Topic 2: Introduction to R

This topic gives overview of basics of R software. In particular, we consider

- **R Software** for spatial analysis.
- **Key concepts of R.**

R Software for spatial analysis

The R software has a number of packages useful for spatial analysis and modelling of various sorts. Some R packages that were developed for analysing spatial data include:

ads	spatial point pattern analysis
DCluster	detecting clusters in spatial count data fields curve and function fitting
geoR	model-based geostatistical methods
geoRglm	model-based geostatistical methods
GeoXB	interactive spatial exploratory data analysis grasp spatial prediction
maptools	geographical information systems
rgdal	interface to GDAL geographical data analysis
sp	base library for some spatial data analysis packages
spatclus	detecting clusters in spatial point pattern data
spatialCovariance	spatial covariance for data on grids
spatialkernel	interpolation and segregation of point patterns
spatstat	Spatial point pattern analysis and modelling
spBayes	Gaussian spatial process MCMC (grid data)
spdep	spatial statistics for variables observed at fixed sites
spgwr	geographically weighted regression
splancs	spatial and space-time point pattern analysis
spsurvey	spatial survey methods
trip	analysis of spatial trip data

R is a language and environment for data analysis. R is free software with an open-source licence. You can download it from **r-project.org** and it should be easy to install on any computer (see the instructions at the website).

In addition to the basic R system, the R website also offers many add-on modules (libraries or packages) contributed by users. These can be downloaded from **cran.r-project.org** (under Contributed Packages).

RStudio IDE is a powerful and productive user interface for R. Its free and open source, and works great on Windows, Mac, and Linux. To download RStudio, please use **www.rstudio.com**.

To make use of a package, you need to:

1. download the package code (once only) without unpacking;
2. install the package on your system (once only);
3. load the package into your current R session using the command library (each time you start a new R session).

The installation step is performed automatically using R, not by manually unpacking the code. Installation is usually a very easy process.

- If you are running Windows and use an R GUI, use the pull-down menu item Packages – Install packages. If this menu item is available, then you will be able to download and install any desired packages by simply selecting the package name from the pulldown list.
- If this menu item is not available (for internet security reasons), you can manually download packages by going to the CRAN website under Contributed packages – Windows binaries and downloading the desired zip files of Windows binary files. Then, start an R session and use the menu item Packages Install from local zip files to install.
- If you use RStudio, use the tab Tools $\rightarrow$ Install Packages and start typing a name of a package. Select it and click Install.

R session

You can run an R session using either a point-and-click interface or a line-by-line command interpreter. In these notes, R commands are printed as they would appear when typed at the command line. So a typical series of R commands looks similar to this:

```
> pi/2
> sin(pi/2)
> x <- sqrt(2)
> x
```

Note that you are not meant to type the $>$ symbol; this is just the prompt for command input in R.
If the input is too long, R will break it into several lines, and print the character $+$ to indicate that the input continues from the previous line. (You don't type the $+$). Also if you type an expression involving brackets and hit Return before all the open brackets have been closed, then R will print a $+$ indicating that it expects you to finish the expression.

Classes in R

R is an object-oriented language. A dataset with some kind of structure on it (e.g. a contingency table, a time series, a point pattern) is treated as a single object. For example, R includes a dataset sunspots which is a time series containing monthly sunspot counts from 1749 to 1983. This dataset can be manipulated as if it were a single object:

```
> plot(sunspots)
> summary(sunspots)
> X <- sunspots
```

Each object in R is identified as belonging to a particular type or class depending on its structure. For example, the sunspots dataset is a time series:

```
> class(sunspots)
[1] "ts"
```

The command TYPEOF returns the storage mode of the object x:

```
> typeof(sunspots)
[1] "double"
```

Standard operations, such as printing, plotting, or calculating the sample mean, are defined separately for each class of object. For example, typing

```
> plot(sunspots)
```

invokes the generic command plot. As sunspots is an object of class "ts" representing a time series, and there is a special method for plotting time series, called PLOT.TS. So the system executes

```
> plot.ts(sunspots)
```

It is said that the plot command is dispatched to the method plot.ts. The plot method for time series produces a display that is sensible for time series, with axes properly annotated.

Data frames

Using an example from the standard data set cars, we can see that it is an object of class data.frame, stored in a list, which is a vector whose components can be arbitrary objects.

data.frame has both names and summary methods:

```
> class(cars)
[1] "data.frame"
> typeof(cars)
[1] "list"
> names(cars)
[1] "speed" "dist"
> summary(cars)
speed dist
[...]
```

The data.frame contains two variables, one recording the speed of the observed cars in mph, the other the stopping distance measured in feet.

The STR method often gives a clear digest, including the size and class:

```
>  str(cars)
'data.frame': 50 obs. of 2 variables:
$ speed:num 4 4 7 7 8 ...
$ dist :num 2 10 4 22 16 ...
```

If you would like to see specific element or variables in your data.frame, you can use data.frame indexing by $ or [...]:

```
>  cars$speed
>  cars[ , 1]
>  cars[3, 1]
```

Select a specific column

Selects the first column of data

Select the specific element of row/column.

Functions

Every function in R returns a value. The return value may be null, or a single number, a list, or any kind of object. When you type an R expression on the command line, the result of evaluating the expression is printed.

```
>  1 + 1
[1] 2
>  sin(pi/3)
[1] 0.8660254
```

A function which performs a complicated analysis of your data will typically return an object belonging to a special class. This is a convenient way to handle calculations that yield large or complicated output. It enables you to store the result for later use, and provides methods for handling the result.

The ? command displays the function's help. In the RStudio editor you can set the cursor at a function name and use F1 to find function's help.

Formula

A formula is most often two-sided, with a response variable to the left of the tilde operator, and in this case a determining variable on the right:

```
> class(dist ~ speed)
[1] "formula"
```

These objects are typically used as arguments to model fitting functions, such as lm (to fit linear models). They will usually be accompanied by a data argument, indicating where the variables are to be found:

```
> lm(dist ~ speed, data = cars)
Call:
lm(formula = dist ~ speed, data = cars)
Coefficients:
(Intercept) speed
-17.579 3.932
```

Key R commands

plot(x)	*plots an R object*
summary(x)	*produces result summaries of an R object*
class(x)	*gives object's class*
typeof(x)	*determines the type of an object*
names(x)	*gives names of an object*
str(x)	*display the structure of an R object*
lm(formula, data)	*fits a linear model*
curve(expression, from, to)	*draws a curve corresponding to an expression over the interval [from, to]*
hist(x, ...)	*computes a histogram of the given data*
mean(x)	*computes the arithmetic mean*
sd(x)	*computes the standard deviation*