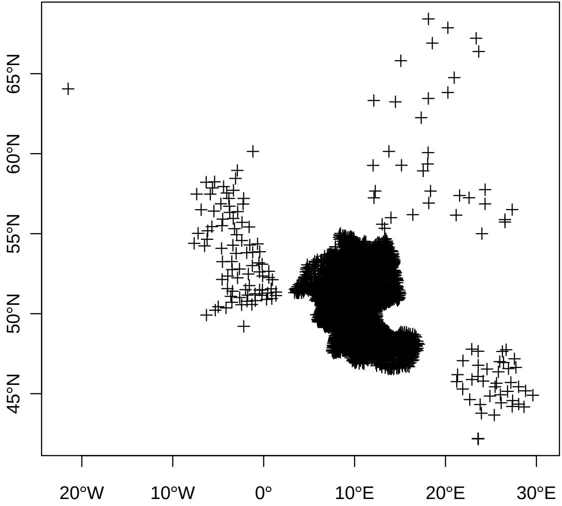
```
Geospatial Data Abstraction Library extensions to R success
Loaded runtime: GDAL 1.4.1.0, released 2007/04/09
```

```
> proj4string(eurdep) = CRS("+init=epsg:4326")
> eurdep.tr = spTransform(eurdep, CRS("+init=epsg:3035"))
```

Export untransformed data to GE:

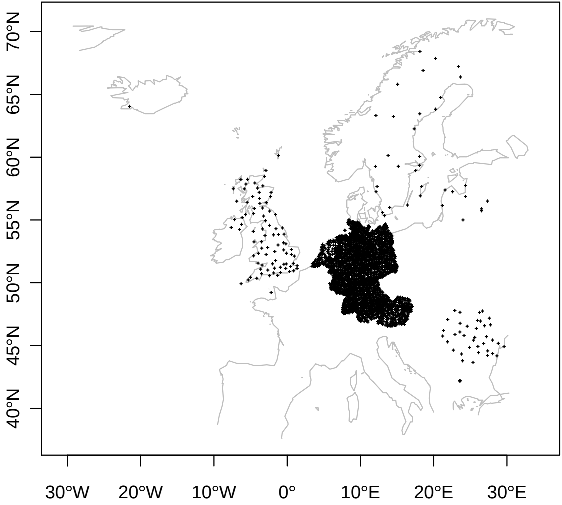
```
> writeOGR(eurdep, "eurdep.kml", "eurdep.kml", driver = "KM
```

```
> plot(eurdep, axes = TRUE)
```

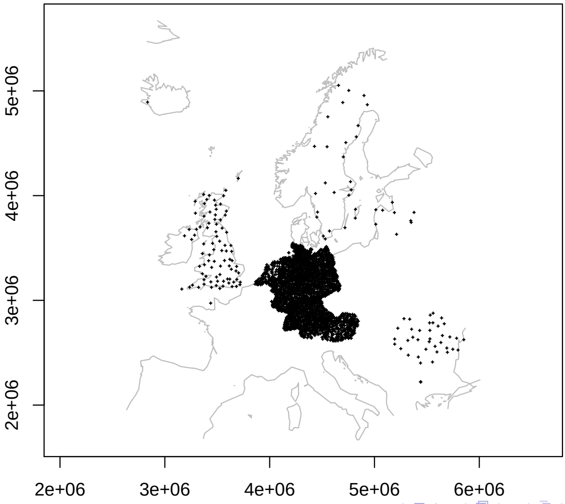


```
> plot(eurdep, axes = TRUE)
> library(maps)
> library(mapdata)
> library(maptools)
> wrld = map("world", interior = FALSE, plot = FALSE,
+     xlim = c(-25, 30), ylim = c(40, 70))
> wrld = pruneMap(wrld)
> wrld.sp = map2SpatialLines(wrld, proj4string = CRS("+init
```

```
> plot(wrld.sp, axes = TRUE, col = "grey")
> points(eurdep, pch = 3, cex = 0.2)
```



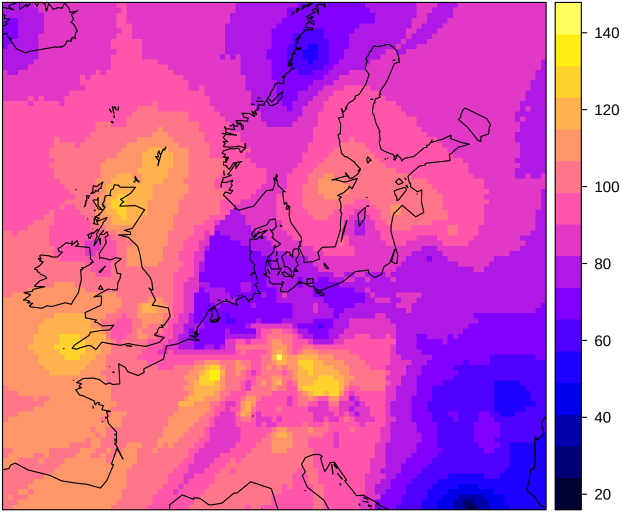
```
> wrld.sp = spTransform(wrld.sp, CRS("+init=epsg:3035"))
> plot(wrld.sp, axes = TRUE, col = "grey")
> points(eurdep.tr, pch = 3, cex = 0.2)
```



Methods in package sp

- `print`, `summary`: print, summarize
- `plot`, `spplot`: plot methods
- `bbox`: retrieve spatial bounding box
- `coordinates`, `coordinates<-`
- `polygons`, `polygons<-`: retrieve or set polygons
- `coordnames`, `coordnames<-`: get/set coordinate names
- `gridded`: convert points to grid or reverse
- `overlay`: overlay two layers
- `spsample`: spatial sampling

```
> library(gstat)
> eurdep.tr = eurdep.tr[eurdep$VALUE < 200, ]
> v = variogram(VALUE ~ 1, eurdep.tr, cutoff = 2e+05)
> plot(v)
> v.fit = fit.variogram(v, vgm(1, "Exp", 1e+05,
+     1))
> plot(v, v.fit)
> grd = makegrid(eurdep.tr)
> grd.sp = SpatialPoints(grd)
> gridded(grd.sp) = TRUE
> proj4string(grd.sp) = CRS(proj4string(eurdep.tr))
> zd = zerodist(eurdep.tr)
> out = krige(VALUE ~ 1, eurdep.tr[-zd[, 1], ],
+     grd.sp, v.fit, nmax = 100)

[using ordinary kriging]

> spplot(out[1], col.regions = bpy.colors(), sp.layout = li
+     wrld.sp))
```


140
120
100
80
60
40
20

Better backdrop data

```
> library(rgdal)
> nuts1 = readOGR("NUTS_RG_10M_2007", "NUTS_RG_10M_2007")
> nuts1.tr = spTransform(nuts1, CRS(proj4string(eurdep.tr))
> plot(eurdep.tr, cex = 0.2, col = "red")
> plot(nuts1.tr, add = T, border = "grey")
> layout = list("sp.polygons", nuts1.tr, first = FALSE)
> spplot(out[1], col.regions = bpy.colors(), sp.layout = la
```