Applied geostatistics
Exercise 2: Visualizing spatial structure

D G Rossiter
University of Twente, Faculty of Geo-Information Science & Earth Observation (ITC)

June 26, 2010

Contents

1 Introduction **1**

2 Reading data from files **1**
2.1 Answers 4

3 Data types **4**
3.1 Spatially-explicit objects 6
3.2 Saving R objects 8
3.3 Answers 9

4 Spatial distribution of point observations **9**
4.1 Answers 12

5 Postplots **12**
5.1 * Panel functions 15
5.2 Answers 17

6 Visualizing regional trends **17**
6.1 Answers 24

7 Visualizing local spatial dependence **25**
7.1 The variogram cloud 25

Version 1.92. Copyright © 2006-2010 University of Twente, Faculty ITC All rights reserved. Reproduction and dissemination of the work as a whole (not parts) freely permitted if this original copyright notice is included. Sale or placement on a web site where payment must be made to access this document is strictly prohibited.
To adapt or translate please contact the author (http://www.itc.nl/personal/rossiter).

7.2 The empirical variogram 30
7.3 Answers 34

8 Visualizing anisotropy **35**
8.1 Answers 41

9 Quitting R **42**

10 Self-test **42**

References **45**

Index of R concepts **47**

入乡随俗 (Rù xiāng suí sú)
"When you enter a village, follow its customs"
– Chinese proverb

1 Introduction

In this exercise you will learn some ways to visualize spatial data in R and the spatial packages `gstat` and `sp`.

After completing this exercise you should be able to:

1. Generate scatterplots of point locations in space;
2. Generate postplots showing the relative value of some attribute in feature space by symbol size, colour, or both;
3. Generate a first- and second-order trend surface from a point set;
4. Generate variogram clouds and experimental variograms;
5. Generate variogram surfaces and directional variograms.

Task 1 : If R is not already running, start it. If you haven't already done so, load the `gstat` and `sp` libraries, as shown in the previous exercise. Also load the `lattice` graphics library, which is used for many of the plots. •

Note that the `require` method checks whether a package is already loaded, and only loads it if needed.

```
> require(sp)
> require(gstat)
> require(lattice)
```

2 Reading data from files

In the previous exercise we used example datasets that are supplied with R or one of its add-in packages. Of course we usually want to work on our own datasets, which are generally not supplied already in R format. In this section we see how to read data from text files and store them in R format.

Our example is a set of soil samples from the Swiss Jura, supplied by J.-P. Dubois of the Ecole Polytechnique Fèdérale in Lausanne (CH). This dataset is used as a running example in the well-known text of Goovaerts [3]; there are also many research papers [1, 14, 4, 2, 7, 6, 15] that use it as a test case for geostatistical methods.

> **Note:** Since these notes were first written, the Jura dataset has been made available as an example dataset in the `gstat` package. We continue with the exercise of loading it from an external file, because it is a skill you will need when working with your own datasets.
>
> The built-in version is accessed with the `data(jura)` command; see `?jura` for more information on the contents.

This dataset is supplied on the course CD, in directory `datasets\jura`, as a text file named `juraset.dat`; it is also on the course website.

Task 2 :

1. Create a new **working directory** for the results of your exercises; in MS-Windows this is a new **folder**.
2. Copy the file `juraset.dat` to this working directory.
3. Make this R's working directory.

•

For the last part of the task, we use the `setwd` ("set working directory") method; we check the current directory with the `getwd` ("get [current] working directory") method.

> **Note:** Running R under the MS-Windows GUI, you can also change the working directory with the `File | Change dir ...` menu choice. Running R under the Mac OS/X GUI, this is `Misc | Change working directory ...` or `(Apple key)+D`.

Your output from these commands will be different from what is shown here, because you will be working in a different directory structure. I have chosen to set up a directory named `dgeostat` ("distance geostatistics course") in my `tmp` ("temporary") directory, in my home folder.

```
> getwd()

[1] "/Users/rossiter/data/edu/dgeostats/ex"

> setwd("/Users/rossiter/tmp/dgeostat")
> getwd()

[1] "/Users/rossiter/tmp/dgeostat"
```

Task 3 : Examine the contents of text file `juraset.dat`. •

You could view the file contents in a plain-text editor; e.g. on Windows you could use Wordpad or Notepad, but we will use R's `file.show` method, which shows the file contents in a separate window.

```
> file.show("juraset.dat")

The JURA data set
Provided by J.-P. Dubois, IATE-Pedologie,
Ecole Polytechnique Federale de Lausanne, 1015 Lausanne (CH)

Printed as Appendix C in: Goovaerts, P., 1997, Geostatistics for natural
resources evaluation, Oxford University Press.

Spatial coordinates and values of categorial and continuous attributes
at the 359 sampled sites (original German names in parentheses)
Rock Types: 1: Argovian, 2: Kimmeridgian, 3: Sequanian, 4: Portlandian,
   5: Quaternary.
```

```
Land uses: 1: Forest, 2: Pasture (Weide(land), Wiese, Grasland),
   3: Meadow (Wiese, Flur, Matte, Anger), 4: Tillage (Ackerland, bestelltes Land)

References:
(1) Atteia, O., Dubois, J.-P., Webster, R., 1994,
Geostatistical analysis of soil contamination in the Swiss Jura.
Environmental Pollution 86, 315-327;
(2) Webster, R., Atteia, O., Dubois, J.-P., 1994,
Coregionalization of trace metals in the soil in the Swiss Jura.
European Journal of Soil Science 45, 205-218

X       Y    Rock  Land    Cd       Cu      Pb      Co       Cr      Ni      Zn
2.386   3.077   3     3     1.740   25.72   77.36   9.32    38.32   21.32   92.56
2.544   1.972   2     2     1.335   24.76   77.88   10.00   40.20   29.72   73.56
...
2.593   3.312   3     3     0.325   8.08    26.20   10.60   30.00   14.00   54.96
```

This text file contains three kinds of information:

1. A description of the dataset (**metadata**): 22 lines (count them!);
2. Names of the different variables (fields): one line;
3. The data themselves; one line (**record**) per observation;

Q1 : *Which is the line with the variable names?* *Jump to A1* •

Q2 : *How are the fields in each record separated from each other?* *Jump to A2* •

To work on this dataset, we need to read it into R. Fortunately, R has good facilities for importing and exporting in many formats; for details see the *R Data Import/Export* manual supplied with R [12]; this is found in the `doc/manual` subdirectory of your R installation. Here we will use a simple approach: read the data from a text file.

Task 4 : Read the contents of the file into a **data frame**; list the first and last several lines to make sure it was read in correctly. •

We use the `read.table` method, which is a very flexible method for importing text files. In the present case we have seen that the first 22 lines are documentation; these must be skipped with the optional `skip` argument. Then we have a **header** line giving the variable names; we ask R to use this line to name the fields in the data frame by using the optional `header` argument.

To check the first and last records we use the `head` and `tail` methods.

```
> jura.all <- read.table("juraset.dat", header = T, skip = 22)
> head(jura.all)

      X     Y Rock Land    Cd    Cu    Pb    Co    Cr    Ni    Zn
1 2.386 3.077    3    3 1.740 25.72 77.36  9.32 38.32 21.32 92.56
```

```
2 2.544 1.972    2    2 1.335 24.76 77.88 10.00 40.20 29.72 73.56
3 2.807 3.347    3    2 1.610  8.88 30.80 10.60 47.00 21.40 64.80
4 4.308 1.933    2    3 2.150 22.70 56.40 11.92 43.52 29.72 90.00
5 4.383 1.081    5    3 1.565 34.32 66.40 16.32 38.52 26.20 88.40
6 3.244 4.519    5    3 1.145 31.28 72.40  3.50 40.40 22.04 75.20
```

```
> tail(jura.all)
```

```
        X     Y Rock Land    Cd    Cu    Pb    Co    Cr    Ni    Zn
354 4.713 1.302    2    3 0.845 10.28 33.64  9.68 29.92 23.80 65.60
355 4.248 2.489    4    1 1.220  5.52 48.80  5.24 27.04 21.04 46.40
356 3.784 3.677    2    1 0.640  6.68 34.32 13.92 28.32 18.00 55.20
357 4.324 2.835    2    1 1.650  8.88 60.00  8.72 33.36 22.72 80.00
358 3.859 4.022    3    3 1.433 43.60 60.80 13.32 47.60 29.12 87.20
359 2.593 3.312    3    3 0.325  8.08 26.20 10.60 30.00 14.00 54.96
```

Task 5 : Determine the number of records and fields in the data frame. •

Since the data frame is just a matrix with row and column names, we can use the `dim` ("matrix dimension") method:

```
> dim(jura.all)
```

```
[1] 359  11
```

Q3 : *How many records (observations, rows) are there in this data frame? How many fields (variables, columns)?* *Jump to A3* •

2.1 Answers

A1 : *The line with the field (variable) names is this one:*

```
X  Y  Rock Land Cd  Cu  Pb  Co  Cr  Ni  Zn
```

Return to Q1 •

A2 : *By one or more spaces (blanks).* *Return to Q2* •

A3 : *359 records, each with 11 fields.* *Return to Q3* •

3 Data types

Task 6 : Examine the structure of the `jura.all` object. •

The `str` method gives the data structure:

```
> str(jura.all)
```

```
'data.frame':        359 obs. of  11 variables:
 $ X   : num  2.39 2.54 2.81 4.31 4.38 ...
 $ Y   : num  3.08 1.97 3.35 1.93 1.08 ...
 $ Rock: int  3 2 3 2 5 5 5 1 1 3 ...
 $ Land: int  3 2 2 3 3 3 3 3 3 3 ...
 $ Cd  : num  1.74 1.33 1.61 2.15 1.56 ...
 $ Cu  : num  25.72 24.76 8.88 22.7 34.32 ...
 $ Pb  : num  77.4 77.9 30.8 56.4 66.4 ...
 $ Co  : num  9.32 10 10.6 11.92 16.32 ...
 $ Cr  : num  38.3 40.2 47 43.5 38.5 ...
 $ Ni  : num  21.3 29.7 21.4 29.7 26.2 ...
 $ Zn  : num  92.6 73.6 64.8 90 88.4 ...
```

Q4 : *What are the data types of each field?* *Jump to A4* •

Q5 : *Do these data types match the meaning of each field?* *Jump to A5* •

We see that the **categorical variables** `Rock` and `Land` are not really numbers, they are numeric **codes** that refer to specific rock and land types. This is because the `read.table` method tries to determine the data type of each field; in these field there are only integers, so the field's data type was set to `int`. The import method had no way of knowing that these codes in fact refer to categories. So, we have to inform R of this.

We are told in the metadata what the codes refer to; for example code `1` represents forest lands. We can not use these arbitrary codes `1, 2 ...` as if they were numbers! For example, it would be a serious error to compute a linear regression of one of the metals (say, `Zn`) on the land use.

R solves this problem by representing categorical variables as `factors`; this name comes from their use in "factorial" analysis of variance.

Task 7 : Convert the two categorical variables to R `factors`, and assign the correct names. Again check the data types of each field. •

R provides the `factor` method to specify factors; here we use the optional `labels` argument to assign the names. Note the use of the `c` ('catenate', which is Latin for 'make a chain') method to make a **list** of names, which are then used as the labels.

```
> jura.all$Rock <- factor(jura.all$Rock, labels = c("Argovian",
+     "Kimmeridgian", "Sequanian", "Portlandian", "Quaternary"))
> jura.all$Land <- factor(jura.all$Land, labels = c("Forest",
+     "Pasture", "Meadow", "Tillage"))
> str(jura.all)

'data.frame':        359 obs. of  11 variables:
 $ X   : num  2.39 2.54 2.81 4.31 4.38 ...
 $ Y   : num  3.08 1.97 3.35 1.93 1.08 ...
 $ Rock: Factor w/ 5 levels "Argovian","Kimmeridgian",..: 3 2 3 2 5 5 5 1 1 3 ...
 $ Land: Factor w/ 4 levels "Forest","Pasture",..: 3 2 2 3 3 3 3 3 3 3 ...
 $ Cd  : num  1.74 1.33 1.61 2.15 1.56 ...
```

```
$ Cu  : num  25.72 24.76 8.88 22.7 34.32 ...
$ Pb  : num  77.4 77.9 30.8 56.4 66.4 ...
$ Co  : num  9.32 10 10.6 11.92 16.32 ...
$ Cr  : num  38.3 40.2 47 43.5 38.5 ...
$ Ni  : num  21.3 29.7 21.4 29.7 26.2 ...
$ Zn  : num  92.6 73.6 64.8 90 88.4 ...
```

3.1 Spatially-explicit objects

The data types for the `X` and `Y` fields in the data frame are `num`, i.e. numeric. These are indeed numbers, but of a special kind: they are **coördinates** in geographic space.

It is certainly possible to do some visualization and analysis in R with the data frame, but it is more elegant, and gives many more possibilities, if geographic data is explicitly recognized as such. This was the motivation behind the **R Spatial Project**, which resulted in the `sp` package [9] which provides **classes** (data types and methods for these) for spatial data.

The `sp` package adds a number of **spatial data types**, i.e. new **object classes**; these are then recognized by methods in other packages that are built on top of `sp`, most notably (for our purposes) the `gstat` package.

To take advantage of the power of an explicit spatial representation, we must convert the data frame to the most appropriate `sp` class.

Task 8 : Convert the `jura.all` dataframe to class `SpatialPointsDataFrame`. •

We do this by adding the computed coordinates to the data frame with the `coordinates` method; this automatically converts to the spatial data type defined by the `sp` package:

```
> class(jura.all)

[1] "data.frame"

> coordinates(jura.all) <- c("X", "Y")
> class(jura.all)

[1] "SpatialPointsDataFrame"
attr(,"package")
[1] "sp"
```

Note the use of the `c` ('catenate') method to make a list of the names of the two fields where the coördinates were stored, i.e. `X` and `Y`. These must be quoted, otherwise they would refer to variables named `X` and `Y`. Then the list is **assigned** to the `coordinates` method. This is a syntax we haven't seen before; the `coordinates` method can appear either on the right or the left of the assignment operator.

> **Note:** There are several other syntaxes for specifying the coördinates; it is commonly written with the **formula operator** `~`, which we haven't met yet (but will below, when we compute a trend surface):

```
coordinates(jura.all) <- ~ X + Y
```

Q6 : *What is now the data type of the* `jura.all` *object?* *Jump to A6* •

Task 9 : View the structure and data summary of the spatial object. •

As usual, the structure is displayed by the `str` method; we then summarize the object with the generic `summary` method and view the first few records in the dataframe with the `head` method:

```
> str(jura.all)

Formal class 'SpatialPointsDataFrame' [package "sp"] with 5 slots
  ..@ data       :'data.frame':        359 obs. of  9 variables:
  .. ..$ Rock: Factor w/ 5 levels "Argovian","Kimmeridgian",..: 3 2 3 2 5 5 5 1 1 3 ...
  .. ..$ Land: Factor w/ 4 levels "Forest","Pasture",..: 3 2 2 3 3 3 3 3 3 3 ...
  .. ..$ Cd  : num [1:359] 1.74 1.33 1.61 2.15 1.56 ...
  .. ..$ Cu  : num [1:359] 25.72 24.76 8.88 22.7 34.32 ...
  .. ..$ Pb  : num [1:359] 77.4 77.9 30.8 56.4 66.4 ...
  .. ..$ Co  : num [1:359] 9.32 10 10.6 11.92 16.32 ...
  .. ..$ Cr  : num [1:359] 38.3 40.2 47 43.5 38.5 ...
  .. ..$ Ni  : num [1:359] 21.3 29.7 21.4 29.7 26.2 ...
  .. ..$ Zn  : num [1:359] 92.6 73.6 64.8 90 88.4 ...
  ..@ coords.nrs : int [1:2] 1 2
  ..@ coords     : num [1:359, 1:2] 2.39 2.54 2.81 4.31 4.38 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : NULL
  .. .. ..$ : chr [1:2] "X" "Y"
  ..@ bbox       : num [1:2, 1:2] 0.491 0.524 4.92 5.69
  .. ..- 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

> head(jura.all@data)

          Rock    Land    Cd    Cu    Pb    Co    Cr    Ni    Zn
1    Sequanian  Meadow 1.740 25.72 77.36  9.32 38.32 21.32 92.56
2 Kimmeridgian Pasture 1.335 24.76 77.88 10.00 40.20 29.72 73.56
3    Sequanian Pasture 1.610  8.88 30.80 10.60 47.00 21.40 64.80
4 Kimmeridgian  Meadow 2.150 22.70 56.40 11.92 43.52 29.72 90.00
5   Quaternary  Meadow 1.565 34.32 66.40 16.32 38.52 26.20 88.40
6   Quaternary  Meadow 1.145 31.28 72.40  3.50 40.40 22.04 75.20

> summary(jura.all@data)

          Rock          Land           Cd               Cu
 Argovian    : 76   Forest : 51   Min.   :0.1350   Min.   :  3.550
 Kimmeridgian:124   Pasture: 82   1st Qu.:0.6525   1st Qu.: 10.470
 Sequanian   : 89   Meadow :218   Median :1.1000   Median : 17.200
 Portlandian :  6   Tillage:  8   Mean   :1.2874   Mean   : 23.585
 Quaternary  : 64                 3rd Qu.:1.6800   3rd Qu.: 26.920
                                  Max.   :5.1290   Max.   :166.400
       Pb               Co               Cr
 Min.   : 18.680   Min.   : 1.5500   Min.   : 3.320
 1st Qu.: 36.320   1st Qu.: 6.6600   1st Qu.:27.640
```

```
 Median : 46.800   Median : 9.8400   Median :34.800
 Mean   : 54.631   Mean   : 9.4379   Mean   :35.018
 3rd Qu.: 60.200   3rd Qu.:12.1000   3rd Qu.:41.460
 Max.   :300.000   Max.   :20.6000   Max.   :70.000
       Ni                Zn
 Min.   : 1.980   Min.   : 25.000
 1st Qu.:14.600   1st Qu.: 54.600
 Median :20.680   Median : 73.560
 Mean   :20.019   Mean   : 75.882
 3rd Qu.:25.380   3rd Qu.: 90.000
 Max.   :53.200   Max.   :259.840
```

The spatial object now has several **slots**, marked with the `@` symbol in the structure listing. These include:

- `@data`: the attributes data frame, without coördinates; methods such as `head` that work on dataframes now must name this slot;
- `@coords`: the coördinates;
- `@bbox`: the bounding box (i.e. rectangle enclosing the area).

Q7 : *What is the **spatial** information for this object?* *Jump to A7* •

Q8 : *What is the **non-spatial** information for this object?* *Jump to A8* •

Let's see one example of a spatial operation which requires an `sp` object, the `bbox` method.

Task 10 : Determine the **bounding box** of the observations, i.e. the coördinates inside which all the observations are found. •

```
> bbox(jura.all)

    min  max
X 0.491 4.92
Y 0.524 5.69
```

3.2 Saving R objects

We have done quite a bit of work to get the Jura dataset into the proper form. We don't want to have to repeat all this if we interrupt our session.

At the end of the session we can save our entire workspace; however it is possible to save one or more objects in their own files.

Task 11 : Save the `jura.all` object in a file, as an R object that can be loaded in later sessions. •

We use the `save` method, naming the object(s) to save and the file in which to save them. The file extension `.RData` is conventional for R data:

```
> save(jura.all, file = "Jura.RData")
```

There should now be a file named `Jura.RData` in your working directory.

3.3 Answers

A4 : `X, Y` *and* `Cd, Cu, Pb, Co, Cr, Ni, Zn` *are numeric (data type* `num`*;* `Rock` *and* `Land` *are integers (data type* `int`*).* *Return to Q4* •

A5 : *The metals (*`Cd` *etc.) are indeed numeric values; they are attributes in feature space. The coördinates* `X` *and* `Y` *are also numbers, but they refer to geographic space. However,* `Rock` *and* `Land` *are not really numbers, they are numeric* ***codes*** *that refer to specific rock and land types.* *Return to Q5* •

A6 : *The data type is now* `SpatialPointsDataFrame`*; this is defined in the* `sp` *package.* *Return to Q6* •

A7 : *Spatial information includes (1) a* ***bounding box****, i.e. limits of the the coördinates; (2) the geographic* ***projection****, in this case marked as* `NA` *("not applicable") because we haven't informed R about it.* *Return to Q7* •

A8 : *Non-spatial information is the data frame less the coördinates, i.e. all the feature (attribute) space information: the two categorical variables and the seven metal contents.* *Return to Q8* •

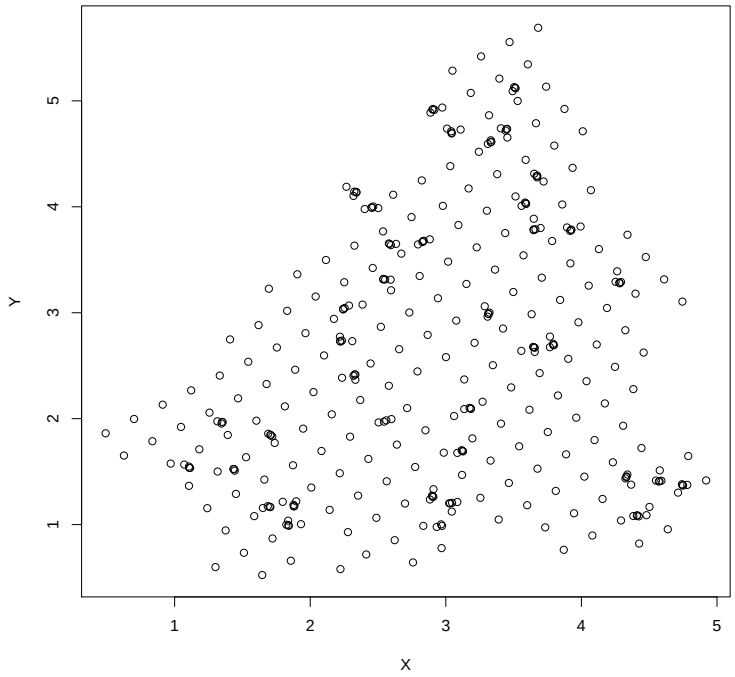
4 Spatial distribution of point observations

The first visualization of any spatial data set is the **locations** of the observations, i.e. their **distribution in space**.

Task 12 : Plot the spatial locations of the observations in the study area. •

The `plot` base graphics method is used to make a scatterplot of the coördinates, which are extracted with the `coordinates` method:

```
> plot(coordinates(jura.all))
```

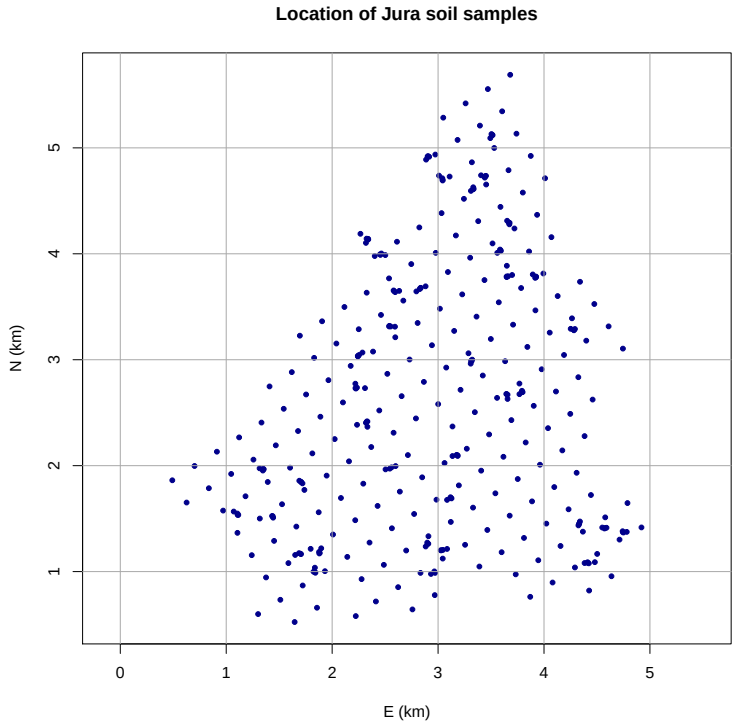


Note that R automatically scaled the axes to so that the plot is square; this results in a distorted map.

We can easily make this plot more attractive, adding:

- a title with the `main` graphics argument;
- axis labels with the `xlab` and `ylab` graphics argument;;
- colouring the points with the `col` graphics argument;
- specifying a filled circle as the printing character with the `pch` graphics argument;
- restoring the correct map aspect with the `asp` graphics argument;
- adding an overprinted grid with the `grid` method.

```
> plot(coordinates(jura.all), pch = 20, col = "darkblue",
+     asp = 1, main = "Location of Jura soil samples",
+     xlab = "E (km)", ylab = "N (km)")
> grid(lty = 1, col = "darkgray")
```



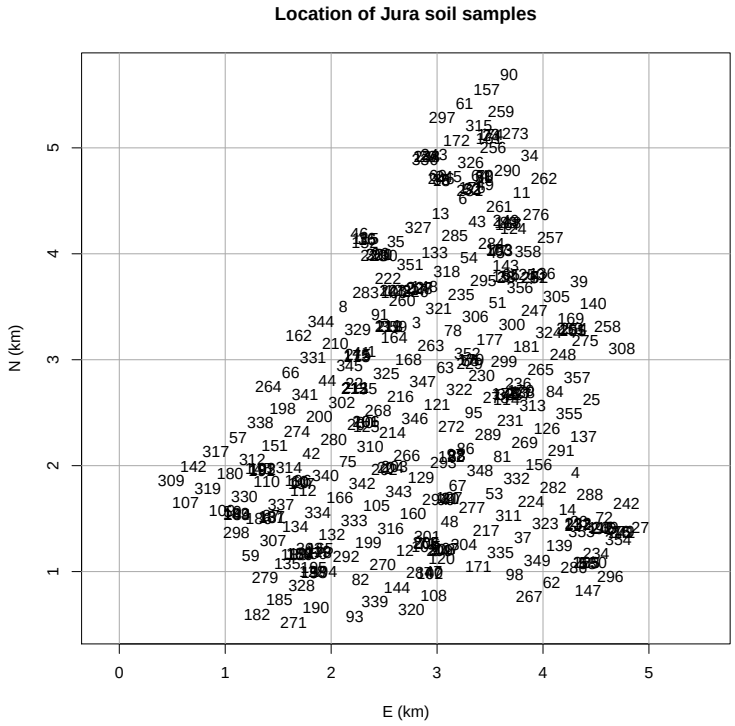
This is a typical R base graphics customization; see Rossiter [10, § 5.1], the online help for `plot` and `plot.default`, and Venables et al. [13, § 12.5]) for details of the various optional base graphics parameters.

> **Note:** Notice that the grid cells are 1 km by 1 km squares, and shown as square in the plot. This is because we specified that the **map aspect** (ratio of unit height to unit width) be unity, with the `asp=1` optional graphics argument. So, one km E is represented by the same plot distance as one km N. Without this, the `plot` method stretches the two axes to make a square; in this case the N dimensions are somewhat smaller than the E, so the map is distorted, as we saw in the previous figure.

Q9 : *Describe the spatial distribution of the Jura soil samples.* *Jump to A9* •

Optional: An interesting variation is to show the observation numbers, which are extracted from the data frame slot `jura.all@data` with the `row.names` method. Note the `type="n"` optional argument to `plot`; this sets up the plot frame (axes etc.) but doesn't plot any points. This allows us to print the numbers with the `text` graphics method.

```
> plot(coordinates(jura.all), type = "n", asp = 1, main = "Location of Jura soil samples",
+     xlab = "E (km)", ylab = "N (km)")
> grid(lty = 1, col = "darkgray")
> text(coordinates(jura.all), row.names(jura.all@data))
```



We leave the `jura.all` object in the workspace.

4.1 Answers

A9 : *Many are on the **regular grid** oriented approximately NE-SW; but there are several **clusters** of observations near some of the grid points.* *Return to Q9* •

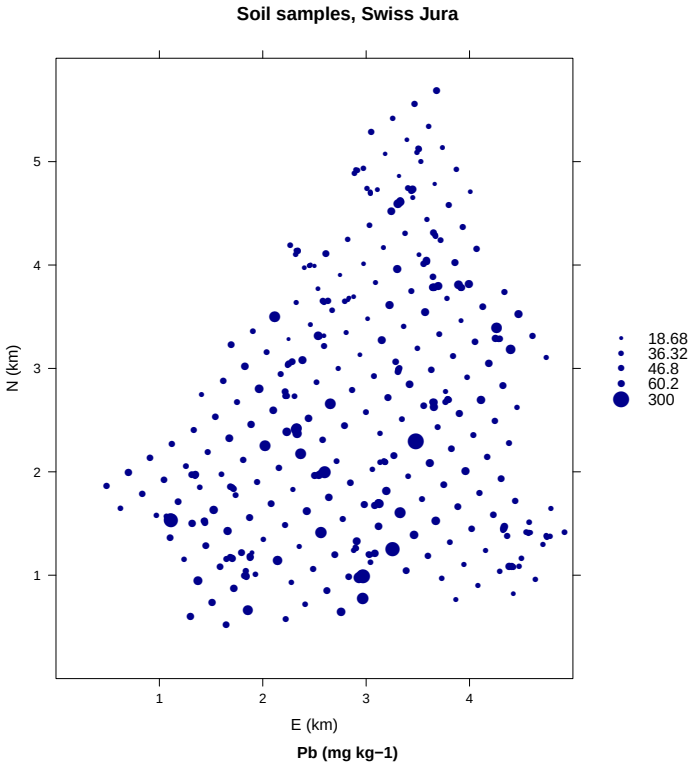
5 Postplots

The most basic visualization of spatial distribution in two dimensions is the **postplot**. It shows the location of each observation point in **geographic space**, with the value of the attribute in **feature space** shown by the symbol. For continuous variables, this is generally by making the **symbol size** proportional to the attribute value.

Task 13 : Make a postplot of the Pb values, showing the relative value by size. •

We use the `bubble` method of the `sp` package to make a so-called "bubble" plot, which is another name for a postplot where the values are shown by relative size. That is, there are different-sized "bubbles":

```
> print(bubble(jura.all, "Pb", main = "Soil samples, Swiss Jura",
+     sub = "Pb (mg kg-1)", col = "darkblue", scales = list(draw = TRUE),
+     maxsize = 2, xlab = "E (km)", ylab = "N (km)", xlim = c(0,
+         5), ylim = c(0, 6), aspect = "iso"))
```



Note: The `print` method here is actually the `print.trellis` method, i.e. a specialization of the generic `print` method, which is automatically called by `print` when the argument (here, the results of the `bubble` function of the `lattice` package) is a `trellis` object.

The `print.trellis` method is the default print method for objects of class `trellis` which are produced by calls to `lattice` functions like `xyplot` or `bubble`; the `spplot` method of the `sp` package also uses lattice graphics and hence `print.trellis`.

In interactive use it is called automatically when a trellis object is produced, so just using the `bubble` method should produce the output on your screen. However, to prepare this document with Sweave, the `print.trellis` method must be called explicitly.

Q10 : *Do you get a clear impression of the spatial distribution of Pb values from this plot?* *Jump to A10* •

As with all R graphics, there are a lot of optional arguments, giving you control over the visualization. In the above example we use:

- `main`: graph title;
- `sub`: graph subtitle (at bottom);
- `xlab`, `ylab`: axis text
- `xlim`, `ylim`: axis limits (lowest, highest values)
- `maxsize`: the character size of the largest symbol, relative to a default 1 computed to match the plot size;

- `scales`: instructions on how to draw tick marks and labels on the axes; `draw = TRUE` says to draw the axis.

> **Note:** Most of these are used throughout the `lattice` graphics package and are explained (to some degree) in the help for the `xyplot` method. Th explanations are not always so clear; experimentation is advised.

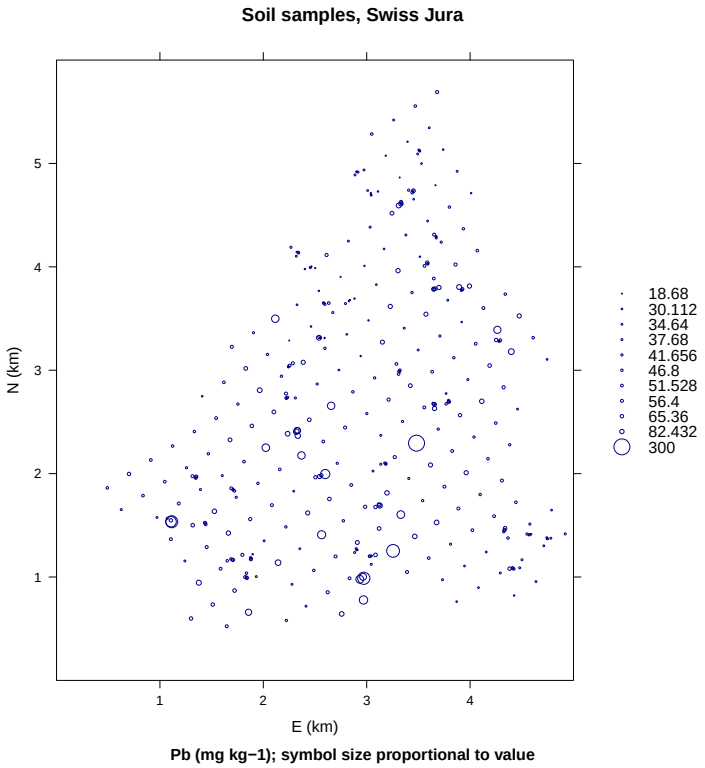
By default there are five symbol classes, with the breakpoints being the quartiles. These can be specified explicitly with the `key.entries` optional argument to `bubble`.

By default the "bubbles" are filled; they can be left open with the `fill=FALSE` optional argument.

A useful option to exaggerate the differences in size is the `do.sqrt=FALSE` optional argument. This scales the symbol size by the value, not its square root.

Here is the adjusted postplot:

```
> print(bubble(jura.all, "Pb", main = "Soil samples, Swiss Jura",
+     sub = "Pb (mg kg-1); symbol size proportional to value",
+     col = "darkblue", scales = list(draw = TRUE), maxsize = 2,
+     fill = FALSE, xlab = "E (km)", ylab = "N (km)", xlim = c(0,
+         5), ylim = c(0, 6), aspect = "iso", do.sqrt = FALSE,
+     key.entries = quantile(jura.all$Pb, seq(0, 1, by = 0.1))))
```



Q11 : *What is the difference in visualization between the* `do.sqrt=TRUE` *and* `do.sqrt=FALSE` *options?* *Jump to A11* •

Q12 : *What is the visualization advantage of open circles (*`fill=FALSE`*), compared to filled circles (*`fill=TRUE`*, the default)?* *Jump to A12* •

Q13 : *Looking at the postplots, can you detect any* ***regional trend****, i.e. systematic increase of attribute values in any direction across the whole region?* *Jump to A13* •

5.1 * Panel functions

This **optional** section shows how to add several objects to the same `lattice` graphics plot using **panel functions**. These are called within the function which is an argument to the `panel` graphics argument.

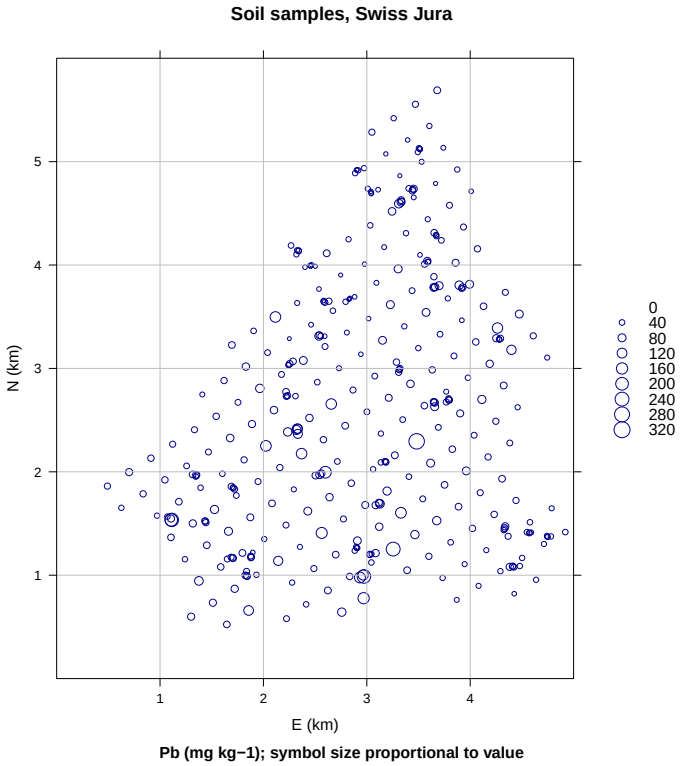
In this case we have two things to plot together:

1. The bubble plot itself, with the `panel.xyplot` method; note that `bubble` method of the `sp` package is just a wrapper around the underlying `xyplot` method of the `lattice` graphics package;
2. An overprinted grid, with the `panel.grid` method.

> **Note:** There are many panel functions; see `?panel.functions` or the help for any one of them, e.g. `?panel.grid` for details.

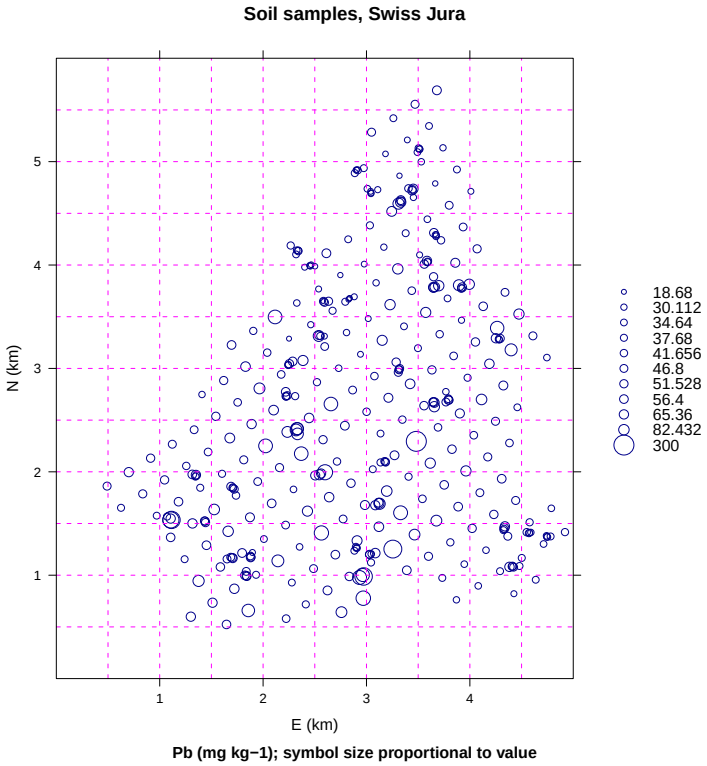
We'll also change the break points to every 40 mg kg^{-1} from 0 to the maximum, using the `seq` method to specify the sequence of values and the `key.entries` graphics argument:

```
> print(bubble(jura.all, "Pb", main = "Soil samples, Swiss Jura",
+     sub = "Pb (mg kg-1); symbol size proportional to value",
+     col = "darkblue", scales = list(draw = TRUE), maxsize = 2,
+     fill = F, xlab = "E (km)", ylab = "N (km)", xlim = c(0,
+         5), ylim = c(0, 6), aspect = "iso", panel = function(x,
+         y, ...) {
+         panel.xyplot(x, y, ...)
+         panel.grid(h = -1, v = -1, col = "gray", lty = 1)
+     }, key.entries = seq(0, 320, by = 40)))
```



The `h=-1, v=-1` argument to `panel.grid` is shorthand for "place the grid lines at the same locations as the axis ticks". If these are positive numbers they specify the number of grid lines in the two dimensions. Here we have a 5 by 6 km area, so to get grid lines every 500 m we specify 11 horizontally (leaving out the top, which already is drawn as part of the box) and 9 vertically.

```
> print(bubble(jura.all, "Pb", main = "Soil samples, Swiss Jura",
+     sub = "Pb (mg kg-1); symbol size proportional to value",
+     col = "darkblue", scales = list(draw = TRUE), maxsize = 2.5,
+     fill = F, xlab = "E (km)", ylab = "N (km)", xlim = c(0,
+         5), ylim = c(0, 6), aspect = "iso", panel = function(x,
+         y, ...) {
+         panel.xyplot(x, y, ...)
+         panel.grid(h = 11, v = 9, col = "magenta", lty = 2)
+     }, key.entries = quantile(jura.all$Pb, seq(0, 1,
+         by = 0.1))))
```



5.2 Answers

A10 : *Yes, the higher values in feature space are the larger circles.* *Return to Q10* •

A11 : *The* `do.sqrt=FALSE` *option exaggerates the extreme values; the* `do.sqrt=TRUE` *shows a more even sequence.* *Return to Q11* •

A12 : *With open symbols we can visualize all the values, if observations overlap or even if a smaller-valued observation is completely within a larger-valued one. With filled symbols the smaller-valued observation would be completely missed.* *Return to Q12* •

A13 : *The largest values are in the southern half of the region, but the trend, if any, is very weak.* *Return to Q13* •

6 Visualizing regional trends

We use a different dataset to visualize geographic trends. This is a small part of the dataset of Yemefack et al. [16]. This subset is supplied on the course CD, in directory `datasets\tcp`, as an R data file named `tcp.RData`; it is also on the course website.

Task 14 : Load the R data file `tcp.RData` into your working directory and examine its structure. •

First you have to copy it from the CD to your working directory, and then use the `load` method (the inverse of the `save` method we used above) to read in a file that is already in R's internal data format; we then use the `ls` method to list the objects in the workspace and the `str` method to see the structure of the new object.

```
> load("tcp.RData")
> ls()

[1] "jura.all" "tcp"

> str(tcp)

'data.frame':        147 obs. of  4 variables:
 $ UTM_E : num  702638 701659 703488 703421 703358 ...
 $ UTM_N : num  326959 326772 322133 322508 322846 ...
 $ clay35: num  78 80 66 61 53 57 70 72 70 62 ...
 $ pH35  : num  4.8 4.4 4.2 4.54 4.4 ...
```

There are four fields:

UTM_E : Easting

UTM_N : Northing

clay35 : clay content of the fine earth, % by weight, 30–50 cm layer

pH35 : reaction of the soil in 1:1 (by volume) water, 30–50 cm layer

We will look at the regional trend of the subsoil clay content.

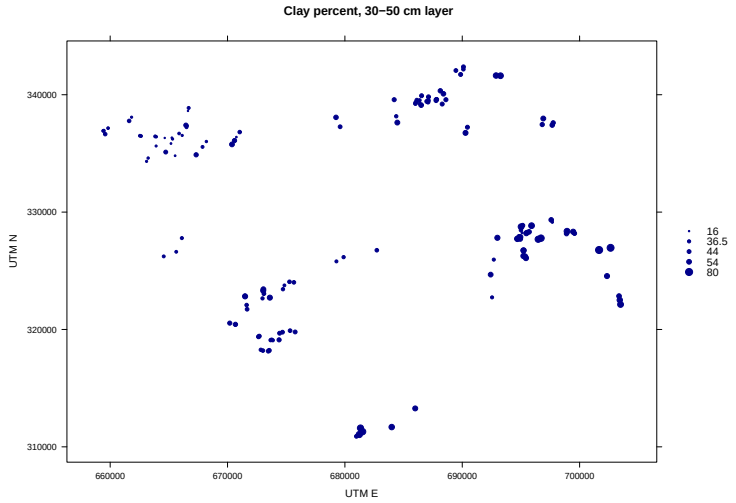
Task 15 : Convert the `tcp` data frame to a spatial object. •

```
> coordinates(tcp) <- c("UTM_E", "UTM_N")
> str(tcp)

Formal class 'SpatialPointsDataFrame' [package "sp"] with 5 slots
  ..@ data       :'data.frame':        147 obs. of  2 variables:
  .. ..$ clay35: num [1:147] 78 80 66 61 53 57 70 72 70 62 ...
  .. ..$ pH35  : num [1:147] 4.8 4.4 4.2 4.54 4.4 ...
  ..@ coords.nrs : int [1:2] 1 2
  ..@ coords     : num [1:147, 1:2] 702638 701659 703488 703421 703358 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : NULL
  .. .. ..$ : chr [1:2] "UTM_E" "UTM_N"
  ..@ bbox       : num [1:2, 1:2] 659401 310897 703488 342379
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2] "UTM_E" "UTM_N"
  .. .. ..$ : chr [1:2] "min" "max"
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slots
  .. .. ..@ projargs: chr NA
```

Notice how we used the **formula** syntax to specify the coördinates.

Task 16 : Make a postplot of the clay content. •

```
> bubble(tcp, zcol = "clay35", do.sqrt = F, pch = 20, col = "darkblue",
+     scales = list(draw = TRUE), maxsize = 2, xlab = "UTM E",
+     ylab = "UTM N", aspect = "iso", main = "Clay percent, 30-50 cm layer")
```



Q14 : *Describe the regional trend of clay content in the 30-50 cm layer. Does it appear that a plane could fit this? A bowl?* *Jump to A14* •

In the next exercise we will see how to fit the trend and evaluate the goodness-of-fit. Here we are just trying to visualize the trend and evaluate the fit by eye.

Task 17 : Compute a second-order trend surface of the clay content; display it as a raster map with a resolution of 500×500m, with the postplot of the point observations on top of the raster. •

There are several steps here:

1. Create the raster;
2. Compute the trend surface on the raster;
3. Display the computed trend surface;
4. Superimpose the observations.

To create the grid, we first determine the bounding box:

```
> bbox(tcp)
```

```
         min    max
UTM_E 659401 703488
UTM_N 310897 342379
```

Then we make a grid with the `expand.grid` method, with limits somewhat bigger than the bounding box. Note that we must name the coördinates the **same** as the observation points:

```
> g <- expand.grid(UTM_E = seq(659000, 704000, by = 500),
+     UTM_N = seq(310000, 343000, by = 500), KEEP.OUT.ATTRS = F)
> str(g)

'data.frame':        6097 obs. of  2 variables:
 $ UTM_E: num  659000 659500 660000 660500 661000 ...
 $ UTM_N: num  310000 310000 310000 310000 310000 310000 310000 310000 310000 310000 ...

> coordinates(g) <- c("UTM_E", "UTM_N")
> str(g)

Formal class 'SpatialPoints' [package "sp"] with 3 slots
  ..@ coords     : num [1:6097, 1:2] 659000 659500 660000 660500 661000 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : NULL
  .. .. ..$ : chr [1:2] "UTM_E" "UTM_N"
  ..@ bbox       : num [1:2, 1:2] 659000 310000 704000 343000
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2] "UTM_E" "UTM_N"
  .. .. ..$ : chr [1:2] "min" "max"
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slots
  .. .. ..@ projargs: chr NA
```

Q15 : *How many grid cells are there in this raster?* *Jump to A15* •

To compute a trend surface on a raster, we use the `krige` method of the `gstat` package. Despite its name, `krige` not only interpolates by kriging (as we will see in later exercises) but also can interpolate by inverse distance weighting (IDW) and trend surfaces. The **syntax** of `krige` is somewhat complicated, but you will get used to it as we go along

```
> ts2 <- krige(clay35 ~ I(UTM_E^2) + I(UTM_N^2) + UTM_E +
+     UTM_N, loc = tcp, newdata = g, model = NULL)

[ordinary or weighted least squares prediction]
```

The arguments used for `krige` here are:

1. The **formula** showing the **linear model** of any trend surface; here it is a second-order model. This is the same formula syntax as is used in many other contexts, most notably the `lm` ("linear models") method we will meet in a later lesson;

   (The formula operators are explained in Rossiter [10, § 4.17] and Venables et al. [13, § 11]);

2. `loc` (short for `location`), the object which stores the coördinates of the point observations; here this is `tcp`;

3. `data`, the object with the attribute data; if this is not named (as in this example) the data is in the same object as the location;

4. `newdata`, the object with coördinates of the locations where new data should be computed; here it is the grid `g`;

5. `model`, the model of **local spatial dependence**; we will use this extensively in kriging, but for a trend surface this is `NULL`, i.e. we do not consider local dependence, only a **regional trend**.

   (Note that **local** spatial dependence will be visualized in §7 later in this exercise.)

The syntax of the **formula**, here `clay35 ~ I(UTM_E^2) + I(UTM_N^2) + UTM_E + UTM_N`, needs some more comment:

- The **tilde** `~` separates the formula into a **left-hand side** and a **right-hand side**; the formula can be read "left-hand side depends on (is modelled by) right hand side";
- The **left-hand side** is the attribute being modelled, in this case the clay content `clay35`;
- The **right-hand side** is a formula combining the predictors, i.e. the attributes or coördinates which model the left-hand side;
- Here the two predictors are the coördinates `UTM_E` and `UTM_N`;
- Since we are modelling the clay content as a **second-order** trend surface, we have to include the predictors both in their **linear** form (e.g. `UTM_E`) and in their **quadratic** form (e.g. `I(UTM_E^2)`, where the caret `^` is the operator for exponentiation;
- The quadratic form **must** be written inside the `I` ("as is") method, because inside formulas the `^` operator does not mean exponentiation, it has another purpose.

Sorry for the complexity, but it is all needed for more sophisticated modelling.

We examine the structure of the trend surface:

```
> str(ts2)

Formal class 'SpatialPointsDataFrame' [package "sp"] with 5 slots
  ..@ data       :'data.frame':        6097 obs. of  2 variables:
  .. ..$ var1.pred: num [1:6097] 52 52.2 52.4 52.5 52.7 ...
  .. ..$ var1.var : num [1:6097] 110 109 108 107 106 ...
  ..@ coords.nrs : int [1:2] 1 2
  ..@ coords     : num [1:6097, 1:2] 659000 659500 660000 660500 661000 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : NULL
  .. .. ..$ : chr [1:2] "UTM_E" "UTM_N"
  ..@ bbox       : num [1:2, 1:2] 659000 310000 704000 343000
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2] "UTM_E" "UTM_N"
  .. .. ..$ : chr [1:2] "min" "max"
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slots
  .. .. ..@ projargs: chr NA
```

Q16 : *Where in the computed trend surface* `ts2` *is the predicted clay content?* *Jump to A16* •

Now we can display the trend surface. For this we use the `levelplot` method from the `lattice` package, which is an implementation of the Trellis graphics system [11, 8].

Task 18 : Load the `lattice` package. •

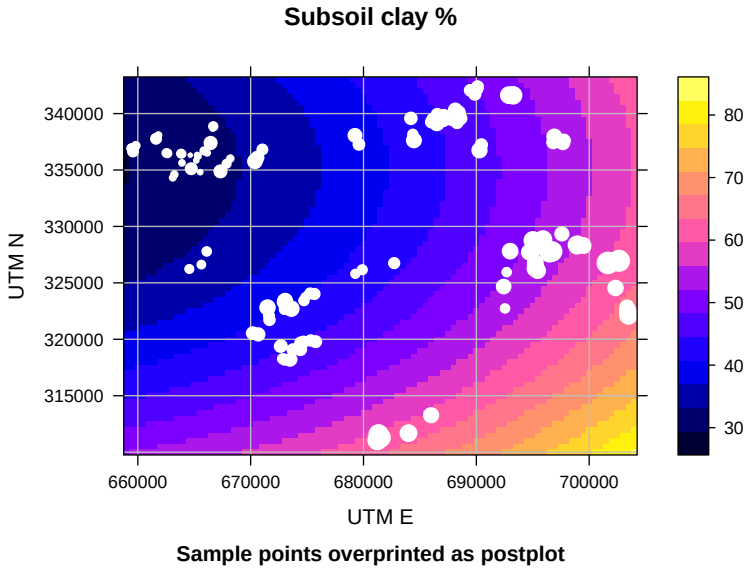
```
> library(lattice)
```

This `levelplot` method display some attribute against coördinates specified in a **model formula** as explained above. We also use the **panel functions** explained above in §5.1. Here you can see there are three graphics objects placed on the same panel:

1. The levelplot itself, with the `panel.levelplot` method;
2. An overprinted grid, with the `panel.grid` method;
3. The observation points, with the `panel.points` method.

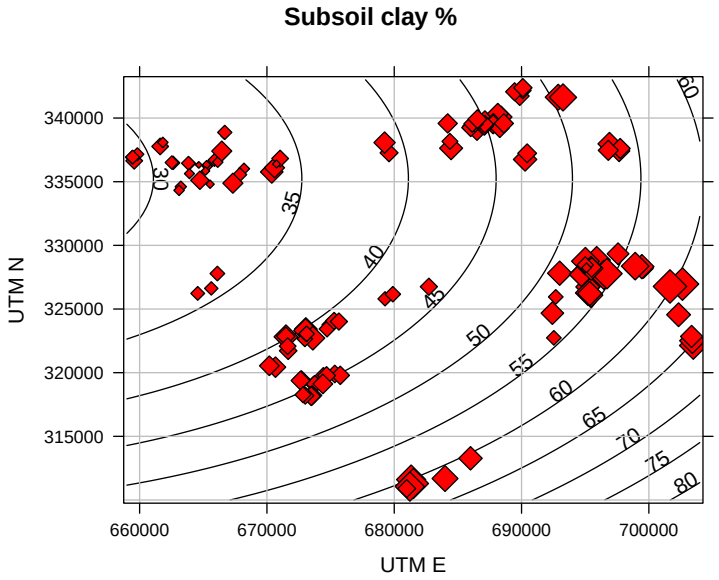
All of these are called within the a function which is an argument to the `panel` graphics argument.

```
> print(levelplot(var1.pred ~ UTM_E + UTM_N, as.data.frame(ts2),
+     col.regions = bpy.colors(64), asp = "iso", main = "Subsoil clay %",
+     sub = "Sample points overprinted as postplot", xlab = "UTM E",
+     ylab = "UTM N", panel = function(x, y, z, ...) {
+         panel.levelplot(x, y, z, ...)
+         panel.grid(h = -1, v = -1, col = "gray", lty = 1)
+         panel.points(coordinates(tcp), cex = 3 * tcp$clay35/max(tcp$clay35),
+             pch = 20, col = "white")
+     }))
```



Another way to visualize this is with the very similar `contourplot` method, also from the `lattice` package. We use the `at` graphics argument to specify the contour intervals. Note also the different graphics treatment of the overprinted points: here black (`col="black"`) diamonds (`pch=23`) with red fill (`fill="red"`), rather than white filled circles as in the levelplot:

```
> print(contourplot(var1.pred ~ UTM_E + UTM_N, as.data.frame(ts2),
+     asp = "iso", main = "Subsoil clay %", at = seq(0,
+         100, by = 5), sub = "Sample points overprinted as postplot",
+     xlab = "UTM E", ylab = "UTM N", panel = function(x,
+         y, z, ...) {
+         panel.contourplot(x, y, z, ...)
+         panel.grid(h = -1, v = -1, col = "gray", lty = 1)
+         panel.points(coordinates(tcp), cex = 2.5 * tcp$clay35/max(tcp$clay35),
+             pch = 23, col = "black", fill = "red")
+     }))
```



Sample points overprinted as postplot

Q17 : *Describe the shape of the second-order trend surface.* *Jump to A17*
•

Q18 : *How well does the second-order trend surface appear to match the observations?* *Jump to A18* •

Task 19 : Remove the workspace objects created in this section. •

```
> rm(tcp, g, ts2)
```

6.1 Answers

A14 : *The lowest values are in the NW; they increase both to the E and S. It looks bowl-shaped.* *Return to Q14* •

A15 : 6097 *Return to Q15* •

A16 : *In the* `var1.pred` *("variable number 1 prediction") field of the* `@data` *slot.* *Return to Q16* •

A17 : *An elogated (stretched) bowl, lowest in the NW corner and increasing both*

to the E and S; more rapid increase towards the S, hence the elongation towards the E. *Return to Q17* •

A18 : *It fits the general pattern quite well; however in each of the clusters we can see some exceptions to the general trend.* *Return to Q18* •

7 Visualizing local spatial dependence

A post-plot (§5) sometimes shows a **trend**, as seen in the previous section. However, it sometimes shows a **local spatial dependence**, i.e. close-by points in **geographic** space are also close-by in **attribute** space. So far we only have had a subjective impression of this from looking at the post-plot, and there was no way to determine the **range** or **strength** of the local spatial dependence. For this we can use (semi-)**variograms**.

7.1 The variogram cloud

We begin with the **variogram cloud**, which shows the semivariances between all point pairs. Recall that the semivariance is defined as:

$$\gamma(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{2}[z(\mathbf{x}_i) - z(\mathbf{x}_j)]^2$$

where $\mathbf{x}$ is a geographic point and $z(\mathbf{x})$ is its **attribute value**.

This pair of points is separated by a vector $\mathbf{h}$; to begin with we just consider this vector as the **Euclidean distance** between the points:

$$\mathbf{h} = ||\mathbf{x}_i, \mathbf{x}_j|| = \sqrt{\sum_{k=1}^{n} (x_{i,k} - x_{j,k})^2}$$

where n is the number of dimensions (in this example, 2).

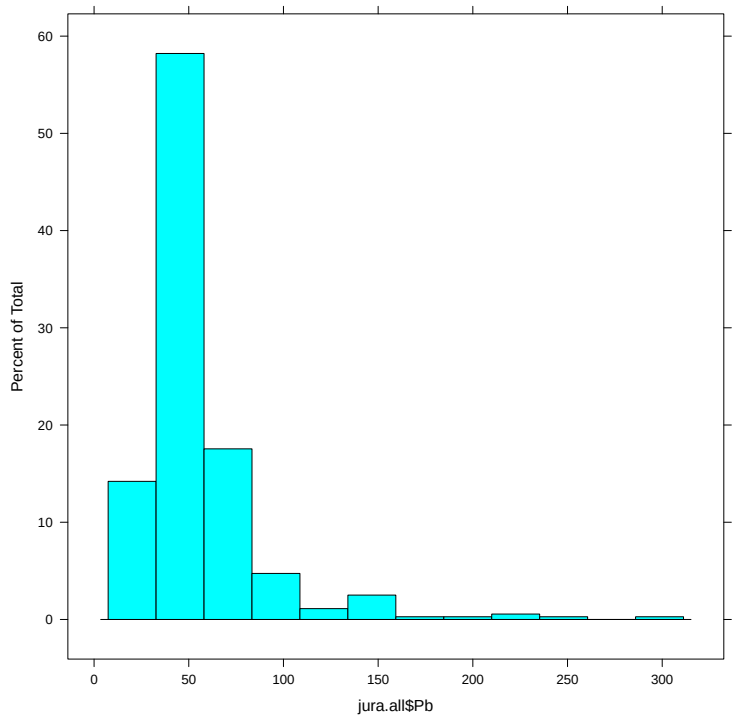
Since the differences between values are **squared**, in a **skewed** distribution of the attribute value, the few high values will have an excessive effect on the variogram. It is common to **transform** such attributes to a more symmetric form.

In Exercise 1, §7, we examined the distribution of Pb values in the Meuse river data set; now we do the same for the Jura soil samples:

Task 20 : Visualise the distribution of Pb in the Jura soil samples. •

We take this opportunity to introduce the `lattice` version of the histogram, the `histogram` method. This is very much like the base graphics `hist` method but has different options and can be used easily in multivariate visualisation.

```
> histogram(jura.all$Pb, nint = 12)
```



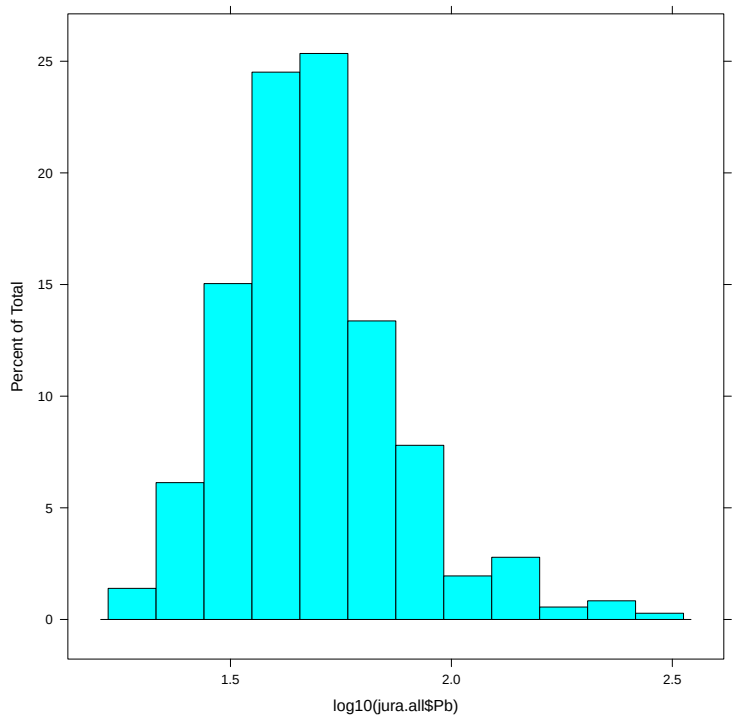
Q19 : *Is this distribution symmetric or skewed?* *Jump to A19* •

A right-skewed distribution like this one can often be made more symmetric with the **logarithmic transformation**.

Task 21 : Visualise the distribution of the **base-10 logarithm** Pb in the Jura soil samples. •

The logarithm is computed with the `log` method; the base-10 logarithm with the `log10` method. The advantage of base-10 logarithms is that we can back-transform easily, since the integer logarithms represent powers of 10.

```
> histogram(log10(jura.all$Pb), nint = 12)
```



Q20 : *Is this distribution symmetric or skewed?* *Jump to A20* •

Now that we have a symmetric distribution, we can compute semivariances.

Task 22 : Compute the **semivariance** between the first two observations. •

We use the `diff` method as a shorthand for subtraction. Note how the `^` exponentiation operator is **vectorized**, i.e. applied to each of the two elements of the vector `jura.all$Pb[1:2]` separately.

```
> jura.all[1:2, ]

     coordinates         Rock    Land    Cd    Cu    Pb    Co    Cr
1 (2.386, 3.077)    Sequanian  Meadow 1.740 25.72 77.36  9.32 38.32
2 (2.544, 1.972) Kimmeridgian Pasture 1.335 24.76 77.88 10.00 40.20
     Ni    Zn
1 21.32 92.56
2 29.72 73.56

> jura.all$Pb[1:2]

[1] 77.36 77.88

> log10(jura.all$Pb[1:2])

[1] 1.8885165 1.8914259

> diff(log10(jura.all$Pb[1:2]))

[1] 0.0029094818
```

```
> diff(log10(jura.all$Pb[1:2])^2)

[1] 0.010997674

> 0.5 * (diff(log10(jura.all$Pb[1:2]))^2)

[1] 4.2325421e-06
```

This last result is the semi-variance.

Q21 : *What is the semi-variance between the base-10 logarithms of the Pb values for the first two observations?* *Jump to A21* •

Task 23 : Compute the metric **distance** between the first two observations. •

```
> coordinates(jura.all)[1:2, ]

          X     Y
[1,] 2.386 3.077
[2,] 2.544 1.972

> diff(coordinates(jura.all)[1:2, "X"])

[1] 0.158

> diff(coordinates(jura.all)[1:2, "X"])^2

[1] 0.024964

> diff(coordinates(jura.all)[1:2, "Y"])

[1] -1.105

> diff(coordinates(jura.all)[1:2, "Y"])^2

[1] 1.221025

> sqrt(diff(coordinates(jura.all)[1:2, "X"])^2 + diff(coordinates(jura.all)[1:2,
+     "Y"])^2)

[1] 1.1162388
```

Q22 : *How far apart are the first two observations?* *Jump to A22* •

We now want to repeat these calculations for **all point-pairs** in the data set, and then graph them.

Task 24 : Compute the number of point-pairs in this dataset. •

We use the `dim` method to get the dimensions of the dataset; the first element of this list is the number of rows, i.e. observations. Then the formula is:

$$[n \cdot (n-1)]/2$$

```
> (dim(jura.all)[1] * (dim(jura.all)[1] - 1))/2

[1] 64261
```

Q23 : *How many point-pairs are there?* *Jump to A23* •

The `variogram` method of the `gstat` package, with optional argument `cloud=T`, computes the semivariances and distances for all of these:

Task 25 : Compute the variogram cloud of the base-10 logarithm of the Pb values and display the results for the first few point-pairs. •

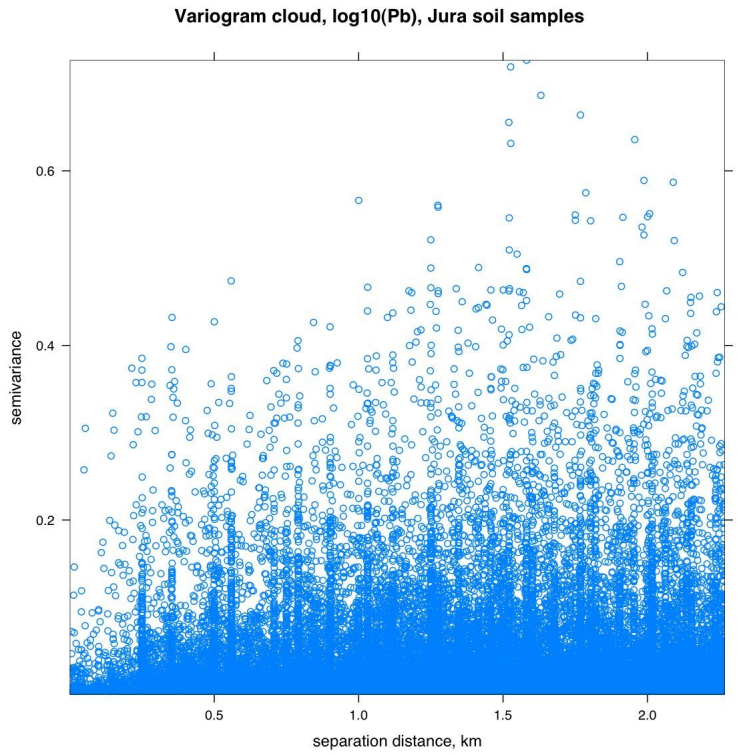
```
> vc <- variogram(log10(Pb) ~ 1, loc = jura.all, cloud = T)
> head(vc)

        dist         gamma dir.hor dir.ver   id left right
1 1.11623877 4.2325421e-06       0       0 var1    2     1
2 0.50014098 7.9986298e-02       0       0 var1    3     1
3 1.39992643 8.1154224e-02       0       0 var1    3     2
4 2.23669846 9.4170461e-03       0       0 var1    4     1
5 1.76443107 9.8205682e-03       0       0 var1    4     2
6 2.06213409 3.4513103e-02       0       0 var1    4     3
```

Note the **formula** `log10(Pb) ~ 1` which specifies the form of the spatial dependence. The right-hand side `1` means to model the left-hand side `log10(Pb)` only as **local** spatial dependence, i.e. without any trend or other explanatory factors. In a later exercise we will see situations where the right-hand side models both local and regional dependence.

Q24 : *What is the semi-variance and distance between observations 1 and 2, as computed by the* `variogram` *method of the* `gstat` *package?* *Jump to A24* •

Task 26 : Plot the variogram cloud cloud of the base-10 logarithm of the Pb values. •

```
> plot(vc, main = "Variogram cloud, log10(Pb)", xlab = "separation distance, km")
```



Q25 : *Describe the overall pattern of semi-variances vs. separations.* *Jump to A25* •

Q26 : *There is some vertical "striping" of point pairs, e.g. at 0.25 km, 0.5 km, ≈ 0.707 km. Explain this from the spatial distribution of the observations.* *Jump to A26* •

7.2 The empirical variogram

The variogram cloud shows **all** point-pairs, but is clearly far too detailed to be useful for examining the general pattern of spatial dependence. For that we use the **empirical** variogram, which organizes the cloud into bins, like a histogram. And just like the histogram we have lots of options.

Task 27 : Compute the default empirical variogram of the base-10 logarithm of the Pb values and list it. •

We use the same `variogram` method of the `gstat` package, but without the `cloud=T` optional argument:

```
> v <- variogram(log10(Pb) ~ 1, loc = jura.all)
> print(v)

     np        dist       gamma dir.hor dir.ver   id
1   442 0.068586139 0.017122607       0       0 var1
2  1074 0.241666170 0.029133051       0       0 var1
```

```
3  1355 0.375441718 0.030139988       0       0 var1
4  2240 0.530222674 0.033467424       0       0 var1
5  2044 0.691275796 0.031947445       0       0 var1
6  2810 0.838972590 0.033689144       0       0 var1
7  2555 0.996701472 0.033847360       0       0 var1
8  2841 1.124313631 0.037649135       0       0 var1
9  3836 1.287400668 0.037476127       0       0 var1
10 3051 1.442574514 0.039668342       0       0 var1
11 3447 1.579720539 0.040094959       0       0 var1
12 3732 1.741068281 0.036854731       0       0 var1
13 3101 1.888856181 0.038330998       0       0 var1
14 3231 2.036956126 0.039319359       0       0 var1
15 3286 2.192333772 0.039134177       0       0 var1
```

Q27 : *For the first bin (closest separation), what is the **number of point-pairs**, the **average separation**, and the **average semi-variance**?* *Jump to A27* •

Q28 : *Observations 1 and 2 are separated by 1.11624 km (see above); in which bin does this point-pair fall?* *Jump to A28* •

> **Note:** The default method to estimate the "average" semi-variance in a bin is with the ordinary arithmetic mean. If optional argument `cressie=T` is given, Cressie's robust variogram estimate is computed; the help page for `variogram` gives the reference.

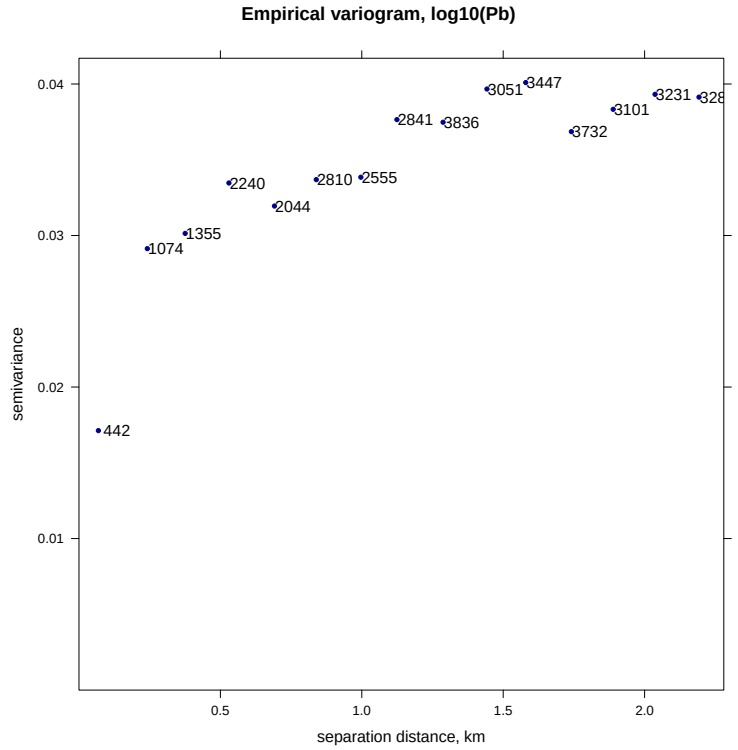
Task 28 : Plot the empirical variogram, also showing the number of point-pairs that contributed to each estimate. •

We call the `plot` graphics method to plot the variogram object `v`.

> **Note:** Because of the object-oriented R class system, R can determine that `v` has class `gstatVariogram`, and the call to the generic `plot` method is automatically replaced with a call to the `plot.gstatVariogram` method. This shows the power of object-oriented languages.
>
> To see the class of this object:
>
> ```
> > class(v)
>
> [1] "gstatVariogram" "data.frame"
> ```

We use the optional `plot.numbers=T` argument to see the number of point-pairs; we also add a title, change the symbol colour and x-axis label:

```
> print(plot(v, plot.numbers = T, main = "Empirical variogram, log10(Pb)",
+     xlab = "separation distance, km", col = "darkblue",
+     pch = 20))
```



Q29 : *Describe the overall pattern of semi-variances vs. separations. What are the approximate* ***sill****,* ***range*** *and* ***nugget****?* *Jump to A29* •

As with a histogram, there is a default computation of the bin width; this is 1/15 of the maximum separation. There is also a default maximum separation; this is 1/3 of the maximum separation of the entire data set, i.e. the furthest distance between any point pairs.

Q30 : *Looking at the spatial distribution of the samples (§4), what is the approximate maximum separation in this data set? What is then 1/3 of this?* *Jump to A30* •

In a later lesson we will see how to decide on an appropriate cutoff and number of bins; here we just see how to specify them for visualization. Let's look at a finer resolution for the shorter separations:

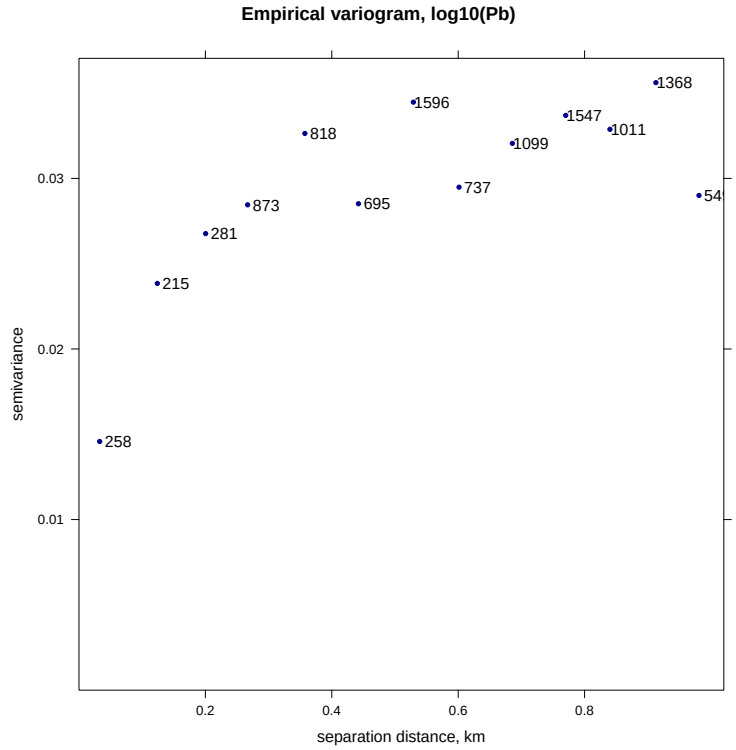
Task 29 : Compute and plot the empirical variogram for the base-10 logarithm of Pb with a cutoff 1 km and 80 m wide bins. •

The cutoff (maximum separation) is specified with the `cutoff` optional argument, the bin width with the `width` optional argument to the `variogram` method:

```
> v.08 <- variogram(log10(Pb) ~ 1, loc = jura.all, cutoff = 1,
+     width = 0.08)
> print(v.08)
```

```
   np        dist       gamma dir.hor dir.ver   id
1  258 0.032795793 0.014579025       0       0 var1
2  215 0.123960083 0.023843849       0       0 var1
3  281 0.200432011 0.026763673       0       0 var1
4  873 0.266864883 0.028450561       0       0 var1
5  818 0.357455593 0.032634730       0       0 var1
6  695 0.442230471 0.028513872       0       0 var1
7 1596 0.529125811 0.034470608       0       0 var1
8  737 0.601342758 0.029486156       0       0 var1
9 1099 0.685708964 0.032054914       0       0 var1
10 1547 0.770174468 0.033693325       0       0 var1
11 1011 0.840118163 0.032878238       0       0 var1
12 1368 0.912869641 0.035614075       0       0 var1
13  549 0.981037293 0.029000762       0       0 var1
```

```
> print(plot(v.08, pl = T, main = "Empirical variogram, log10(Pb)",
+     xlab = "separation distance, km", col = "darkblue",
+     pch = 20))
```



Q31 : *Describe the differences between this and the default variogram.* *Jump to A31* •

We now remove the workspace objects created in this section:

```
> rm(v, v.08, vc)
```

7.3 Answers

A19 : *Heavily right-skewed.* *Return to Q19* •

A20 : *Almost symmetric.* *Return to Q20* •

A21 : $4.2325 \cdot 10^{-6} \cdot log_{10}\text{Pb}^2$, *i.e.* 0.0000042325 *Return to Q21* •

A22 : 1.1162km *Return to Q22* •

A23 : *64,261* *Return to Q23* •

A24 : *The point-pairs are identified by the* `left` *and* `right` *fields. In this output the first line has* `left = 2` *and* `right = 1`*; this is then the first point-pair which we computed above. The results are in fields* `dist` *(separation) and* `gamma` *(semivariance), here 1.1162 and 4.2325e-06, respectively.*

These answers are exactly the same as we computed directly from the two points.
Return to Q24 •

A25 : *Semivariances generally increase with separation.* *Return to Q25* •

A26 : *There are many point-pairs with the exact same separation; these are pairs of points on the regular grid.* *Return to Q26* •

A27 : *The number of point-pairs is shown in field* `np` *as 442. The average separation is shown in field* `dist` *as 0.0686 km. The average semi-variance is shown in field* `gamma` *as 0.0171* $log_{10}\text{Pb}^2$. *Return to Q27* •

A28 : *The end points of the bins are not given, only the average separations. This point-pair is likely in bin 8, with average separation 1.1243 km.* *Return to Q28* •

A29 : *The average semi-variance increases with separation; this is abrupt at first and then approaches a sill. Estimated variogram parameters are: sill =* 0.04 $(log_{10}\text{Pb})^2$, *range* 1.5 km, *nugget* 0.012 $(log_{10}\text{Pb}^2)$. *Return to Q29* •

A30 : *Observations 90 and and 182 look like the furthest separation. Visually this appears to be about 5.5 km; it is actually:*

```
> (maxdist <- sqrt(diff(coordinates(jura.all)[c(90, 182),
+     "X"])^2 + diff(coordinates(jura.all)[c(90, 182),
+     "Y"])^2))
```

```
[1] 5.6198471
```

In this example, note the use of `c` *method to make a list of the two observation numbers we wanted to compare. Then one-third of this is*

```
> 5.5/3
```

```
[1] 1.8333333
```

km, approximately, or

```
> maxdist/3
```

```
[1] 1.8732824
```

km, exactly.

Each of the 15 bins is then

```
> maxdist/(3 * 15)
```

```
[1] 0.12488549
```

km wide.

Clean up:

```
> rm(maxdist)
```

Return to Q30 •

A31 *:*

- *There are fewer point-pairs in each bin of the shorter-range variogram;*
- *The shorter-range variogram is more erratic, due to the smaller number of point-pairs in each estimate;*
- *The shorter-range variogram doesn't quite reach a definite sill;*
- *It is easier to estimate the nugget in the shorter-range variogram, since there is a bin near the origin (zero separation).*

Return to Q31 •

8 Visualizing anisotropy

In the previous sections we have assumed that the spatial dependence is the same in all directions; this is called **isotropic**, from the Greek "iso-" (same) + "tropic" (direction). However, the autocorrelated spatial process that produced the dependence may be stronger in one direction than in others. This type of spatial dependence is called **anisotropic**, from the Greek "an-" (not) + isotropic. In practice, we consider the case where there is one direction of

strongest spatial dependence, with the weakest spatial dependence at right angles (orthogonal) to it.

We will use two visualization methods to detect anisotropy:

1. the **variogram surface** (also called "variogram map");
2. **directional variograms**

To appreciate these visualization methods we need a dataset with strong anisotropy; a good choice is the Meuse soil pollution dataset we saw in Exercise 1.

Task 30 : Load the `meuse` sample dataset from the `sp` package and convert it from a dataframe to a spatial object by specifying its coördinates. •

```
> data(meuse)
> coordinates(meuse) <- ~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 ...
  .. ..$ lead   : num [1:155] 299 277 199 116 117 137 132 150 133 80 ...
  .. ..$ zinc   : num [1:155] 1022 1141 640 257 269 ...
  .. ..$ elev   : num [1:155] 7.91 6.98 7.8 7.66 7.48 ...
  .. ..$ dist   : num [1:155] 0.00136 0.01222 0.10303 0.19009 0.27709 ...
  .. ..$ om     : num [1:155] 13.6 14 13 8 8.7 7.8 9.2 9.5 10.6 6.3 ...
  .. ..$ ffreq  : Factor w/ 3 levels "1","2","3": 1 1 1 1 1 1 1 1 1 1 ...
  .. ..$ soil   : Factor w/ 3 levels "1","2","3": 1 1 1 2 2 2 2 1 1 2 ...
  .. ..$ lime   : Factor w/ 2 levels "0","1": 2 2 2 1 1 1 1 1 1 1 ...
  .. ..$ landuse: Factor w/ 15 levels "Aa","Ab","Ag",..: 4 4 4 11 4 11 4 2 2 15 ...
  .. ..$ dist.m : num [1:155] 50 30 150 270 380 470 240 120 240 420 ...
  ..@ 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
```

We begin with the variogram surface.

Task 31 : Display a variogram surface of the natural logarithm of the Zn (zinc) content in the Meuse data set, up to 800m and with a bin width of 100 m. •

> **Note:** This variable is strongly right-skewed; the logarithmic transform makes it much more symmetrical and thus easier to model. If you wish, you can verify this with histograms of the original and transformed variables.

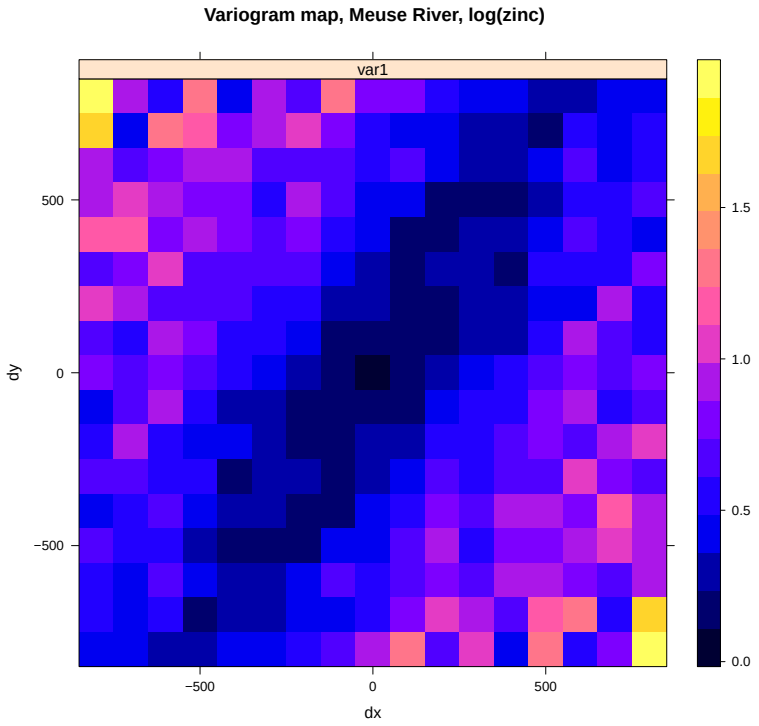
We use the `variogram` method with the optional `map=TRUE` argument; this produces a variogram surface instead of an omnidirectional empirical variogram. We then plot it with the generic `plot` method, which, for a variogrqam surface object, specializes to `plot.gstatVariogram`, which uses methods in the `lattice` graphics package.

Raster (grid) maps can be displayed with different **colour ramp** (series of colours) from the lowest to the highest values, giving different visualizations. For `lattice` graphics this is determined by the `col.regions` argument. There are several pre-defined methods that create colour ramps:

- `heat.colors`: from red to white-hot;
- `terrain.colors`:
- `topo.colors`: as used in topographic maps;
- `cm.colors`: cyan through magenta (pastels);
- `bpy.colors`: blue-purple-yellow; looks good printed
- `hsv`: custom ramp specifying hue, saturation and value;
- `rainbow`: custom ramp from a given start to stop colour;
- `gray`: gray scale

You can experiment with these to see which visualization you prefer. We start with `bpy.colors`:

```
> print(plot(variogram(log(zinc) ~ 1, meuse, map = TRUE,
+     cutoff = 800, width = 100), main = "Variogram map, Meuse River, log(zinc)",
+     col.regions = bpy.colors(64)))
```



Note: The cutoff (maximum separation), specified with the `cutoff` optional argument, must be an **integer multiple** of the bin width, specified with the `width` optional argument, in order to obtain a (correct) symmetric variogram map. In the above example `cutoff=800`, which is exactly eight times the `width=100`.

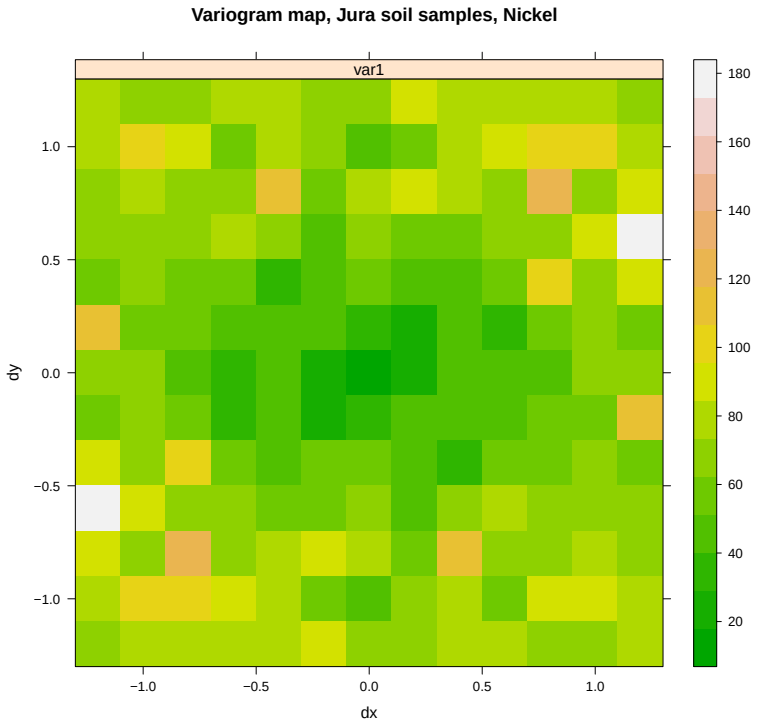
Q32 : *In which direction from the origin (centre of the plot) do the colours (representing the semi-variances) remain similar furthest from the origin?* *Jump to A32* •

Not all datasets show anisotropy; before continuing with the anisotropic example we show a counter-example from a different data set.

Task 32 : Display a variogram map of the Ni (nickel) content in the Jura data set, up to 1.2 km and with a bin width of 200 m. •

For this display, we'll use the `terrain.colors` method to produce a colour ramp typical of old-style atlases: deep green for "sea level", passing to light brown and then white at the "mountaintops":

```
> print(plot(variogram(Ni ~ 1, jura.all, cutoff = 1.2,
+     width = 0.2, map = TRUE), main = "Variogram map, Jura soil samples, Nickel",
+     col.regions = terrain.colors(64)))
```

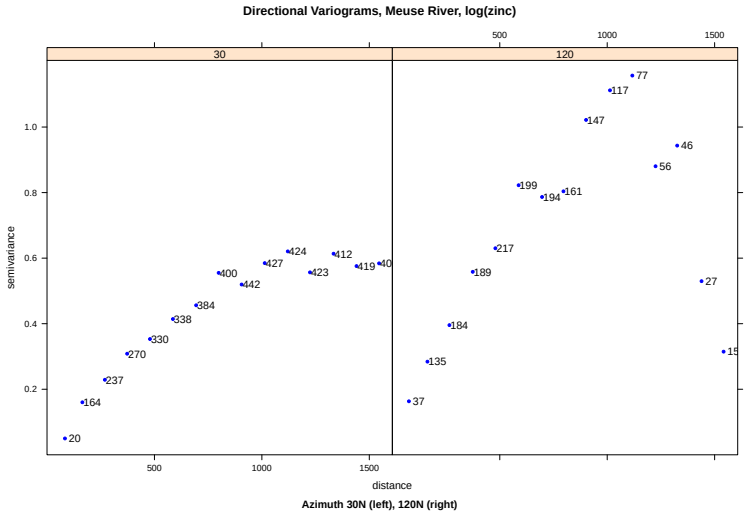


Q33 : *In which direction from the origin (centre of the plot) do the colours (representing the semi-variances) remain similar furthest from the origin?*
Jump to A33 •

Returning to the anisotropic structure (zinc in the Meuse data set), we now compute directional variograms with the `variogram` method, but using the optional `alpha` argument to specify one or more angles.

Task 33 : Display the directional variograms of the logarithm of Zn content at 30° N and 120° N, i.e. the suspected major and minor axes of the anisotropy elipse. •

```
> plot(variogram(log(zinc) ~ 1, meuse, alpha = c(30, 120),
+     cutoff = 1600), main = "Directional Variograms, Meuse River, log(zinc)",
+     sub = "Azimuth 30N (left), 120N (right)", pl = T,
+     pch = 20, col = "blue")
```



Q34 : *Do the two directions have similar variograms? (Consider sill, range, nugget)* *Jump to A34* •

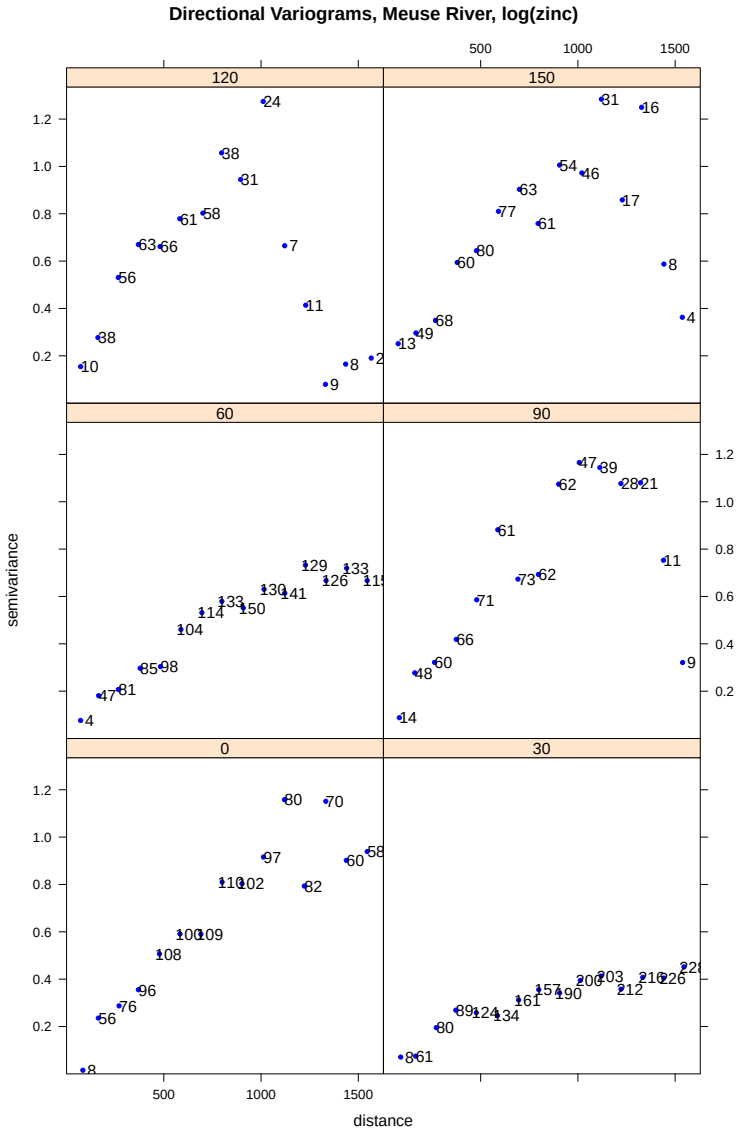
Q35 : *In which of the two perpendicular axes is the spatial dependence stronger (longer range, lower nugget to sill ratio)?* *Jump to A35* •

Optional: More on directional variograms Directional variograms may be further controlled by the `tol.hor` optional argument; this controls the angular range included in the bin. By default it is 90° divided by the number of angles specified in the `alpha` optional argument. So in this case since there are two angles the tolerance is 45° on either side of each axis; i.e. all point-pairs are included either in the 30° N or the 120° N bins. This has the effect of diluting any anisotropy, making it harder to detect.

For finer resolution we can either specify more angles or a narrower tolerance. The problem is that each bin now has fewer point-pairs, leading to erratic variograms which are difficult to model.

Task 34 : Display the directional variograms of the logarithm of Zn content at 30° intervals from 0° N through 150° N. •

```
> plot(variogram(log(zinc)~1, meuse, alpha=seq(0, 150, by=30), cutoff=1600 ),
+      main="Directional Variograms, Meuse River, log(zinc)",
+      pl=T, pch=20, col="blue")
```



Q36 : *How many directional variograms are produced? What then is the horizontal tolerance angle?* *Jump to A36* •

Q37 : *Compare the 30° and 120° N directional variograms of this figure (tolerance angle 15°) with those from the previous figure (tolerance angle 45°). What are the principal differences?* *Jump to A37* •

8.1 Answers

A32 : *At an azimuth (angle from N) of approximately 30° N – 210° N the blue colours do not change much; orthogonal to this there is a rapid change. The spatial*

dependence is ***anisotropic****.* Return to Q32 •

A33 : *There is no apparent direction where the colours persist; the spatial dependence is* ***isotropic****, also called* ***omnidirectional****.* Return to Q33 •

A34 : *They are quite different. The 30N variogram has a very regular form, with almost zero nugget, range about 1100 m, and sill near 0.6. The 120N variogram is irregular (partly because of the smaller number of point-pairs in that direction), with a nugget near 0.1, a range that is quite difficult to estimate but which may be placed near 800 m where the first sill of about 0.8 is reached.* Return to Q34 •

A35 : *30N.* Return to Q35 •

A36 : *Six directions; so each has a tolerance of 90/6 = 15˚ on either side, i.e. a total width of 30˚.* Return to Q36 •

A37 :

1. *Fewer point-pairs in each bin with the narrower angle;*
2. *In the major axis (30˚ N) a lower sill (0.4 instead of 0.6) and longer range; this due to less dilution of the anisotropic effect;*
3. *Very similar in the minor axis.*

Return to Q37 •

9 Quitting R

We will use both the Meuse and Jura datasets again, so let's leave them in the workspace and save them in `.RData`.

Task 35 : Leave R, saving the workspace. •

10 Self-test

This section is a small self-test of how well you mastered this exercise. You should be able to complete the tasks and answer the questions with the knowledge you have gained from the exercise. Please **submit your answers (including graphical output) to the instructor** for grading and sample answers.

For this self-test we work with the famous Walker Lake dataset of Isaaks and Srivastava [5]. This has coördinates `X` and `Y`, two attributes `V` and `U` and a categorical attribute `T`; on purpose the meanings of these are not given, so we concentrate purely on their spatial statistics with no prejudice.

Task 1 : Load the example dataset `walker` which is provided with the `gstat` package and convert it to a spatial object by specifying the coordinates. •

Q1 : *What is the bounding box of this dataset?* •

Task 2 : Plot the location of the sample points. •

Q2 : *Describe the* ***spatial distribution*** *of the observation points in the study area, i.e. the* ***point-pattern****.* •

Task 3 : Make a histogram of attribute `V`. •

Q3 : *Describe the feature-space distribution of attribute* `V`*.* •

Task 4 : Make a histogram of the **square root** of attribute `V`, using the `sqrt` method. •

Q4 : *Describe the feature-space distribution of the square root of attribute* `V`*.* •

From here we work with the square root of attribute `V`. The simplest way to proceed is to define it as a separate value in the data frame.

Task 5 : Add the square root of attribute `V` to the data frame of the `walker` spatial object. •

Since we haven't seen this syntax before, the solution is given here. We name the new attribute `sqrtV`:

```
> walker$sqrtV <- sqrt(walker$V)
> str(walker@data)
```

Task 6 : Make two post-plots of the square root of attribute `V`: (1) relative attribute value shown by a colour ramp of your choosing; (2) relative attribute value shown by size of the plotting symbol. •

Q5 : *Describe the spatial structure of attribute V: (1) any trend? if so, specify; (2) any local spatial dependence? if so, specify; (3) if there is local structure, does it appear to be isotropic or anisotropic?* •

Task 7 : Compute and display the empirical omnidirectional (isotropic) variogram of the square root of attribute `V`. •

Q6 : *Is this evidence of local spatial dependence? What are the approximate sill, range and nugget? Please include the units of measure for these. (Hint: see the on-line help for this dataset).* •

Task 8 : Compute and display a variogram surface of the square root of attribute `V` with a cutoff of 40 m and bin width of 8 m. •

Q7 : *Is there any evidence for anisotropy in the spatial dependence of attribute V? If so, what is the principal axis?* •

Task 9 : Compute and display directional variograms at azimuths $0, 30, \ldots 150°$ N. •

Q8 : *Are the variograms similar in all six directions? Describe any differences.* •

Task 10 : Remove the `walker` dataset and any temporary variables from the workspace. •

References

[1] O Atteia, J-P Dubois, and R Webster. Geostatistical analysis of soil contamination in the Swiss Jura. *Environmental Pollution*, 86(3):315–327, 1994. 1

[2] X Emery and J M Ortiz. Weighted sample variograms as a tool to better assess the spatial variability of soil properties. *Geoderma*, 140: 81–89, 2007. 1

[3] P Goovaerts. *Geostatistics for natural resources evaluation*. Applied Geostatistics. Oxford University Press, New York; Oxford, 1997. 1

[4] P Goovaerts, R Webster, and J-P Dubois. Assessing the risk of soil contamination in the Swiss Jura using indicator geostatistics. *Environmental and Ecological Statistics*, 4(1):31–48, 1997. 1

[5] E H Isaaks and R M Srivastava. *An introduction to applied geostatistics*. Oxford University Press, New York, 1990. 42

[6] R M Lark. Modelling complex soil properties as contaminated regionalized variables. *Geoderma*, 106(3-4):173–190, 2002. 1

[7] B P Marchant and R M Lark. Robust estimation of the variogram by residual maximum likelihood. *Geoderma*, 140:62–72, 2007. 1

[8] P Murrell. *R Graphics*. Chapman & Hall/CRC, Boca Raton, FL, 2005. ISBN 1-584-88486-X. 22

[9] E J Pebesma and R S Bivand. Classes and methods for spatial data in R. *R News*, 5(2):9–13, November 2005. URL http://CRAN.R-project.org/doc/Rnews/. 6

[10] D G Rossiter. *Introduction to the R Project for Statistical Computing for use at ITC*. International Institute for Geo-information Science & Earth Observation (ITC), Enschede (NL), 3.7 edition, 2009. URL http://www.itc.nl/personal/rossiter/teach/R/RIntro_ITC.pdf. 11, 20

[11] Deepayan Sarkar. Lattice. *R News*, 2(2):19–23, June 2002. URL http://CRAN.R-project.org/doc/Rnews/. 22

[12] R Development Core Team. *R Data Import/Export*. The R Foundation for Statistical Computing, Vienna, version 2.7.2 (2007-08-25) edition, 2008. http://cran.r-project.org/doc/manuals/R-data.pdf. 3

[13] W N Venables, D M Smith, and R Development Core Team. *An Introduction to R; Notes on R: A Programming Environment for Data Analysis and Graphics*. R Foundation for Statistical Computing, Vienna, version 2.6.1 (2007-11-26) edition, 2007. URL http://www.R-project.org. 11, 20

[14] R Webster, O Atteia, and J-P Dubois. Coregionalization of trace metals in the soil in the Swiss Jura. *European Journal of Soil Science*, 45(2): 205–218, 1994. 1

[15] T Yao. Nonparametric cross-covariance modeling as exemplified by soil heavy metal concentrations from the Swiss Jura. *Geoderma*, 88(1-2): 13–38, 1999. 1

[16] M Yemefack, D G Rossiter, and R Njomgang. Multi-scale characterization of soil variability within an agricultural landscape mosaic system in southern Cameroon. *Geoderma*, 125(1-2):117–143, 2005. 17

Index of R Concepts

^ operator, 21, 27

asp graphics argument, 10
at graphics argument, 23

bbox (package:sp), 8
bpy.colors (package:sp), 37
bpy.colors, 37
bubble (package:lattice), 13
bubble (package:sp), 12, 14, 15

c, 5, 6, 35
cm.colors, 37
col graphics argument, 10, 23
col.regions lattice graphics argument, 37
contourplot (package:lattice), 23
coordinates (package:sp), 6, 9

diff, 27
dim, 4, 28

expand.grid, 19

factor, 5
file.show, 2
fill graphics argument, 23

getwd, 2
gray, 37
grid, 10
gstat package, 1, 6, 20, 29, 30, 43

head, 3, 7, 8
heat.colors, 37
hist, 25
histogram (package:lattice), 25
hsv, 37

I, 21

krige (package:gstat), 20

lattice package, 1, 13–15, 22, 23, 25, 37
levelplot (package:lattice), 22
lm, 20
load, 18
log, 26
log10, 26
ls, 18

main lattice graphics argument, 13
main graphics argument, 10
meuse dataset, 36

panel lattice graphics argument, 15, 22
panel.grid (package:lattice), 15, 16, 22
panel.levelplot (package:lattice), 22
panel.points (package:lattice), 22
panel.xyplot (package:lattice), 15
pch graphics argument, 10, 23
plot, 9, 11, 31, 37
plot.default, 11
plot.gstatVariogram (package:gstat), 31, 37
print (package:lattice), 13
print, 13
print.trellis (package:lattice), 13

rainbow, 37
read.table, 3, 5
require, 1
row.names, 11

save, 8, 18
scales lattice graphics argument, 14
seq, 15
setwd, 2
sp package, 1, 6, 8, 9, 12, 13, 15, 36
spplot (package:sp), 13
sqrt, 43
str, 4, 7, 18
sub lattice graphics argument, 13
summary, 7

tail, 3
terrain.colors, 37, 38
text, 11
topo.colors, 37
trellis class, 13

variogram (package:gstat), 29–32, 37, 39

walker dataset, 43, 44

xlab lattice graphics argument, 13
xlab graphics argument, 10
xlim lattice graphics argument, 13
xyplot (package:lattice), 13–15

ylab lattice graphics argument, 13

`ylab` graphics argument, 10
`ylim lattice graphics argument`, 13