Displaying Time Series, Spatial, and Space-Time Data with R

Oscar Perpiñán Lamigueiro

Contents

Contents **i**

1 Introduction **1**
1.1 What This Book Is About 1
1.2 What You Will *Not* Find in This Book 2
1.3 How to Read This Book 3
1.4 R Graphics 4
1.5 ggplot2 6
1.6 Packages 7
1.7 Software Used to Write This Book 8
1.8 About the Author 8
1.9 Acknowledgments 9

I Time Series **11**

2 Displaying Time Series: Introduction **13**
2.1 Packages 14
2.2 Further Reading 16

3 Time on the Horizontal Axis **17**
3.1 Time Graph of Different Meteorological Variables 17
3.2 Time Series of Variables with the Same Scale 24
3.3 Stacked Graphs 35

4 Time as a Conditioning or Grouping Variable **45**
4.1 Scatterplot Matrix: Time as a Grouping Variable 45
4.2 Scatterplot with Time as a Conditioning Variable 50

5 Time as a Complementary Variable **55**
5.1 Polylines 56
5.2 Choosing Colors 57
5.3 Labels to Show Time Information 61
5.4 Country Names: Positioning Labels 62
5.5 A Panel for Each Year 65
5.6 Traveling Bubbles 71

6 About the Data **75**

II Spatial Data **77**

7 Displaying Spatial Data: Introduction **79**
7.1 Packages 80
7.2 Further Reading 85

8 Thematic Maps **87**
8.1 Proportional Symbol Mapping 87
8.2 Choropleth Maps 105
8.3 Raster Maps 112
8.4 Vector Fields 130

9 Reference and Physical Maps **135**
9.1 Physical Maps 135
9.2 OpenStreetMap with Hill Shade Layers 139

10 About the Data **147**
10.1 Air Quality in Madrid 147
10.2 Spanish General Elections 151
10.3 CM SAF 152
10.4 Land Cover and Population Rasters 153

III Space-Time Data **155**

11 Displaying Spatiotemporal Data: Introduction **157**

11.1 Packages 158
11.2 Further Reading 160

12 Spatiotemporal Raster Data **161**
12.1 Introduction 161
12.2 Level Plots 162
12.3 Graphical Exploratory Data Analysis 163
12.4 Space-Time and Time Series Plots 165
12.5 Animation 169

13 Spatiotemporal Point Observations **175**
13.1 Introduction 175
13.2 Data and Spatial Information 175
13.3 Graphics with `spacetime` 177
13.4 Animation 180

Bibliography **187**

Index **193**

Chapter 1

Introduction

1.1 What This Book Is About

A data graphic is not only a static image but also tells a story about the data. It activates cognitive processes that are able to detect patterns and discover information not readily available with the raw data. This is particularly true for time series, spatial, and space-time datasets.

There are several excellent books about data graphics and visual perception theory, with guidelines and advice for displaying information, including visual examples. Let's mention *The Elements of Graphical Data* (Cleveland 1994) and *Visualizing Data* (Cleveland 1993) by W. S. Cleveland, *Envisioning Information* (Tufte 1990) and *The Visual Display of Quantitative Information* (Tufte 2001) by E. Tufte, *The Functional Art* by A. Cairo (Cairo 2012), and *Visual Thinking for Design* by C. Ware (Ware 2008). Ordinarily, they do not include the code or software tools to produce those graphics.

On the other hand, there is a collection of books that provides code and detailed information about the graphical tools available with R. Commonly they do not use real data in the examples and do not provide advice for improving graphics according to visualization theory. Three books are the unquestioned representatives of this group: *R Graphics* by P. Murrell (Murrell 2011), *Lattice: Multivariate Data Visualization with R* by D. Sarkar (Sarkar 2008), and *ggplot2: Elegant Graphics for Data Analysis* by H. Wickham (Wickham 2009).

This book proposes methods to display time series, spatial, and space-time data using R, and aims to be a synthesis of both groups providing code and detailed information to produce high-quality graphics with practical examples.

1.2 What You Will *Not* Find in This Book

- **This is not a book to learn R.**

  Readers should have a fair knowledge of programming with R to understand the book. In addition, previous experience with the `zoo`, `sp`, `raster`, `lattice`, `ggplot2`, and `grid` packages is helpful.

  If you need to improve your R skills, consider these information sources:

  - Introduction to R[1].
  - Official manuals[2].
  - Contributed documents[3].
  - Mailing lists[4].
  - R-bloggers[5].
  - Books related to R[6] and particularly *Software for Data Analysis* by John M. Chambers (Chambers 2008).

- **This book does not provide an exhaustive collection of visualization methods**.

  Instead, it illustrates what I found to be the most useful and effective methods. Notwithstanding, each part includes a section titled "Further Reading" with bibliographic proposals for additional information.

- **This book does not include a complete review or discussion of R packages**.

[1] http://cran.r-project.org/doc/manuals/R-intro.html
[2] http://cran.r-project.org/manuals.html
[3] http://cran.r-project.org/other-docs.html
[4] http://www.r-project.org/mail.html
[5] http://www.r-bloggers.com
[6] http://www.r-project.org/doc/bib/R-books.html

Their most useful functions, classes, and methods regarding data and graphics are outlined in the introductory chapter of each part, and conveniently illustrated with the help of examples. However, if you need detailed information about a certain aspect of a package, you should read the correspondent package manual or vignette. Moreover, if you want to know additional alternatives, you can navigate through the CRAN Task Views about Time Series[7], Spatial Data[8], Spatiotemporal Data[9], and Graphics[10].

- **Finally, this book is not a handbook of data analysis, geostatistics, point pattern analysis, or time series theory**.

  Instead, this book is focused on the exploration of data with visual methods, so it may be framed in the Exploratory Data Analysis approach. Therefore, this book may be a useful complement for superb bibliographic references where you will find plenty of information about those subjects. For example, (Chatfield 2003), (Cressie and Wikle 2011), (Slocum 2005) and (R. S. Bivand, E. J. Pebesma, and Gomez-Rubio 2008).

1.3 How to Read This Book

This book is organized into three parts, each devoted to different types of data. Each part comprises several chapters according to the various visualization methods or data characteristics. The chapters are structured as independent units so readers can jump directly to a certain chapter according to their needs. Of course, there are several dependencies and redundancies between the sets of chapters that have been conveniently signaled with cross-references.

The content of each chapter illustrates how to display a dataset starting with an easy and direct approach. Often this first result is not entirely satisfactory so additional improvements are progressively added. Each step involves additional complexity which, in some cases, can be overwhelming during a first reading. Thus, some sections, marked with the sign ⚽, can be safely skipped for later reading.

Although I have done my best to help readers understand the methods and code, you should not expect to understand it after one reading. The

[7] http://cran.r-project.org/web/views/TimeSeries.html
[8] http://cran.r-project.org/web/views/Spatial.html
[9] http://cran.r-project.org/web/views/SpatioTemporal.html
[10] http://cran.r-project.org/web/views/Graphics.html

key is practical experience, and the best way is to try out the code with the provided data **and** modify it to suit your needs with your own data. There is a website and a code repository to help you in this task.

1.3.1 Website and Code Repository

The book website with the main graphics of this book is located at

http://oscarperpinan.github.com/spacetime-vis/

The full code is freely available from the repository:

https://github.com/oscarperpinan/spacetime-vis

On the other hand, the datasets used in the examples are either available at the repository or can be freely obtained from other websites. It must be underlined that the combination of code and data freely available allows this book to be fully reproducible.

I have chosen the datasets according to two main criteria:

- They are freely available without restrictions for public use.
- They cover different scientific and professional fields (meteorology and climate research, economy and social sciences, energy and engineering, environmental research, epidemiology, etc.).

The repository and the website can be downloaded as compressed files[11], and if you use `git`, you can clone the repository with

```
git clone https://github.com/oscarperpinan/spacetime-vis.git
```

1.4 R Graphics

There are two distinct graphics systems built into R, referred to as traditional and grid graphics. Grid graphics are produced with the `grid` package (Murrell 2011), a flexible low-level graphics toolbox. Compared with the traditional graphics model, it provides more flexibility to modify or

[11] Repository: https://github.com/oscarperpinan/spacetime-vis/archive/master.zip, Website: https://github.com/oscarperpinan/spacetime-vis/archive/gh-pages.zip

add content to an existent graphical output, better support for combining different outputs easily, and more possibilities for interaction. All the graphics in this book have been produced with the grid graphics model.

Other packages are constructed over it to provide high-level functions, most notably the `lattice` and `ggplot2` packages.

1.4.1 lattice

The `lattice` package (Sarkar 2008) is an independent implementation of Trellis graphics, which were mostly influenced by *The Elements of Graphing Data* (Cleveland 1994). Trellis graphics often consist of a rectangular array of panels. The `lattice` package uses a *formula* interface to define the structure of the array of panels with the specification of the variables involved in the plot. The result of a `lattice` high-level function is a `trellis` object.

For bivariate graphics, the formula is generally of the form `y ~ x` representing a single panel plot with `y` versus `x`. This formula can also involve expressions. The main function for bivariate graphics is `xyplot`.

Optionally, the formula may be `y ~ x | g1 * g2` and `y` is represented against `x` conditional on the variables `g1` and `g2`. Each unique combination of the levels of these conditioning variables determines a subset of the variables `x` and `y`. Each subset provides the data for a single panel in the Trellis display, an array of panels laid out in columns, rows, and pages.

For example, in the following code, the variable `wt` of the dataset `mtcars` is represented against the `mpg`, with a panel for each level of the categorical variable `am`. The points are grouped by the values of the `cyl` variable.

```
xyplot(wt ~ mpg | am, data = mtcars, groups = cyl)
```

For trivariate graphics, the formula is of the form `z ~ x * y`, where `z` is a numeric response, and `x` and `y` are numeric values evaluated on a rectangular grid. Once again, the formula may include conditioning variables, for example `z ~ x * y | g1 * g2`. The main function for these graphics is `levelplot`.

The plotting of each panel is performed by the panel function, specified in a high-level function call as the `panel` argument. Each high-level `lattice` function has a default panel function, although the user can create new Trellis displays with custom panel functions.

`lattice` is a member of the recommended packages list so it is commonly distributed with R itself. There are more than 250 packages depending on it, and the most important packages for our purposes (`zoo`, `sp`, and `raster`) define methods to display their classes using `lattice`.

On the other hand, the `latticeExtra` package (Sarkar and Andrews 2012) provides additional flexibility for the somewhat rigid structure of the Trellis framework implemented in `lattice`. This package complements the `lattice` with the implementation of layers via the `layer` function, and superposition of `trellis` objects and layers with the `+.trellis` function. Using both packages, you can define a graphic with the formula interface (under the `lattice` model) and overlay additional content as layers (following the `ggplot2` model).

1.5 ggplot2

The `ggplot2` package (Wickham 2009) is an implementation of the system proposed in *The Grammar of Graphics* (Wilkinson 1999), a general scheme for data visualization that breaks up graphs into semantic components such as scales and layers. Under this framework, the definition of the graphic with `ggplot2` is done with a combination of several functions that provides the components, instead of the formula interface of `lattice`.

With `ggplot2`, a graphic is composed of:

- A dataset, `data`, and a set of mappings from variables to aesthetics, `aes`.
- One or more layers, each composed of: a geometric object, `geom_*`, to control the type of plot you create (points, lines, etc.); a statistical transformation, `stat_*`; and a position adjustment (and optionally, additional dataset and aesthetic mappings).
- A scale, `scale_*`, to control the mapping from data to aesthetic attributes. Scales are common across layers to ensure a consistent mapping from data to aesthetics.
- A coordinate system, `coords_*`.
- Optionally, a faceting specification, `facet_*`, the equivalent of Trellis graphics with panels.

The function `ggplot` is typically used to construct a plot incrementally, using the `+` operator to add layers to the existing ggplot object. For instance, the following code (equivalent to the previous `lattice` example) uses `mtcars` as the dataset, and maps the `mpg` variable on the x-axis and the `wt` variable on the y-axis. The geometric object is the point using the `cyl` variable to control the color. Finally, the levels of the `am` variable define the panels of the graphic.

```
ggplot(mtcars, aes(mpg, wt)) +
   geom_point(aes(colour=factor(cyl))) +
   facet_grid(. ~ am)
```

This package is increasingly popular, with a list of more than ninety packages depending on it. On the other hand, few packages provide method definitions based on `ggplot2` to display their classes. In our context, only the `zoo` package defines the `autoplot` function based on it.

1.5.1 Comparison between lattice and ggplot2

Which package to choose is, for a wide range of datasets, a question of personal preferences. You may be interested in a comparison between them published in a series of blog posts[12]. However, the major drawback of `ggplot2` is its considerably slower speed when dealing with large datasets[13], so you should be cautious with large spatial and spatiotemporal data. Consequently, most of the code in Part **??** contains alternatives defined both with `lattice` and with `ggplot2`. However, because of the speed problem and the absence of `ggplot2` functions in the corresponding packages, only a minor fraction of the code in Parts **??** and **??** contains graphics defined with `ggplot2`.

1.6 Packages

Throughout the book, several R packages are used. All of them are available from CRAN, and you must install them before using the code. Most of them are loaded at the start of the code of each chapter, although some of them are loaded later if they are used only inside optional sections (marked with ✿). You should install the last version available at CRAN to ensure correct functioning of the code.

Although the introductory chapter of each part includes a section with an outline of the most relevant packages, some of them deserve to be highlighted here:

- `zoo` (Zeileis and Grothendieck 2005) provides infrastructure for time series using arbitrary classes for the time stamps (Section 2.1.1).

[12] http://learnr.wordpress.com/2009/06/28/ggplot2-version-of-figures-in-lattice-multivari

[13] Take a look at the time comparison published as the final result of the previous series of blog posts, http://learnr.files.wordpress.com/2009/08/latbook.pdf

- `sp` (E. Pebesma 2012) provides a coherent set of classes and methods for the major spatial data types: points, lines, polygons, and grids (Section 7.1.1). `spacetime` (E. Pebesma 2012) defines classes and methods for spatiotemporal data, and methods for plotting data as map sequences or multiple time series (Section 11.1.1).

- `raster` (R. J. Hijmans 2013) is a major extension of gridded spatial data classes. It provides a unified access method to different raster formats, permitting large objects to be analyzed with the definition of basic and high-level processing functions (Sections 7.1.2 and 11.1.2). `rasterVis` (Perpiñán and R. Hijmans 2013) provides enhanced visualization of raster data with methods for spatiotemporal rasters (Sections 7.1.3 and 11.1.3).

- `gridSVG` (Murrell and Potter 2013) converts any grid scene to an SVG document. The `grid.hyperlink` function allows a hyperlink to be associated with any component of the scene, the `grid.animate` function can be used to animate any component of a scene, and the `grid.garnish` function can be used to add SVG attributes to the components of a scene. By setting event handler attributes on a component, plus possibly using the `grid.script` function to add JavaScript to the scene, it is possible to make the component respond to user input such as mouse clicks.

1.7 Software Used to Write This Book

This book has been written using different computers running Debian GNU Linux and using several gems of open-source software:

- org-mode (Schulte et al. 2012), LaTeX, and AUCTeX, for authoring text and code.

- R (R Development Core Team 2013) with Emacs Speaks Statistics (Rossini et al. 2004).

- GNU Emacs as development environment.

1.8 About the Author

During the past 15 years, my main area of expertise has been photovoltaic solar energy systems, with a special interest in solar radiation. Initially I

worked as an engineer for a private company and I was involved in several commercial and research projects. The project teams were partly integrated by people with low technical skills who relied on the input from engineers to complete their work. I learned how a good visualization output eased the communication process.

Now I work as a professor and researcher at the university. Data visualization is one of the most important tools I have available. It helps me embrace and share the steps, methods, and results of my research. With students, it is an inestimable partner in helping them understand complex concepts.

I have been using R to simulate the performance of photovoltaic energy systems and to analyze solar radiation data, both as time series and spatial data. As a result, I have developed packages that include several graphical methods to deal with multivariate time series (namely, `solaR` (Perpiñán 2012)) and space-time data (`rasterVis`).

1.9 Acknowledgments

Writing a book is often described as a solitary activity. It is certainly difficult to write when you are with friends or spending time with your family,... although with three little children at home I have learned to write prose and code while my baby wants to learn typing and my daughters need help to share a family of dinosaurs.

Seriously speaking, solitude is the best partner of a writer. But when I am writing or coding I feel I am immersed in a huge collaborative network of past and present contributors. Piotr Kropotkin described it with the following words (Kropotkin 1906):

> Thousands of writers, of poets, of scholars, have laboured to increase knowledge, to dissipate error, and to create that atmosphere of scientific thought, without which the marvels of our century could never have appeared. And these thousands of philosophers, of poets, of scholars, of inventors, have themselves been supported by the labour of past centuries. They have been upheld and nourished through life, both physically and mentally, by legions of workers and craftsmen of all sorts.

And Lewis Mumford claimed (Mumford 1934):

> Socialize Creation! What we need is the realization that the creative life, in all its manifestations, is necessarily a social product.

I want to express my deepest gratitude and respect to all those women and men who have contributed and contribute to strengthening the communities of free software, open data, and open science. My special thanks go to the people of the R community: users, members of the R Core Development Team, and package developers.

With regard to this book in particular, I would like to thank John Kimmel for his constant support, guidance, and patience.

Last, and most importantly, thanks to Candela, Marina, and Javi, my crazy little shorties, my permanent source of happiness, imagination, and love. Thanks to María, *mi amor, mi cómplice y todo*.

Part I

Time Series

Chapter 2

Displaying Time Series: Introduction

A time series is a sequence of observations registered at consecutive time instants. When these time instants are evenly spaced, the distance between them is called the sampling interval. The visualization of time series is intended to reveal changes of one or more quantitative variables through time, and to display the relationships between the variables and their evolution through time.

The standard time series graph displays the time along the horizontal axis. Several variants of this approach can be found in Chapter 3. On the other hand, time can be conceived as a grouping or conditioning variable (Chapter 4). This solution allows several variables to be displayed together with a scatterplot, using different panels for subsets of the data (time as a conditioning variable) or using different attributes for groups of the data (time as a grouping variable). Moreover, time can be used as a complementary variable that adds information to a graph where several variables are confronted (Chapter 5).

These chapters provide a variety of examples to illustrate a set of useful techniques. These examples make use of several datasets (available at the book website) described in Chapter 6.

2.1 Packages

The CRAN Tasks View "Time Series Analysis" [1] summarizes the packages for reading, vizualizing, and analyzing time series. This section provides a brief introduction to the `zoo` and `xts` packages. Most of the information has been extracted from their vignettes, webpages, and help pages. You should read them for detailed information.

Both packages extensively use the time classes defined in `R`. The interested reader will find an overview of the different time classes in `R` in (Ripley and Hornik 2001) and (Grothendieck and Petzoldt 2004).

2.1.1 zoo

The `zoo` package (Zeileis and Grothendieck 2005) provides an `S3` class with methods for indexed totally ordered observations. Its key design goals are independence of a particular index class and consistency with base `R` and the `ts` class for regular time series.

Objects of class `zoo` are created by the function `zoo` from a numeric vector, matrix, or a factor that is totally ordered by some index vector. This index is usually a measure of time but every other numeric, character, or even more abstract vector that provides a total ordering of the observations is also suitable. It must be noted that this package defines two new index classes, `yearmon` and `yearqtr`, for representing monthly and quarterly data, respectively.

The package defines several methods associated with standard generic functions such as `print`, `summary`, `str`, `head`, `tail`, and `[` (subsetting). In addition, standard mathematical operations can be performed with `zoo` objects, although only for the intersection of the indexes of the objects.

On the other hand, the data stored in `zoo` objects can be extracted with `coredata`, which drops the index information, and can be replaced by `coredata<-`. The index can be extracted with `index` or `time`, and can be modified by `index<-`. Finally, the `window` and `window<-` methods extract or replace time windows of `zoo` objects.

Two `zoo` objects can be merged by common indexes with `merge` and `cbind`. The `merge` method combines the columns of several objects along the union or the intersection of the indexes. The `rbind` method combines the indexes (rows) of the objects.

[1] http://CRAN.R-project.org/view=TimeSeries

The `aggregate` method splits a `zoo` object into subsets along a coarser index grid, computes a function (`sum` is the default) for each subset, and returns the aggregated `zoo` object.

This package provides four methods for dealing with missing observations:

1. `na.omit` removes incomplete observations.
2. `na.contiguous` extracts the longest consecutive stretch of non-missing values.
3. `na.approx` replaces missing values by linear interpolation.
4. `na.locf` replaces missing observations by the most recent non-=NA= prior to it.

The package defines interfaces to `read.table` and `write.table` for reading, `read.zoo`, and writing, `write.zoo`, `zoo` series from or to text files. The `read.zoo` function expects either a text file or connection as input or a `data.frame`. `write.zoo` first coerces its argument to a `data.frame`, adds a column with the index, and then calls `write.table`.

2.1.2 xts

The `xts` package (Ryan and Ulrich 2013) extends the `zoo` class definition to provide a general time-series object. The index of an `xts` object must be of a time or date class: `Date`, `POSIXct`, `chron`, `yearmon`, `yearqtr`, or `timeDate`. With this restriction, the subset operator `[` is able to extract data using the ISO:8601 [2] time format notation `CCYY-MM-DD HH:MM:SS`. It is also possible to extract a range of times with a `from/to` notation, where both from and to are optional. If either side is missing, it is interpreted as a request to retrieve data from the beginning, or through the end of the data object.

Furthermore, this package provides several time-based tools:

- `endpoints` identifies the endpoints with respect to time.
- `to.period` changes the periodicity to a coarser time index.
- The functions `period.*` and `apply.*` evaluate a function over a set of non-overlapping time periods.

[2] http://en.wikipedia.org/wiki/ISO_8601

2.2 Further Reading

- (Wills 2011) provides a systematic analysis of the visualization of time series, and a section of (Jeffrey Heer, Bostock, and Ogievetsky 2010) summarizes the main techniques to display time series.
- (Cleveland 1994) includes a section about time series visualization with a detailed discussion of the banking to 45° technique and the cut-and-stack method. (J. Heer and Agrawala 2006) propose the multi-scale banking, a technique to identify trends at various frequency scales.
- (Few 2008; J. Heer, Kong, and Agrawala 2009) explain in detail the foundations of the horizon graph (Section 3).
- The *small multiples* concept (Sections 3.2 and **??**) is illustrated in (Tufte 2001; Tufte 1990).
- Stacked graphs are analyzed in (Byron and Wattenberg 2008), and the ThemeRiver technique is explained in (Havre et al. 2002).
- (Cleveland 1994; Friendly and Denis 2005) study the scatterplot matrices (Section **??**), and (D. B. Carr et al. 1987) provide information about hexagonal binning.
- (Harrower and Fabrikant 2008) discuss the use of animation for the visualization of data. (Few 2007) exposes a software tool resembling the Trendalyzer.
- The `D3` gallery [3] shows several great examples of time-series visualizations using the JavaScript library `D3.js`.

[3] https://github.com/mbostock/d3/wiki/Gallery

Chapter 3

Time on the Horizontal Axis

The most frequent visualization method of a time series uses the horizontal axis to depict the time index. This chapter illustrates several variants to display multivariate time series: multiple time series with different scales, variables with the same scale, and stacked graphs.

3.1 Time Graph of Different Meteorological Variables

There is a variety of scientific research interested in the relationship among several meteorological variables. A suitable approach is to display the time evolution of all of them using a panel for each of the variables. The superposition of variables with different characteristics is not very useful (unless their values were previously rescaled), so this option is postponed for Section 3.2.

For this example we will use the 8 years of daily data from the SIAR meteorological station located at Aranjuez (Madrid). This multivariate time series can be displayed with the `xyplot` method of `lattice` for `zoo` objects with a panel for each variable (Figure 3.1).

```
load('data/aranjuez.RData')
library(zoo)
## The layout argument arranges panels in rows
xyplot(aranjuez, layout=c(1, ncol(aranjuez)))
```

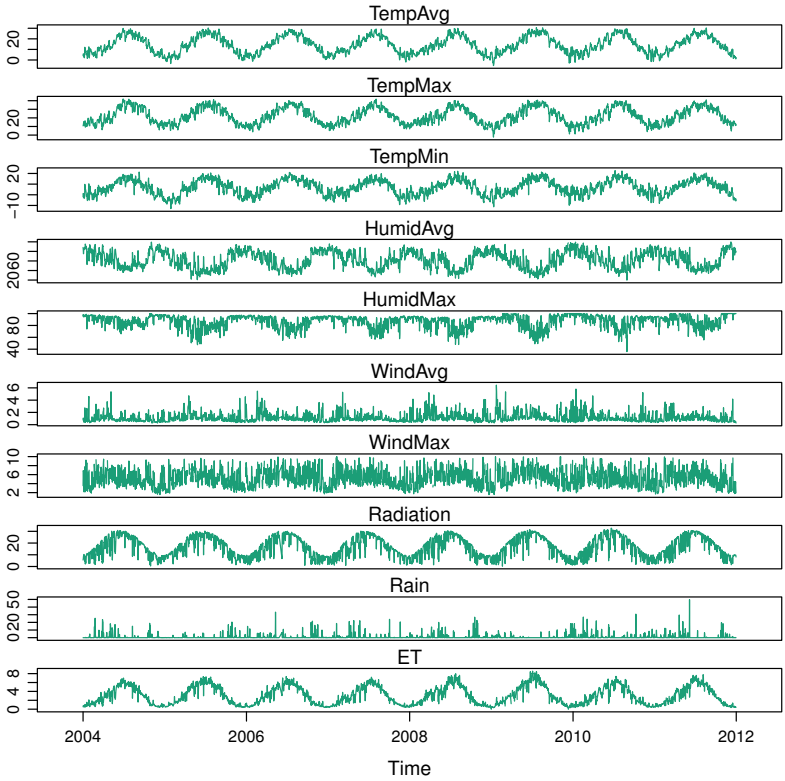

FIGURE 3.1: Time plot of the collection of meteorological time series of the Aranjuez station (`lattice` version).

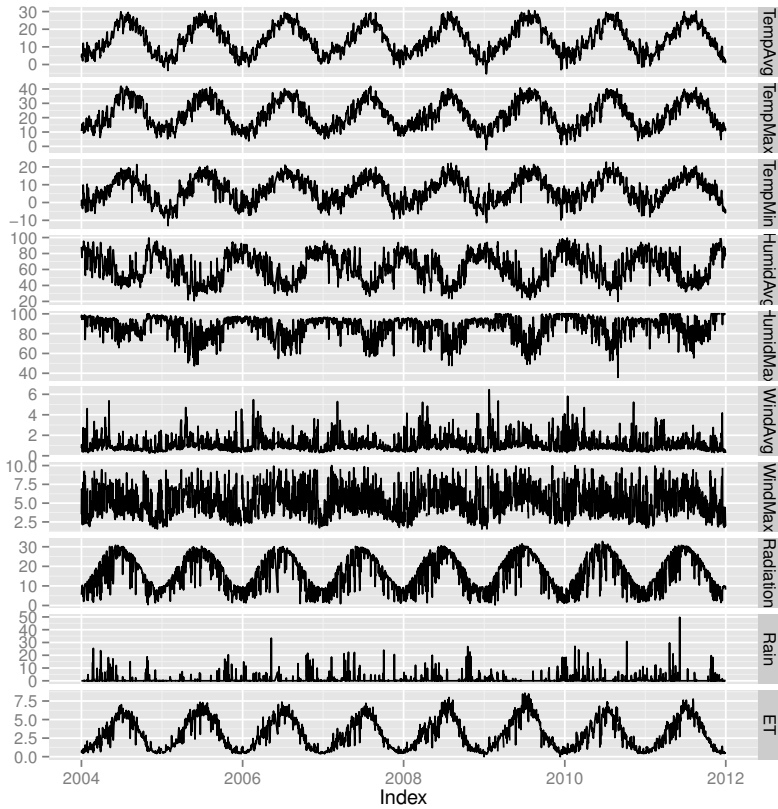

FIGURE 3.2: Time plot of the collection of meteorological time series of the Aranjuez station (`ggplot2` version).

The package `ggplot2` provides the generic method `autoplot` to automate the display of certain classes with a simple command. The package `zoo` provides an `autoplot` method for the `zoo` class with a result similar to that obtained with `xyplot` (Figure 3.2).

```
autoplot(aranjuez) + facet_free()
```

3.1.1 Annotations to Enhance the Time Graph

These first attempts can be improved with a custom panel function that generates the content of each panel using the information processed by

xyplot, or overlaying additional layers with autoplot. One of the main enhancements is to highlight certain time regions that fulfill certain conditions. The package latticeExtra provides a nice solution for xyplot with panel.xblocks. The result is displayed in Figure 3.3:

- The label of each time series is displayed with text inside each panel instead of using the strips mechanism. The panel.text prints the name of each variable with the aid of panel.number.
- The alternating of years is displayed with blocks of gray and white color using the panel.xblocks function from latticeExtra. The year is extracted (as character) from the time index of the zoo object with format.POSIXlt.
- Those values below the mean of each variable are highlighted with short red color blocks at the bottom of each panel, again with the panel.xblocks function.
- The maxima and minima are highlighted with small blue triangles.

Because the functions included in the panel function are executed consecutively, their order determines the superposition of graphical layers.

```
library(grid)
library(latticeExtra)

## Auxiliary function to extract the year value of a POSIXct time
## index
Year <- function(x)format(x, "%Y")

xyplot(aranjuez, layout=c(1, ncol(aranjuez)), strip=FALSE,
      scales=list(y=list(cex=0.6, rot=0)),
      panel=function(x, y, ...){
        ## Alternation of years
        panel.xblocks(x, Year,
                      col = c("lightgray", "white"),
                      border = "darkgray")
        ## Values under the average highlighted with red regions
        panel.xblocks(x, y<mean(y, na.rm=TRUE),
                      col = "indianred1",
                      height=unit(0.1, 'npc'))
        ## Time series
```

```
      panel.lines(x, y, col='royalblue4', lwd=0.5, ...)
      ## Label of each time series
      panel.text(x[1], min(y, na.rm=TRUE),
                 names(aranjuez)[panel.number()],
                 cex=0.6, adj=c(0, 0), srt=90, ...)
      ## Triangles to point the maxima and minima
      idxMax <- which.max(y)
      panel.points(x[idxMax], y[idxMax],
                   col='black', fill='lightblue', pch=24)
      idxMin <- which.min(y)
      panel.points(x[idxMin], y[idxMin],
                   col='black', fill='lightblue', pch=25)
    })
```

There is no equivalent `panel.xblocks` function that can be used with `ggplot2`. Therefore, the `ggplot2` version must explicitly compute the corresponding bands (years and regions below the average values):

- The first step in working with `ggplot` is to transform the `zoo` object into a `data.frame` in long format. `fortify` returns a `data.frame` with three columns: the time `Index`, a factor indicating the `Series`, and the corresponding `Value`.

```
timeIdx <- index(aranjuez)

long <- fortify(aranjuez, melt=TRUE)
```

- The bands of values below the average can be easily extracted with `scale` because these regions are negative when the `data.frame` is centered.

```
## Values below mean are negative after being centered
scaled <- fortify(scale(aranjuez, scale=FALSE), melt=TRUE)
## The 'scaled' column is the result of the centering.
## The new 'Value' column store the original values.
scaled <- transform(scaled, scaled=Value, Value=long$Value)
underIdx <- which(scaled$scaled <= 0)
## 'under' is the subset of values below the average
under <- scaled[underIdx,]
```

- The years bands are defined with the function `endpoints` from the `xts` package:

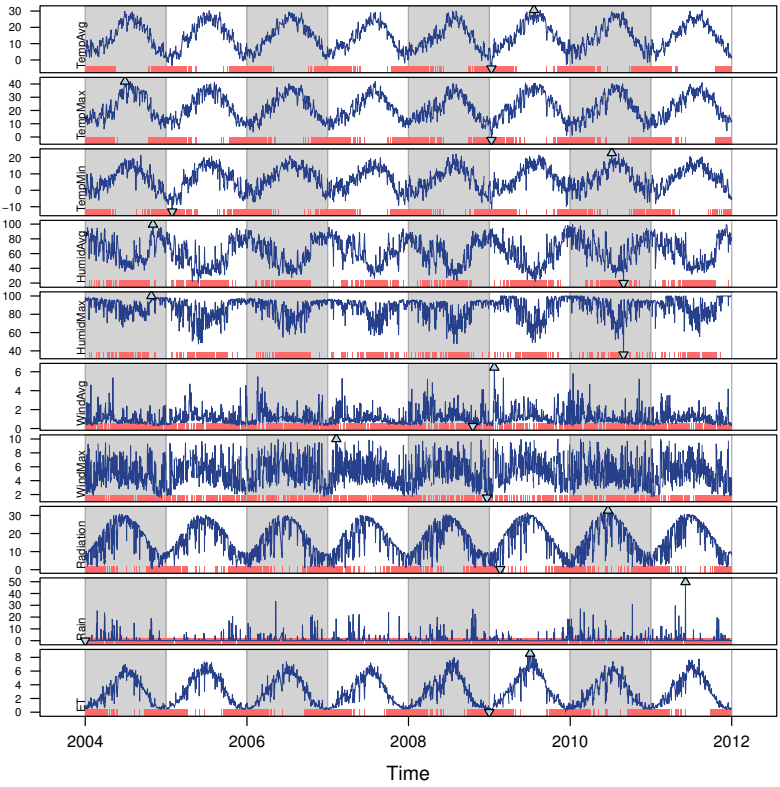

FIGURE 3.3: Enhanced time plot of the collection of meteorological time series of the Aranjuez station.

```
library(xts)
ep <- endpoints(timeIdx, on='years')
N <- length(ep[-1])
## 'tsp' is start and 'tep' is the end of each band
tep <- timeIdx[ep]
tsp <- timeIdx[ep[-(N+1)]+1]
## 'cols' is a vector with the color of each band
cols <- rep_len(c('gray', 'white'), N)
```

- The minima and maxima points of each variable are extracted with `apply`:

```
minIdx <- timeIdx[apply(aranjuez, 2, which.min)]
minVals <- apply(aranjuez, 2, min, na.rm=TRUE)
mins <- data.frame(Index=minIdx,
                   Value=minVals,
                   Series=names(aranjuez))

maxIdx <- timeIdx[apply(aranjuez, 2, which.max)]
maxVals <- apply(aranjuez, 2, max, na.rm=TRUE)
maxs <- data.frame(Index=maxIdx,
                   Value=maxVals,
                   Series=names(aranjuez))
```

- With `ggplot` we define the canvas, and the layers of information are added successively:

```
ggplot(data=long, aes(Index, Value)) +
   ## Time series of each variable
   geom_line(colour = "royalblue4", lwd = 0.5) +
   ## Year bands
   annotate(geom='rect', ymin = -Inf, ymax = Inf,
            xmin=tsp, xmax=tep,
            fill = cols, alpha = 0.4) +
   ## Values below average
   geom_rug(data=under,
          sides='b', col='indianred1') +
   ## Minima
   geom_point(data=mins, pch=25) +
   ## Maxima
   geom_point(data=maxs, pch=24) +
```

```
## Axis labels and theme definition
labs(x='Time', y=NULL) +
theme_bw() +
## Each series is displayed in a different panel with an
## independent y scale
facet_free()
```

Some messages from Figure 3.3:

- The radiation, temperature, and evotranspiration are quasi-periodic and are almost synchronized between them. Their local maxima appear in the summer and the local minima in the winter. Obviously, the summer values are higher than the average.
- The average humidity varies in oposition to the temperature and radiation cycle, with local maxima located during winter.
- The average and maximum wind speed, and rainfall vary in a more erratic way and do not show the evident periodic behavior of the radiation and temperature.
- The rainfall is different from year to year. The remaining variables do not show variations between years.
- The fluctuations of solar radiation are more apparent than the temperature fluctuations. There is hardly any day with temperatures below the average value during summer, while it is not difficult to find days with radiation below the average during this season.

3.2 Time Series of Variables with the Same Scale

As an example of time series of variables with the same scale, we will use measurements of solar radiation from different meteorological stations.

The first attempt to display this multivariate time series makes use of the `xyplot.zoo` method. The objective of this graphic is to display the behavior of the collection as a whole: the series are superposed in the same panel (`superpose=TRUE`) without legend (`auto.key=TRUE`), using thin lines and partial transparency[1]. Transparency softens overplotting problems and reveals density clusters because regions with more overlapping lines are darker. Figure 3.4 displays the variations around the time average (`avRad`).

[1] A similar result can be obtained with `autoplot` using `facets=NULL`.

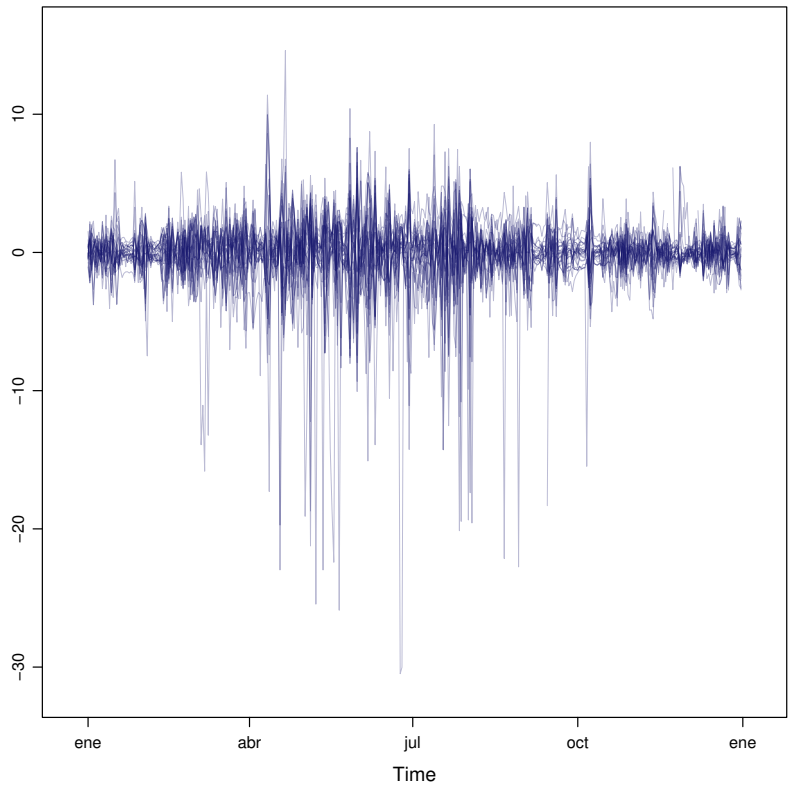

FIGURE 3.4: Time plot of the variations around time average of solar radiation measurements from the meteorological stations of Navarra.

```
load('data/navarra.RData')
```

```
avRad <- zoo(rowMeans(navarra, na.rm=1), index(navarra))
pNavarra <- xyplot(navarra - avRad,
                   superpose=TRUE, auto.key=FALSE,
                   lwd=0.5, alpha=0.3, col='midnightblue')
pNavarra
```

This result can be improved with different methods: the cut-and-stack method, the horizon graph with `horizonplot`, and dynamic labeling with the `gridSVG` package.

3.2.1 Aspect Ratio and Rate of Change

When a graphic is intended to inform about the rate of change, special attention must be paid to the aspect ratio of the graph, defined as the ratio of the height to the width of the graphical window. Cleveland analyzed the importance of the aspect ratio for judging rate of change. He concluded that we visually decode the information about the relative local rate of change of one variable with another by comparing the orientations of the local line segments that compose the polylines. The recommendation is to choose the aspect ratio so that the absolute values of the orientations of the segments are centered on 45° (banking to 45°).

The problem with banking to 45° is that the resulting aspect ratio is frequently too small. A suitable solution to minimize wasted space is the cut-and-stack method. The `xyplot.ts` method implement this solution with the combination of the arguments `aspect` and `cut`. The version of Figure 3.4 using banking to 45° and the cut-and-stack method is produced with

```
xyplot(navarra - avRad,
     aspect='xy', cut=list(n=3, overlap=0.1),
     strip=FALSE,
     superpose=TRUE, auto.key=FALSE,
     lwd=0.5, alpha=0.3, col='midnightblue')
```

3.2.2 The Horizon Graph

The horizon graph is useful in examining how a large number of series changes over time, and does so in a way that allows both comparisons between the individual time series and and independent analysis of each series. Moreover, extraordinary behaviors and predominant patterns are easily distinguished (J. Heer, Kong, and Agrawala 2009; Few 2008).

This graph displays several stacked series collapsing the y-axis to free vertical space:

- Positive and negative values share the same vertical space. Negative values are inverted and placed above the reference line. Sign is encoded using different hues (positive values in blue and negative values in red).
- Differences in magnitude are displayed as differences in color intensity (darker colors for greater differences).

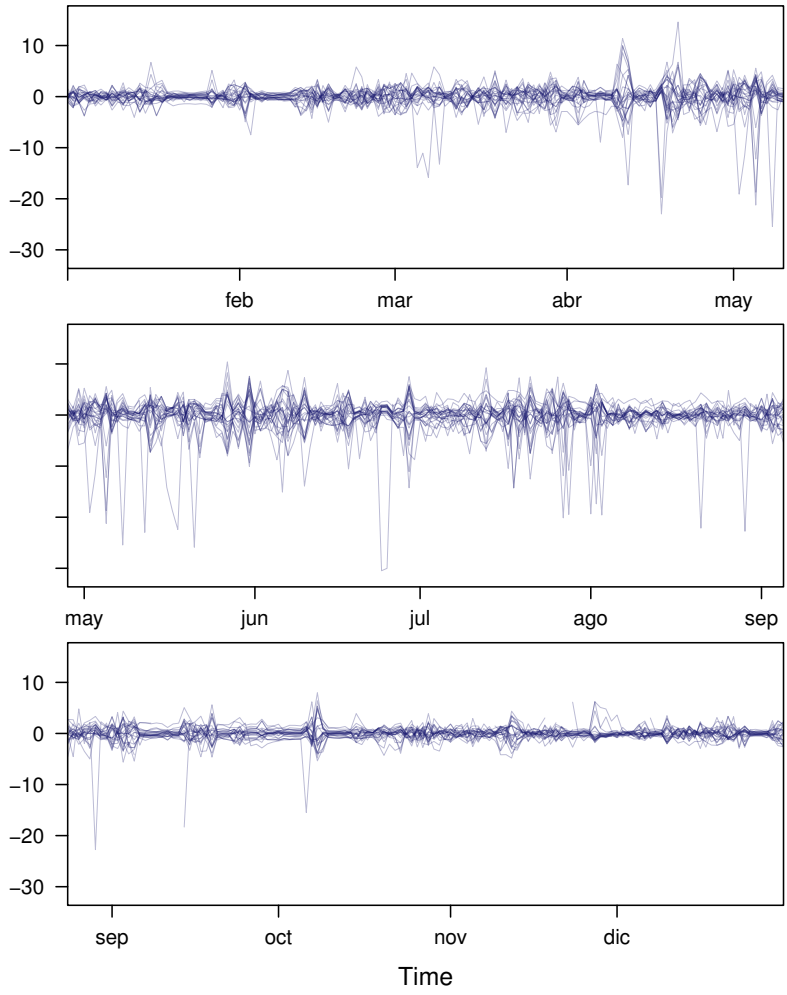

FIGURE 3.5: Cut-and-stack plot with banking to 45°.

- The color bands share the same baseline and are superposed, with darker bands in front of the ligther ones.

Because the panels share the same design structure, once this technique is understood, it is easy to establish comparisons or spot extraordinary events. This method is what Tufte described as small multiples (Tufte 1990).

Figure 3.6 displays the variations of solar radiation around the time average with an horizon graph using a row for each time series.

```
library(latticeExtra)

horizonplot(navarra-avRad,
          layout=c(1, ncol(navarra)),
          origin=0, colorkey=TRUE)
```

Figure 3.6 allows several questions to be answered:

- Which stations consistently measure above and below the average?
- Which stations resemble more closely the average time series?
- Which stations show erratic and uniform behavior?
- In each of the stations, is there any day with extraordinary measurements?
- Which part of the year is associated with more intense absolute fluctuations across the set of stations?

3.2.3 Time Graph of the Differences between a Time Series and a Reference

The horizon graph is also useful in revealing the differences between a univariate time series and another reference. For example, we might be interested in the departure of the observed temperature from the long-term average, or in other words, the temperature change over time.

Let's illustrate this approach with the time series of daily average temperatures measured at the meteorological station of Aranjuez. The reference is the long-term daily average calculated with `ave`.

```
Ta <- aranjuez$TempAvg
timeIndex <- index(aranjuez)
longTa <- ave(Ta, format(timeIndex, '%j'))
diffTa <- (Ta - longTa)
```

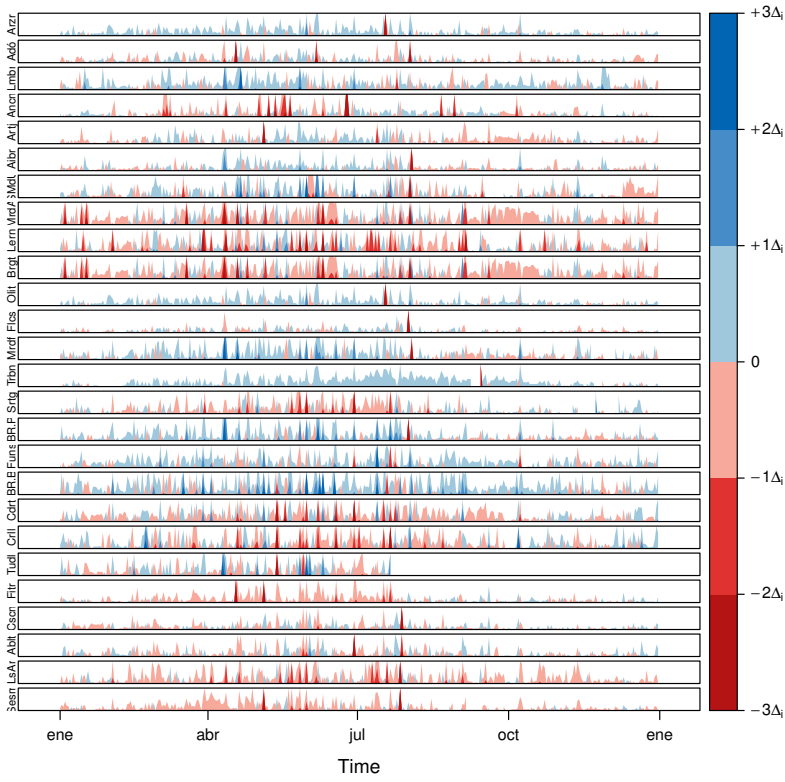

FIGURE 3.6: Horizon plot of variations around time average of solar radiation measurements from the meteorological stations of Navarra.

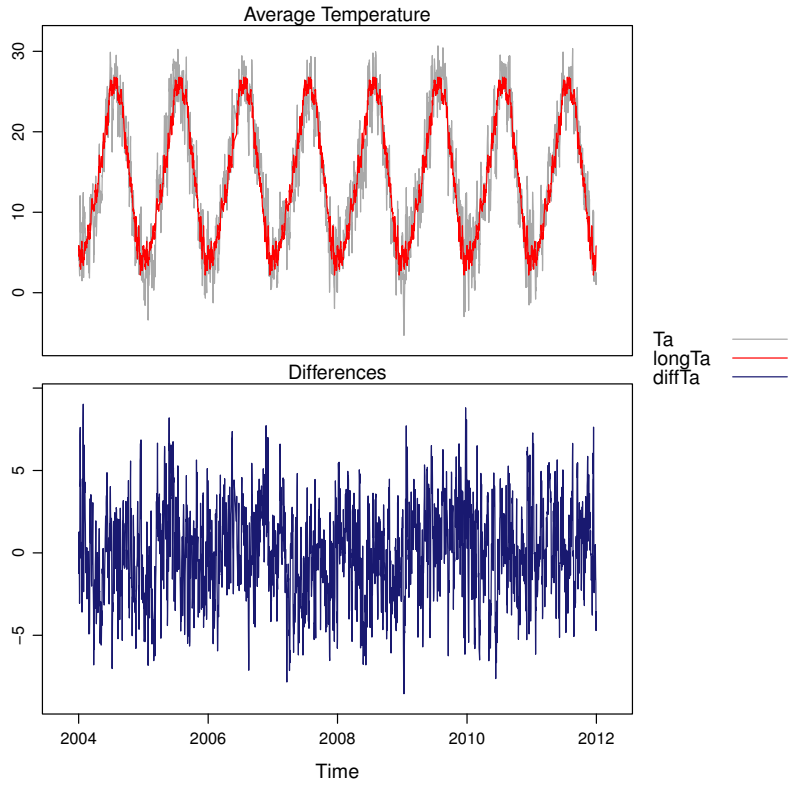

FIGURE 3.7: Daily temperature time series, its long-term average and the differences between them.

The temperature time series, the long-term average and the differences between them can be displayed with the `xyplot` method, now using `screens` to use a different panel for the differences time series (Figure 3.7)

```
xyplot(cbind(Ta, longTa, diffTa),
      col=c('darkgray', 'red', 'midnightblue'),
      superpose=TRUE, auto.key=list(space='right'),
      screens=c(rep('Average␣Temperature', 2), 'Differences'))
```

The horizon graph is better suited for displaying the differences. The next code again uses the cut-and-stack method (Figure 3.5) to distinguish between years. Figure 3.8 shows that 2004 started clearly above the aver-

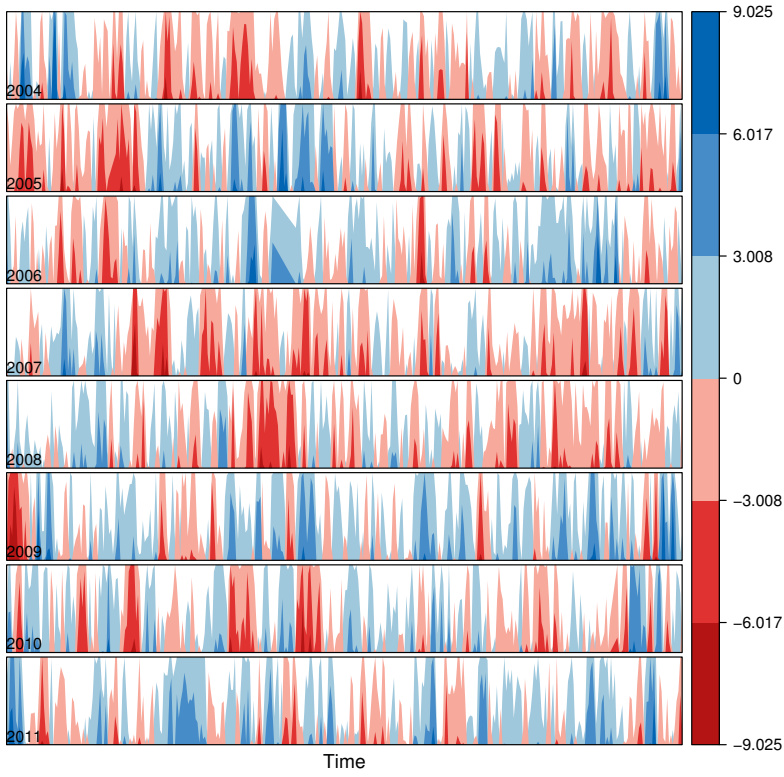

FIGURE 3.8: Horizon graph displaying differences between a daily temperature time series and its long-term average.

age while 2005 and 2009 did the contrary. Year 2007 was frequently below the long-term average but 2011 was more similar to that reference.

```
years <- unique(format(timeIndex, '%Y'))

horizonplot(diffTa, cut=list(n=8, overlap=0),
          colorkey=TRUE, layout=c(1, 8),
          scales=list(draw=FALSE, y=list(relation='same')),
          origin=0, strip.left=FALSE) +
   layer(grid.text(years[panel.number()], x = 0, y = 0.1,
                gp=gpar(cex=0.8),
                just = "left"))
```

A different approach to display this information is to produce a level plot displaying the time series using parts of its time index as independent and conditioning variables[2]. The following code displays the differences with the day of month on the horizontal axis and the year on the vertical axis, with a different panel for each month number. Therefore, each cell of Figure 3.9 corresponds to a certain day of the time series. If you compare this figure with the horizon plot, you will find the same previous findings but revealed now in more detail. On the other hand, while the horizon plot of Figure 3.8 clearly displays the yearly evolution, the combination of variables of the level plot focuses on the comparison between years in a certain month.

```
year <- function(x)as.numeric(format(x, '%Y'))
day <- function(x)as.numeric(format(x, '%d'))
month <- function(x)as.numeric(format(x, '%m'))
```

```
myTheme <- modifyList(custom.theme(region=brewer.pal(9, 'RdBu')),
                            list(
                              strip.background=list(col='gray'),
                              panel.background=list(col='gray')))

maxZ <- max(abs(diffTa))

levelplot(diffTa ~ day(timeIndex) * year(timeIndex) | factor(month(
    timeIndex)),
        at=pretty(c(-maxZ, maxZ), n=8),
        colorkey=list(height=0.3),
        layout=c(1, 12), strip=FALSE, strip.left=TRUE,
        xlab='Day', ylab='Month',
        par.settings=myTheme)
```

3.2.4 Interaction with `gridSVG`

The `gridSVG` package provides functions to convert `grid`-based `R` graphics to an SVG format. It provides several functions to add dynamic and interactive capabilities to `R` graphics. In this section we will use `grid.script`, a function to add JavaScript code to a plot.

The first step is to specify which component of the scene will run the JavaScript code. The `grid.ls` function returns a listing of the names of

[2]This approach was inspired by the `strip` function of the `metvurst` package (http://metvurst.blogspot.com.es/2012/11/plotting-large-amounts-of-atmospheric_4.html).

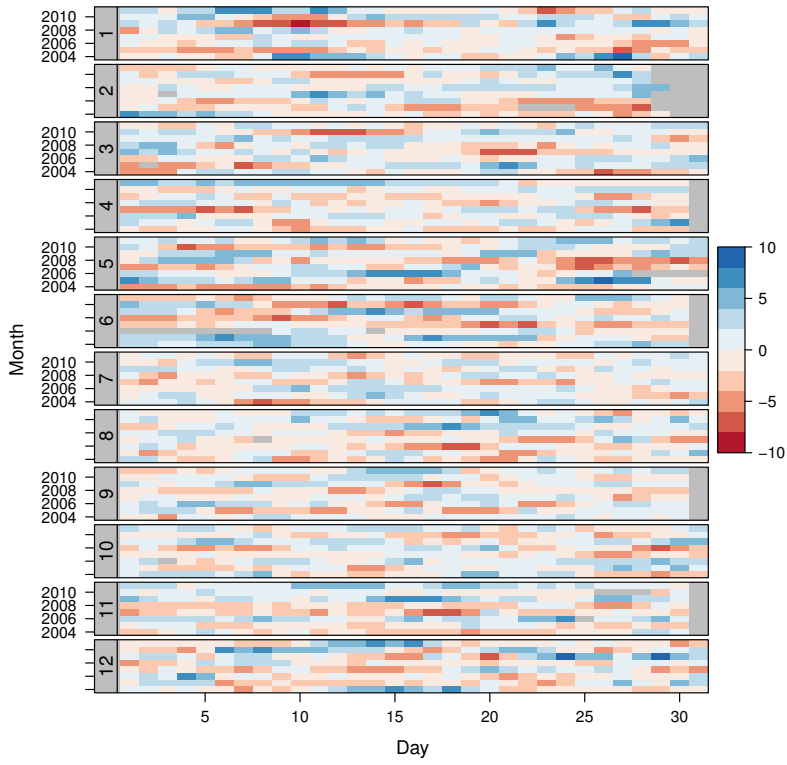

FIGURE 3.9: Level plot of differences between a daily temperature time series and its long-term average.

grobs or viewports included in the graphic output: only the lines will be connected with the JavaScript code.

```
library(gridSVG)
## grobs in the graphical output
pNavarra
grobs <- grid.ls(print=FALSE)
## only interested in some of them
nms <- grobs$name[grobs$type == "grobListing"]
idxNames <- grep('lines', nms)
IDs <- nms[idxNames]
```

The second step is to modify each `grob` (graphical object) to add attributes that specify when it will call JavaScript code. For each line identified with the elements of the `IDs` vector and associated to a meteorological station, the `navarra` object is accessed to extract the annual mean value of the daily radiation and the abbreviated name of the corresponding station (`info`). The `grid.garnish` function adds attributes to the `grob` of each line so that when the mouse moves over a `grob`, the line is highlighted and colored in red (`highlight`). When the mouse hovers out of the `grob`, the `hide` function sets back the default values of line width and transparency, but uses the green color to denote that this line has been already visited. In addition, because the browsers display the content of the title attribute with a default tooltip, `grid.garnish` sets this attribute to `info`.

```
for (id in unique(IDs)){
  ## extract information from the data
  ## according to the ID value
  i <- strsplit(id, '\\.')
  i <- sapply(i, function(x)as.numeric(x[5]))
  ## Information to be attached to each line: annual mean of daily
  ## radiation and abbreviated name of the station
  dat <- round(mean(navarra[,i], na.rm=TRUE), 2)
  info <- paste(names(navarra)[i], paste(dat, collapse=','),
              sep=': ')
  ## attach SVG attributes
  grid.garnish(id,
             onmouseover="highlight(evt)",
             onmouseout="hide(evt)",
             title=info)
}
```

These JavaScript functions are included in a script file named `highlight.js` (available at the website of the book). It can be added as an additional object with `grid.script`.

```
grid.script(filename="highlight.js")
```

This script is easy to understand, even without previous JavaScript knowledge:

```
highlight = function(evt){',
    evt.target.setAttribute('opacity', '1');
    evt.target.setAttribute('stroke', 'red');
    evt.target.setAttribute('stroke-width', '1');
}
```

```
hide = function(evt){
    evt.target.setAttribute('opacity', '0.3');
    evt.target.setAttribute('stroke', green');
    evt.target.setAttribute('stroke-width', '0.3');
}
```

Finally, `gridToSVG` exports the whole scene to SVG.

```
grid.export('figs/navarraRadiation.svg')
```

A snapshot of the result, as viewed in a browser with a line highlighted, is shown in Figure 3.10. Open the SVG file with your browser, explore it using the horizon graph (Figure 3.6) as a reference, and try to answer the questions raised with that graphic.

3.3 Stacked Graphs

If the variables of a multivariate time series can be summed to produce a meaningful global variable, they may be better displayed with stacked graphs. For example, the information on unemployment in the United States provides data of unemployed persons by industry and class of workers, and can be summed to give a total unemployment time series.

```
load('data/unemployUSA.RData')
```

The time series of unemployment can be directly displayed with the `xyplot.zoo` method (Figure 3.11).

```
xyplot(unemployUSA, superpose=TRUE, par.settings=custom.theme,
     auto.key=list(space='right'))
```

This graphical output is not very useful: the legend is confusing, with too many items; the vertical scale is dominated by the largest series, with several series buried in the lower part of the scale; the trend, variations and structure of the total and individual contributions cannot be deduced from this graph.

A suitable improvement is to display the multivariate time series as a set of stacked colored polygons to follow the macro/micro principle proposed by Tufte (Tufte 1990): Show a collection of individual time series and also display their sum. A traditional stacked graph is easily obtained with `geom_area`:

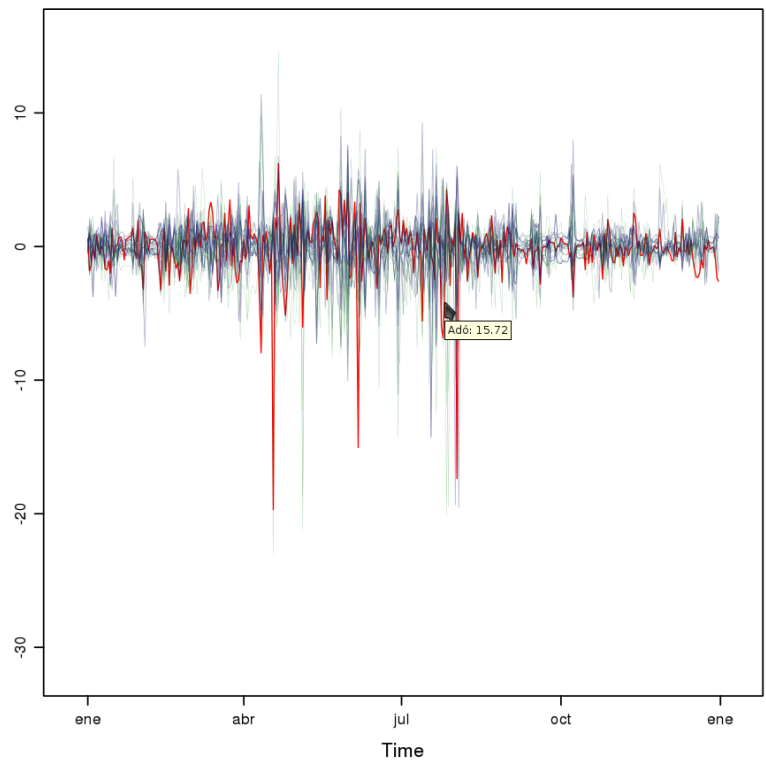

FIGURE 3.10: Snapshot of an SVG graphic produced with `gridSVG`.

```
library(scales) ## scale_x_yearmon needs scales::pretty_breaks
autoplot(unemployUSA, facets=NULL, geom='area') +
   geom_area(aes(fill=Series)) +
   scale_x_yearmon()
```

Traditional stacked graphs have their bottom on the x-axis which makes the overall height at each point easy to estimate. On the other hand, with this layout, individual layers may be difficult to distinguish. The *ThemeRiver* (Havre et al. 2002) (also named *streamgraph* in (Byron and Wattenberg 2008)) provides an innovative layout method in which layers are symmetrical around the x-axis at their center. At a glance, the pattern of the global sum and individual variables, their contribution to conform the global sum, and the interrelation between variables can be perceived.

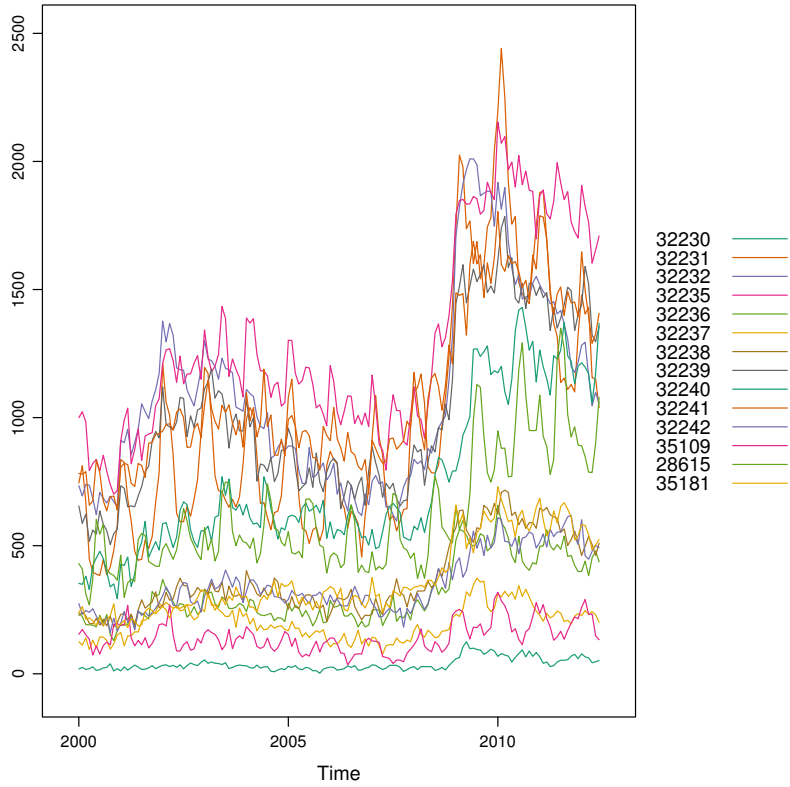

FIGURE 3.11: Time series of unemployment with `xyplot` using the default panel function.

FIGURE 3.12: Time series of unemployment with stacked areas using `geom_area`.

I have defined a panel and prepanel functions[3] to implement a ThemeRiver with `xyplot`. The result is displayed in Figure 3.13 with a vertical line to indicate one of main milestones of the financial crisis, whose effect on the overall unemployment results is clearly evident.

```
library(colorspace)
## We will use a qualitative palette from colorspace
nCols <- ncol(unemployUSA)
pal <- rainbow_hcl(nCols, c=70, l=75, start=30, end=300)
myTheme <- custom.theme(fill=pal, lwd=0.2)

sep2008 <- as.numeric(as.yearmon('2008-09'))

xyplot(unemployUSA, superpose=TRUE, auto.key=FALSE,
     panel=panel.flow, prepanel=prepanel.flow,
     origin='themeRiver', scales=list(y=list(draw=FALSE)),
     par.settings=myTheme) +
   layer(panel.abline(v=sep2008, col='gray', lwd=0.7))
```

This figure can help answer several questions. For example:

- What is the industry or class of worker with the lowest/highest unemployment figures during this time period?
- What is the industry or class of worker with the lowest/highest unemployment increases due to the financial crisis?
- There are a number of local maxima and minima of the total unemployment numbers. Are all the classes contributing to the maxima/minima? Do all the classes exhibit the same fluctuation behavior as the global evolution?

More questions and answers can be found in the "Current Employment Statistics" reports from the Bureau of Labor Statistics[4].

3.3.1 Panel and Prepanel Functions to Implement the ThemeRiver with `xyplot`

The `xyplot` function displays information according to the class of its first argument (methods) and to the `panel` function. We will use the `xyplot.zoo` method (equivalent to the `xyplot.ts` method) with a new custom `panel` function. This new panel function has four main arguments,

[3] The code of these panel and prepanel functions is explained in Section 3.3.1.

[4] The March 2012 highlights report is available at http://www.bls.gov/ces/highlights032012.pdf.

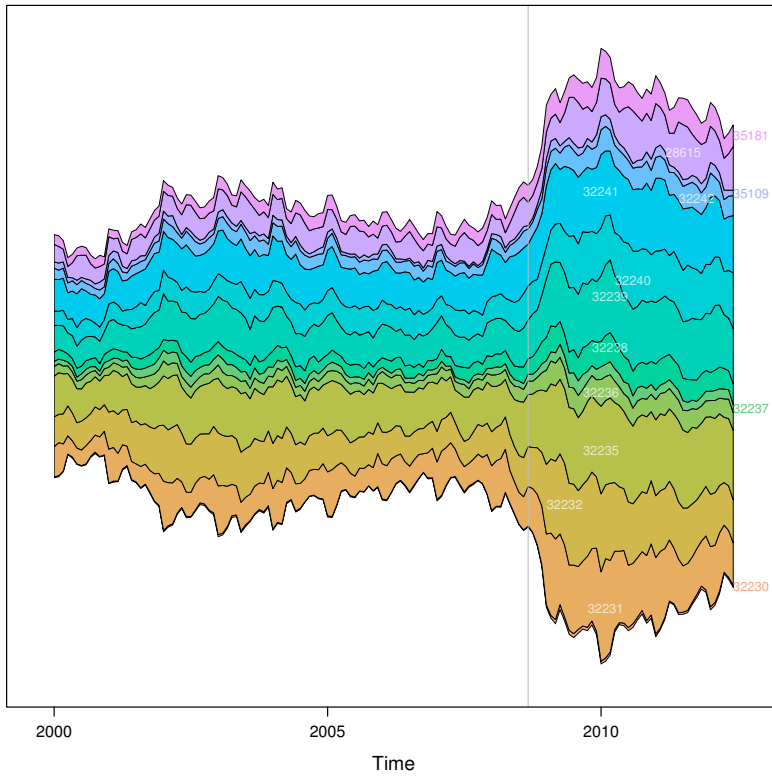

FIGURE 3.13: ThemeRiver of unemployment in the United States.

three of them calculated by `xyplot` (`x`, `y` and `groups`) and a new one, `origin`. Of course, it includes the `...` argument to provide additional arguments.

The first step is to create a `data.frame` with coordinates and with the `groups` factor. The value and number of the levels will be used in the main step of this `panel` function. With this `data.frame` we have to calculate the `y` and `x` coordinates for each group to get a stacked set of polygons.

This `data.frame` is in the *long* format, with a row for each observation, and where the `group` column identifies the variable. Thus, it must be transformed to the *wide* format, with a column for each variable. With the `unstack` function, a new `data.frame` is produced whose columns are defined according to the formula `y ~ groups` and with a row for each time

position. The stack of polygons is the result of the cumulative sum of each row (`apply(yWide, 1, cumsum)`). The origin of this sum is defined with the corresponding `origin` argument: with `themeRiver`, the polygons are arranged in a symmetric way.

Each column of this matrix of cumulative sums defines the `y` coordinate of each variable (where `origin` is now the first variable). The polygon of each variable is between this curve (`iCol+1`) and the one of the previous variable (`iCol`). In order to get a closed polygon, the coordinates of the inferior limit are in reverse order. This new `data.frame` (`Y`) is in the *wide* format, but `xyplot` requires the information in the *long* format: the `y` coordinates of the polygons are extracted from the `values` column of the *long* version of this `data.frame`.

The `x` coordinates are produced in an easier way. Again, `unstack` produces a `data.frame` with a column for each variable and a row for each time position, but now, because the `x` coordinates are the same for the set of polygons, the corresponding vector is constructed directly using a combination of concatenation and repetition.

Finally, the `groups` vector is produced, repeating each element of the columns of the original `data.frame` (`dat$groups`) twice to account for the forward and reverse curves of the corresponding polygon.

The final step before displaying the polygons is to acquire the graphical settings. The information retrieved with `trellis.par.get` is transferred to the corresponding arguments of `panel.polygon`.

Everything is ready for constructing the polygons. With a `for` loop, the coordinates of the corresponding group are extracted from the `x` and `y` vectors, and a polygon is displayed with `panel.polygon`. The labels of each polygon (the `levels` of the original `groups` variable, `groupLevels`) are printed inside the polygon if there is enough room for the text (`hChar>1`) or at the right if the polygon is too small, or if it is the first or last variable of the set. Both the polygons and the labels share the same color (`col[i]`).

```
panel.flow <- function(x, y, groups, origin, ...){
  dat <- data.frame(x=x, y=y, groups=groups)
  nVars <- nlevels(groups)
  groupLevels <- levels(groups)

  ## From long to wide
  yWide <- unstack(dat, y~groups)
  ## Where are the maxima of each variable located? We will use
  ## them to position labels.
  idxMaxes <- apply(yWide, 2, which.max)
```

```
##Origin calculated following Havr.eHetzler.ea2002
if (origin==’themeRiver’) origin= -1/2*rowSums(yWide)
else origin=0
yWide <- cbind(origin=origin, yWide)
## Cumulative sums to define the polygon
yCumSum <- t(apply(yWide, 1, cumsum))
Y <- as.data.frame(sapply(seq_len(nVars),
                    function(iCol)c(yCumSum[,iCol+1],
                                    rev(yCumSum[,iCol]))))
names(Y) <- levels(groups)
## Back to long format, since xyplot works that way
y <- stack(Y)$values

## Similar but easier for x
xWide <- unstack(dat, x~groups)
x <- rep(c(xWide[,1], rev(xWide[,1])), nVars)
## Groups repeated twice (upper and lower limits of the polygon)
groups <- rep(groups, each=2)

## Graphical parameters
superpose.polygon <- trellis.par.get("superpose.polygon")
col = superpose.polygon$col
border = superpose.polygon$border
lwd = superpose.polygon$lwd

## Draw polygons
for (i in seq_len(nVars)){
  xi <- x[groups==groupLevels[i]]
  yi <- y[groups==groupLevels[i]]
  panel.polygon(xi, yi, border=border,
              lwd=lwd, col=col[i])
}

## Print labels
for (i in seq_len(nVars)){
  xi <- x[groups==groupLevels[i]]
  yi <- y[groups==groupLevels[i]]
  N <- length(xi)/2
  ## Height available for the label
  h <- unit(yi[idxMaxes[i]], ’native’) -
    unit(yi[idxMaxes[i] + 2*(N-idxMaxes[i]) +1], ’native’)
  ##...converted to "char" units
  hChar <- convertHeight(h, ’char’, TRUE)
```

```
    ## If there is enough space and we are not at the first or
    ## last variable, then the label is printed inside the polygon.
    if((hChar >= 1) && !(i %in% c(1, nVars))){
      grid.text(groupLevels[i],
                xi[idxMaxes[i]],
                (yi[idxMaxes[i]] +
                 yi[idxMaxes[i] + 2*(N-idxMaxes[i]) +1])/2,
                gp = gpar(col='white', alpha=0.7, cex=0.7),
                default.units='native')
    } else {
      ## Elsewhere, the label is printed outside

      grid.text(groupLevels[i],
                xi[N],
                (yi[N] + yi[N+1])/2,
                gp=gpar(col=col[i], cex=0.7),
                just='left', default.units='native')
    }
  }
}
```

With this panel function, `xyplot` displays a set of stacked polygons corresponding to the multivariate time series (Figure 3.14). However, the graphical window is not large enough, and part of the polygons fall out of it. Why?

```
xyplot(unemployUSA, superpose=TRUE, auto.key=FALSE,
       panel=panel.flow, origin='themeRiver',
       par.settings=myTheme, cex=0.4, offset=0,
       scales=list(y=list(draw=FALSE)))
```

The problem is that `lattice` makes a preliminary estimate of the window size using a default `prepanel` function that is unaware of the internal calculations of our new `panel.flow` function. The solution is to define a new `prepanel.flow` function.

The input arguments and first lines are the same as in `panel.flow`. The output is a list whose elements are the limits for each axis (`xlim` and `ylim`), and the sequence of differences (`dx` and `dy`) that can be used for the aspect and banking calculations.

The limits of the x-axis are defined with the range of the time index, while the limits of the y-axis are calculated with the minimum of the first column of `yCumSum` (the origin line) and with the maximum of its last column (the upper line of the cumulative sum).

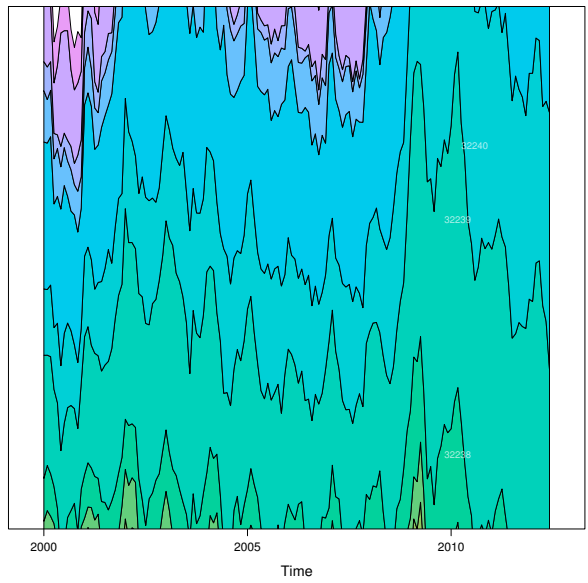

FIGURE 3.14: First attempt of ThemeRiver.

```
prepanel.flow <- function(x, y, groups, origin,...){
  dat <- data.frame(x=x, y=y, groups=groups)
  nVars <- nlevels(groups)
  groupLevels <- levels(groups)
  yWide <- unstack(dat, y~groups)
  if (origin=='themeRiver') origin= -1/2*rowSums(yWide)
  else origin=0
  yWide <- cbind(origin=origin, yWide)
  yCumSum <- t(apply(yWide, 1, cumsum))

  list(xlim=range(x),
       ylim=c(min(yCumSum[,1]), max(yCumSum[,nVars+1])),
       dx=diff(x),
       dy=diff(c(yCumSum[,-1])))
}
```

Chapter 4

Time as a Conditioning or Grouping Variable

In Section 3.1 we learned to display the time evolution of multiple time series with different scales. But, what if instead of displaying the time evolution we want to confront the variables between them? Section **??** proposes the scatterplot matrix solution with time as a grouping variable. Section **??** uses an enhanced scatterplot with time as a conditioning variable. Section **??** includes a digression about the hexagonal binning for large datasets.

4.1 Scatterplot Matrix: Time as a Grouping Variable

The scatterplot matrices are based on the technique of small multiples (Tufte 1990): small, thumbnail-sized representations of multiple images displayed all at once, which allows the reader to immediately, and in parallel, compare the inter-frame differences. A scatterplot matrix is a display of all pairwise bivariate scatterplots arranged in a $p \times p$ matrix for p variables. Each subplot shows the relation between the pair of variables at the intersection of the row and column indicated by the variable names in the diagonal panels (Friendly and Denis 2005).

This graphical tool is implemented in the `splom` function[1]. The following code displays the relation between the set of meteorological variables

[1] `ggplot2` users may wish to explore the `ggpairs` function from the `GGally` package.

using a sequential palette from the ColorBrewer catalog (`RbBu`, with black added to complete a twelve-color palette) to encode the month. The order of colors of this palette is chosen in order to display summer months with intense colors and to distinguish between the first and second half of the year with red and blue, respectively (Figure 4.1).

```
load('data/aranjuez.RData')

## Red-Blue palette with black added (12 colors)
colors <- c(brewer.pal(n=11, 'RdBu'), '#000000')
## Rearrange according to months (darkest for summer)
colors <- colors[c(6:1, 12:7)]

splom(~as.data.frame(aranjuez),
      groups=format(index(aranjuez), '%m'),
     auto.key=list(space='right',
        title='Month', cex.title=1),
     pscale=0, varname.cex=0.7, xlab='',
      par.settings=custom.theme(symbol=colors,
         pch=19), cex=0.3, alpha=0.1)
```

Let's explore Figure 4.1. For example,

- The highest values of ambient temperature (average, maximum, and mimimun), solar radiation, and evotranspiration can be found during the summer.
- These variables are almost linearly related. The relation between radiation and temperature is different during both halves of the year (red and blue regions can be easily distinguished).
- The humidity reaches its highest values during winter without appreciable differences between the first and second half of the year. The temperature and humidity may be related with an exponential function.

A bit of interactivity can be added to this plot with the identification of some points. This task is easy with `panel.link.splom`. The points are selected via mouse clicks (and highlighted in green). Clicks other than left-clicks terminate the procedure. The output of this function is the index of chosen points.

```
trellis.focus('panel', 1, 1)
idx <- panel.link.splom(pch=13, cex=0.6, col='green')
aranjuez[idx,]
```

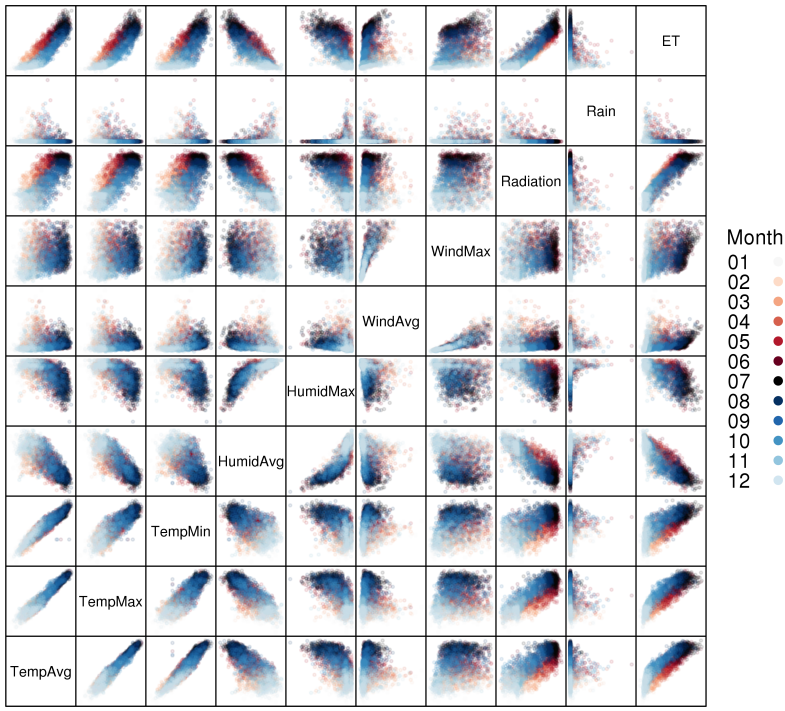

FIGURE 4.1: Scatter plot matrix of the collection of meteorological time series of the Aranjuez station.

4.1.1 Hexagonal Binning

For large datasets, the display of a large number of points in a scatterplot produces hidden point density, long computation times, and slow displays. These problems can be circumvented with the estimation and representation of points densities. A common encoding uses gray scales, pseudo colors or partial transparency. An improved scheme encodes density as the size of hexagon symbols inscribed within hexagonal binning regions (D. B. Carr et al. 1987).

The `hexbin` package (D. Carr, Lewin-Koh, and Maechler 2013) includes several functions for hexagonal binning. The `panel.hexbinplot` is a good substitute for the default panel function. In addition, our first attempt with `splom` can be improved with several modifications (Figure **??**):

- The scale's ticks and labels are suppressed with `pscale=0`.
- The panels of the lower part of the matrix (`lower.panel`) will include a locally weighted scatterplot smoothing (loess) with `panel.loess`.
- The diagonal panels (`diag.panel`) will display the kernel density estimate of each variable. The `density` function computes this estimate. The result is adjusted to the panel limits (calculated with `current.panel.limits`). The kernel density is plotted with `panel.lines` and the `diag.panel.splom` function completes the content of each diagonal panel.
- The point density is encoded with the palette `BTC` (ligther colors for high density values and darker colors for almost empty regions, with a gradient of blue hues for intermediate values).

```
library(hexbin)

splom(~as.data.frame(aranjuez),
         panel=panel.hexbinplot, xlab='',
         colramp=BTC,
         diag.panel = function(x, ...){
           yrng <- current.panel.limits()$ylim
           d <- density(x, na.rm=TRUE)
           d$y <- with(d, yrng[1] + 0.95 * diff(yrng) * y / max(y))
           panel.lines(d)
           diag.panel.splom(x, ...)
         },
         lower.panel = function(x, y, ...){
           panel.hexbinplot(x, y, ...)
           panel.loess(x, y, ..., col = 'red')
         },
         pscale=0, varname.cex=0.7
         )
```

A drawback of the matrix of scatterplots with hexagonal binning is that each panel is drawn independently, so it is impossible to compute a common color key for all of them. In other words, two cells with exactly the same color in different panels encode different point densities.

It is possible to display a reduced set of variables against another one and generate a common color key using the `hexbinplot` function. First, the dataset must be reshaped from the wide format (one colum for each

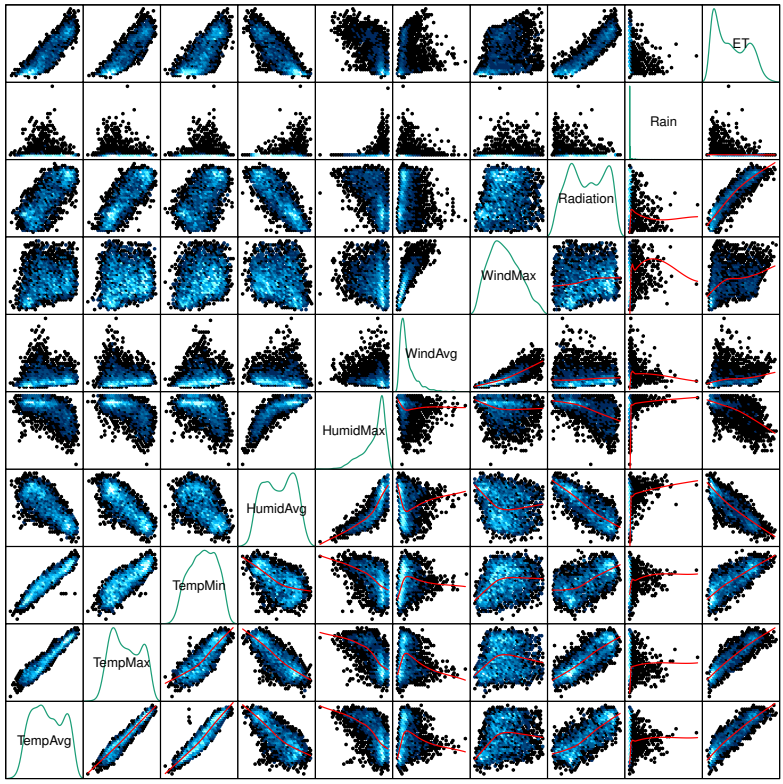

FIGURE 4.2: Scatterplot matrix of the collection of meteorological time series of the Aranjuez station using hexagonal binning.

variable) to the long format (only one column for the values with one row for each observation).

The `reshape` function needs several arguments to perform the conversion. The most important is the `data.frame` to be transformed. Then there are the names of variables to be mapped to a single variable in the long dataset (the three ambient temperatures). The name of this variable can be set with `v.names`. Finally, `timevar` is the name of the column in long format that differentiates multiple observations from the same variable. The values of this column are defined with the `times` argument.

```
aranjuezDF <- data.frame(aranjuez,
                    month=format(index(aranjuez), '%m'))
aranjuezRshp <- reshape(aranjuezDF, direction='long',
                    varying=list(names(aranjuez)[1:3]),
                    v.names='Temperature',
                    times=names(aranjuez)[1:3],
                    timevar='Statistic')
```

```
head(aranjuezRshp)
```

The `hexbinplot` displays this dataset with a different panel for each type of temperature (average, maximum, and minimum) but with a common color key encoding the point density (Figure 4.3). Now, two cells with the same color in different panels encode the same value.

```
hexbinplot(Radiation~Temperature|Statistic, data=aranjuezRshp,
         layout=c(1, 3), colramp=BTC) +
   layer(panel.loess(..., col = 'red'))
```

The ggplot2 version uses `stat_binhex`.

```
ggplot(data=aranjuezRshp, aes(Temperature, Radiation)) +
   stat_binhex(ncol=1) +
   stat_smooth(se=FALSE, method='loess', col='red') +
   facet_wrap(~Statistic, ncol=1) +
   theme_bw()
```

4.2 Scatterplot with Time as a Conditioning Variable

After discussing the hexagonal binning, let's recover the time variable. Figure 4.1 uses colors to encode months. Instead, we will now display separate scatterplots with a panel for each month. In addition, the statistic

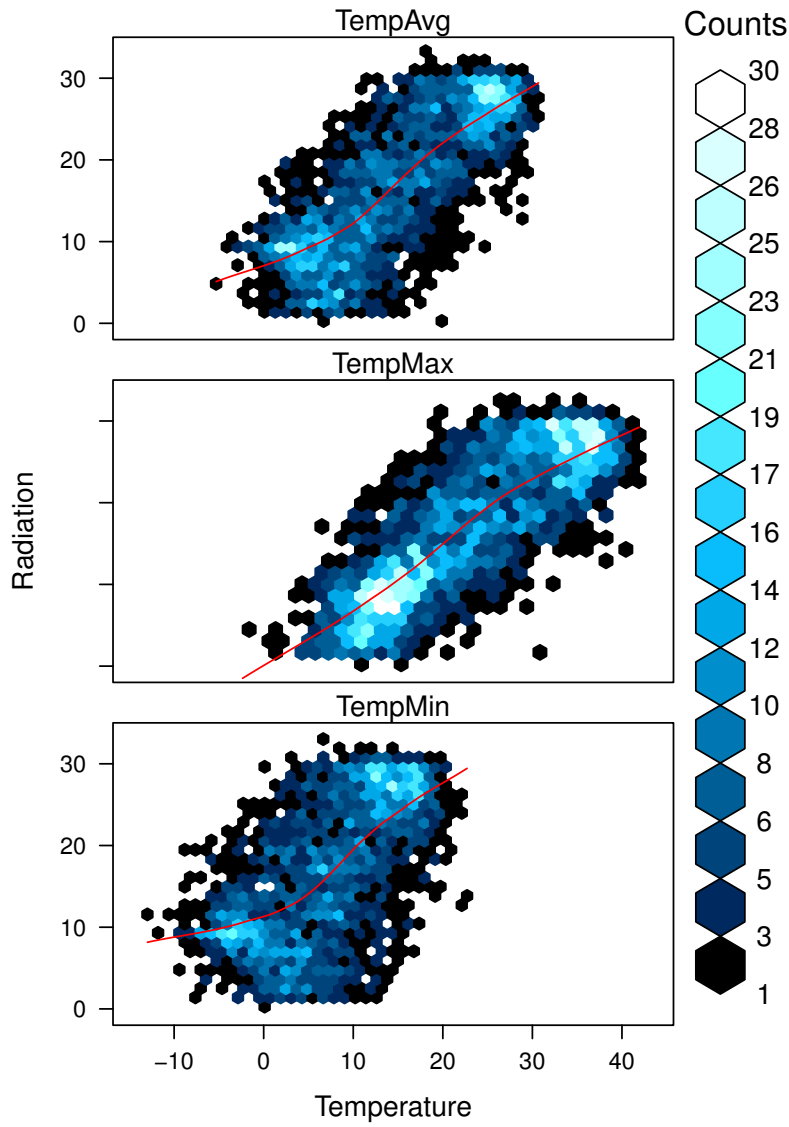

FIGURE 4.3: Scatterplot with hexagonal binning of temperature versus solar radiation using data of the Aranjuez station (`lattice` version).

type (average, maximum, minimum) is included as an additional conditioning variable.

This matrix of panels can be displayed with `ggplot` using `facet_grid`. The code of Figure 4.4 uses partial transparency to cope with overplotting, small horizontal and vertical segments (`geom_rug`) to display points density on both variables, and a smooth line in each panel.

```
ggplot(data=aranjuezRshp, aes(Radiation, Temperature)) +
   facet_grid(Statistic ~ month) +
   geom_point(col='skyblue4', pch=19, cex=0.5, alpha=0.3) +
   geom_rug() +
   stat_smooth(se=FALSE, method='loess', col='indianred1', lwd
       =1.2) +
   theme_bw()
```

The version with `lattice` needs the `useOuterStrips` function from the `latticeExtra` package, which prints the names of the conditioning variables on the top and left outer margins (Figure 4.5).

```
useOuterStrips(xyplot(Temperature ~ Radiation | month * Statistic,
                  data=aranjuezRshp,
                  between=list(x=0),
                  col='skyblue4', pch=19,
                  cex=0.5, alpha=0.3)) +
   layer({
      panel.rug(..., col.line='indianred1', end=0.05, alpha=0.6)
      panel.loess(..., col='indianred1', lwd=1.5, alpha=1)
   })
```

These figures show the typical seasonal behavior of solar radiation and ambient temperature. Additionally, it displays in more detail the same relations between radiation and temperature already discussed with Figure 4.3.

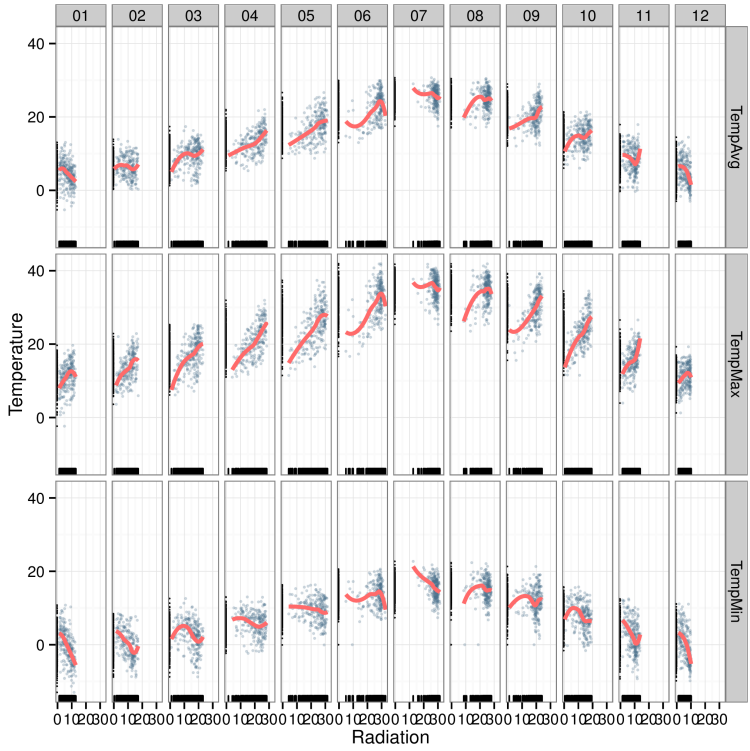

FIGURE 4.4: Scatterplot of temperature versus solar radiation for each month using data of the Aranjuez station (`ggplot2` version).

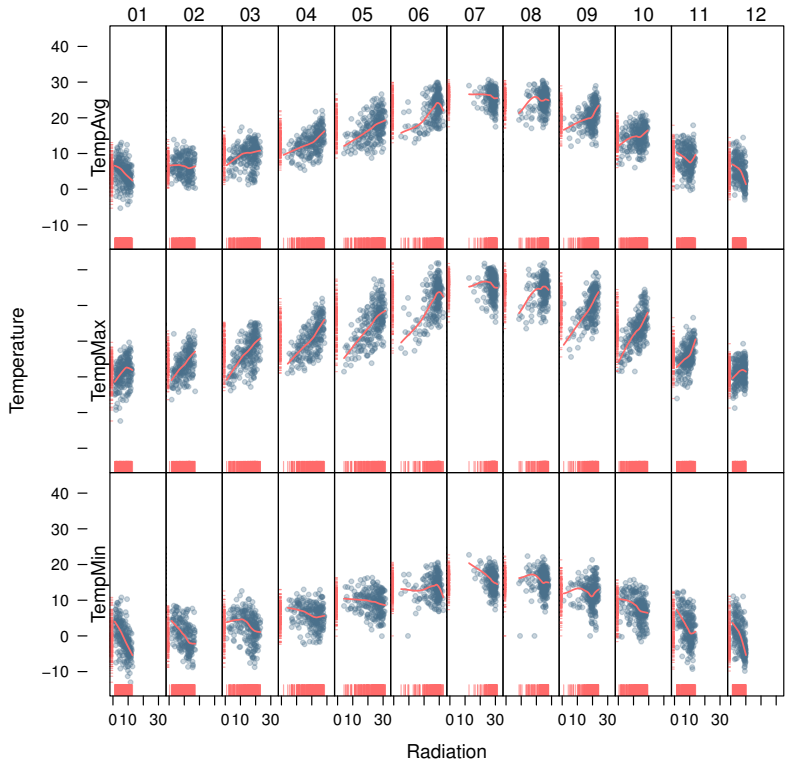

FIGURE 4.5: Scatterplot of temperature versus solar radiation for each month using data of the Aranjuez station (lattice version).

Chapter 5

Time as a Complementary Variable

Gapminder [1] is an independent foundation based in Stockholm, Sweden. Its mission is "to debunk devastating myths about the world by offering free access to a fact-based world view." They provide free online tools, data, and videos "to better understand the changing world." The initial development of Gapminder was the Trendalyzer software, used by Hans Rosling in several sequences of his documentary "The Joy of Stats."

The information visualization technique used by Trendalyzer is an interactive bubble chart. By default it shows five variables: two numeric variables on the vertical and horizontal axes, bubble size and color, and a time variable that may be manipulated with a slider. The software uses brushing and linking techniques for displaying the numeric value of a highlighted country.

This software was acquired by Google in 2007, and is now available as a Motion Chart gadget and as the Public Data Explorer.

In this chapter, time will be used as a complementary variable which adds information to a graph where several variables are confronted. We will illustrate this approach with the evolution of the relationship between Gross National Income (GNI) and carbon dioxide (CO_2) emissions for a set

[1] http://www.gapminder.org/

of countries extracted from the database of the World Bank Open Data. We will try several solutions to display the relationship between CO_2 emissions and GNI over the years using time as a complementary variable. The final method will produce an animated plot resembling the Trendalyzer solution.

5.1 Polylines

The first solution is a Motion Chart the `googleVis` package (Gesmann and Castillo 2011), an interface between R and the Google Visualisation API. With its `gvisMotionChart` function it is easy to produce a Motion Chart that can be displayed using a browser with Flash enabled (Figure 5.1).

```
load('data/CO2.RData')
```

```
library(googleVis)
pgvis <- gvisMotionChart(CO2data, idvar='Country.Name', timevar='
    Year')
```

Although the `gvisMotionChart` is quite easy to use, the global appearance and behavior are completely determined by Google API[2]. Moreover, you should carefully read their Terms of Use before using it for public distribution.

Our next attempt is to display the entire data in a panel with a scatterplot using country names as the grouping factor. Points of each country are connected with polylines to reveal the time evolution (Figure 5.2).

```
## lattice version
xyplot(GNI.capita ~ CO2.capita, data=CO2data,
     xlab="Carbon␣dioxide␣emissions␣(metric␣tons␣per␣capita)",
     ylab="GNI␣per␣capita,␣PPP␣(current␣international␣$)",
     groups=Country.Name, type='b')
```

```
## ggplot2 version
ggplot(data=CO2data, aes(x=CO2.capita, y=GNI.capita,
         color=Country.Name)) +
   xlab("Carbon␣dioxide␣emissions␣(metric␣tons␣per␣capita)") +
   ylab("GNI␣per␣capita,␣PPP␣(current␣international␣$)") +
   geom_point() + geom_path() + theme_bw()
```

Three improvements can be added to this graphical result:

[2]You should read the Google API Terms of Service before using `googleVis`: https://developers.google.com/terms/.

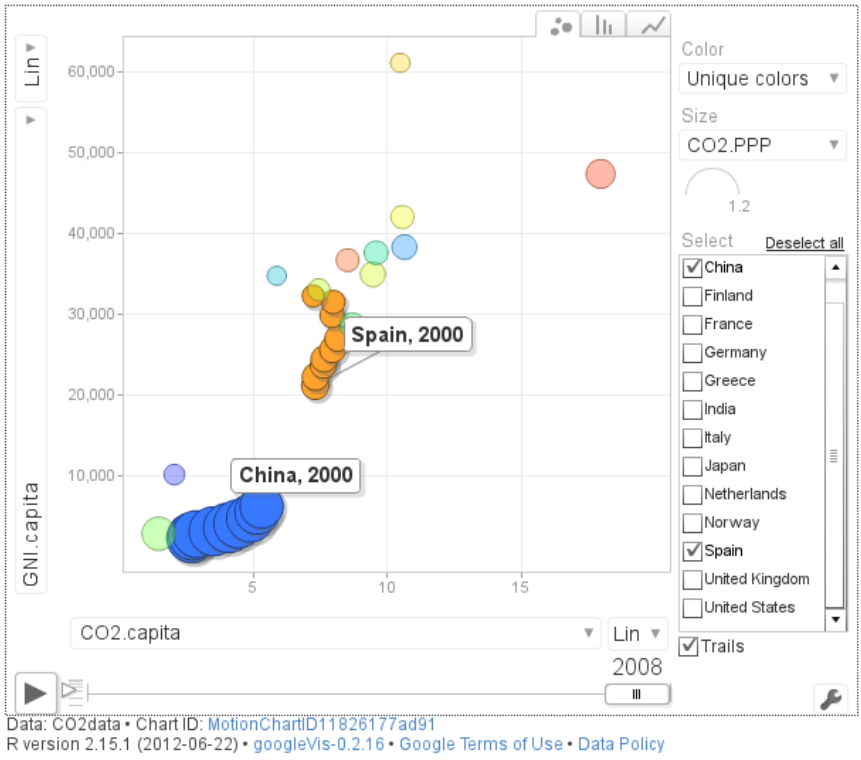

FIGURE 5.1: Snapshot of a Motion Chart produced with googleVis.

1. Define a better palette to enhance visual discrimination between countries.
2. Display time information with labels to show year values.
3. Label each polyline with the country name instead of a legend.

5.2 Choosing Colors

The `Country.Name` categorical variable will be encoded with a qualitative palette, namely the first five colors of `Set1` palette[3] from the `RColorBrewer`

[3] `http://colorbrewer2.org/`

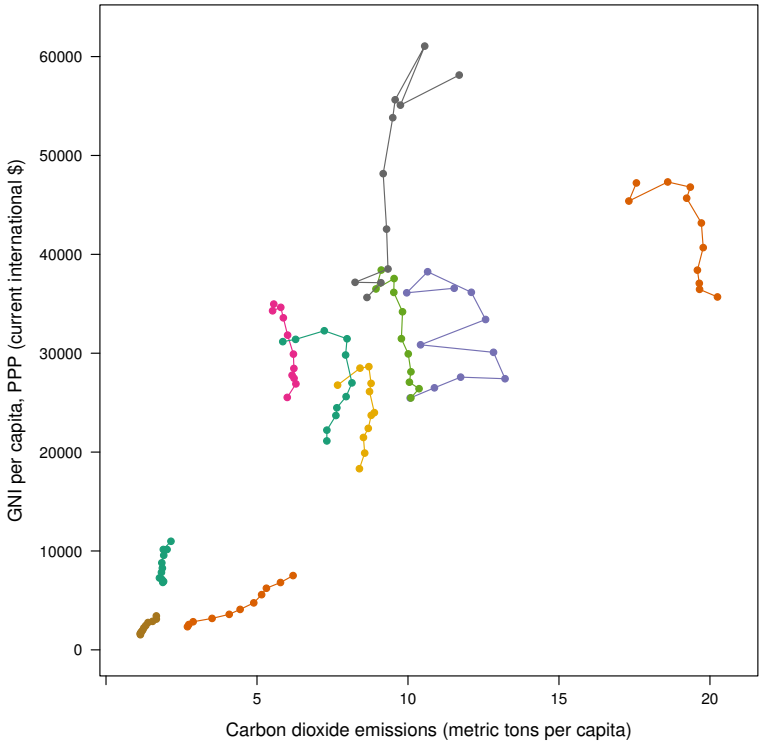

FIGURE 5.2: GNI per capita versus CO_2 emissions per capita (`lattice` version).

package (Neuwirth 2011). Because there are more countries than colors, we have to repeat some colors to complete the number of levels of the variable `Country.Name`. The result is a palette with non-unique colors, and thus some countries will share the same color. This is not a problem because the curves will be labeled, and countries with the same color will be displayed at enough distance.

```
library(RColorBrewer)

nCountries <- nlevels(CO2data$Country.Name)
pal <- brewer.pal(n=5, 'Set1')
pal <- rep(pal, length = nCountries)
```

Adjacent colors of this palette are chosen to be easily distinguishable. Therefore, the connection between colors and countries must be in such a way that nearby lines are encoded with adjacent colors of the palette.

A simple approach is to calculate the annual average of the variable to be represented along the x-axis (`CO2.capita`), and extract colors from the palette according to the order of this value.

```
## Rank of average values of CO2 per capita
CO2mean <- aggregate(CO2.capita ~ Country.Name, data=CO2data, FUN=
    mean)
palOrdered <- pal[rank(CO2mean$CO2.capita)]
```

A more sophisticated solution is to use the ordered results of a hierarchical clustering of the time evolution of the CO_2 per capita values (Figure 5.3). The data is extracted from the original CO_2 `data.frame`.

```
CO2capita <- CO2data[, c('Country.Name', 'Year', 'CO2.capita')]
CO2capita <- reshape(CO2capita, idvar='Country.Name', timevar='Year'
    , direction='wide')
hCO2 <- hclust(dist(CO2capita[, -1]))

oldpar <- par(mar=c(0, 2, 0, 0) + .1)
plot(hCO2, labels=CO2capita$Country.Name,
    xlab='', ylab='', sub='', main='')
par(oldpar)
```

The colors of the palette are assigned to each country with `match`, which returns a vector of the positions of the matches of the country names in alphabetical order in the country names ordered according to the hierarchical clustering.

```
idx <- match(levels(CO2data$Country.Name),
```

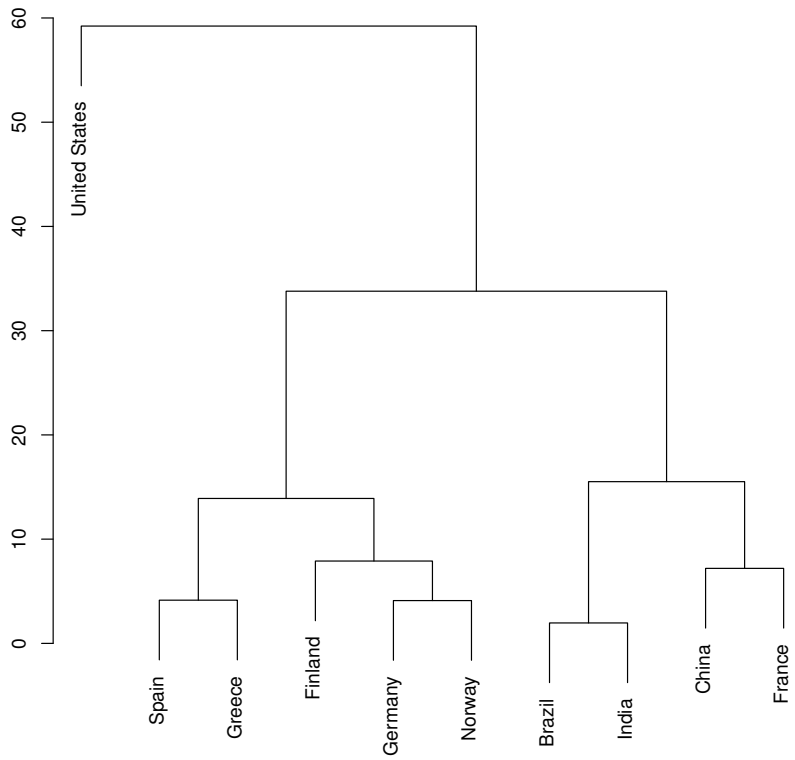

FIGURE 5.3: Hierarchical clustering of the time evolution of CO_2 per capita values.

```
          CO2capita$Country.Name[hCO2$order])
palOrdered <- pal[idx]
```

It must be highlighted that this palette links colors with the levels of `Country.Name` (country names in alphabetical order), which is exactly what the `groups` argument provides. The following code produces a curve for each country using different colors to distinguish them.

```
## simpleTheme encapsulates the palette in a new theme for xyplot
myTheme <- simpleTheme(pch=19, cex=0.6, col=palOrdered)

pCO2.capita <- xyplot(GNI.capita ~ CO2.capita,
                  xlab="Carbon␣dioxide␣emissions␣(metric␣tons␣per␣
                      capita)",
```

```
                ylab="GNI␣per␣capita,␣PPP␣(current␣international␣$)
                    ",
                groups=Country.Name, data=CO2data,
                par.settings=myTheme,
                type='b')
```

```
gCO2.capita <- ggplot(data=CO2data, aes(x=CO2.capita, y=GNI.capita,
                color=Country.Name)) +
   geom_point() + geom_path() +
   scale_color_manual(values=palOrdered, guide=FALSE) +
   xlab('CO2␣emissions␣(metric␣tons␣per␣capita)') +
   ylab('GNI␣per␣capita,␣PPP␣(current␣international␣$)') +
   theme_bw()
```

5.3 Labels to Show Time Information

This result can be improved with labels displaying the years to show the time evolution. A panel function with `panel.text` to print the year labels and `panel.superpose` to display the lines for each group is a solution. In the panel function, `subscripts` is a vector with the integer indices representing the rows of the `data.frame` to be displayed in the panel.

```
xyplot(GNI.capita ~ CO2.capita,
     xlab="Carbon␣dioxide␣emissions␣(metric␣tons␣per␣capita)",
     ylab="GNI␣per␣capita,␣PPP␣(current␣international␣$)",
     groups=Country.Name, data=CO2data,
     par.settings=myTheme,
     type='b',
     panel=function(x, y, ..., subscripts, groups){
       panel.text(x, y, ...,
                labels=CO2data$Year[subscripts],
                pos=2, cex=0.5, col='gray')
       panel.superpose(x, y, subscripts, groups,...)
     }
     )
```

The same result with a clearer code is obtained with the combination of `+.trellis`, `glayer_` and `panel.text`. Using `glayer_` instead of `glayer`, we ensure that the labels are printed below the lines.

```
pCO2.capita <- pCO2.capita +
   glayer_(panel.text(..., labels=CO2data$Year[subscripts],
                   pos=2, cex=0.5, col='gray'))
```

```
gCO2.capita <- gCO2.capita + geom_text(aes(label=Year),
                                colour='gray',
                                size=2.5,
                                hjust=0, vjust=0)
```

5.4 Country Names: Positioning Labels

The common solution to link each curve with the group value is to add a legend. However, a legend can be confusing with too many items. In addition, the reader must carry out a complex task: Choose the line, memorize its color, search for it in the legend, and read the country name.

A better approach is to label each line using nearby text with the same color encoding. A suitable method is to place the labels close to the end of each line (Figure 5.4). Labels are placed with the `panel.pointLabel` function from the `maptools` package. This function use optimization routines to find locations without overlaps.

```
library(maptools)
## group.value provides the country name; group.number is the
## index of each country to choose the color from the palette.
pCO2.capita +
   glayer(panel.pointLabel(mean(x), mean(y),
                       labels= group.value,
                       col=palOrdered[group.number],
                       cex=.8,
                       fontface=2, fontfamily='Palatino'))
```

However, this solution does not solve the overlapping between labels and lines. The package `directlabels` (Hocking 2013) includes a wide repertory of positioning methods to cope with this problem. The main function, `direct.label`, is able to determine a suitable method for each plot, although the user can choose a different method from the collection or even define a custom method. For the `pCO2.capita` object, I have obtained the best results with `extreme.grid` (Figure 5.5).

```
library(directlabels)
direct.label(pCO2.capita, method='extreme.grid')
```

```
direct.label(gCO2.capita, method='extreme.grid')
```

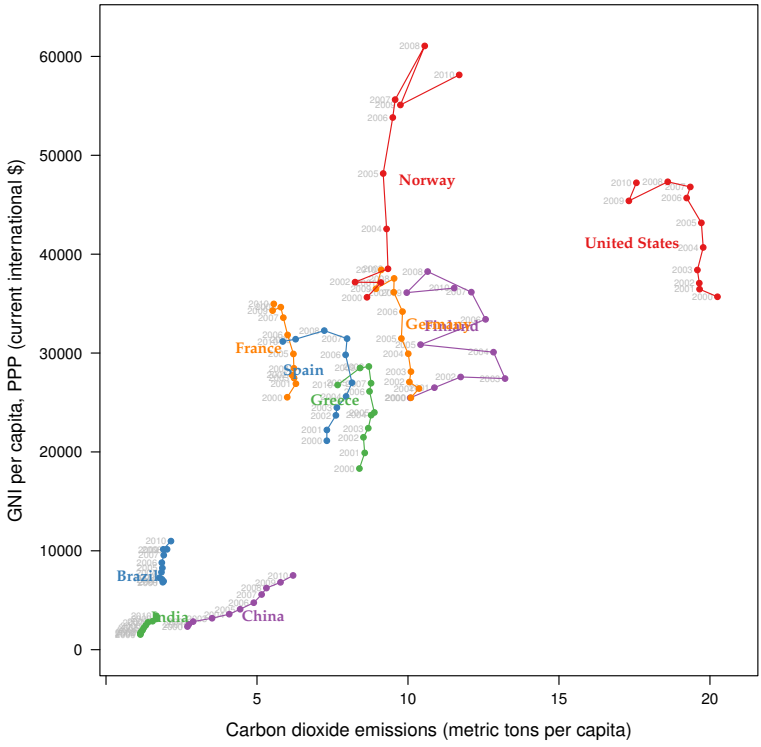

FIGURE 5.4: CO_2 emissions versus GNI per capita. Labels are placed with `panel.pointLabel`.

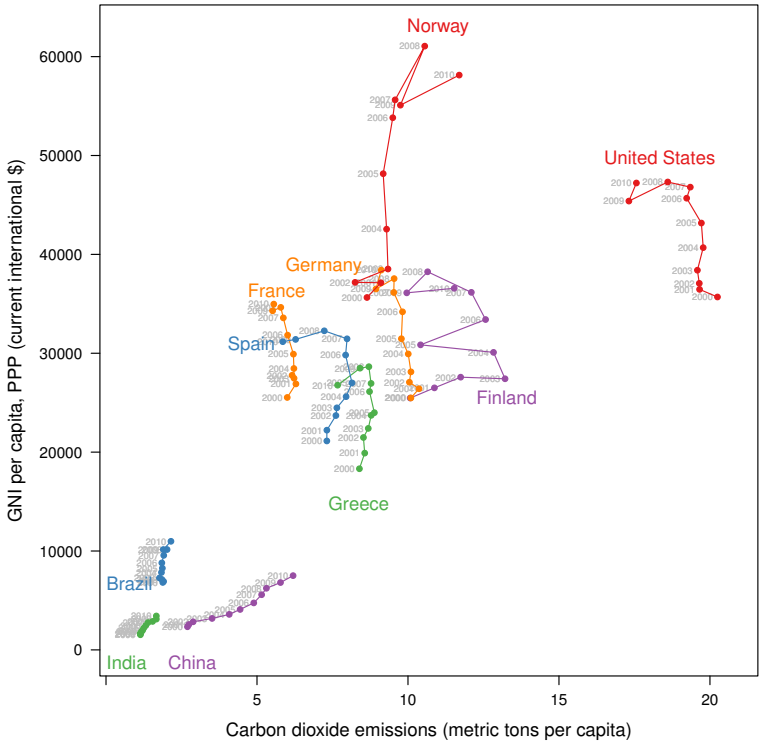

FIGURE 5.5: CO_2 emissions versus GNI per capita. Labels are placed with the `extreme.grid` method of the `directlabels` package.

5.5 A Panel for Each Year

Time can be used as a conditioning variable (as shown in previous sections) to display subsets of the data in different panels. Figure 5.6 is produced with the same code as in Figure 5.2, now including `|factor(Year)` in the lattice version and `facet_wrap(~ Year)` in the `ggplot2` version.

```
xyplot(GNI.capita ~ CO2.capita | factor(Year), data=CO2data,
     xlab="Carbon␣dioxide␣emissions␣(metric␣tons␣per␣capita)",
     ylab="GNI␣per␣capita,␣PPP␣(current␣international␣$)",
     groups=Country.Name, type='b',
     auto.key=list(space='right'))
```

```
ggplot(data=CO2data, aes(x=CO2.capita, y=GNI.capita, colour=Country.
    Name)) +
   facet_wrap(~ Year) + geom_point(pch=19) +
   xlab('CO2␣emissions␣(metric␣tons␣per␣capita)') +
   ylab('GNI␣per␣capita,␣PPP␣(current␣international␣$)') +
   theme_bw()
```

Because the grouping variable, `Country.Name`, has many levels, the legend is not very useful. Once again, point labeling is recommended (Figure 5.7).

```
xyplot(GNI.capita ~ CO2.capita | factor(Year), data=CO2data,
     xlab="Carbon␣dioxide␣emissions␣(metric␣tons␣per␣capita)",
     ylab="GNI␣per␣capita,␣PPP␣(current␣international␣$)",
     groups=Country.Name, type='b',
     par.settings=myTheme) +
   glayer(panel.pointLabel(x, y, labels=group.value,
                       col=palOrdered[group.number], cex=0.7))
```

5.5.1 Using Variable Size to Encode an Additional Variable

Instead of using simple points, we can display circles of different radius to encode a new variable. This new variable is `CO2.PPP`, the ratio of CO_2 emissions to the Gross Domestic Product with purchasing power parity (PPP) estimations.

To use this numeric variable as an additional grouping factor, its range must be divided into different classes. The typical solution is to use `cut` to coerce the numeric variable into a `factor` whose levels correspond to uniform intervals, which could be unrelated to the data distribution. The `classInt` package (R. Bivand 2013) provides several methods to partition data into classes based on natural groups in the data distribution.

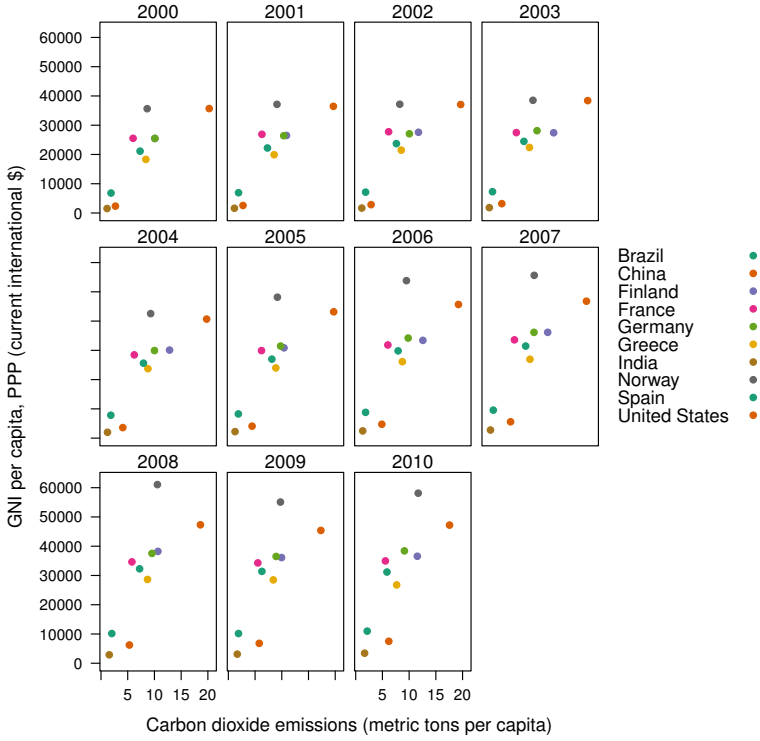

FIGURE 5.6: CO_2 emissions versus GNI per capita with a panel for each year.

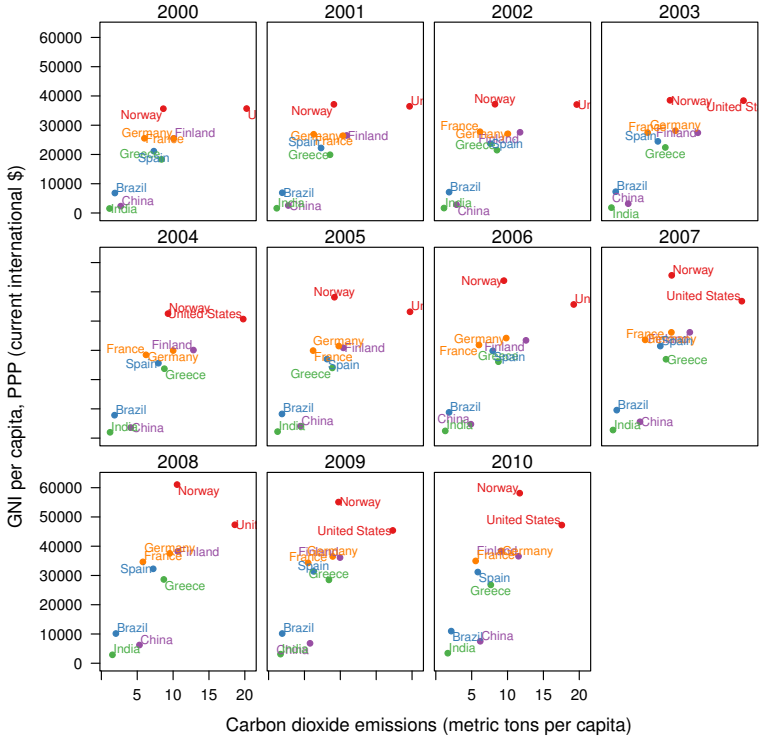

FIGURE 5.7: CO_2 emissions versus GNI per capita with a panel for each year.

```
library(classInt)
z <- CO2data$CO2.PPP
intervals <- classIntervals(z, n=4, style='fisher')
```

Although the functions of this package are mainly intended to create color palettes for maps, the results can also be associated to point sizes. `cex.key` defines the sequence of sizes (to be displayed in the legend) associated with each `CO2.PPP` using the `findCols` function.

```
nInt <- length(intervals$brks) - 1
cex.key <- seq(0.5, 1.8, length=nInt)

idx <- findCols(intervals)
CO2data$cexPoints <- cex.key[idx]
```

The graphic will display information on two variables (`GNI.capita` and `CO2.capita` in the vertical and horizontal axes, respectively) with a conditioning variable (`Year`) and two grouping variables (`Country.Name`, and `CO2.PPP` through `cexPoints`) (Figure 5.8).

```
ggplot(data=CO2data, aes(x=CO2.capita, y=GNI.capita, colour=Country.
    Name)) +
   facet_wrap(~ Year) + geom_point(aes(size=cexPoints), pch=19) +
   xlab('Carbon␣dioxide␣emissions␣(metric␣tons␣per␣capita)') +
   ylab('GNI␣per␣capita,␣PPP␣(current␣international␣$)') +
   theme_bw()
```

The `auto.key` mechanism of the `lattice` version is not able to cope with two grouping variables. Therefore, the legend, whose main componens are the labels (`intervals`) and the point sizes (`cex.key`), should be defined manually (Figure 5.9).

```
op <- options(digits=2)
tab <- print(intervals)
options(op)

key <- list(space='right',
          title=expression(CO[2]/GNI.PPP),
          cex.title=1,
          ## Labels of the key are the intervals strings
          text=list(labels=names(tab), cex=0.85),
          ## Points sizes are defined with cex.key
          points=list(col='black', pch=19,
            cex=cex.key, alpha=0.7))
```

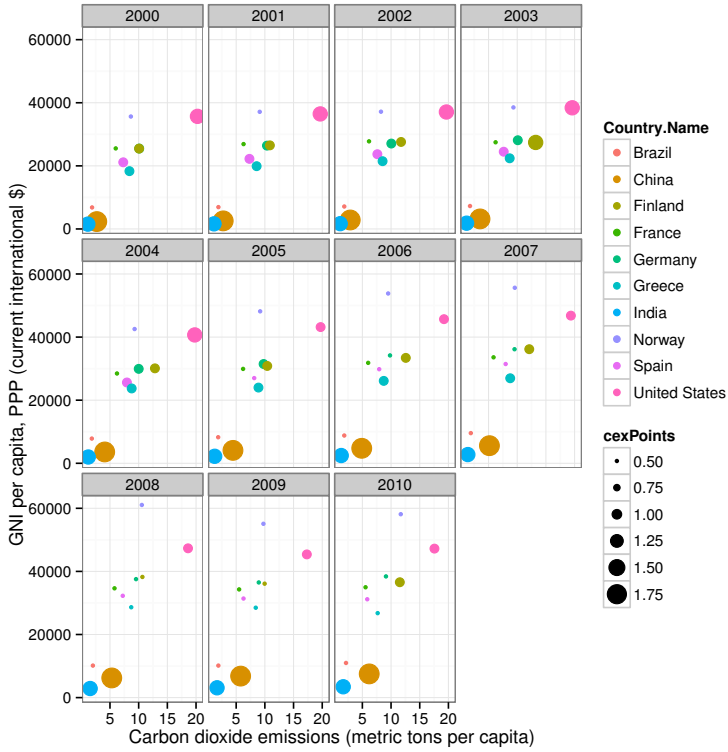

FIGURE 5.8: CO_2 emissions versus GNI per capita for different intervals of the ratio of CO_2 emissions to the GDP PPP estimations.

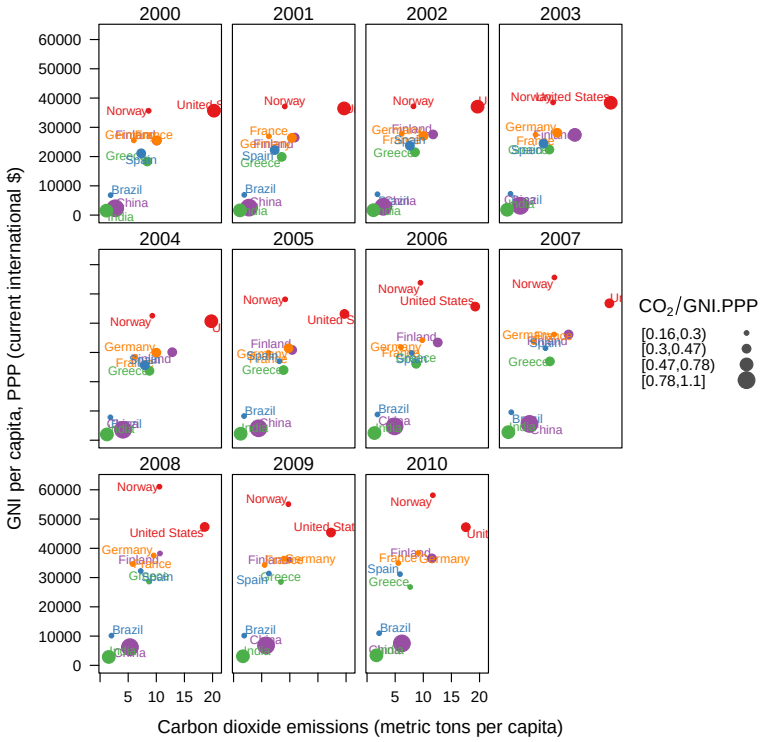

FIGURE 5.9: CO_2 emissions versus GNI per capita for different intervals of the ratio of CO_2 emissions to the GDP PPP estimations.

```
xyplot(GNI.capita ~ CO2.capita|factor(Year), data=CO2data,
      xlab="Carbon␣dioxide␣emissions␣(metric␣tons␣per␣capita)",
      ylab="GNI␣per␣capita,␣PPP␣(current␣international␣$)",
      groups=Country.Name, key=key, alpha=0.7,
      col=palOrdered, cex=CO2data$cexPoints) +
   glayer(panel.pointLabel(x, y, labels=group.value,
                           col=palOrdered[group.number], cex=0.7))
```

5.6 Traveling Bubbles

The final solution to display this multivariate time series is with animation via the function `grid.animate` of the `gridSVG` package. We will mimic the Trendalyzer/Motion Chart solution, using traveling bubbles of different colors and with radius proportional to `CO2.PPP`.

The first step is to draw the initial state of the bubbles. Their colors are again defined by the `palOrdered` palette, although the `adjustcolor` function is used for a ligther `fill` color. Because there will not be a legend, there is no need to define class intervals, and thus the radius is directly proportional to the value of `CO2data$CO2.PPP`.

```
library(gridSVG)

xyplot(GNI.capita ~ CO2.capita, data=CO2data,
      xlab="Carbon␣dioxide␣emissions␣(metric␣tons␣per␣capita)",
      ylab="GNI␣per␣capita,␣PPP␣(current␣international␣$)",
      subset=Year==2000, groups=Country.Name,
      ## The limits of the graphic are defined
      ## with the entire dataset
      xlim=extendrange(CO2data$CO2.capita),
      ylim=extendrange(CO2data$GNI.capita),
      panel=function(x, y, ..., subscripts, groups) {
        color <- palOrdered[groups[subscripts]]
        radius <- CO2data$CO2.PPP[subscripts]
        ## Size of labels
        cex <- 1.1*sqrt(radius)
        ## Bubbles
        grid.circle(x, y, default.units="native",
                    r=radius*unit(.25, "inch"),
                    name=trellis.grobname("points", type="panel"),
                    gp=gpar(col=color,
                      ## Fill color ligther than border
                      fill=adjustcolor(color, alpha=.5),
                      lwd=2))
        ## Country labels
        grid.text(label=groups[subscripts],
                  x=unit(x, 'native'),
                  ## Labels above each bubble
                  y=unit(y, 'native') + 1.5 * radius *unit(.25, 'inch')
                      ,
                  name=trellis.grobname('labels', type='panel'),
                  gp=gpar(col=color, cex=cex))
      })
```

From this initial state, `grid.animate` creates a collection of animated graphical objects with the result of `animUnit`. This function produces a set of values that will be interpreted by `grid.animate` as intermediate states of a feature of the graphical object. Thus, the bubbles will travel across the values defined by `x_points` and `y_points`, while their labels will use `x_points` and `x_labels`.

The use of `rep=TRUE` ensures that the animation will be repeated indefinitely.

```
## Duration in seconds of the animation
duration <- 20

nCountries <- nlevels(CO2data$Country.Name)
years <- unique(CO2data$Year)
nYears <- length(years)

## Intermediate positions of the bubbles
x_points <- animUnit(unit(CO2data$CO2.capita, 'native'),
                id=rep(seq_len(nCountries), each=nYears))
y_points <- animUnit(unit(CO2data$GNI.capita, 'native'),
                id=rep(seq_len(nCountries), each=nYears))
## Intermediate positions of the labels
y_labels <- animUnit(unit(CO2data$GNI.capita, 'native') +
                1.5 * CO2data$CO2.PPP * unit(.25, 'inch'),
                id=rep(seq_len(nCountries), each=nYears))
## Intermediate sizes of the bubbles
size <- animUnit(CO2data$CO2.PPP * unit(.25, 'inch'),
                id=rep(seq_len(nCountries), each=nYears))

grid.animate(trellis.grobname("points", type="panel", row=1, col=1),
          duration=duration,
          x=x_points,
          y=y_points,
          r=size,
          rep=TRUE)

grid.animate(trellis.grobname("labels", type="panel", row=1, col=1),
          duration=duration,
          x=x_points,
          y=y_labels,
          rep=TRUE)
```

A bit of interactivity can be added with the `grid.hyperlink` function. For example, the following code adds the corresponding Wikipedia link to a mouse click on each bubble.

```
countries <- unique(CO2data$Country.Name)
URL <- paste('http://en.wikipedia.org/wiki/', countries, sep='')
grid.hyperlink(trellis.grobname('points', type='panel', row=1, col
    =1),
            URL, group=FALSE)
```

Finally, the time information: The year is printed in the lower right corner, using the `visibility` attribute of an animated `textGrob` object to show and hide the values.

```
visibility <- matrix("hidden", nrow=nYears, ncol=nYears)
diag(visibility) <- "visible"
yearText <- animateGrob(garnishGrob(textGrob(years, .9, .15,
                                    name="year",
                                    gp=gpar(cex=2, col="grey")),
                            visibility="hidden"),
                    duration=20,
                    visibility=visibility,
                    rep=TRUE)
grid.draw(yearText)
```

The SVG file produced with `grid.export` is available at the website of the book (Figure 5.10). Because this animation does not trace the paths, Figure 5.5 provides this information as a static complement.

```
grid.export("figs/bubbles.svg")
```

Now, sit down in your favorite easy chair and watch the magistral video "200 Countries, 200 Years, 4 Minutes"[4]. After that, you are ready to open the SVG file of traveling bubbles: It is easier, a short time period with less than twenty countries.

[4] http://www.gapminder.org/videos/200-years-that-changed-the-world-bbc/

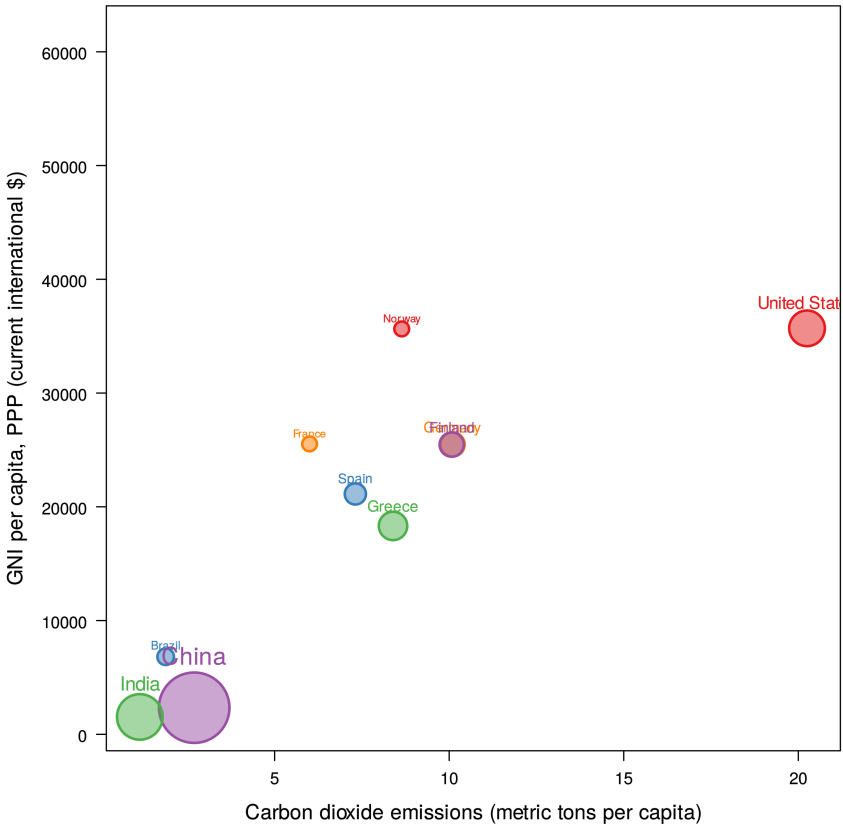

FIGURE 5.10: Animated bubbles produced with `gridSVG`.

Chapter 6

About the Data

Part II

Spatial Data

Chapter 7

Displaying Spatial Data: Introduction

Spatial data (also known as geospatial data) are directly or indirectly referenced to a location on the surface of the Earth. Their spatial reference is composed of coordinate values and a system of reference for these coordinates. Spatial data are often accessed, manipulated, or analyzed through Geographic Information Systems (GIS).

Real objects represented by GIS data can be divided into two abstractions: discrete objects (e.g., a road or a river) represented with vector data (points, lines, and polygons), and continuous fields (such as elevation or solar radiation) represented with raster data. The `sp` package is the preferred option to use vector data in `R`, and the `raster` package is the choice for raster data [1].

This part exposes several examples where vector and raster data are displayed to show geographic location of features and physical landscape features of a place (reference and physical maps, Chapter 9) or a specific variable in the context of a geographic reference (thematic maps, Chapter 8). These examples make use of several datasets (available at the book website) described in Chapter 10.

[1] Although `sp` and `raster` are the most important packages, there are an increasing number of packages designed to work with spatial data. They are summarized in the corresponding CRAN Task View. Read Section 7.2 for details.

7.1 Packages

The CRAN Tasks View "Analysis of Spatial Data" [2] summarizes the packages for reading, vizualizing, and analyzing spatial data. This section provides a brief introduction to `sp`, `raster`, `rasterVis`, `maptools`, `rgdal`, `gstat`, and `maps`. Most of the information has been extracted from their vignettes, webpages, and help pages. You should read them for detailed information.

7.1.1 sp

The `sp` package (E. J. Pebesma and R. S. Bivand 2005) provides classes and methods for dealing with spatial data in `R`. The spatial data classes implemented are points (`SpatialPoints`), grids (`SpatialPixels` and `SpatialGrid`), lines (`Line`, `Lines` and `SpatialLines`), rings, and polygons (`Polygon`, `Polygons`, and `SpatialPolygons`), each of them without data or with data (for example, `SpatialPointsDataFrame` or `SpatialLinesDataFrame`).

Selecting, retrieving, or replacing certain attributes in spatial objects with data is done using standard methods:

- `[` selects rows (items) and columns in the `data.frame`.
- `[[` selects a column from the `data.frame`
- `[[<-` assigns or replaces values to a column in the `data.frame`.

A number of spatial methods are available for the classes in `sp`:

- `coordinates(object) <- value` sets spatial coordinates to create spatial data. It promotes a `data.frame` into a `SpatialPointsDataFrame`. *value* may be specified by a formula, a character vector, or a numeric matrix or `data.frame` with the actual coordinates.
- `coordinates(object, ...)` returns a matrix with the spatial coordinates. If used with `SpatialPolygons` it returns a matrix with the centroids of the polygons.
- `bbox` returns a matrix with the coordinates bounding box.
- `proj4string(object)` and `proj4string(object) <- value` retrieve or set projection attributes on spatial classes.

[2] http://CRAN.R-project.org/view=Spatial

- `spTransform` transforms from one coordinate reference system (geographic projection) to another (requires package `rgdal`).
- `spplot` plots attributes combined with spatial data: Points, lines, grids, polygons.

7.1.2 raster

The `raster` package (R. J. Hijmans 2013) has functions for creating, reading, manipulating, and writing raster data. The package provides general raster data manipulation functions. The package also implements raster algebra and most functions for raster data manipulation that are common in Geographic Information Systems (GIS).

The raster package can work with raster datasets stored on disk if they are too large to be loaded into memory. The package can work with large files because the objects it creates from these files only contain information about the structure of the data, such as the number of rows and columns, the spatial extent, and the filename, but it does not attempt to read all the cell values in memory. In computations with these objects, the data are processed in chunks.

The package defines a number of `S4` classes. `RasterLayer`, `RasterBrick`, and `RasterStack` are the most important:

- A `RasterLayer` object represents single-layer (variable) raster data. It can be created with the function `raster`. This function is able to create a `RasterLayer` from another object, including another `Raster*` object, or from a `SpatialPixels*` and `SpatialGrid*` object, or even a matrix. In addition, it can create a `RasterLayer` reading data from a file. The `raster` package can use raster files in several formats, some of them via the `rgdal` package. Supported formats for reading include GeoTIFF, ESRI, ENVI, and ERDAS.
- `RasterBrick` and `RasterStack` are classes for multilayer data. A `RasterStack` is a list of `RasterLayer` objects with the same spatial extent and resolution. It can be formed with a collection of files in different locations or even mixed with `RasterLayer` objects that only exist in memory. A `RasterBrick` is truly a multilayered object, and processing it can be more efficient than processing a `RasterStack` representing the same data.

The raster package defines a number of methods for raster algebra with Raster* objects: arithmetic operators, logical operators, and functions such as abs, round, ceiling, floor, trunc, sqrt, log, log10, exp, cos, sin, max, min, range, prod, sum, any, and all. In these functions, Raster* objects can be mixed with numbers.

There are several functions to modify the content or the spatial extent of Raster* objects, or to combine Raster* objects:

- The crop function takes a geographic subset of a larger Raster* object. trim crops a RasterLayer by removing the outer rows and columns that only contain NA values. extend adds new rows and/or columns with NA values.
- The merge function merges two or more Raster* objects into a single new object.
- projectRaster transforms values of a Raster* object to a new object with a different coordinate reference system.
- With overlay, multiple Raster* objects can be combined (for example, multiply them).
- mask removes all values from one layer that are NA in another layer, and cover combines two layers by taking the values of the first layer except where these are NA.
- calc computes a function for a Raster* object. With RasterLayer objects, another RasterLayer is returned. With multilayer objects the result depends on the function: With a summary function (sum, max, etc.), calc returns a RasterLayer object, and a RasterBrick object otherwise.
- stackApply computes summary layers for subsets of a RasterStack or RasterBrick.
- cut and reclassify replace ranges of values with single values.
- zonal computes zonal statistics, that is, summarizes a Raster* object using zones (areas with the same integer number) defined by another RasterLayer.

7.1.3 rasterVis

The `rasterVis` package (Perpiñán and R. Hijmans 2013) complements the `raster` package, providing a set of methods for enhanced visualization and interaction. This package defines visualization methods (`levelplot`) for quantitative data and categorical data, both for univariate and multivariate rasters.

It also includes several methods in the frame of the Exploratory Data Analysis approach: scatterplots with `xyplot`, histograms and density plots with `histogram` and `densityplot`, violin and boxplots with `bwplot`, and a matrix of scatterplots with `splom`.

On the other hand, this package is able to display vector fields using arrows, `vectorplot`, or with streamlines (Wegenkittl and Gröller 1997), `streamplot`. In this last method, for each point, *droplet*, of a jittered regular grid, a short streamline portion, *streamlet*, is calculated by integrating the underlying vector field at that point. The main color of each streamlet indicates local vector magnitude (slope). Streamlets are composed of points whose sizes, positions, and color degradation encode the local vector direction (aspect).

7.1.4 maptools

The `maptools` package (R. Bivand and Lewin-Koh 2013) provides a set of tools for manipulating and reading geographic data, in particular ESRI (Environmental Systems Research Institute) shapefiles. The package also provides interface wrappers for exchanging spatial objects with packages such as PBSmapping, spatstat, maps, RArcInfo, Stata tmap, WinBUGS, Mondrian, and others. The main functions in the context of this book are

- `readShapePoints` reads data from a points shapefile into a `SpatialPointsDataFrame` object.
- `writePointsShape` writes data from a `SpatialPointsDataFrame` object to a shapefile.
- `readShapeLines` reads data from a line shapefile into a `SpatialLinesDataFrame` object.
- `writeLinesShape` writes data from a `SpatialLinesDataFrame` object to a shapefile.
- `readShapePoly` reads data from a polygon shapefile into a `SpatialPolygonsDataFrame` object.

- `writePolyShape` writes data from a `SpatialPolygonsDataFrame` object to a shapefile.
- `map2SpatialPolygons` and `map2SpatialLines` may be used to convert map objects returned by the `map` function in the `maps` package to the classes defined in the `sp` package.
- `spCbind` provides cbind-like methods for `Spatial*DataFrame` and `data.frame` objects.

The topology operations on geometries performed by this package (for example, `unionSpatialPolygons`) use the package `rgeos`, an interface to the Geometry Engine Open Source (GEOS) [3].

7.1.5 rgdal

The `rgdal` package (R. Bivand, Keitt, and Rowlingson 2013) provides bindings to the Geospatial Data Abstraction Library (GDAL) [4]. With `readOGR` and `readGDAL`, both GDAL raster and OGR vector map data can be imported into `R`, and GDAL raster data and OGR vector data can be exported with `writeGDAL` and `writeOGR`.

In addition, this package provides access to projection and transformation operations from the PROJ.4 library [5]. This package implements several `spTransform` methods providing transformation between datums and conversion between projections using PROJ.4 projection arguments.

7.1.6 gstat

The `gstat` package (E. J. Pebesma 2004) provides functions for geostatistical modeling, prediction, and simulation, including variogram modeling and simple, ordinary, universal, and external drift kriging.

Most of the functionality of this package is beyond the scope of this book. However, some functions must be mentioned:

- `variogram` calculates the sample variogram from data, or for the residuals if a linear model is given. `vgm` generates a variogram and `fit.variogram` fit ranges and/or sills from a variogram model to a sample variogram.

[3] http://trac.osgeo.org/geos/
[4] http://www.gdal.org/
[5] https://trac.osgeo.org/proj/

- `krige` is the function for simple, ordinary or universal kriging. `gstat` is the function for univariate or multivariate geostatistical prediction.

7.1.7 maps

The `maps` (Becker, Wilks, Brownrigg, and Minka 2013), `mapdata` (Becker, Wilks, and Brownrigg 2013), and `mapproj` (McIlroy et al. 2013) packages are useful to draw or create geographical maps. `mapdata` contains higher resolution databases, and `mapproj` converts latitude/longitude coordinates into projected coordinates.

7.2 Further Reading

- (Slocum 2005) and (Dent, Torguson, and Hodler 2008) are comprehensive books on thematic cartography and geovisualization. They include chapters devoted to data classification, scales, map projections, color theory, typography, and proportional symbol, choropleth, dasymetric, isarithmic, and multivariate mapping. Several resources are available at their accompanying websites [6].

- (R. S. Bivand, E. J. Pebesma, and Gomez-Rubio 2008) is the essential reference to work with spatial data in R. R. Bivand and E. Pebesma are the authors of the fundamental `sp` package, and they are the authors or maintainers of several important packages such as `gstat`, for geostatistical modeling, prediction, and simulation, `rgdal`, `rgeos` and `maptools`. Chapter 3 is devoted to the visualization of spatial data. Code, figures, and data of the book are available at the accompanying website [7].

- (Hengl 2009) is an open-access book with seven spatial data analysis exercises. The author is the creator and maintainer of the Spatial-Analyst webpage [8].

- The CRAN Tasks View "Analysis of Spatial Data" [9] summarizes the packages for reading, vizualizing, and analyzing spatial data. The

[6] http://www.pearsonhighered.com/slocum3e/ and http://highered.mcgraw-hill.com/sites/0072943823/

[7] http://www.asdar-book.org/

[8] http://spatial-analyst.net

[9] http://CRAN.R-project.org/view=Spatial

packages in development published at R-Forge are listed in the "Spatial Data & Statistics" topic view [10]. The R-SIG-Geo mailing list [11] is a powerful resource for obtaining help.

- The "Spatial Analysis" [12] and "Kartograph" [13] webpages publish a variety of beautiful visualization examples.

[10] http://r-forge.r-project.org/softwaremap/trove_list.php?form_cat=353
[11] https://stat.ethz.ch/mailman/listinfo/R-SIG-Geo/
[12] http://spatialanalysis.co.uk/map-gallery/
[13] http://kartograph.org/

Chapter 8

Thematic Maps

A thematic map focuses on a specific theme or variable, commonly using geographic data such as coastlines, boundaries, and places as points of reference for the variable being mapped. These maps provide specific information about particular locations or areas (proportional symbol mapping and choropleth maps) and information about spatial patterns (isarithmic and raster maps). The following sections illustrate the code you need to produce these maps, with a final section devoted to the visualization of vector fields.

8.1 Proportional Symbol Mapping

8.1.1 Introduction

The proportional symbol technique uses symbols of different sizes to represent data associated with areas or point locations, with circles being the most frequently used geometric symbol. The data and the size of symbols can be related through different types of scaling: mathematical scaling sizes areas of point symbols in direct proportion to the data; perceptual scaling corrects the mathematical scaling to account for visual understimation of larger symbols; and range grading, where data are grouped, and each class is represented with a single symbol size.

In this chapter we display data from a grid of sensors belonging to the Integrated Air Quality system of the Madrid City Council (Section 10.1)

with circles as the proportional symbol, and range grading as the scaling method. The objective when using range grading is to discriminate between classes instead of estimating an exact value from a perceived symbol size. However, because human perception of symbol size is limited, it is always recommended to add a second perception channel to improve the discrimination task. Colors from a sequential palette will complement symbol size to encode the groups.

8.1.2 Proportional Symbol with `spplot`

The `NO2sp` `SpatialPointsDataFrame` can be easily displayed with the `spplot` method provided by the `sp` package, based on `xyplot` from the `lattice` package. Both color and size can be combined in a unique graphical output because `spplot` accepts both of them (Figure 8.1). I define a sequential palette whose colors denote the value of the variable (green for lower values of the contaminant, brown for intermediate values, and black for highest values).

```
library(sp)

load('data/NO2sp.RData')
```

```
airPal <- colorRampPalette(c('springgreen1', 'sienna3', 'gray5'))(5)

spplot(NO2sp["mean"], col.regions=airPal, cex=sqrt(1:5),
     edge.col='black', scales=list(draw=TRUE),
     key.space='right')
```

The `ggplot2` version of this code needs to transform the `SpatialPointsDataFrame` to a conventional `data.frame` (which will contain two columns with latitude and longitude values).

```
NO2df <- data.frame(NO2sp)
NO2df$Mean <- cut(NO2sp$mean, 5)

ggplot(data=NO2df, aes(long, lat, size=Mean, fill=Mean)) +
   geom_point(pch=21, col='black') + theme_bw() +
   scale_fill_manual(values=airPal)
```

8.1.3 Optimal Classification and Sizes to Improve Discrimination

Two main improvements can be added to Figure 8.1:

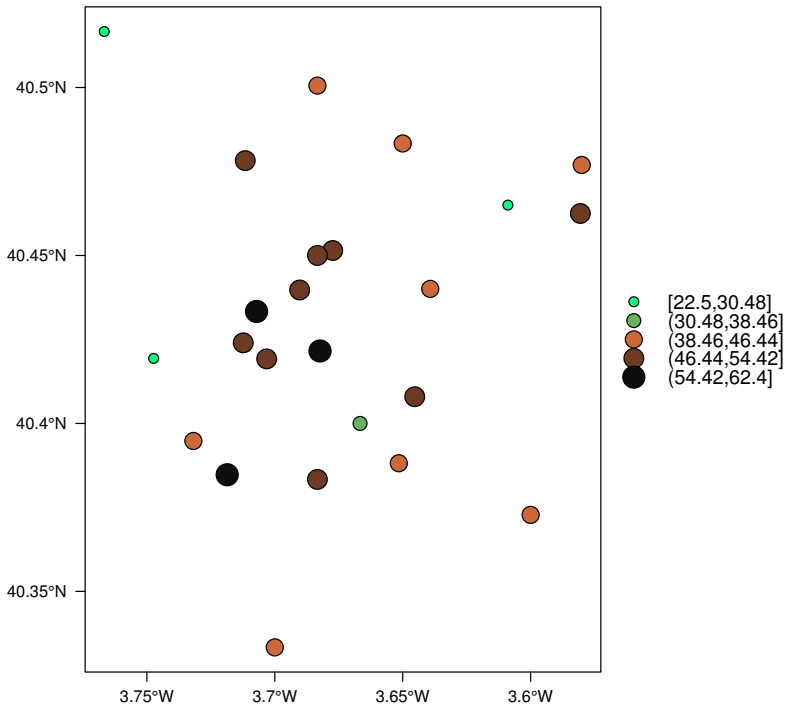

FIGURE 8.1: Annual average of NO_2 measurements in Madrid. Values are shown with different symbol sizes and colors for each class with the `spplot` function.

- Define classes dependent on the data structure (instead of the uniform distribution assumed with `cut`). A suitable approach is the `classInterval` function of the `classInt` package, which implements the Fisher-Jenks optimal classification algorithm.

```
library(classInt)
## The number of classes is chosen between the Sturges and the
## Scott rules.
nClasses <- 5
intervals <- classIntervals(NO2sp$mean, n=nClasses, style='fisher')
## Number of classes is not always the same as the proposed number
nClasses <- length(intervals$brks) - 1
```

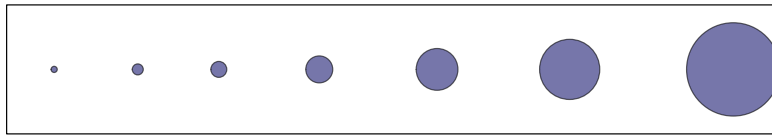

FIGURE 8.2: Symbol sizes proposed by Borden Dent.

```
op <- options(digits=4)
tab <- print(intervals)
options(op)
```

- Encode each group with a symbol size (circle area) such that visual discrimination among classes is enhanced. The next code uses the set of radii proposed in (Dent, Torguson, and Hodler 2008) (Figure 8.2). This set of circle sizes is derived from studies by Meihoefer (Meihoefer 1969). He derived a set of ten circle sizes that were easily and consistently discriminated by his subjects. The alternative proposed by Dent et al. improves the discrimination between some of the circles.

```
## Complete Dent set of circle radii (mm)
dent <- c(0.64, 1.14, 1.65, 2.79, 4.32, 6.22, 9.65, 12.95, 15.11)
## Subset for our dataset
dentAQ <- dent[seq_len(nClasses)]
## Link Size and Class: findCols returns the class number of each
## point; cex is the vector of sizes for each data point
idx <- findCols(intervals)
cexNO2 <- dentAQ[idx]
```

These two enhancements are included in Figure 8.3, which displays the categorical variable `classNO2` (instead of `mean`) whose levels are the intervals previously computed with `classIntervals`. In addition, this figure includes an improved legend.

```
NO2sp$classNO2 <- factor(names(tab)[idx])
```

```
## ggplot2 version
NO2df <- data.frame(NO2sp)

ggplot(data=NO2df, aes(long, lat, size=classNO2, fill=classNO2)) +
```

```
   geom_point(pch=21, col='black') + theme_bw() +
   scale_fill_manual(values=airPal) +
   scale_size_manual(values=dentAQ*2)
```

```
## spplot version

## Definition of an improved key with title and background
NO2key <- list(x=0.98, y=0.02, corner=c(1, 0),
           title=expression(NO[2]~~(paste(mu, plain(g))/m^3)),
           cex.title=.75, cex=0.7,
           background='gray92')

pNO2 <- spplot(NO2sp["classNO2"],
            col.regions=airPal, cex=dentAQ,
            edge.col='black',
            scales=list(draw=TRUE),
            key.space=NO2key)
pNO2
```

8.1.4 Spatial Context with Underlying Layers and Labels

The spatial distribution of the stations is better understood if we add underlying layers with information about the spatial context.

8.1.4.1 Static Image

A suitable method is to download data from a provider such as Google Maps™ or OpenStreetMap and transform it adequately. There are several packages that provide an interface to query several map servers. On one hand, `RGoogleMaps`, `OpenStreetMaps`, and `ggmap` provide raster images from static maps obtained from Google Maps, Stamen, OpenStreetMap, etc.; on the other hand, `osmar` is able to access OpenStreetMap data and convert it into classes provided by existing R packages (mainly `sp` and `igraph0` objects).

Among these options, I have chosen the Stamen watercolor maps available through the `ggmap` (Kahle and Wickham 2013) and `OpenStreetMaps` packages (Fellows and Stotz 2013). It is worth noting that these map tiles are published by Stamen Design under a Creative Commons licence CC BY-3.0 (Attribution). They produce these maps with data by OpenStreetMap also published under a Creative Commons licence BY-SA (Attribution - ShareAlike).

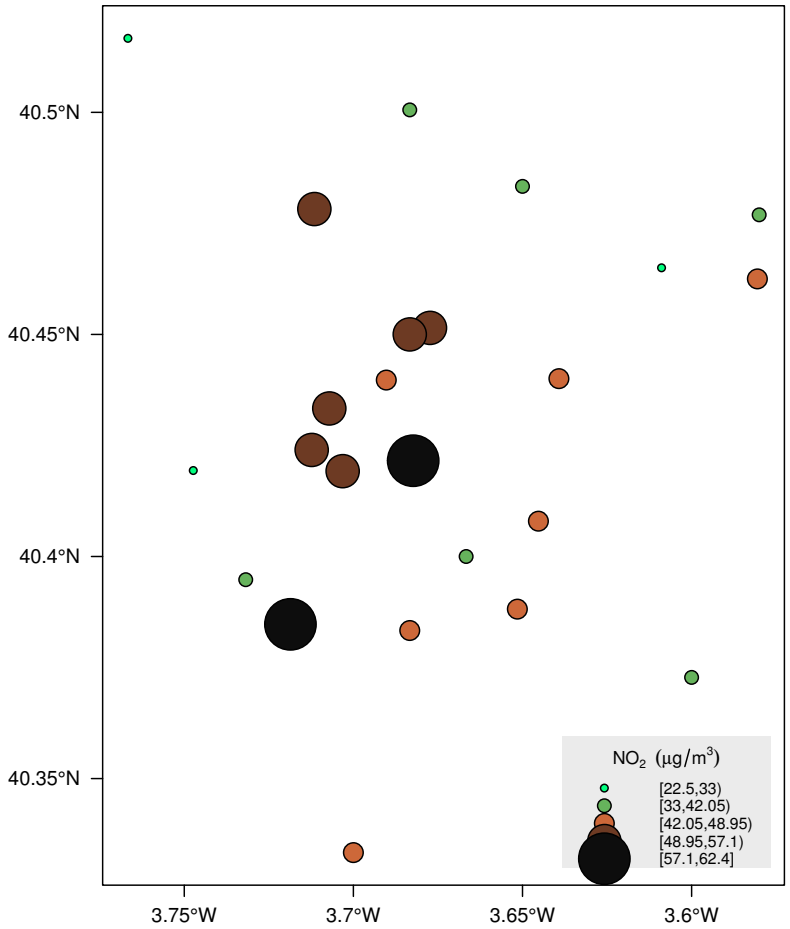

FIGURE 8.3: Annual average of NO_2 measurements in Madrid.

```
madridBox <- bbox(NO2sp)

## ggmap solution
library(ggmap)
madridGG <- get_map(c(madridBox), maptype='watercolor', source='
    stamen')
```

```
## OpenStreetMap solution
library(OpenStreetMap)
ul <- madridBox[c(4, 1)]
lr <- madridBox[c(2, 3)]
madridOM <- openmap(ul, lr, type='stamen-watercolor')
madridOM <- openproj(madridOM)
```

```
NO2df <- data.frame(NO2sp)

## ggmap
ggmap(madridGG) +
   geom_point(data=NO2df,
            aes(long, lat, size=classNO2, fill=classNO2),
            pch=21, col='black') +
     scale_fill_manual(values=airPal) +
     scale_size_manual(values=dentAQ*2)

##OpenStreetMap
autoplot(madridOM) +
   geom_point(data=NO2df,
            aes(long, lat, size=classNO2, fill=classNO2),
            pch=21, col='black') +
   scale_fill_manual(values=airPal) +
   scale_size_manual(values=dentAQ*2)
```

Although ggmap is designed to work with the ggplot2 package, the result of get_map is only a raster object with attributes. Therefore, it can be easily displayed with grid.raster as an underlying layer of the previous spplot result (Figure 8.4).

```
## the 'bb' attribute stores the bounding box of the get_map result
bbMap <- attr(madridGG, 'bb')
## This information is needed to resize the image with grid.raster
height <- with(bbMap, ur.lat - ll.lat)
width <- with(bbMap, ur.lon - ll.lon)

pNO2 + layer(grid.raster(madridGG,
```

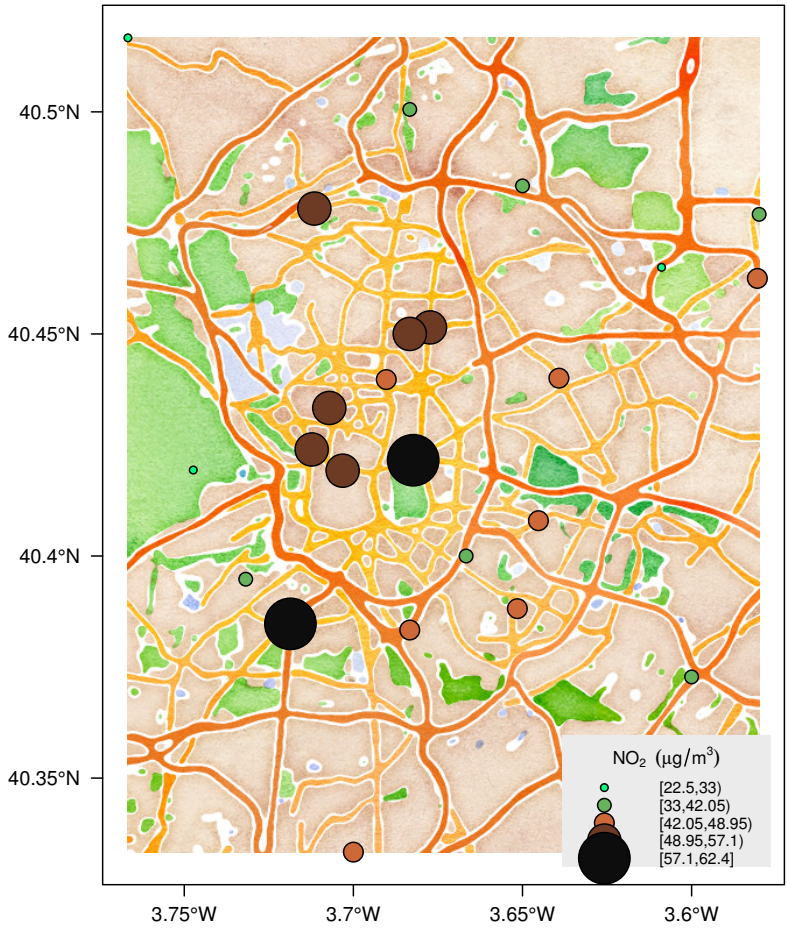

FIGURE 8.4: Annual average of NO_2 measurements in Madrid.

```
                    width=width, height=height,
                    default.units='native'),
          under=TRUE)
```

The result of `openmap` is more sophisticated but can also be converted and displayed with `grid.raster`.

```
tile <- madridOM$tile[[1]]
```

```
height <- with(tile$bbox, p1[2] - p2[2])
width <- with(tile$bbox, p2[1] - p1[1])

colors <- as.raster(matrix(tile$colorData,
                      ncol=tile$yres,
                      nrow=tile$xres,
                      byrow=TRUE))

pNO2 + layer(grid.raster(colors,
                    width=width,
                    height=height,
                    default.units='native'),
          under=TRUE)
```

8.1.4.2 Vector Data

A major problem with the previous solution is that the user can neither modify the image nor use its content to produce additional information. A different approach is to use digital vector data (points, lines, and polygons). A popular format for vectorial data is the shapefile, commonly used by public and private providers to distribute information. A shapefile can be read with `readShapePoly` and `readShapeLines` from the `rgdal` package. These functions produce a `SpatialPolygonsDataFrame` and a `SpatialLinesDataFrame` objects, respectively. These objects can be displayed with the `sp.polygons` and `sp.lines` functions provided by the `sp` package.

For our example, the Madrid district and streets are available as shapefiles from the nomecalles web service[1].

```
library(maptools)
library(rgdal)

## nomecalles http://www.madrid.org/nomecalles/Callejero_madrid.icm
## Form at http://www.madrid.org/nomecalles/DescargaBDTCorte.icm

## Madrid districts
unzip('Distritos␣de␣Madrid.zip')
distritosMadrid <- readShapePoly('Distritos␣de␣Madrid/200001331')
proj4string(distritosMadrid) <- CRS("+proj=utm␣+zone=30")
```

[1] http://www.madrid.org/nomecalles/

```
distritosMadrid <- spTransform(distritosMadrid, CRS=CRS("+proj=
    longlat␣+ellps=WGS84"))

## Madrid streets
unzip('Callejero_␣Ejes␣de␣viales.zip')
streets <- readShapeLines('Callejero_␣Ejes␣de␣viales/call2011.shp')
streetsMadrid <- streets[streets$CMUN=='079',]
proj4string(streetsMadrid) <- CRS("+proj=utm␣+zone=30")
streetsMadrid <- spTransform(streetsMadrid, CRS=CRS("+proj=longlat␣+
    ellps=WGS84"))
```

These shapefiles can be included in the plot with the `sp.layout` mechanism accepted by `spplot` or with the `layer` and `+.trellis` functions from the `latticeExtra` package. The station codes are placed with this same procedure using the `sp.pointLabel` function from the `maptools` package. Figure 8.5 displays the final result.

```
spDistricts <- list('sp.polygons', distritosMadrid, fill='gray97',
    lwd=0.3)
spStreets <- list('sp.lines', streetsMadrid, lwd=0.05)
spNames <- list(sp.pointLabel, NO2sp,
            labels=substring(NO2sp$codEst, 7),
            cex=0.6, fontfamily='Palatino')

spplot(NO2sp["classNO2"], col.regions=airPal, cex=dentAQ,
     edge.col='black', alpha=0.8,
     sp.layout=list(spDistricts, spStreets, spNames),
     scales=list(draw=TRUE),
     key.space=NO2key)
```

```
pNO2 +
   layer(sp.pointLabel(NO2sp,
                  labels=substring(NO2sp$codEst, 7),
                  cex=0.8, fontfamily='Palatino')
       ) +
   layer_({
      sp.polygons(distritosMadrid, fill='gray97', lwd=0.3)
      sp.lines(streetsMadrid, lwd=0.05)
   })
```

The `ggplot2` package is not able to work directly with `SpatialLines*` or `SpatialPolygon*` objects. Instead, it includes several `fortify` methods to convert objects from these classes into a conventional `data.frame`. You should beware that the `fortify` process for large objects (such as the

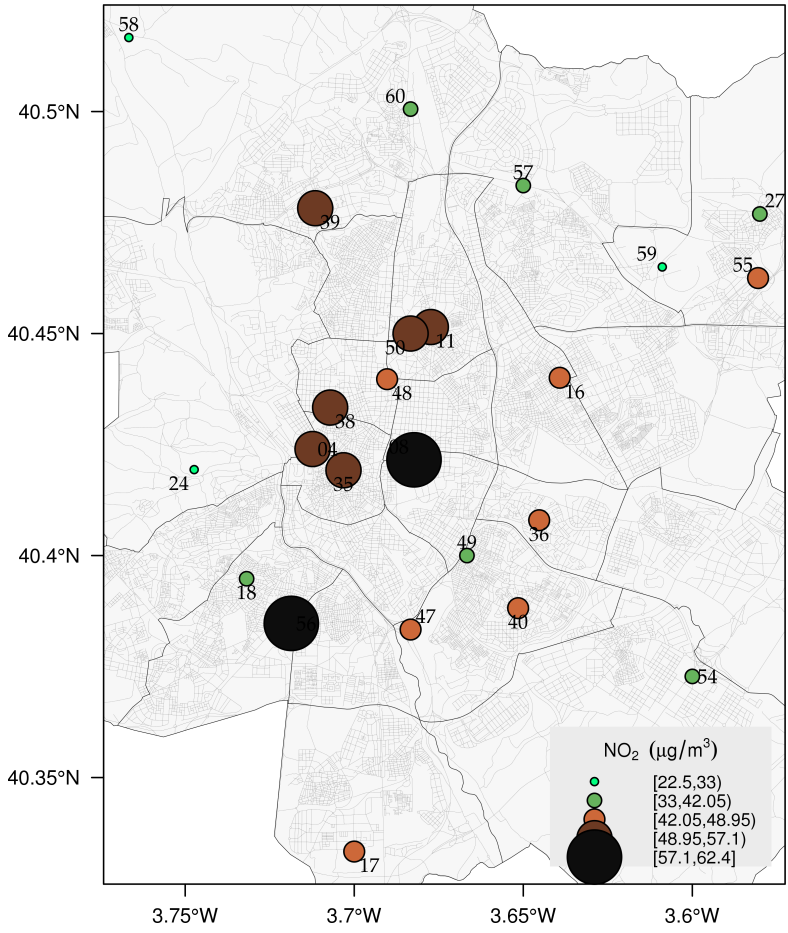

FIGURE 8.5: Annual average of NO_2 measurements in Madrid using shapefiles (lines and polygons) and text as geographical context.

SpatialLinesDataFrame in our example) requires too much time to be completed.

8.1.5 Spatial Interpolation

The measurements at discrete points give limited information about the underlying process. It is quite common to approximate the spatial distribution of the measured variable with the interpolation between measurement locations. Selection of the optimal interpolation method is outside the scope of this book. The following code illustrates an easy solution using inverse distance weighted (IDW) interpolation with the gstat package (E. J. Pebesma 2004) *only* for illustration purposes.

```
library(gstat)

airGrid <- spsample(NO2sp, type='regular', n=1e5)
gridded(airGrid) <- TRUE
airKrige <- krige(mean ~ 1, NO2sp, airGrid)
```

The result is a SpatialPixelsDataFrame that can be displayed with spplot and combined with the previous layers and the measurement station points (Figure 8.6).

```
spplot(airKrige["var1.pred"],
     col.regions=colorRampPalette(airPal)) +
 layer({
   sp.polygons(distritosMadrid, fill='transparent', lwd=0.3)
   sp.lines(streetsMadrid, lwd=0.07)
   sp.points(NO2sp, pch=21, alpha=0.8, fill='gray50', col='black')
   })
```

8.1.6 Export to Other Formats

A different approach is to use an external data viewer, due to its features or its large community of users. Two tools deserve to be mentioned: GeoJSON rendered within GitHub repositories, and KML files imported in Google Earth™.

8.1.6.1 GeoJSON and OpenStreetMap

GeoJSON is an open computer file format for encoding collections of simple geographical features along with their nonspatial attributes using JavaScript Object Notation (JSON). These files can be easily rendered within

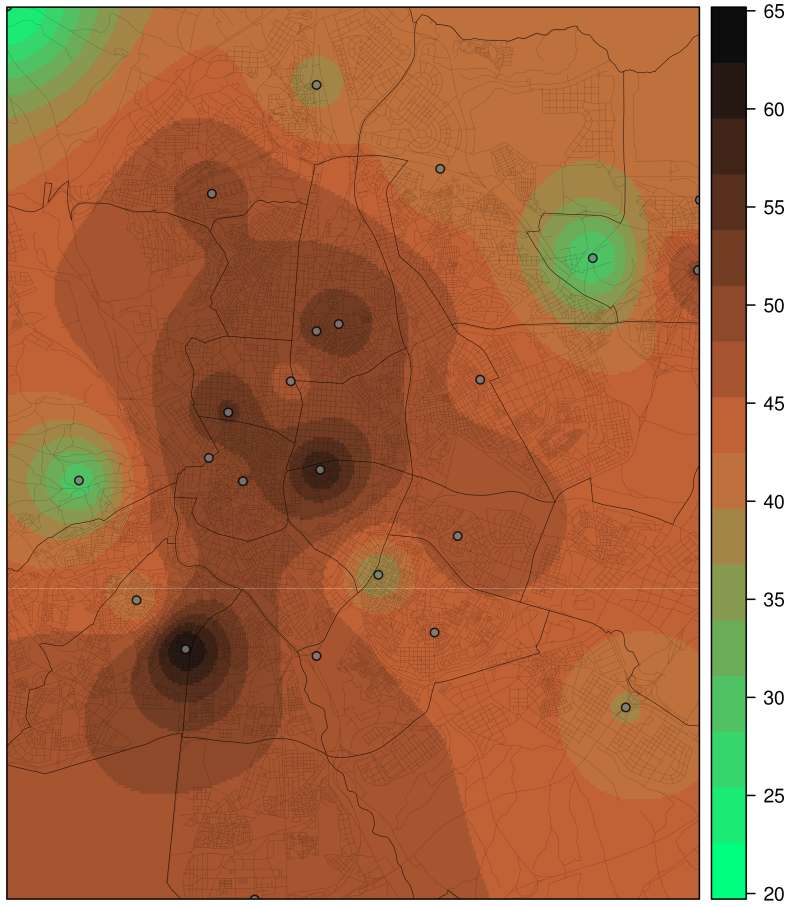

FIGURE 8.6: Kriging annual average of NO_2 measurements in Madrid.

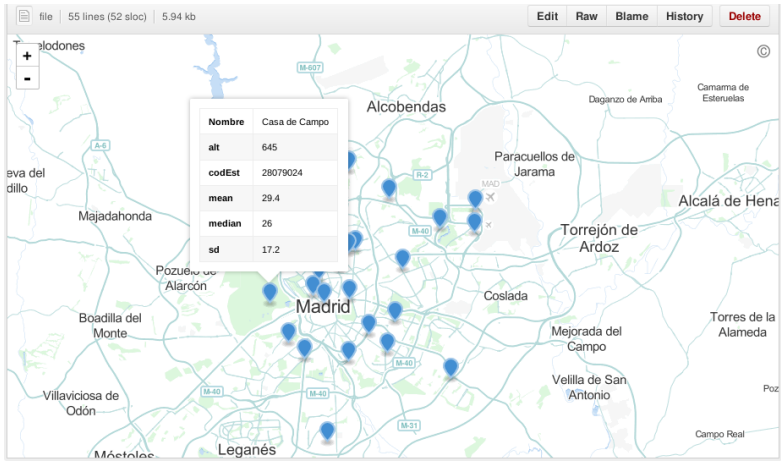

FIGURE 8.7: NO_2 data in a GeoJSON file rendered within the GitHub repository.

GitHub repositories. GitHub uses Leaflet.js[2] to represent the data and MapBox[3] with OpenStreetMap[4] for the underlying map data.

Our `SpatialPointsDataFrame` can be converted to a GeoJSON file with `writeOGR` from the `rgdal` package.

```
library(rgdal)
writeOGR(NO2sp, 'data/NO2.geojson', 'NO2sp', driver='GeoJSON')
```

Figure 8.7 shows a snapshot of the rendering of this GeoJSON file, available from the GitHub repository. There you can zoom on the map and click on the stations to display the data.

8.1.6.2 Keyhole Markup Language

Keyhole Markup Language (KML) is a file format to display geographic data within Internet-based, two-dimensional maps and three-dimensional Earth browsers. KML uses a tag-based structure with nested elements and attributes, and is based on the XML standard. KML became an international standard of the Open Geospatial Consortium in 2008. Google

[2] http://leafletjs.com/
[3] http://www.mapbox.com/
[4] http://www.openstreetmap.org/

Earth was the first program able to view and graphically edit KML files, although Marble, an open-source project, also offers KML support.

There are several packages able to generate KML files. For example, the `writeOGR` function from the `rgdal` package can also write KML files:

```
library(rgdal)
writeOGR(NO2sp, dsn='NO2_mean.kml', layer='mean', driver='KML')
```

However, the `plotKML` package provides a simpler interface and includes a wide set of options:

```
library(plotKML)
plotKML(NO2sp["mean"], points_names=NO2sp$codEst)
```

Both functions produce a file that can be directly opened with Google Earth or Marble.

8.1.7 Additional Information with Tooltips and Hyperlinks

Now, let's suppose you need to know the median and standard deviation of the time series of a certain station. Moreover, you would like to watch the photography of that station; or even better, you wish to visit its webpage for additional information. A frequent solution is to produce interactive graphics with tooltips and hyperlinks.

The `gridSVG` package is able to create an SVG graphic, where each component owns a `title` attribute; the content of this attribute is commonly displayed as a tooltip when the mouse hovers over the element. The content of this attribute can be modified thanks to the `grid.garnish` function. Moreover, the `grid.hyperlink` function can add hyperlinks to the correspondent graphical element.

The tooltips will display the photography of the station, the name of the station, and the statistics previously calculated with `aggregate` in the first step of this chapter. The station images are downloaded from the Munimadrid webpage. The `htmlParse` function from the `XML` package parses each station page, and the station photograph is extracted with `getNodeSet` and `xmlAttrs`.

```
library(XML)

old <- setwd('images')
for (i in 1:nrow(NO2df)){
  codEst <- NO2df[i, "codEst"]
  ## Webpage of each station
  codURL <- as.numeric(substr(codEst, 7, 8))
```

```
  rootURL <- 'http://www.mambiente.munimadrid.es'
  stationURL <- paste(rootURL,
                  '/opencms/opencms/calaire/contenidos/estaciones/
                      estacion',
                  codURL, '.html', sep='')
  content <- htmlParse(stationURL, encoding='utf8')
  ## Extracted with http://www.selectorgadget.com/
  xPath <- '//*[contains(concat(␣"␣",␣@class,␣"␣"␣),␣concat(␣"␣",␣"
      imagen_1",␣"␣"␣))]'
  imageStation <- getNodeSet(content, xPath)[[1]]
  imageURL <- xmlAttrs(imageStation)[1]
  imageURL <- paste(rootURL, imageURL, sep='')
  download.file(imageURL, destfile=paste(codEst, '.jpg', sep=''))
}
setwd(old)
```

Next, we attach the hyperlink and the SVG information to each circle.

```
print(pNO2 + layer_(sp.polygons(distritosMadrid, fill='gray97', lwd
    =0.3)))
```

```
library(gridSVG)

NO2df <- as.data.frame(NO2sp)

tooltips <- sapply(seq_len(nrow(NO2df)), function(i){
  codEst <- NO2df[i, "codEst"]
  ## Information to be attached to each line
  stats <- paste(c('Mean', 'Median', 'SD'),
              signif(NO2df[i, c('mean', 'median', 'sd')], 4),
              sep='␣=␣', collapse='<br␣/>')
  ## Station photograph
  imageURL <- paste('images/', codEst, '.jpg', sep='')
  imageInfo <- paste("<img␣src=", imageURL,
                 "␣width='100'␣height='100'␣/>", sep='')
  ## Text to be included in the tooltip
  nameStation <- paste('<b>',
                   as.character(NO2df[i, "Nombre"]),
                   '</b>', sep='')
  info <- paste(nameStation, stats, sep='<br␣/>')
  ## Tooltip includes the image and the text
  paste(imageInfo, info, sep='<br␣/>')
})
grid.garnish('points.panel', title=tooltips, grep=TRUE, group=FALSE)
```

```
## Webpage of each station
rootURL <- 'http://www.mambiente.munimadrid.es'
urlList <- sapply(seq_len(nrow(NO2df)), function(i){
  codEst <- NO2df[i, "codEst"]
  codURL <- as.numeric(substr(codEst, 7, 8))
  stationURL <- paste(rootURL,
                  '/opencms/opencms/calaire/contenidos/estaciones/
                      estacion',
                  codURL, '.html', sep='')
  })

grid.hyperlink('points.panel', urlList, grep=TRUE, group=FALSE)
```

The `title` attribute can be accessed with the JavaScript plug-ins jQuery[5] and jQuery UI[6] to display tooltips when the mouse hovers over each station. The `grid.script` function creates objects containing links to these plug-ins. And `grid.export` uses these objects to produce an SVG document with script elements.

```
## Add jQuery and jQuery UI scripts
grid.script(file='http://code.jquery.com/jquery-1.8.3.js')
grid.script(file='http://code.jquery.com/ui/1.9.2/jquery-ui.js')
## Simple JavaScript code to initialize the tooltip
grid.script(file='js/myTooltip.js')
## Produce the SVG graphic: the results of grid.garnish,
## grid.hyperlink and grid.script are converted to SVG code
grid.export('figs/airMadrid.svg')
```

These plug-ins will work only after the file `airMadrid.svg` created by `grid.export` is inserted in a HTML file with standard headers. Figure 8.8 shows a capture of the result.

```
htmlBegin <- '<!DOCTYPE␣html>
<html>
<head>
<title>Tooltips␣with␣jQuery␣and␣gridSVG</title>
<link␣rel="stylesheet"␣type="text/css"␣href="http://code.jquery.com/
    ui/1.9.2/themes/smoothness/jquery-ui.css"␣/>
<meta␣charset="utf-8">
</head>
<body>'
```

[5] http://jquery.com/
[6] http://jqueryui.com/

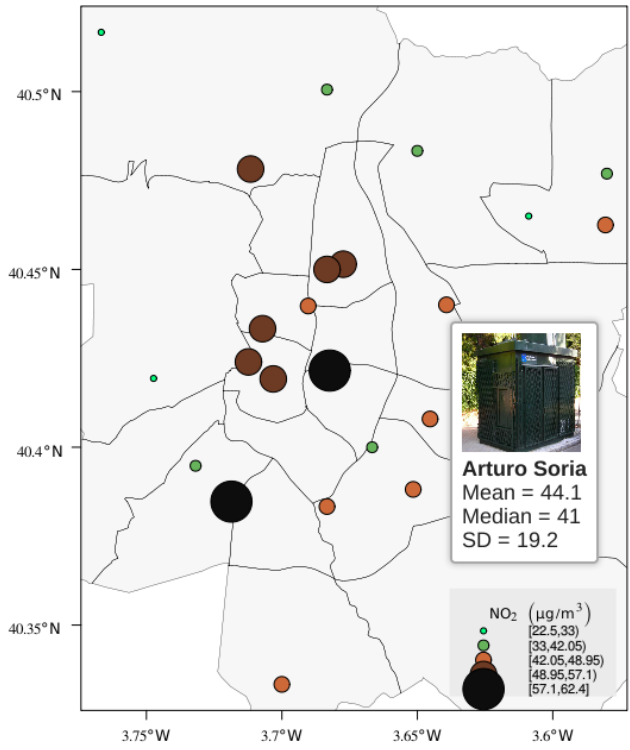

FIGURE 8.8: Tooltips generated with `gridSVG` using jQuery and jQuery UI.

```
htmlEnd <- '</body>␣</html>'

svgText <- paste(readLines('figs/airMadrid.svg'), collapse='\n')

writeLines(paste(htmlBegin, svgText, htmlEnd, sep='\n'),
         'airMadrid.html')
```

8.2 Choropleth Maps

A choropleth map shades regions according to the measurement of a variable displayed on the map. The choropleth map is an appropiate tool to visualize a variable uniformly distributed within each region, changing only at the region boundaries. This method performs correctly with homogeneous regions, both in size and shape.

This section details how to create a multivariate choropleth map to show the results of the 2011 Spanish general elections. It is inspired by the infographic from the *New York Times*[7], a multivariate choropleth map of the inmigration behavior in the United States.

```
votes2011 <- read.csv('data/votes2011.csv',
                  colClasses=c('factor', 'factor', 'numeric', '
                      numeric'))
```

The next section describes how to define a `SpatialPolygonsDataFrame` with the data from this `data.frame` and the spatial information of the administrative boundaries from a shapefile. You can skip it for later reading if you are not interested in this procedure and jump to the section 8.2.2 where the maps are produced.

8.2.1 Administrative Boundaries

The Spanish administrative boundaries are available as shapefiles at the INE (Instituto Nacional de Estadística) webpage[8]. Both the municipalities, `espMap`, and province boundaries, `provinces`, are read as `SpatialPolygonsDataFrame` with `readShapePoly`.

```
library(sp)
library(maptools)
```

```
old <- setwd(tempdir())
download.file('http://goo.gl/TIvr4', 'mapas_completo_municipal.rar')
system2('unrar', c('e', 'mapas_completo_municipal.rar'))
espMap <- readShapePoly(fn="esp_muni_0109")
Encoding(levels(espMap$NOMBRE)) <- "latin1"

provinces <- readShapePoly(fn="spain_provinces_ag_2")
setwd(old)
```

[7] http://www.nytimes.com/interactive/2009/03/10/us/20090310-immigration-explorer.html

[8] http://www.ine.es/ > Products and services > Publications > Download the PC-Axis program > Municipal maps

Some of the polygons are repeated and can be dissolved with `unionSpatialPolygons` (the `rgeos` package must be installed).

```
## dissolve repeated polygons
espPols <- unionSpatialPolygons(espMap, espMap$PROVMUN)
```

Spanish maps are commonly displayed with the Canarian islands next to the peninsula. First we have to extract the polygons of the islands and the polygons of the peninsula, and then shift the coordinates of the islands with `elide`. Finally, a new `SpatialPolygons` object binds the shifted islands with the peninsula.

```
## Extract Canarias islands from the SpatialPolygons object
canarias <- sapply(espPols@polygons, function(x)substr(x@ID, 1, 2) %
    in% c("35", "38"))
peninsulaPols <- espPols[!canarias]
islandPols <- espPols[canarias]

## Shift the island extent box to position them at the bottom right
    corner
dy <- bbox(peninsulaPols)[2,1] - bbox(islandPols)[2,1]
dx <- bbox(peninsulaPols)[1,2] - bbox(islandPols)[1,2]
islandPols2 <- elide(islandPols, shift=c(dx, dy))
bbIslands <- bbox(islandPols2)

## Bind Peninsula (without islands) with shifted islands
espPols <- rbind(peninsulaPols, islandPols2)
```

The final step is to link the data with the polygons. The `ID` slot of each polygon is the key to find the correspondent registry in the `votes2011` dataset.

```
## Match polygons and data using ID slot and PROVMUN column
IDs <- sapply(espPols@polygons, function(x)x@ID)
idx <- match(IDs, votes2011$PROVMUN)

##Places without information
idxNA <- which(is.na(idx))

##Information to be added to the SpatialPolygons object
dat2add <- votes2011[idx, ]

## SpatialPolygonsDataFrame uses row names to match polygons with
    data
row.names(dat2add) <- IDs
```

```
espMapVotes <- SpatialPolygonsDataFrame(espPols, dat2add)

## Drop those places without information
espMapVotes <- espMapVotes[-idxNA, ]
```

8.2.2 Map

The SpatialPolygonsDataFrame constructed in the previous section contains two main variables: whichMax, the name of the predominant political option, and pcMax, the percentage of votes obtained by this political option.

whichMax is a categorical value with four levels: the two main parties (PP and PSOE), the abstention results (ABS), and the rest of the parties (OTH). Figure 8.9 encodes these levels with a qualitative palette with constant hues and varying chroma and luminance for each class using the package colorspace (Zeileis, Hornik, and Murrell 2009). In order to improve the color discrimination, hues are equally spaced along the HCL (Hue, Chroma, Luminance) based color wheel.

```
library(colorspace)

classes <- levels(factor(espMapVotes$whichMax))
nClasses <- length(classes)

qualPal <- rainbow_hcl(nClasses, start=30, end=300)
```

For the definition of a combined palette in the next section, it is interesting to note that the colors provided by rainbow_hcl can be obtained with the following code where the distances between hues and their values are computed explicitly.

```
## distance between hues
step <- 360/nClasses
## hues equally spaced
hue = (30 + step*(seq_len(nClasses)-1))%%360
qualPal <- hcl(hue, c=50, l=70)
```

```
spplot(espMapVotes["whichMax"], col='transparent', col.regions=
    qualPal)
```

On the other hand, pcMax is a quantitative variable that can be adequately displayed with a sequential palette (Figure 8.10).

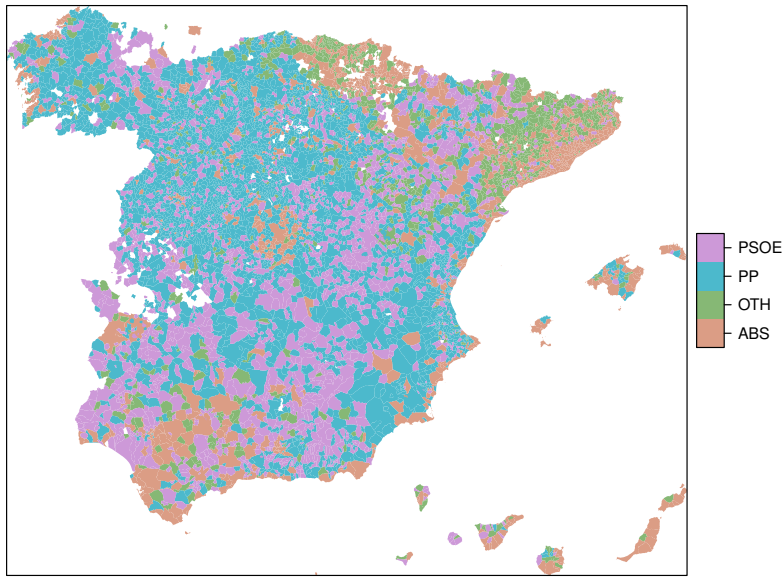

FIGURE 8.9: Categorical choropleth map displaying the name of the predominant political option in each municipality in the 2011 Spanish general elections.

```
quantPal <- rev(heat_hcl(16))
spplot(espMapVotes["pcMax"], col='transparent', col.regions=quantPal
    )
```

8.2.3 Categorical and Quantitative Variables Combined in a Multivariate Choropleth Map

Following the inspiring example of the infographic from the *New York Times*, we will combine both choropleth maps to produce a multivariate map: the hue of each polygon will be determined by the name of the predominant option (`whichMax`) but the chroma and luminance will vary according to the percentage of votes (`pcMax`). Hues are computed with the same method as in Figure 8.9, while the corresponding values of chroma and luminance are calculated with the `sequential_hcl` function.

```
classes <- levels(factor(espMapVotes$whichMax))
nClasses <- length(classes)
```

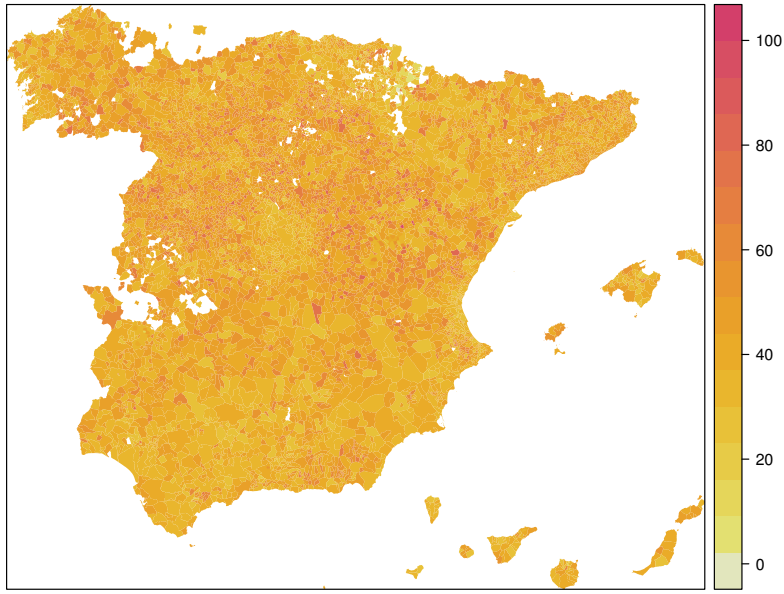

FIGURE 8.10: Quantitative choropleth map displaying the percentage of votes obtained by the predominant political option in each municipality in the 2011 Spanish general elections.

```
step <- 360/nClasses
multiPal <- lapply(1:nClasses, function(i){
   rev(sequential_hcl(16, h = (30 + step*(i-1))%%360))
   })
```

With this multivariate palette we can produce a list of maps extracting the polygons according to each class and filling with the appropiate color from this palette. The resulting list of `trellis` objects can be combined with `Reduce` and the `+.trellis` function of the `latticeExtra` and produce a `trellis` object.

It is important to note that, to ensure the legend's homogeneity, the breakpoints defined by the `at` argument are the same for all the individual maps.

```
pList <- lapply(1:nClasses, function(i){
   ## Only those polygons corresponding to a level are selected
   mapClass <- espMapVotes[espMapVotes$whichMax==classes[i],]
```

```
   pClass <- spplot(mapClass['pcMax'], col.regions=multiPal[[i]],
                col='transparent',
                ## labels only needed in the last legend
                colorkey=(if (i==nClasses) TRUE else list(labels=rep
                    ('', 6))),
                at = seq(0, 100, by=20))
})

p <- Reduce('+', pList)
```

The legend of this `trellis` object must be defined manually. The main operation is to merge the legends from the components of the list of maps to obtain a bivariate legend.

The first step is to add a title to each individual legend. This is a little complex because `levelplot` (the engine under the `spplot` method) does not include a title in its color key. The solution is to define a function to add the title and include it as an argument to the legend component of each `trellis` object. The `print.trellis` method will process this function when displaying the `trellis` object. The `frameGrob` and `packGrob` of the `grid` package will do the main work inside this function.

```
## Function to add a title to a legend
addTitle <- function(legend, title){
 titleGrob <- textGrob(title, gp=gpar(fontsize=8), hjust=1, vjust=1)
 ## retrieve the legend from the trellis object
 legendGrob <- eval(as.call(c(as.symbol(legend$fun), legend$args)))
 ## Layout of the legend WITH the title
 ly <- grid.layout(ncol=1, nrow=2,
               widths=unit(0.9, 'grobwidth', data=legendGrob))
 ## Create a frame to host the original legend and the title
 fg <- frameGrob(ly, name=paste('legendTitle', title, sep='_'))
 ## Add the grobs to the frame
 pg <- packGrob(fg, titleGrob, row=2)
 pg <- packGrob(pg, legendGrob, row=1)
 }

## Access each trellis object from pList...
for (i in seq_along(classes)){
 ## extract the legend (automatically created by spplot)...
 lg <- pList[[i]]$legend$right
 ## ... and add the addTitle function to the legend component of
     each trellis object
 pList[[i]]$legend$right <- list(fun='addTitle',
                          args=list(legend=lg, title=classes[i]))
```

```
}
```

Now that every component of `pList` includes a legend with a title, the legend of the `p` trellis object can be modified to store the merged legends from the set of components of `pList`.

```
## List of legends
legendList <- lapply(pList, function(x){
  lg <- x$legend$right
  clKey <- eval(as.call(c(as.symbol(lg$fun), lg$args)))
  clKey
})

## Function to pack the list of legends in a unique legend
## Adapted from latticeExtra::: mergedTrellisLegendGrob
packLegend <- function(legendList){
  N <- length(legendList)
  ly <- grid.layout(nrow = 1, ncol = N)
  g <- frameGrob(layout = ly, name = "mergedLegend")
  for (i in 1:N) g <- packGrob(g, legendList[[i]], col = i)
  g
}

## The legend of p will include all the legends
p$legend$right <- list(fun = 'packLegend', args = list(legendList =
    legendList))
```

Figure 8.11 displays the result with the province boundaries superposed (only for the peninsula due to a problem with the definition of boundaries the Canarian islands in the file) and a rectangle to separate the Canarian islands from the remainder of the map.

```
canarias <- provinces$PROV %in% c(35, 38)
peninsulaLines <- provinces[!canarias,]

p +
  layer(sp.polygons(peninsulaLines, lwd = 0.1)) +
  layer(grid.rect(x=bbIslands[1,1], y=bbIslands[2,1],
                  width=diff(bbIslands[1,]),
                  height=diff(bbIslands[2,]),
                  default.units='native', just=c('left', 'bottom'),
                  gp=gpar(lwd=0.5, fill='transparent')))
```

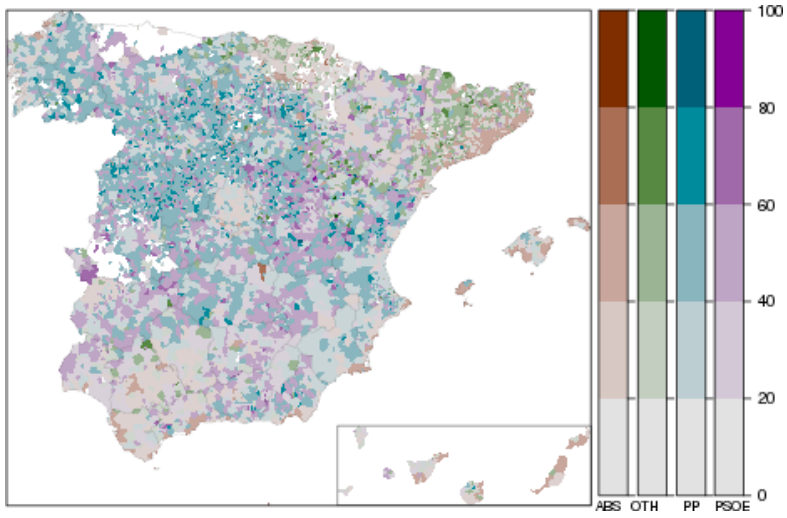

FIGURE 8.11: Spanish general elections results. The map shows the result of the most voted option in each municipality.

8.3 Raster Maps

A raster data structure is a matrix of cells organized into rows and columns where each cell contains a value representing information, such as temperature, altitude, population density, land use, etc. This section describes how to display a raster with two different examples: CM-SAF solar irradiation rasters will illustrate the use of quantitative data, and land cover and population data from the NEO-NASA project will exemplify the display of categorical data and multivariate rasters. Read Chapter 10 for details about these datasets.

8.3.1 Quantitative Data

As an example of quantitative data, this section displays the distribution of annual solar irradiation over the Iberian peninsula using the estimates from CM SAF. The `RasterLayer` object of annual averages of solar irradiation estimated by CM SAF can be easily displayed with the `levelplot` method of the `rasterVis` package. Figure 8.12 illustrates this raster with marginal graphics to show the column (longitude) and row (latitude) sum-

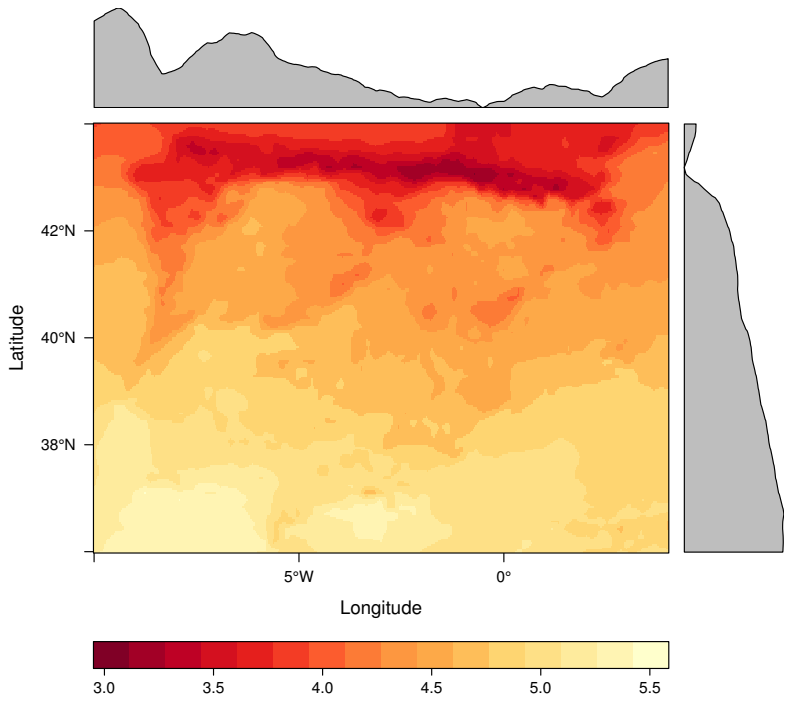

FIGURE 8.12: Annual average of solar radiation displayed with a sequential palette.

maries of the `RasterLayer` object. The summary is computed with the function defined by `FUN.margin` (which uses `mean` as the default value).

```
library(raster)
library(rasterVis)
SISav <- raster('data/SISav')
levelplot(SISav)
```

Although the solar irradiation distribution reveals the physical structure of the region, it is recommended to add the geographic context with a layer of administrative boundaries (Figure 8.13).

```
library(maps)
library(mapdata)
library(maptools)
```

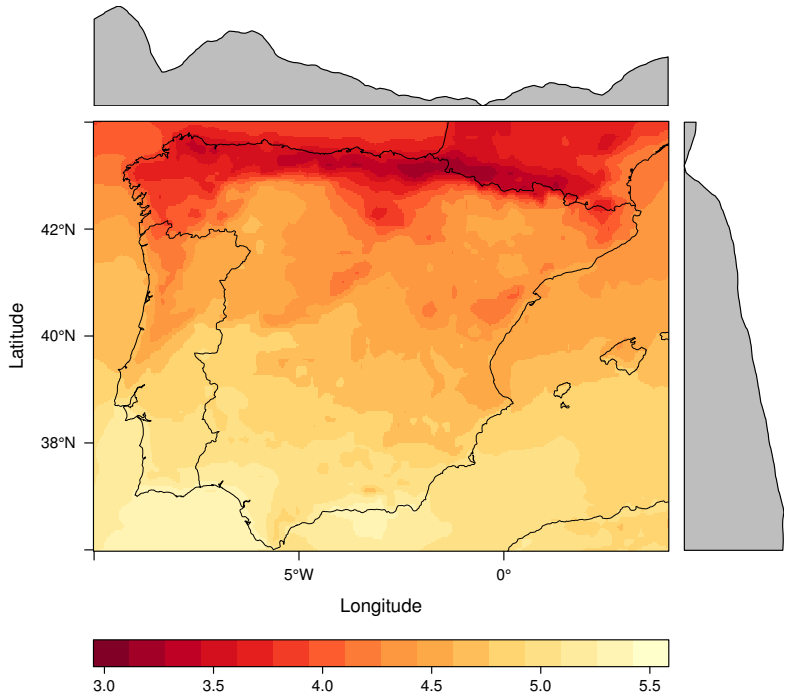

FIGURE 8.13: Annual average of solar radiation with administrative boundaries.

```
ext <- as.vector(extent(SISav))
boundaries <- map('worldHires',
              xlim=ext[1:2], ylim=ext[3:4],
              plot=FALSE)
boundaries <- map2SpatialLines(boundaries,
                        proj4string=CRS(projection(SISav)))
```

```
levelplot(SISav) + layer(sp.lines(boundaries, lwd=0.5))
```

8.3.1.1 Hill Shading

A frequent method to improve the display of meteorological rasters is the hill shading or shaded relief technique, a method of representing relief on

a map by depicting the shadows that would be cast by high ground if light comes from a certain sun position (Figure 8.14).

The procedure is as follows:

- Download a Digital Elevation Model (DEM) from the DIVA-GIS service.

```
old <- setwd(tempdir())
download.file('http://www.diva-gis.org/data/msk_alt/ESP_msk_alt.zip'
    , 'ESP_msk_alt.zip')
unzip('ESP_msk_alt.zip', exdir='.')

DEM <- raster('ESP_msk_alt')
```

- Compute the hill shade raster with `terrain` and `hillShade` from `raster`.

```
slope <- terrain(DEM, 'slope')
aspect <- terrain(DEM, 'aspect')
hs <- hillShade(slope=slope, aspect=aspect,
             angle=20, direction=30)
```

```
setwd(old)
```

- Combine the result with the previous map using semitransparency.

```
## hillShade theme: gray colors and semitransparency
hsTheme <- modifyList(GrTheme(), list(regions=list(alpha=0.6)))

levelplot(SISav, panel=panel.levelplot.raster,
        margin=FALSE, colorkey=FALSE) +
   levelplot(hs, par.settings=hsTheme, maxpixels=1e6) +
   layer(sp.lines(boundaries, lwd=0.5))
```

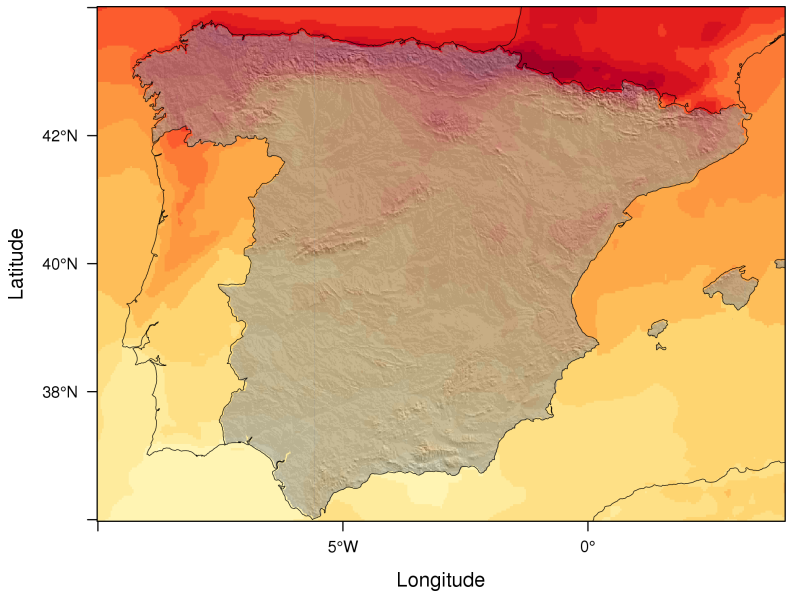

FIGURE 8.14: Hill shading of annual average of solar radiation.

8.3.1.2 Excursus: 3D Visualization

An alternative method for a DEM is 3D visualization where the user can rotate or zoom the figure. This solution is available thanks to the `rgl` package, which provides functions for 3D interactive graphics. The `plot3D` function in the `rasterVis` package is a wrapper to this package for `RasterLayer` objects.

```
plot3D(DEM, maxpixels=5e4)
```

The output scene can be exported to several formats such as WebGL with `writeWebGL` to be rendered in a browser, or `STL` with `writeSTL`, a format commonly used in 3D printing. Files using this format are viewed easily on GitHub (Figure 8.15)

```
writeSTL('figs/DEM.stl')
```

8.3.1.3 Diverging Palettes

Next, instead of displaying the absolute values of each cell, we will analyze the differences between each cell and the global average value. This

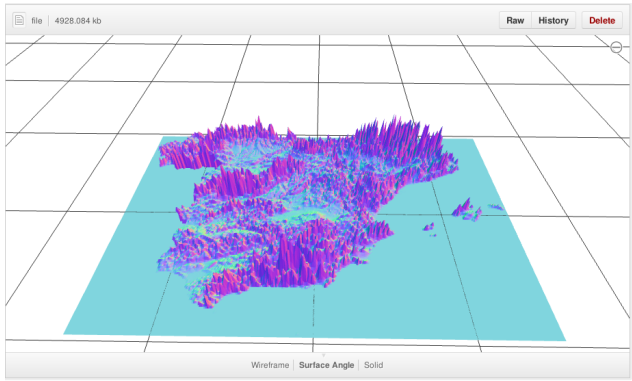

FIGURE 8.15: 3D visualization of a Digital Elevation Model using the STL format in a GitHub repository.

average is computed with the `cellStats` function and substracted from the original `RasterLayer`. Figure 8.16 displays the relation between these scaled values and latitude (`y`), with five different groups defined by the longitude (`cut(x, 5)`). It is evident that larger irradiation values are associated with lower latitudes. However, there is no such clear relation between irradiation and longitude.

```
meanRad <- cellStats(SISav, 'mean')
SISav <- SISav - meanRad
```

```
xyplot(layer ~ y, data = SISav,
      groups=cut(x, 5),
      par.settings=rasterTheme(symbol=plinrain(n=5, end=200)),
      xlab = 'Latitude', ylab = 'Solar␣radiation␣(scaled)',
      auto.key=list(space='right', title='Longitude', cex.title=1.3))
```

Numerical information ranging in an interval including a neutral value is commonly displayed with diverging palettes. These palettes represent neutral classes with light colors, while low and high extremes of the data range are highlighted using dark colors with contrasting hues. I use the Purple-Orange palette from ColorBrewer with purple for positive values and orange for negative values. In order to underline the position of the interval containing zero, the center color of this palette is substituted with pure white. The resulting palette is displayed in Figure 8.17 with the custom `showPal` function. The corresponding correspondent raster map pro-

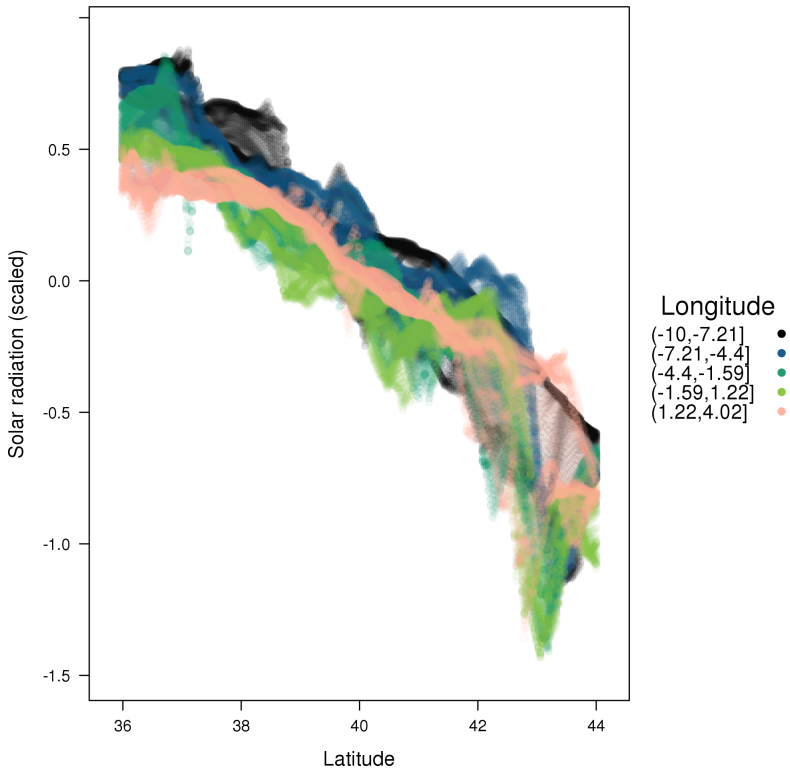

FIGURE 8.16: Relation between scaled annual average radiation and latitude for several longitude groups.

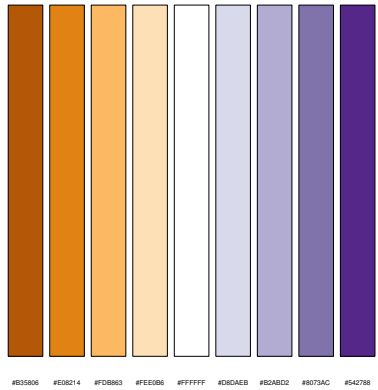

FIGURE 8.17: Purple-Orange diverging palette using white as middle color.

duced with this palette is displayed in Figure 8.18. Although extreme positive and negative values can be easily discriminated, the zero value is not associated with white because the data range is not symmetrical around zero.

```
divPal <- brewer.pal(n=9, 'PuOr')
divPal[5] <- "#FFFFFF"

showPal <- function(pal, labs=pal, cex=0.6, ...){
  barplot(rep(1, length(pal)), col=pal,
          names.arg=labs, cex.names=cex,
          axes=FALSE, ...)
}

showPal(divPal)
```

```
divTheme <- rasterTheme(region=divPal)

levelplot(SISav, contour=TRUE, par.settings=divTheme)
```

The solution is to connect the symmetrical color palette with the asymmetrical data range. The first step is to create a set of breaks such that the zero value is the center of one of the intervals.

```
rng <- range(SISav[])
## Number of desired intervals
```

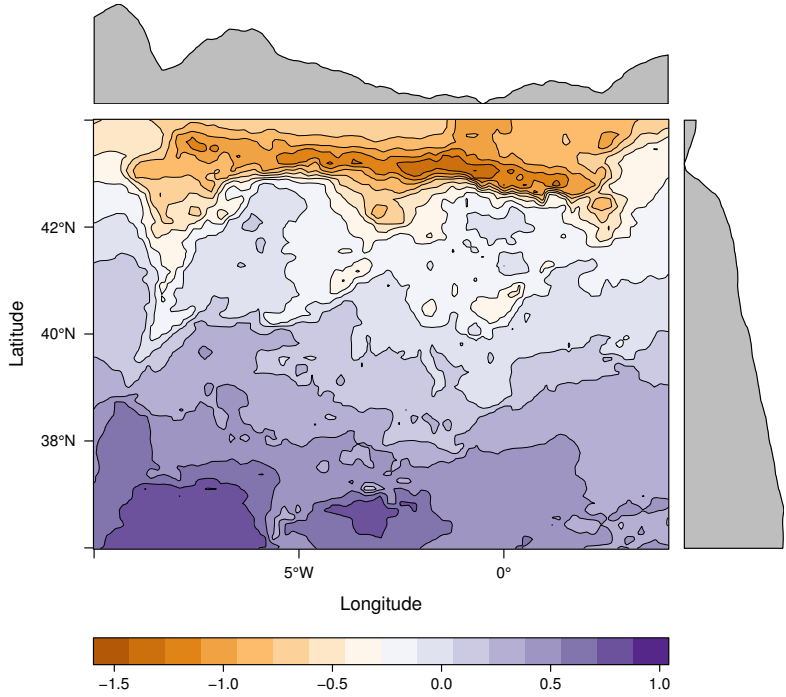

FIGURE 8.18: Asymmetric raster data (scaled annual average irradiation) displayed with a symmetric diverging palette.

```
nInt <- 15
## Increment corresponding to the range and nInt
inc0 <- diff(rng)/nInt
## Number of intervals from the negative extreme to zero
n0 <- floor(abs(rng[1])/inc0)
## Update the increment adding 1/2 to position zero in the center of
     an interval
inc <- abs(rng[1])/(n0 + 1/2)
## Number of intervals from zero to the positive extreme
n1 <- ceiling((rng[2]/inc - 1/2) + 1)
## Collection of breaks
breaks <- seq(rng[1], by=inc, length= n0 + 1 + n1)
```

The next step is to compute the midpoints of each interval. These points represent the data belonging to each interval, and their value will be connected with a color of the palette.

```
## Midpoints computed with the median of each interval
idx <- findInterval(SISav[], breaks, rightmost.closed=TRUE)
mids <- tapply(SISav[], idx, median)
## Maximum of the absolute value both limits
mx <- max(abs(breaks))
mids
```

A simple method to relate the palette and the intervals is with a straight line such that a point is defined by the absolute maximum value, `((mx, 1))`, and another point by zero, `((0, 0.5))`. Why are we using the interval [0, 1] as the y-coordinate of this line, and why is 0.5 the result of zero? The reason is that the input of the `break2pal` function will be the result of `colorRamp`, a function that creates another interpolating function which maps colors with values between 0 and 1. Therefore, a new palette is created, extracting colors from the original palette, such that the central color (white) is associated with the interval containing zero. This palette is displayed in Figure 8.19.

The raster map produced with this new palette is displayed in Figure 8.20. Now zero is clearly associated with the white color.

```
break2pal <- function(x, mx, pal){
  ## x = mx gives y = 1
  ## x = 0 gives y = 0.5
  y <- 1/2*(x/mx + 1)
  rgb(pal(y), maxColorValue=255)
}

## Interpolating function that maps colors with [0, 1]
## rgb(divRamp(0.5), maxColorValue=255) gives "#FFFFFF" (white)
divRamp <- colorRamp(divPal)
## Diverging palette where white is associated with the interval
## containing the zero
pal <- break2pal(mids, mx, divRamp)
showPal(pal, round(mids, 1))
```

```
levelplot(SISav, par.settings=rasterTheme(region=pal),
        at=breaks, contour=TRUE)
```

It is interesting to note two operations carried out internally by the `lattice` package. First, the `custom.theme` function (used by `rasterTheme`)

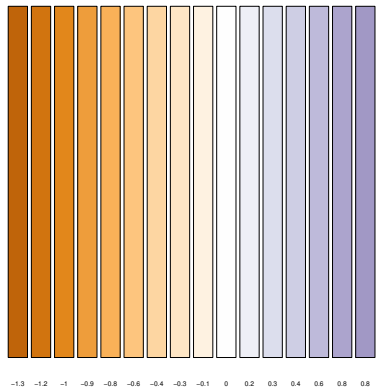

FIGURE 8.19: Modified diverging palette related with the asymmetrical raster data.

creates a new palette with 100 colors using `colorRampPalette` to interpolate the palette passed as an argument. Second, the `level.colors` function makes the arrangement between intervals and colors. If this function receives more colors than intervals, it chooses a subset of the palette disregarding some of the intermediate colors. Therefore, because this function will receive 100 colors from `par.settings`, it is difficult to control exactly which colors of our original palette will be represented.

An alternative way for finer control is to fill the `regions$col` component of the theme with our palette after it has been created (Figure 8.21).

```
divTheme <- rasterTheme()

divTheme$regions$col <- pal
levelplot(SISav, par.settings=divTheme, at=breaks, contour=TRUE)
```

A final improvement to this map is to compute the intervals using a classification algorithm with the `classInt` package. With this approach it is likely that zero will not be perfectly centered in its corresponding interval. The remaining code is exactly the same as above, replacing the `breaks` vector with the result of the `classIntervals` function. Figure 8.22 displays the result.

```
library(classInt)

cl <- classIntervals(SISav[],
```

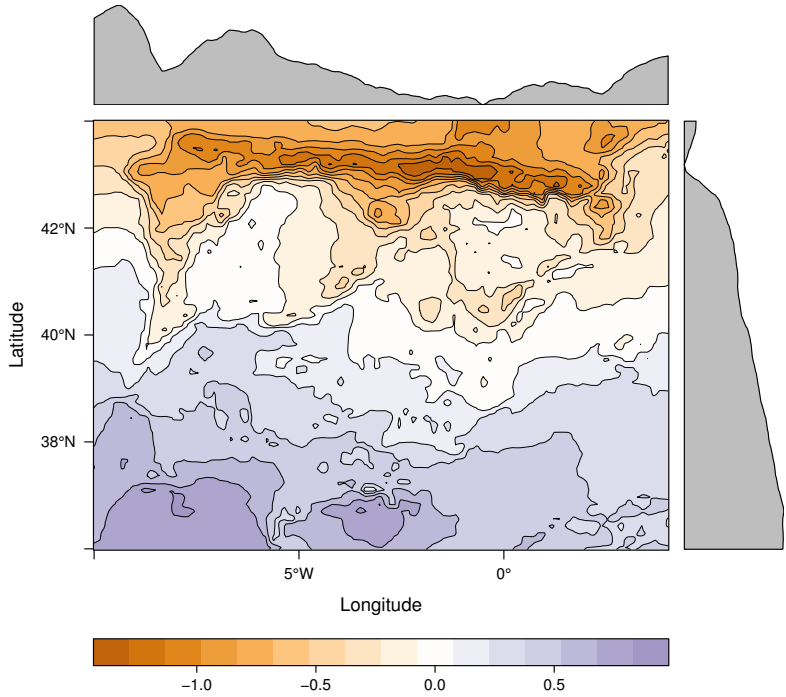

FIGURE 8.20: Asymmetric raster data (scaled annual average irradiation) displayed with a modified diverging palette.

```
                ## n=15, style='equal')
                ## style='hclust')
                ## style='sd')
                style='kmeans')
                ## style='quantile')
cl
breaks <- cl$brks
```

```
idx <- findInterval(SISav[], breaks, rightmost.closed=TRUE)
mids <- tapply(SISav[], idx, median)
mids
mx <- max(abs(breaks))
pal <- break2pal(mids, mx, divRamp)
divTheme$regions$col <- pal
levelplot(SISav, par.settings=divTheme, at=breaks, contour=TRUE)
```

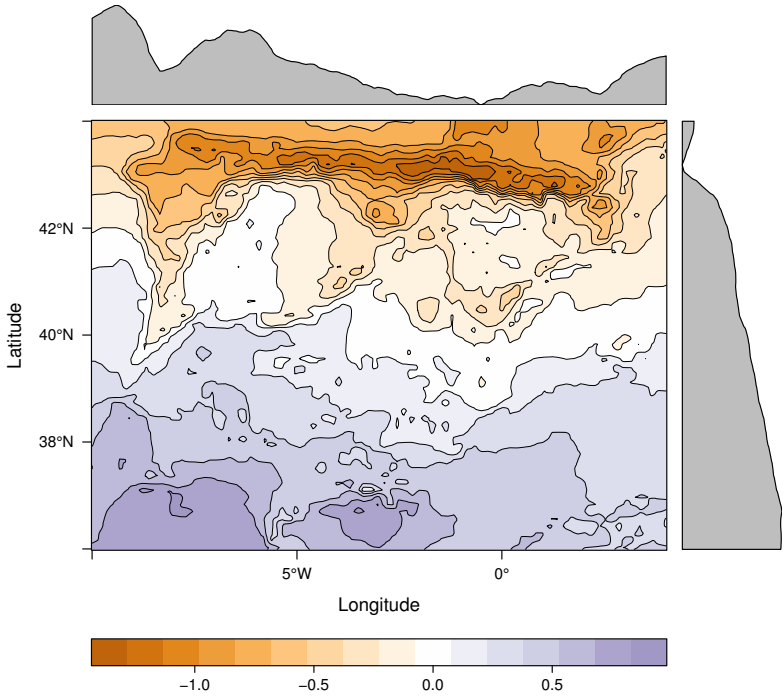

FIGURE 8.21: Same as Figure 8.20 but colors are assigned directly to the `regions$col` component of the theme.

8.3.2 Categorical Data

Land cover is the observed physical cover on the Earth's surface. A set of seventeen different categories is commonly used. Using satellite observations, it is possible to map where on Earth each of these seventeen land surface categories can be found and how these land covers change over time.

This section illustrates how to read and display rasters with categorical information using information from the NEO-NASA project. After the land cover and population density files have been downloaded, two `RasterLayers` can be created with the `raster` package. Both files are read, their geographical extent reduced to the area of India and China, and cleaned (`99999` cells are replaced with `NA`).

```
library(raster)
```

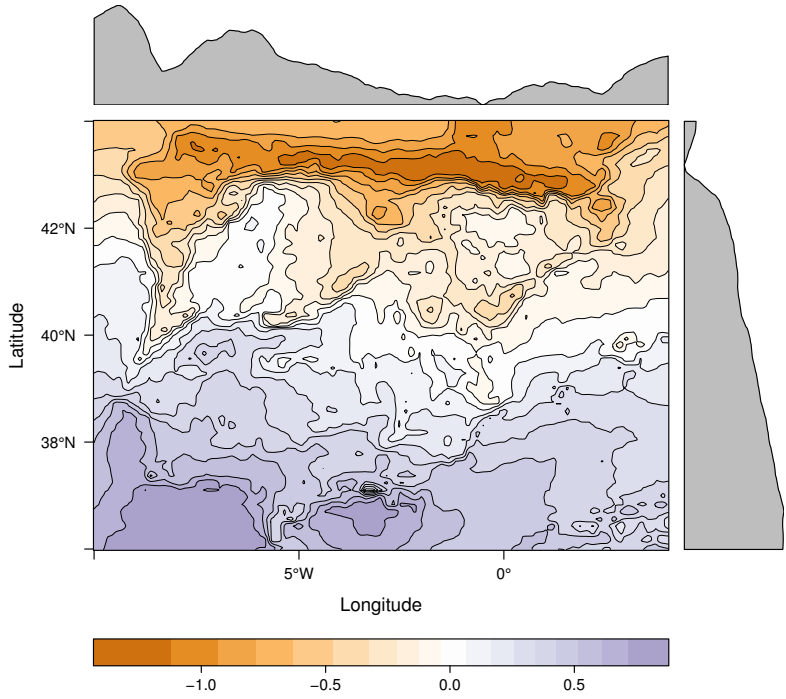

FIGURE 8.22: Same as Figure 8.21 but defining intervals with the optimal classification method.

```
## China and India
ext <- extent(65, 135, 5, 55)

pop <- raster('875430rgb-167772161.0.FLOAT.TIFF')
pop <- crop(pop, ext)
pop[pop==99999] <- NA

landClass <- raster('241243rgb-167772161.0.TIFF')
landClass <- crop(landClass, ext)
```

Each land cover type is designated with a different key: the sea is labeled with 0; forests with 1 to 5; shrublands, grasslands, and wetlands with 6 to 11; agriculture and urban lands with 12 to 14; and snow and barren with 15 and 16. These four groups (sea is replaced by `NA`) will be the levels of the categorical raster. The `raster` package includes the `ratify`

method to define a layer as categorical data, filling it with integer values associated to a Raster Attribute Table (RAT).

```
landClass[landClass %in% c(0, 254)] <- NA
## Only four groups are needed:
## Forests: 1:5
## Shrublands, etc: 6:11
## Agricultural/Urban: 12:14
## Snow: 15:16
landClass <- cut(landClass, c(0, 5, 11, 14, 16))
## Add a Raster Attribute Table and define the raster as categorical
     data
landClass <- ratify(landClass)
## Configure the RAT: first create a RAT data.frame using the
## levels method; second, set the values for each class (to be
## used by levelplot); third, assign this RAT to the raster
## using again levels
rat <- levels(landClass)[[1]]
rat$classes <- c('Forest', 'Land', 'Urban', 'Snow')
levels(landClass) <- rat
```

This categorical raster can be displayed with the `levelplot` method of the `rasterVis` package. Previously, a theme is defined with the background color set to `lightskyblue1` to display the sea areas (filled with `NA` values), and the region palette is defined with adequate colors (Figure 8.23).

```
library(rasterVis)

pal <- c('palegreen4', # Forest
       'lightgoldenrod', # Land
       'indianred4', # Urban
       'snow3') # Snow

catTheme <- modifyList(rasterTheme(),
                  list(panel.background = list(col='lightskyblue1'),
                       regions = list(col= pal)))

levelplot(landClass, maxpixels=3.5e5, par.settings=catTheme,
        panel=panel.levelplot.raster)
```

Let's explore the relation between the land cover and population density rasters. Figure 8.24 displays this latter raster using a logarithmic scale.

```
pPop <- levelplot(pop, zscaleLog=10, par.settings=BTCTheme,
```

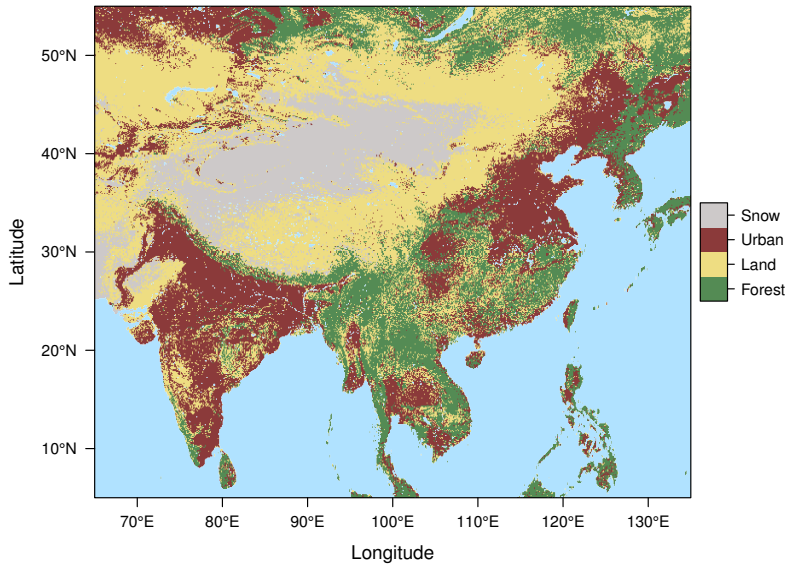

FIGURE 8.23: Land cover raster (categorical data).

```
                maxpixels=3.5e5, panel=panel.levelplot.raster)
pPop
```

Both rasters can be joined together with the `stack` method to create a new `RasterStack` object. Figure 8.25 displays the distribution of the logarithm of the population density associated to each land class.

```
s <- stack(pop, landClass)
names(s) <- c('pop', 'landClass')
histogram(~log10(pop)|landClass, data=s,
        scales=list(relation='free'))
```

8.3.3 Multivariate Legend

We can reproduce the code used to create the multivariate choropleth (Section 8.2) using the `levelplot` function from the `rasterVis` package. Again, the result is a list of `trellis` objects. Each of these objects is the representation of the population density in a particular land class. The `+.trellis`

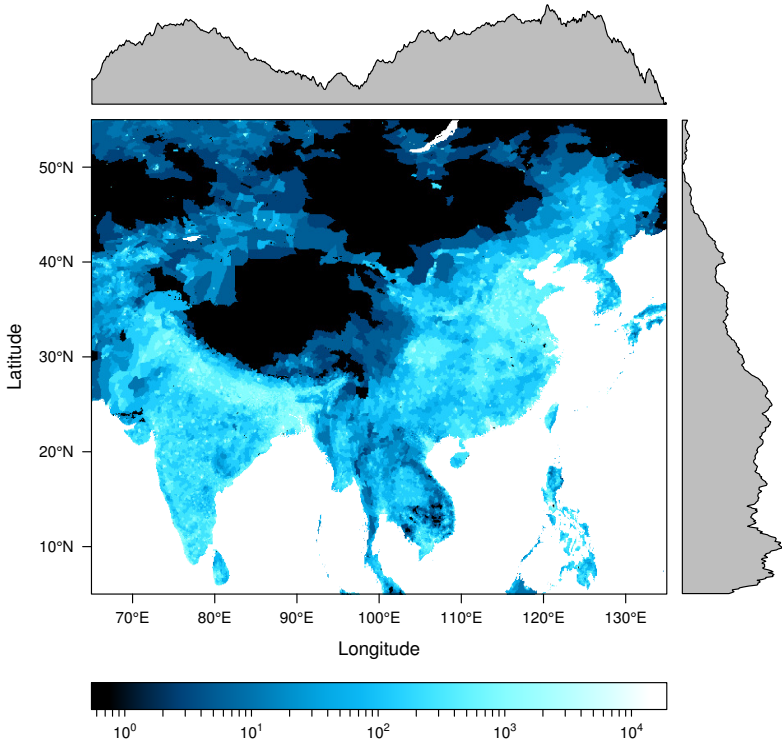

FIGURE 8.24: Population density raster.

function of the `latticeExtra` package with `Reduce` superposes the elements of this list and produces a `trellis` object. Figure 8.26 displays the result.

```
library(colorspace)
## at for each sub-levelplot is obtained from the global levelplot
at <- pPop$legend$bottom$args$key$at
classes <- rat$classes
nClasses <- length(classes)

pList <- lapply(1:nClasses, function(i){
  landSub <- landClass
  ## Those cells from a different land class are set to NA...
  landSub[!(landClass==i)] <- NA
```

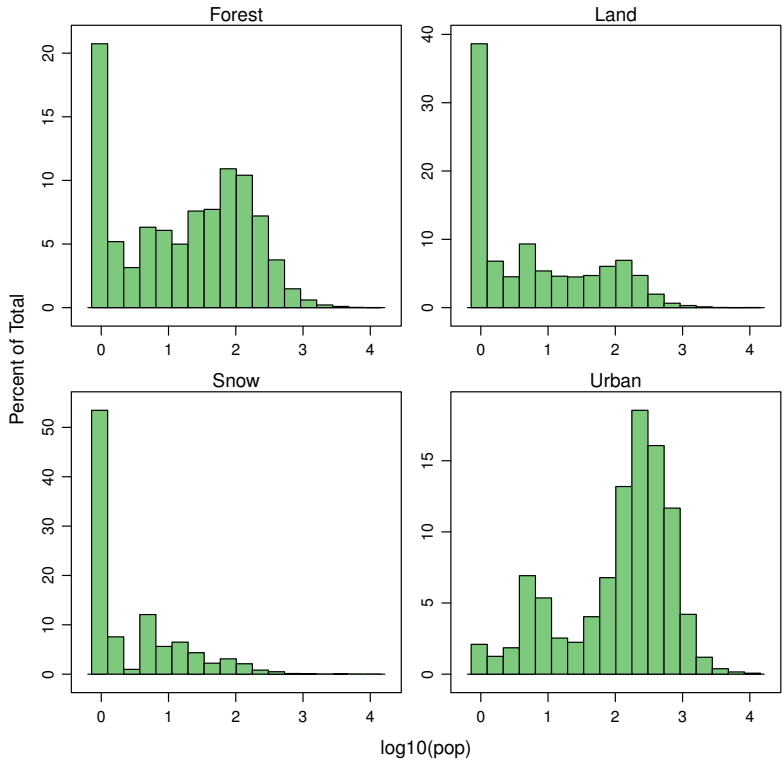

FIGURE 8.25: Distribution of the logarithm of the population density associated to each land class.

```
## ... and the resulting raster masks the population raster
popSub <- mask(pop, landSub)
## The HCL color wheel is divided in nClasses
step <- 360/nClasses
## and a sequential palette is constructed with a hue from one of
## the color wheel parts
cols <- rev(sequential_hcl(16, h = (30 + step*(i-1))%%360))

pClass <- levelplot(popSub, zscaleLog=10, at=at,
                maxpixels=3.5e5,
                ## labels only needed in the last legend
                colorkey=(if (i==nClasses) TRUE else list(labels=
                    list(labels=rep('', 17)))),
```

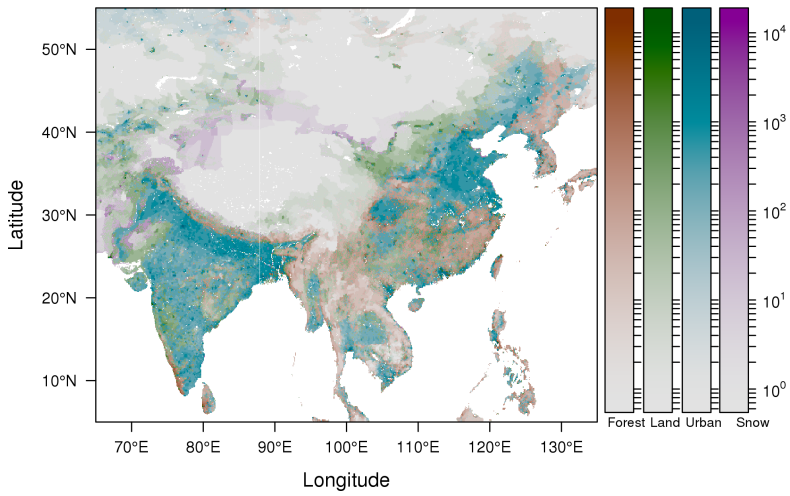

FIGURE 8.26: Population density for each land class (multivariate raster).

```
                    col.regions=cols, margin=FALSE)
})
```

8.4 Vector Fields

Many objects in our natural environment exhibit directional features that are naturally represented by vector data. Vector fields, commonly found in science and engineering, describe the spatial distribution of a vector variable such as fluid flow or electromagnetic forces. A suitable visualization method has to display both the magnitude and the direction of the vectors at any point.

This section illustrates two visualization techniques, arrow plots and stream lines, with the help of the wind direction and speed forecast published by MeteoGalicia (see Section 12.5 for details).

```
library(raster)
library(rasterVis)

wDir <- raster('data/wDir')/180*pi
wSpeed <- raster('data/wSpeed')
windField <- stack(wSpeed, wDir)
names(windField) <- c('magnitude', 'direction')
```

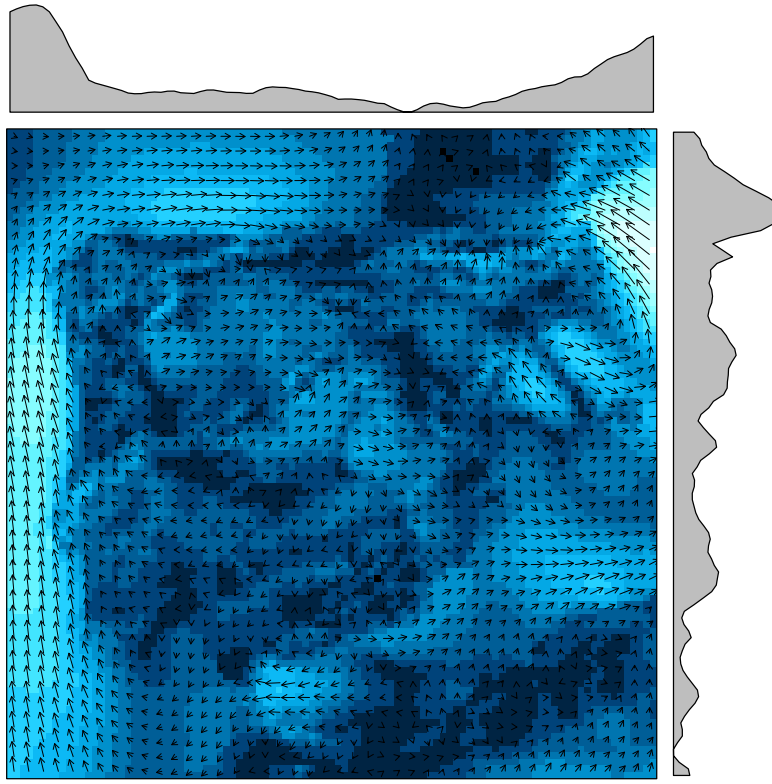

FIGURE 8.27: Arrow plot of the wind vector field.

8.4.1 Arrow Plot

A frequent vector visualization technique is the arrow plot, which draws a small arrow at discrete points within the vector field (Figure 8.27). This approach is best suited for small datasets. If the grid of discrete points gets too dense or if the variations in magnitude are too big, the images tend to be visually confusing.

```
vectorplot(windField, isField=TRUE, par.settings=BTCTheme(),
        colorkey=FALSE, scales=list(draw=FALSE))
```

8.4.2 Streamlines

Another solution is to depict the directional structure of the vector field by its integral curves, also denoted as flow lines or streamlines. There are a variety of algorithms to produce such visualization. The `streamplot` function of `rasterVis` displays streamlines with a procedure inspired by the FROLIC algorithm: For each point, *droplet*, of a jittered regular grid, a short streamline portion, *streamlet*, is calculated by integrating the underlying vector field at that point. The main color of each streamlet indicates local vector magnitude. Streamlets are composed of points whose sizes, positions, and color degradation encode the local vector direction (Figure 8.28).

```
myTheme <- streamTheme(region=rev(brewer.pal(n=4, name='Greys')),
                               symbol=BTC(n=9, beg=20))
streamplot(windField, isField=TRUE,
         par.settings=myTheme,
         droplet=list(pc=12),
         streamlet=list(L=5, h=5),
         scales=list(draw=FALSE),
         panel=panel.levelplot.raster)
```

The magic of Figure 8.28 is that it is able to show the underlying physical structure of the spatial region only displaying wind speed and direction. It is easy to recognize the Iberian Peninsula surrounded by strong winds along the eastern and northern coasts. Another feature easily distinguishable is the Strait of Gibraltar, a channel that connects the Atlantic Ocean to the Mediterranean Sea between the south of Spain and the north of Morocco. Also apparent are the Pyrenees mountains and some of the river valleys.

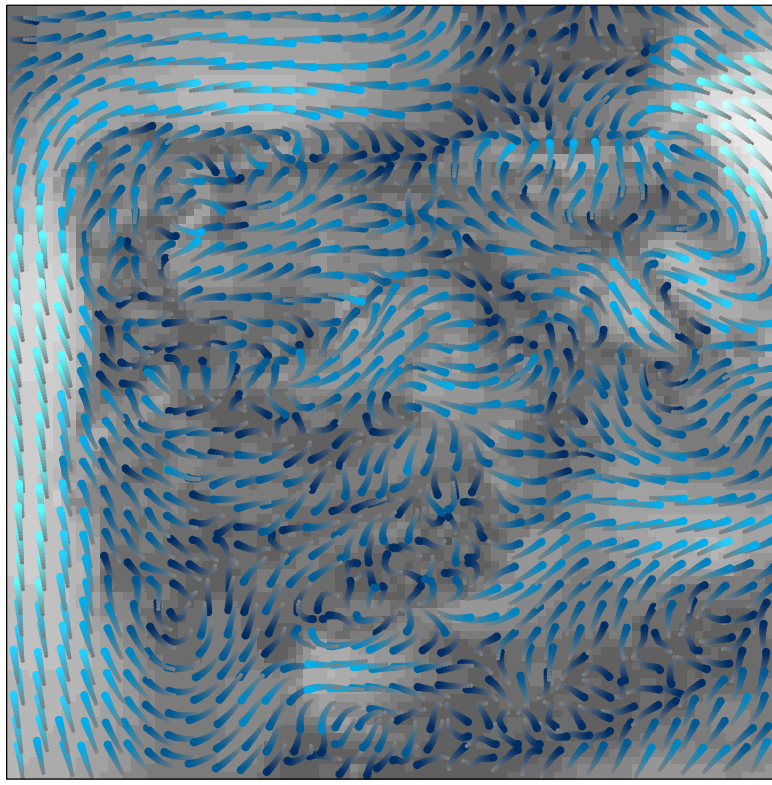

FIGURE 8.28: Streamlines of the wind vector field.

Chapter 9

Reference and Physical Maps

A reference map focuses on the geographic location of features. In these maps, cities are named and major transport routes are identified. In addition, natural features such as rivers and mountains are named, and elevation is shown using a simple color shading. A physical map shows the physical landscape features of a place. Mountains and elevation changes are usually shown with different colors and shades to show relief, using green to show lower elevations and browns for high elevations.

This chapter details how to create a reference map of a northern region of Spain using data from OpenStreetMap and a physical map of Brazil with data from different sources.

9.1 Physical Maps

Brazil[1], the world's fifth largest country, is one of the seventeen megadiverse countries[2], home to diverse wildlife, natural environments, and extensive natural resources in a variety of protected habitats. Throughout this section we will create a physical map of this exceptional country using data from several data services.

```
library(raster)
```

[1] http://en.wikipedia.org/wiki/Brazil
[2] http://en.wikipedia.org/wiki/Megadiverse_countries

```
library(rasterVis)
library(maptools)
library(rgeos)
library(latticeExtra)
library(colorspace)

## Longitude-Latitude projection
proj <- CRS('␣+proj=longlat␣+ellps=WGS84')
```

9.1.1 Retrieving Data

Four types of information are needed: administrative boundaries, terrain elevation, rivers and lakes, and sea depth.

1. The administrative boundaries are available from GADM[3]. The `readShapePoly` function reads data from the downloaded shapefile and creates a `SpatialPolygonsDataFrame` object.

   ```
   old <- setwd(tempdir())

   download.file('http://www.gadm.org/data/shp/BRA_adm.zip',
             'BRA_adm.zip')
   unzip('BRA_adm.zip')
   brazilAdm <- readShapePoly('BRA_adm1.shp', proj4string=proj)
   Encoding(levels(brazilAdm$NAME_1)) <- 'latin1'
   ```

2. The terrain elevation or digital elevation model (DEM) is available from DIVA-GIS[4]. The `raster` function reads the file and creates a `RasterLayer` object.

   ```
   download.file('http://www.diva-gis.org/data/alt/BRA_alt.zip',
             'BRA_alt.zip')
   unzip('BRA_alt.zip')
   brazilDEM <- raster('BRA_alt')
   ```

3. The water lines (rivers and lakes) are available from Natural Earth Data[5]. The `readShapeLines` function reads data from the downloaded shapefile and creates a `SpatialLinesDataFrame` object.

[3] http://gadm.org/
[4] http://www.diva-gis.org/Data
[5] http://www.naturalearthdata.com/

```
## World Water lines (Natural Earth)
download.file('http://www.naturalearthdata.com/http//www.
    naturalearthdata.com/download/10m/physical/ne_10m_rivers_lake_
    centerlines.zip',
           'neRivers.zip')
unzip('neRivers.zip')
worldlRiv <- readShapeLines('ne_10m_rivers_lake_centerlines',
    proj4string = proj)
```

4. Finally, the sea depth is also available from Natural Earth Data[5]. The raster covers the whole world so it must be cropped by the extent of the DEM raster.

```
download.file('http://www.naturalearthdata.com/http//www.
    naturalearthdata.com/download/10m/raster/OB_LR.zip',
           'neSea.zip')
unzip('neSea.zip')
worldSea <- raster('OB_LR.tif')
brazilSea <- crop(worldSea, brazilDEM)
setwd(old)
```

9.1.2 Intersection of Shapefiles and Elevation Model

The rivers and lakes database from Natural Earth Data comprises all the world extent, but we only need the rivers of Brazil. The function `gIntersection` of the package `rgeos` determines the intersection between two geometries. Because these geometries must be defined with classes of the `sp` package, the extent of `brazilDEM` must be first converted to `SpatialPolygons`. The intersection is a new `SpatialLines` object, `brazilRiv`.

```
## only those features labeled as "River" are needed
worlRiv<- worlRiv[worlRiv$featurecla=='River',]

## Define the extent of Brazil as a SpatialPolygons
extBrazil <- as(extent(brazilDEM), 'SpatialPolygons')
proj4string(extBrazil) <- proj

## and intersect it with worldRiv to extract brazilian rivers
## from the world database
brazilRiv <- gIntersection(worldRiv, extBrazil, byid=TRUE)
## and especially the famous Amazonas River
amazonas <- worldRiv[worldRiv$name=='Amazonas',]
```

9.1.3 Labels

Each region of Brazil will be labeled with the name of its corresponding polygon. The locations of the labels are defined by the centroid of each polygon, easily computed with the `coordinates` method. In addition, a larger label with the name of the country will be placed in the average centroid.

```
## Locations of labels of each polygon
centroids <- coordinates(brazilAdm)
## Location of the "Brazil" label (average of the set of polygons
    centroids)
xyBrazil <- apply(centroids, 2, mean)
```

Some region names are too long to be displayed in one line. Thus, a previous step is to split the string if it comprises more than two words.

```
admNames <- strsplit(as.character(brazilAdm$NAME_1), '␣')

admNames <- sapply(admNames,
              FUN=function(s){
                sep=if (length(s)>2) '\n' else '␣'
                paste(s, collapse=sep)
                })
```

9.1.4 Overlaying Layers of Information

Therefore, the physical map (Figure 9.2) is composed of four layers:

1. The sea depth raster displayed with the `levelplot` method of the `rasterVis` package. The palette is defined with `brewer.pal` (Figure 9.1).

   ```
   blueTheme <- rasterTheme(region=brewer.pal(n=9, 'Blues'))

   seaPlot <- levelplot(brazilSea, par.settings=blueTheme,
                  maxpixels=1e6, panel=panel.levelplot.raster,
                  margin=FALSE, colorkey=FALSE)
   ```

2. The altitude raster layer uses a terrain colors palette, as the one produced by the `terrain_hcl` function from the `colorspace` package (Ihaka et al. 2011) (Figure 9.1).

```
terrainTheme <- rasterTheme(region=terrain_hcl(15))

altPlot <- levelplot(brazilDEM, par.settings=terrainTheme,
                maxpixels=1e6, panel=panel.levelplot.raster,
                margin=FALSE, colorkey=FALSE)
```

3. The rivers represented by the `SpatialLinesDataFrame` object. The Amazonas River is labeled with `sp.lineLabel` and printed with a thicker line. The label is created with the `label` method, a wrapper function to extract the `ID` slots from the `SpatialLines` and create a suitable `character` object with the correct `names` values.

```
amazonasLab <- label(amazonas, 'Amazonas')
```

4. The administrative boundaries represented by the `SpatialPolygonsDataFrame` object with their labels printed with the `panel.pointLabel` function. This function uses optimization routines to find good locations for point labels without overlaps.

```
seaPlot + altPlot + layer({
   ## Rivers
   sp.lines(brazilRiv, col='darkblue', lwd=0.2)
   ## Amazonas
   sp.lineLabel(amazonas, amazonasLab,
             lwd=1, col='darkblue', col.line='darkblue',
             cex=0.5, fontfamily='Palatino')
   ## Administrative boundaries
   sp.polygons(brazilAdm, col='black', lwd=0.2)
   ## Centroids of administrative boundaries ...
   panel.points(centroids, col='black')
   ## ... with their labels
   panel.pointLabel(centroids, labels=admNames,
                cex=0.7, fontfamily='Palatino', lineheight=.8)
   ## Country name
   panel.text(xyBrazil[1], xyBrazil[2], labels='B␣R␣A␣Z␣I␣L',
            cex=1.5, fontfamily = 'Palatino', fontface=2)
})
```

9.2 OpenStreetMap with Hill Shade Layers

Although I was born in Madrid, Galicia (north of Spain) is a very special region for me. More precisely, the Cedeira and Valdoviño regions offer a

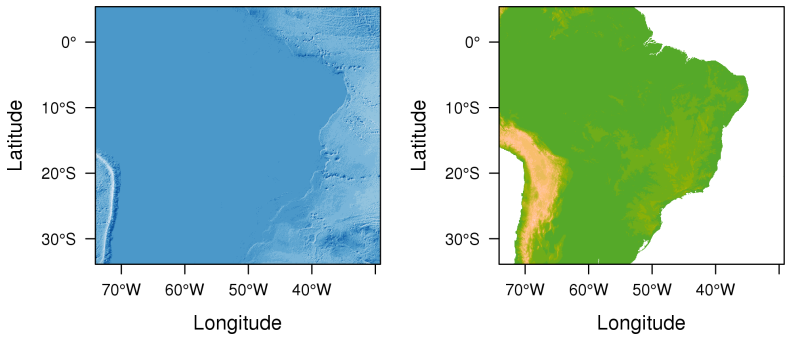

FIGURE 9.1: Sea depth and altitude rasters of Brazil.

wonderful combination of wild sea, secluded beaches, and forests. I will show you a map of these marvelous places.

9.2.1 Retrieving Data from OpenStreetMap

The first step is to acquire information from the OpenStreetMap project. There are several packages to extract data from this service but, while most of them only provide already rendered raster images, the `osmar` package[6] (Eugster and Schlesinger 2010) enables the use of the raw data with classes from the packages `sp` and `igraph`.

The `get_osm` function retrieves a region defined by `corner_bbox` using the OSM API.

```
library('osmar')

api <- osmsource_api()
ymax <- 43.7031
ymin <- 43.6181
xmax <- -8.0224
xmin <- -8.0808
box <- corner_bbox(xmin, ymin, xmax, ymax)
cedeira <- get_osm(box, source=api, full=TRUE)
```

[6]Its webpage http://osmar.r-forge.r-project.org/ proposes two interesting demos.

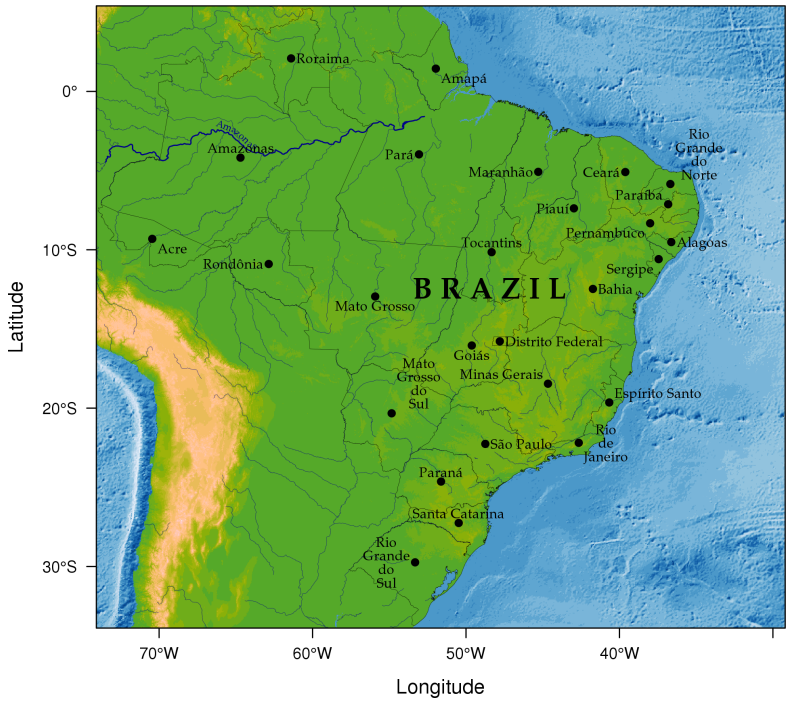

FIGURE 9.2: Physical map of Brazil. Main administrative regions and the Amazonas River are labeled.

The `cedeira` object includes three main components: nodes, ways and relations. These components can be accessed with the functions `find`, `subset`, `way`, `node`, `relation`, and `tags`. Thus, the different kinds of roads can be obtained using `way` and `tags` with the appropiate tag.

```
summary(cedeira$nodes)
```

```
idxHighways <- find(cedeira, way(tags(k=='highway')))
highways <- subset(cedeira, way_ids=idxHighways)
idxStreets <- find(highways, way(tags(v=='residential')))
idxPrimary <- find(highways, way(tags(v=='primary')))
idxSecondary <- find(highways, way(tags(v=='secondary')))
idxTertiary <- find(highways, way(tags(v=='tertiary')))
idxOther <- find(highways,
             way(tags(v=='unclassified' |
```

```
                    v=='footway' |
                    v=='steps')))
```

The result of `find` is the index of each element. The correspondent spatial object is extracted with `find_down` and `subset`, and can be converted to a class defined by the `sp` package with `as_sp`. The following `spFromOSM` function encodes the procedure, and extracts the `SpatialLines` object that represent each type of road.

```
spFromOSM <- function(source, index, type='lines'){
  idx <- find_down(source, index)
  obj <- subset(source, ids=idx)
  objSP <- as_sp(obj, type)
  }

streets <- spFromOSM(cedeira, way(idxStreets))
primary <- spFromOSM(cedeira, way(idxPrimary))
secondary <- spFromOSM(cedeira, way(idxSecondary))
tertiary <- spFromOSM(cedeira, way(idxTertiary))
other <- spFromOSM(cedeira, way(idxOther))
```

A similar procedure can be applied to construct a `SpatialPoints` object with the collection of places with name:

```
idxPlaces <- find(cedeira, node(tags(k=='name')))
places <- spFromOSM(cedeira, node(idxPlaces), 'points')

nms <- subset(cedeira$nodes$tags, subset=(k=='name'), select=c('id',
     'v'))
ord <- match(idxPlaces, nms$id)
nms <- nms[ord,]
places$name <- nms$v[ord]

## Cedeira town will be printed differently
idxCedeira <- which(nms$v=='Cedeira') ##Main town
cedeiraCoords <- coordinates(places[idxCedeira,])
places <- places[-idxCedeira,]
```

9.2.2 Hill Shading

The second step is to produce layers to display the topography. A suitable method is shaded relief or hill shading. This technique simulates the cast shadow thrown from a light source upon a raised relief map. The hill shade layer can be computed from the slope and aspect layers derived

from a Digital Elevation Model (DEM). This layer will underlay the DEM raster, which will be printed using semitransparency.

The DEM for this region is available at the Geonetwork-SECAD service from the Universidad de Extremadura and can be read with `raster`:

```
library(raster)
## Galicia DEM
## http://ide.unex.es/geonetwork/srv/es/main.search?any=MDE_Galicia
## http://ide.unex.es:8180/geonetwork/srv/es/resources.get?id=21&
    fname=dem_gal.7z&access=private

old <- tempdir()
download.file('http://ide.unex.es:8180/geonetwork/srv/es/resources.
    get?id=21&fname=dem_gal.7z&access=private', 'dem_gal.7z')
unzip('dem_gal.7z')
demGalicia <- raster('dem_gal.asc')
setwd(old)
```

The `slope` and `aspect` layers are computed with the `terrain` function, and the hill shade layer is derived with these layers for a fixed sun position. Previously, the useful region of the DEM raster was extracted with the `crop` function:

```
cedeiraSP <- as_sp(cedeira, 'points')
projCedeira <- projection(cedeiraSP)
##extCedeira <- bbox(cedeiraSP)
## or summary(cedeira$nodes)$bbox
extCedeira <- extent(-8.15, -7.95, 43.6, 43.75)
demCedeira <- crop(demGalicia, extCedeira)
projection(demCedeira) <- projCedeira
demCedeira[demCedeira <= 0] <- NA

slope <- terrain(demCedeira, 'slope')
aspect <- terrain(demCedeira, 'aspect')
hsCedeira <- hillShade(slope=slope, aspect=aspect,
                  angle=20, direction=30)
```

9.2.3 Overlaying Layers of Information

And finally, the third step is to display the different layers of information in correct order (Figure 9.3):

- The hill shade layer is created with the `levelplot` method for `Raster` objects defined in the `rasterVis` package. The `GrTheme` is modified to display the sea region with blue color.

- The DEM raster is printed with terrain colors and semitransparency over the hill shade layer.
- The roads are displayed with an auxiliary function (`sp.road`) that produces a colored line over a thicker black line.
- The places are represented with `sp.points` and labeled with the `sp.pointLabel` method, a modification of the `pointLabel` function for `base` graphics, both defined in the `maptools` package. These functions use optimization routines to find good locations for point labels without overlaps.

```
library(maptools)
library(latticeExtra)
library(colorspace)
library(rasterVis)

##Auxiliary function to display the roads. A thicker black line in
##the background and a thinner one with an appropiate color.
sp.road <- function(line, lwd=5, blwd=7,
                col='indianred1', bcol='black'){
  sp.lines(line, lwd=blwd, col=bcol)
  sp.lines(line, lwd=lwd, col=col)
}

## The background color of the panel is set to blue to represent the
     sea
hsTheme <- modifyList(GrTheme(), list(panel.background=list(col='
    skyblue3')))
## DEM with terrain colors and semitransparency
terrainTheme <- modifyList(rasterTheme(region=terrain_hcl(n=15)),
                           list(regions=list(alpha=0.6)))
## Hill shade and DEM overlaid
levelplot(hsCedeira, maxpixels=ncell(hsCedeira),
        par.settings=hsTheme, margin=FALSE, colorkey=FALSE) +
  levelplot(demCedeira, maxpixels=ncell(demCedeira),
          par.settings=terrainTheme) +
  ## Roads and places
  layer({
    ## Street and roads
    sp.road(streets, lwd=1, blwd=2, col='white')
    sp.road(other, lwd=2, blwd=3, col='white')
    sp.road(tertiary, lwd=3, blwd=4, col='palegreen')
```

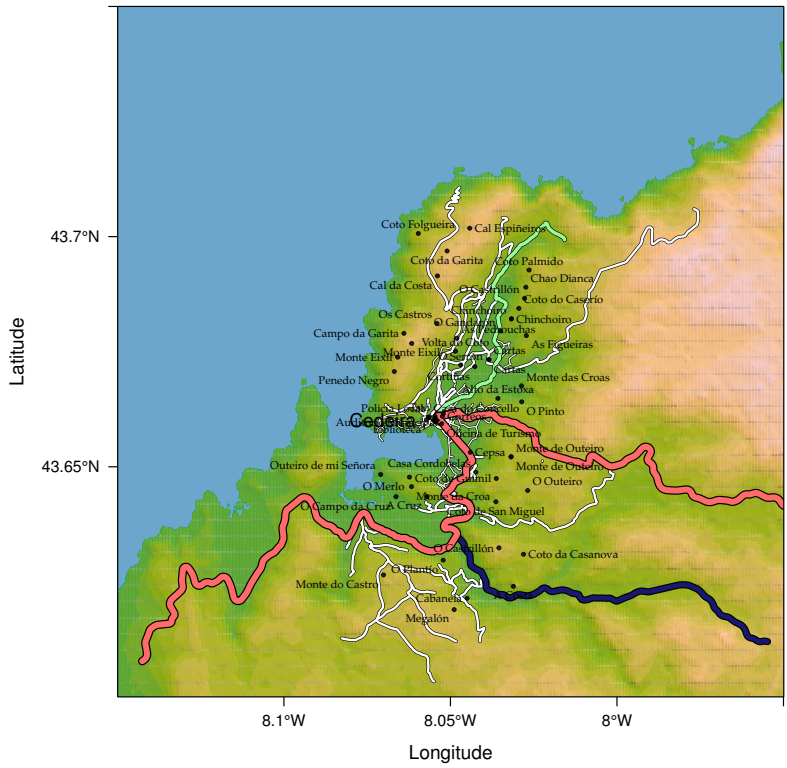

FIGURE 9.3: Main roads near Cedeira, Galicia. Local topography is displayed with the hill shading technique. Some places are highlighted.

```
  sp.road(secondary, lwd=4, blwd=6, col='midnightblue')
  sp.road(primary, col='indianred1')
  ## Places except Cedeira town
  sp.points(places, pch=19, col='black', cex=0.4, alpha=0.8)
  sp.pointLabel(places, labels=places$name,
                fontfamily = 'Palatino',
                cex=0.6, col='black')
  ## Cedeira town
  panel.points(cedeiraCoords, pch=18, col='black', cex=1)
  panel.text(cedeiraCoords, labels='Cedeira', pos=2, offset=1)
  })
```

Chapter 10

About the Data

10.1 Air Quality in Madrid

Air pollution is harmful to health and contributes to respiratory and cardiac diseases, and has a negative impact on natural ecosystems, agriculture, and the built environment. In Spain, the principal pollutants are particulate matter (PM), tropospheric ozone, nitrogen dioxide, and environmental noise[1].

The surveillance system of the Integrated Air Quality system of the Madrid City Council consists of twenty-four remote stations, equipped with analyzers for gases (NO_{X}, CO, ozone, BT_{X}, HCs, SO_{2}) and particles (PM10, PM2.5), which measure pollution in different areas of the urban environment. In addition, many of the stations also include sensors to provide meteorological data.

The detailed information of each measuring station can be retrieved from its own webpage defined by its station code.

```
## codeStations.csv is extracted from the document
## http://www.mambiente.munimadrid.es/opencms/export/sites/default/
    calaire/Anexos/INTPHORA-DIA.pdf,
## table of page 3.
```

[1] http://www.eea.europa.eu/soer/countries/es/

```
codEstaciones <- read.csv2('data/codeStations.csv')
codURL <- as.numeric(substr(codEstaciones$Codigo, 7, 8))

## The information of each measuring station is available at its own
     webpage, defined by codURL
URLs <- paste('http://www.mambiente.munimadrid.es/opencms/opencms/
    calaire/contenidos/estaciones/estacion', codURL, '.html', sep=''
    )
```

10.1.1 Data Arrangement

The station webpage includes several tables that can be extracted with the `readHTMLTable` function of the `XML` package. The longitude and latitude are included in the second table. The `ub2dms` function cleans this table and converts the strings to the `DMS` class defined by the `sp` package to represent degrees, minutes, and decimal seconds.

```
library(XML)
library(sp)

## Access each webpage, retrieve tables and extract long/lat data
coords <- lapply(URLs, function(est){
  tables <- readHTMLTable(est)
  location <- tables[[2]]
  ## Clean the table content and convert to dms format
  ub2dms <- function(x){
    ch <- as.character(x)
    ch <- sub(',', '.', ch)
    ch <- sub('O', 'W', ch) ## Some stations use "O" instead of "W"
    as.numeric(char2dms(ch, "º", "'", "'' "))
  }
  long <- ub2dms(location[2,1])
  lat <- ub2dms(location[2,2])
  alt <- as.numeric(sub(' m.', '', location[2, 3]))

  coords <- data.frame(long=long, lat=lat, alt=alt)

  coords
})

airStations <- cbind(codEstaciones, do.call(rbind, coords))
```

```
## The longitude of "El Pardo" station is wrong (positive instead of
     negative)
airStations$long[22] <- -airStations$long[22]

write.csv2(airStations, file='data/airStations.csv')
```

The 2011 air pollution data are available upon request from the Madrid City Council webpage[2] and at the `data` folder of the book repository. The structure of the file is documented in the INTPHORA-DIA document[3]. The `readLines` function reads the file and a `lapply` loop processes each line. The result is stored in the file `airQuality.csv`

```
## Fill in the form at
## http://www.mambiente.munimadrid.es/opencms/opencms/calaire/
    consulta/descarga.html
## to receive the Diarios11.zip file.
unzip('data/Diarios11.zip')
rawData <- readLines('data/Datos11.txt')
## This loop reads each line and extracts fields as defined by the
## INTPHORA file:
## http://www.mambiente.munimadrid.es/opencms/export/sites/default/
    calaire/Anexos/INTPHORA-DIA.pdf
datos11 <- lapply(rawData, function(x){
  codEst <- substr(x, 1, 8)
  codParam <- substr(x, 9, 10)
  codTec <- substr(x, 11, 12)
  codPeriod <- substr(x, 13, 14)
  month <- substr(x, 17, 18)
  dat <- substr(x, 19, nchar(x))
  ## "N" used for impossible days (31st April)
  idxN <- gregexpr('N', dat)[[1]]
  if (idxN==-1) idxN <- numeric(0)
  nZeroDays <- length(idxN)
  day <- seq(1, 31-nZeroDays)
  ## Substitute V and N with ";" to split data from different days
  dat <- gsub('[VN]+', ';', dat)
  dat <- as.numeric(strsplit(dat, ';')[[1]])
  ## Only data from valid days
  dat <- dat[day]
```

[2] http://www.mambiente.munimadrid.es/opencms/opencms/calaire/consulta/descarga.html

[3] http://www.mambiente.munimadrid.es/opencms/export/sites/default/calaire/Anexos/INTPHORA-DIA.pdf

```
 res <- data.frame(codEst, codParam, ##codTec, codPeriod,
                month, day, year=2011,
                dat)
 })
datos11 <- do.call(rbind, datos11)
write.csv2(datos11, 'data/airQuality.csv')
```

10.1.2 Combine Data and Spatial Locations

Our next step is to combine the data and spatial information. The locations are contained in `airStations`, a `data.frame` that is converted to an `SpatialPointsDataFrame` object with the `coordinates` method.

```
library(sp)

## Spatial location of stations
airStations <- read.csv2('data/airStations.csv')
coordinates(airStations) <- ~ long + lat
## Geographical projection
proj4string(airStations) <- CRS("+proj=longlat␣+ellps=WGS84␣+datum=
    WGS84")
```

On the other hand, the `airQuality` `data.frame` comprises the air quality daily measurements. We will retain only the NO_2 time series.

```
## Measurements data
airQuality <- read.csv2('data/airQuality.csv')
## Only interested in NO2
NO2 <- airQuality[airQuality$codParam==8, ]
```

We will represent each station using aggregated values (mean, median, and standard deviation) computed with `aggregate`:

```
NO2agg <- aggregate(dat ~ codEst, data=NO2,
                FUN = function(x) {
                    c(mean=signif(mean(x), 3),
                      median=median(x),
                      sd=signif(sd(x), 3))
                    })
NO2agg <- do.call(cbind, NO2agg)
NO2agg <- as.data.frame(NO2agg)
```

The aggregated values (a `data.frame`) and the spatial information (a `SpatialPointsDataFrame`) are combined with the `spCbind` method from

the maptools package to create a new SpatialPointsDataFrame. Previously, the data.frame is reordered by matching against the shared key column (airStations$Codigo and NO2agg$codEst):

```
library(maptools)
## Link aggregated data with stations to obtain a
    SpatialPointsDataFrame.
## Codigo and codEst are the stations codes
idxNO2 <- match(airStations$Codigo, NO2agg$codEst)
NO2sp <- spCbind(airStations[, c('Nombre', 'alt')], NO2agg[idxNO2,
    ])
save(NO2sp, file='data/NO2sp.RData')
```

10.2 Spanish General Elections

The results from the 2011 Spanish general elections[4] are available from the Ministry webpage[5] and at the data folder of the book repository. Each region of the map will represent the percentage of votes (pcMax) obtained by the predominant political option (whichMax) at the corresponding municipality. Only four groups are considered: the two main parties (PP and PSOE), the abstention results (ABS), and the remaining parties (OTH). Each region will be identified by the PROVMUN code.

```
dat2011 <- read.csv('data/GeneralSpanishElections2011.gz')

census <- dat2011$Total.censo.electoral
validVotes <- dat2011$Votos.válidos
## Election results per political party and municipality
votesData <- dat2011[, 12:1023]
## Abstention as an additional party
votesData$ABS <- census - validVotes
## Winner party at each municipality
whichMax <- apply(votesData, 1, function(x)names(votesData)[which.
    max(x)])
## Results of the winner party at each municipality
Max <- apply(votesData, 1, max)
## OTH for everything but PP, PSOE and ABS
whichMax[!(whichMax %in% c('PP', 'PSOE', 'ABS'))] <- 'OTH'
## Percentage of votes with the electoral census
pcMax <- Max/census * 100
```

[4] http://en.wikipedia.org/wiki/Spanish_general_election_2011
[5] http://www.infoelectoral.mir.es/docxl/04_201105_1.zip

```
## Province-Municipality code. sprintf formats a number with leading
     zeros.
PROVMUN <- with(dat2011, paste(sprintf('%02d', Código.de.Provincia),
                        sprintf('%03d', Código.de.Municipio),
                        sep=""))

votes2011 <- data.frame(PROVMUN, whichMax, Max, pcMax)
write.csv(votes2011, 'data/votes2011.csv', row.names=FALSE)
```

10.3 CM SAF

The Satellite Application Facility on Climate Monitoring (CM SAF) is a joint venture of the Royal Netherlands Meteorological Institute, the Swedish Meteorological and Hydrological Institute, the Royal Meteorological Institute of Belgium, the Finnish Meteorological Institute, the Deutscher Wetterdienst, Meteoswiss, and the UK MetOffice, along with collaboration of the European Organization for the Exploitation of Meteorological Satellites (EUMETSAT) (CM SAF 2013). The CM-SAF was funded in 1992 to generate and store monthly and daily averages of meteorological data measured in a continuous way with a spatial resolution of 0.03° (15 kilometers). The CM SAF provides two categories of data: operational products and climate data. The operational products are built on data that are validated with on-ground stations and then is provided in near-real-time to develop variability studies in diurnal and seasonal time scales. However, climate data are long-term data series to assess inter-annual variability (Posselt, Mueller, et al. 2012).

In this chapter we will display the annual average of the shortwave incoming solar radiation product (SIS) incident over Spain during 2008, computed from the monthly means of this variable. SIS collates shortwave radiation (0.2 to 4 µm wavelength range) reaching a horizontal unit Earth surface obtained by processing information from geostationary satellites (METEOSAT) and also from polar satellites (MetOp and NOAA) (Schulz et al. 2009) and then validated with high-quality on-ground measurements from the Baseline Surface Radiation Network (BSRN)[6].

The monthly means of SIS are available upon request from the CM SAF webpage (Posselt, Müller, et al. 2011) and at the `data` folder of the book repository. Data from CM-SAF is published as raster files. The `raster`

[6] http://www.bsrn.awi.de/en/home/

package provides the `stack` function to read a set of files and create a `RasterStack` object, where each layer stores the content of a file. Therefore, the twelve raster files of monthly averages produce a `RasterStack` with twelve layers.

```
library(raster)

tmp <- tempdir()
unzip('data/SISmm2008_CMSAF.zip', exdir=tmp)
filesCMSAF <- dir(tmp, pattern='SISmm')
SISmm <- stack(paste(tmp, filesCMSAF, sep='/'))
## CM-SAF data is average daily irradiance (W/m2). Multiply by 24
## hours to obtain daily irradiation (Wh/m2)
SISmm <- SISmm * 24
```

The `RasterLayer` object with annual averages is computed from the monthly means and stored using the native format of the `raster` package.

```
## Monthly irradiation: each month by the corresponding number of
    days
daysMonth <- c(31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)
SISm <- SISmm * daysMonth / 1000 ## kWh/m2
## Annual average
SISav <- sum(SISm)/sum(daysMonth)
writeRaster(SISav, file='SISav')
```

10.4 Land Cover and Population Rasters

The NASA's Earth Observing System (EOS)[7] is a coordinated series of polar-orbiting and low-inclination satellites for long-term global observations of the land surface, biosphere, solid Earth, atmosphere, and oceans. NEO-NASA[8], one of projects included in EOS, provides a repository of global data imagery. We use the population density and land cover classification rasters. Both rasters must be downloaded from their respective webpages as Geo-TIFF files.

```
library(raster)
## http://neo.sci.gsfc.nasa.gov/Search.html?group=64
pop <- raster('875430rgb-167772161.0.FLOAT.TIFF')
## http://neo.sci.gsfc.nasa.gov/Search.html?group=20
landClass <- raster('241243rgb-167772161.0.TIFF')
```

[7] http://eospso.gsfc.nasa.gov/

[8] http://neo.sci.gsfc.nasa.gov

Part III

Space-Time Data

Chapter 11

Displaying Spatiotemporal Data: Introduction

Space-time datasets are indexed both in space and in time. The data may consist of a spatial vector object (for example, points or polygons) or raster data at different times. The first case is representative of data from fixed sensors providing measurements abundant in time but sparse in space. The second case is the typical format of satellite imagery, which produces high spatial resolution data sparse in time (E. Pebesma 2012).

There are several visualization approaches of space-time data trying to cope with the four dimensions of the data (Cressie and Wikle 2011).

On the one hand, the data can be conceived as a collection of snapshots at different times. These snapshots can be displayed as a sequence of frames to produce an animation, or can be printed on one page with different panels for each snapshot using the small-multiple technique described repeatedly in previous chapters.

On the other hand, one of the two spatial dimensions can be collapsed through an appropriate statistic (for example, mean or standard deviation) to produce a space-time plot (also known as a Hovmöller diagram). The axes of this graphic are typically longitude or latitude as the x-axis, and time as the y-axis, with the value of the spatial-averaged value of the raster data represented with color.

Finally, the space-time object can be reduced to a multivariate time series (where each location is a variable or column of the time series) and displayed with the time series visualization techniques described in the Part **??**. This approach is directly applicable to space-time data sparse in space (for example, point measurements at different times). However, it is mandatory to use aggregation in the case of raster data. In this case, the multivariate time series is composed of the evolution of the raster data averaged along a certain direction.

The next chapters, focused on raster space-time data (Chapter 12) and point space-time data (Chapter 13), illustrate with examples how to produce animations, multipanel graphics, hovmöller diagrams, and time-series with R.

11.1 Packages

The CRAN Tasks View "Handling and Analyzing Spatiotemporal Data" [1] summarizes the packages for reading, vizualizing, and analyzing space-time data. This section provides a brief introduction to the `spacetime`, `raster`, and `rasterVis` packages. Most of the information has been extracted from their vignettes, webpages, and help pages. You should read them for detailed information.

11.1.1 spacetime

The `spacetime` package (E. Pebesma 2012) is built upon the classes and methods for spatial data from the `sp` package , and for time series data from the `xts` package. It defines classes to represent four space-time layouts:

1. `STF`, `STFDF`: full space-time grid of observations for spatial features and observation time, with all space-time combinations.
2. `STS`, `STSDF`: sparse grid layout, stores only the non-missing space-time combinations on a lattice
3. `STI`, `STIDF`: irregular layout, time and space points of measured values have no apparent organisation.
4. `STT`, `STTDF`: simple trajectories.

[1] http://cran.r-project.org/web/views/SpatioTemporal.html

Moreover, spacetime provides several methods for the following classes:

- stConstruct, STFDF, and STIDF create objects from single or multiple tables.
- as coerces to other spatiotemporal objects, xts, Spatial, matrix, or data.frame.
- [[selects or replaces data values.
- [selects spatial or temporal subsets, and data variables.
- over retrieves index or data values of one object at the locations and times of another.
- aggregate aggregates data values over particular spatial, temporal, or spatiotemporal domains.
- stplot creates spatiotemporal plots. It is able to produce multipanel plots, space-time plots, animations, and time series plots.

11.1.2 raster

The raster package (R. J. Hijmans 2013) is able to add time information associated with layers of a RasterStack or RasterBrick object with the setZ function. This information can be extracted with getZ.

If a Raster* object includes this information, the zApply function can be used to apply a function over a time series of layers of the object.

11.1.3 rasterVis

rasterVis (Perpiñán and R. Hijmans 2013) provides three methods to display spatiotemporal rasters:

1. hovmoller produces Hovmöller diagrams (Hovmöller 1949). The axes of this kind of diagram are typically longitude or latitude (x-axis) and time (ordinate or y-axis) with the value of some aggregated field represented through color. However, the user can define the direction with dirXY and the summary function with FUN.
2. horizonplot creates horizon graphs (Few 2008), with many time series displayed in parallel by cutting the vertical range into segments and overplotting them with color representing the magnitude and direction of deviation. Each time series corresponds to a geographical zone defined with dirXY and averaged with zonal.

3. `xyplot` displays conventional time series plots. Each time series corresponds to a geographical zone defined with `dirXY` and aggregated with `zonal`.

On the other hand, the `histogram`, `densityplot`, and `bwplot` methods accept a `FUN` argument to be applied to the `z` slot of `Raster*` object (defined by `setZ`). The result of this function is used as the grouping variable of the plot to create different panels.

11.2 Further Reading

- (Cressie and Wikle 2011) is a systematic approach to key quantitative techniques on statistics for spatiotemporal data. The book begins with separate treatments of temporal data and spatial data, and later combines these concepts to discuss spatiotemporal statistical methods. There is a chapter devoted to exploratory methods, including visualization techniques.
- (E. Pebesma 2012) presents the `spacetime` package, which implements a set of classes for spatiotemporal data. This paper includes examples that illustrate how to import, subset, coerce, and export spatiotemporal data, proposes several visualization methods, and discusses spatiotemporal geostatistical interpolation.
- (Slocum 2005) (previously cited in Chapter 7.2) includes a chapter about map animation, discussing several approaches for displaying spatiotemporal data.
- (Hengl 2009) (previously cited in Chapter 7.2) includes a working example with spatiotemporal data to illustrate space-time variograms and interpolation.
- (Harrower and Fabrikant 2008) explore the role of animation in geographic visualization and outline the challenges, both conceptual and technical, involved in the creation and use of animated maps.
- The CRAN Tasks View "Handling and Analyzing Spatiotemporal Data" [2] summarizes the packages for reading, vizualizing, and analyzing space-time data. The R-SIG-Geo mailing list [3] is a powerful resource for obtaining help.

[2] http://cran.r-project.org/web/views/SpatioTemporal.html
[3] https://stat.ethz.ch/mailman/listinfo/R-SIG-Geo/

Chapter 12

Spatiotemporal Raster Data

12.1 Introduction

A space-time raster dataset is a collection of raster layers indexed by time, or in other words, a time series of raster maps. The `raster` package defines the classes `RasterStack` and `RasterBrick` to build multilayer rasters. The index of the collection can be set with the function `setZ` (which is not restricted to time indexes). The `rasterVis` packages provide several methods to display space-time rasters.

12.1.1 Data

Throughout this chapter we will work with a multilayer raster of daily solar radiation estimates from CM SAF (section 10.3) falling in the region of Galicia (north of Spain) during 2011. These data are arranged in a `RasterBrick` with 365 layers using `brick` and time indexed with `setZ`.

```
library(raster)
library(zoo)
library(rasterVis)

SISdm <- brick('data/SISgal')

timeIndex <- seq(as.Date('2011-01-01'), by='day', length=365)
```

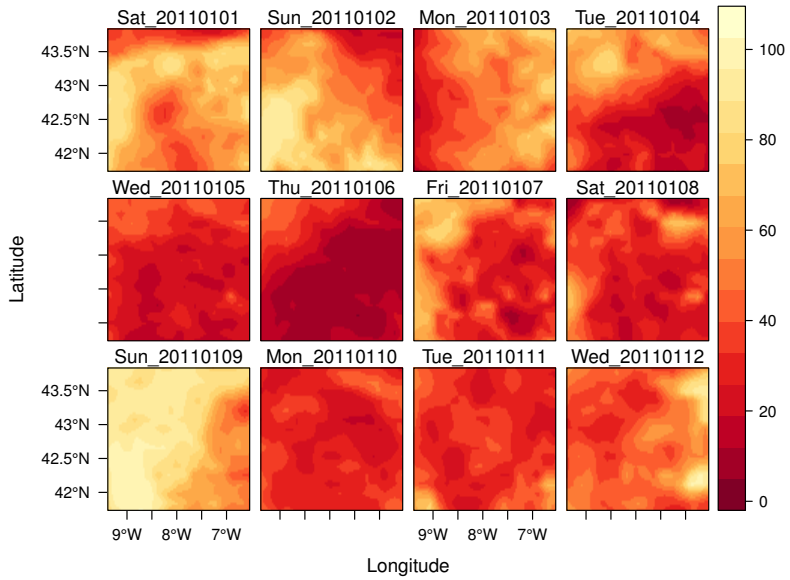

FIGURE 12.1: Level plot of daily averages of solar radiation.

```
SISdm <- setZ(SISdm, timeIndex)
names(SISdm) <- format(timeIndex, '%a_%Y%m%d')
```

12.2 Level Plots

This multilayer raster can be displayed with each snapshot in a panel using the small-multiple technique. The problem with this approach is that only a limited number of panels can be correctly displayed on one page. In this example, we print the first 12 days of the sequence (Figure 12.1).

```
levelplot(SISdm, layers=1:12, panel=panel.levelplot.raster)
```

When the number of layers is very high, a partial solution is to aggregate the data, grouping the layers according to a time condition. For example, we can build a new space-time raster with the monthly averages using `zApply` and `as.yearmon`. This raster can be completely displayed on one page (Figure 12.2), although part of the information of the original data is lost in the aggregation procedure.

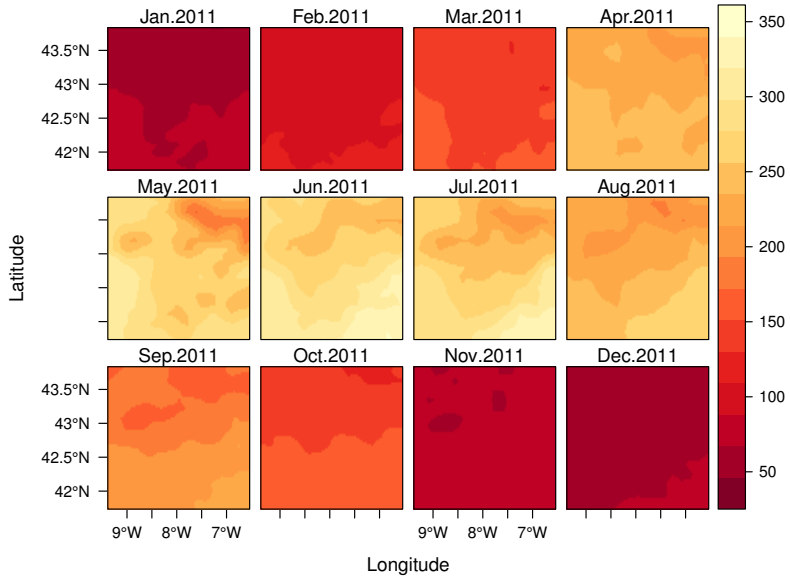

FIGURE 12.2: Level plot of monthly averages of solar radiation.

```
SISmm <- zApply(SISdm, by=as.yearmon, fun='mean')
```

```
levelplot(SISmm, panel=panel.levelplot.raster)
```

12.3 Graphical Exploratory Data Analysis

There are other graphical tools that complement the previous maps. The scatterplot and the matrix of scatterplots, the histogram and kernel density plot, and the boxplot are among the most important tools in the frame of the Exploratory Data Analysis approach. Some of them were previously used with a spatial raster (Chapter 8.3). In this section we will use the histogram (Figure 12.3), the violin plot (a combination of a boxplot and a kernel density plot) (Figure 12.4), and the matrix of scatterplots (section **??**, Figure 12.5).

```
histogram(SISdm, FUN=as.yearmon)
```

```
bwplot(SISdm, FUN=as.yearmon)
```

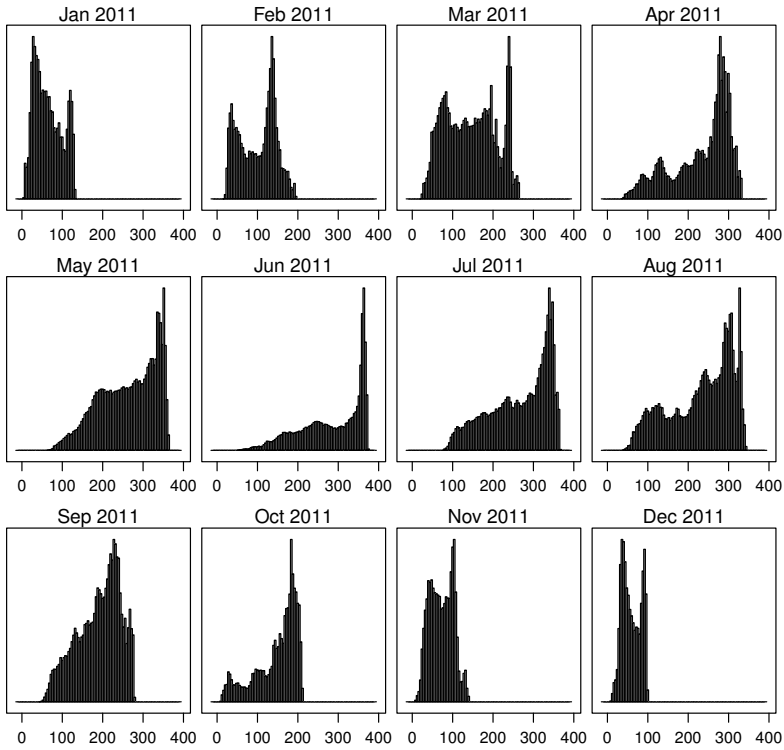

FIGURE 12.3: Histogram of monthly distribution of solar radiation.

```
splom(SISmm, xlab='', plot.loess=TRUE)
```

Both the histogram and the violin plot show that daily solar irradiation is bimodal almost every month. This is related to the predominance of clear sky and overcast days, with several partly cloudy days between these modes. This geographical region receives higher irradiation levels from June to September, and both the levels and the shape of the probability distribution contrast sharply with the winter.

The matrix of scatterplots displays a quasilinear relationship between the central months due to the predominance of clear sky conditions. However, the relationships involving winter months become strongly nonlinear due to the presence of clouds.

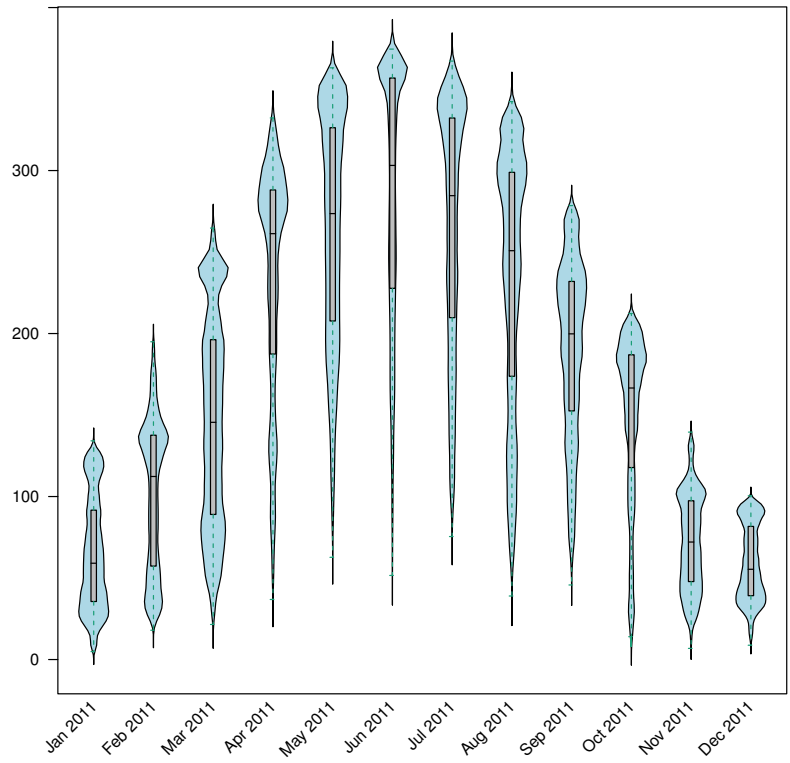

FIGURE 12.4: Violin plot of monthly distribution of solar radiation.

12.4 Space-Time and Time Series Plots

The level plots of Figures 12.1 and 12.2 display the full 3D space-time with a grid of panels where each layer is printed. In other words, the raster is sliced, and the collection of pieces is shown in a table. In the section 12.5, this collection of layers will be displayed sequentially like frames of a movie to build an animation. In this section, the 3D raster is reduced to a 2D matrix with spatial aggregation following a certain direction. For example, Figure 12.6 displays with colors the averaged value of the raster for each latitude zone (using the default value of the argument `dirXY`) with time on the vertical axis.

```
hovmoller(SISdm, par.settings=BTCTheme())
```

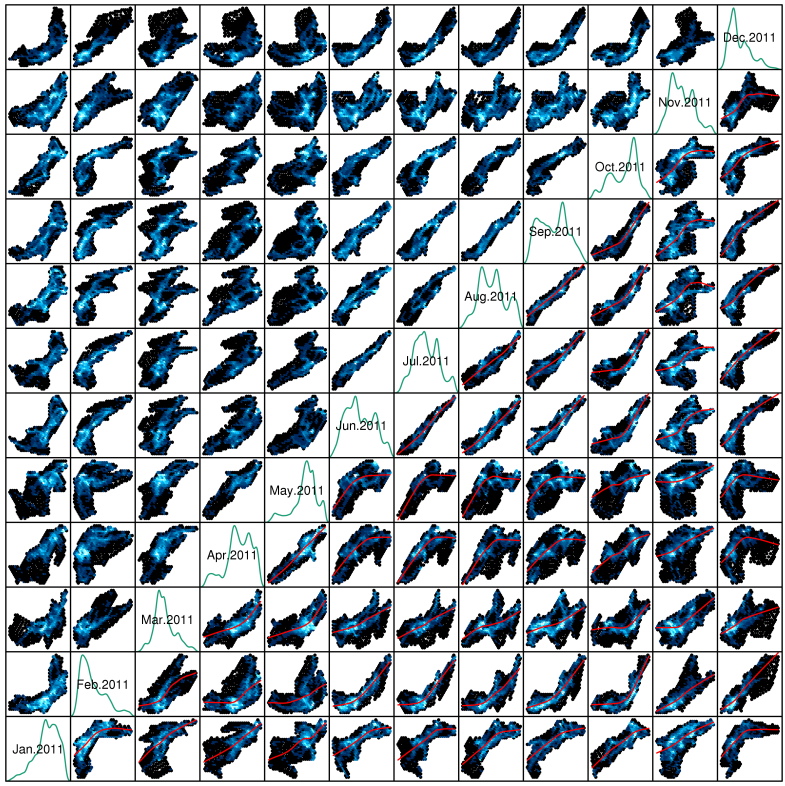

FIGURE 12.5: Scatterplot matrix of monthly averages together with their kernel density estimations in the diagonal frames.

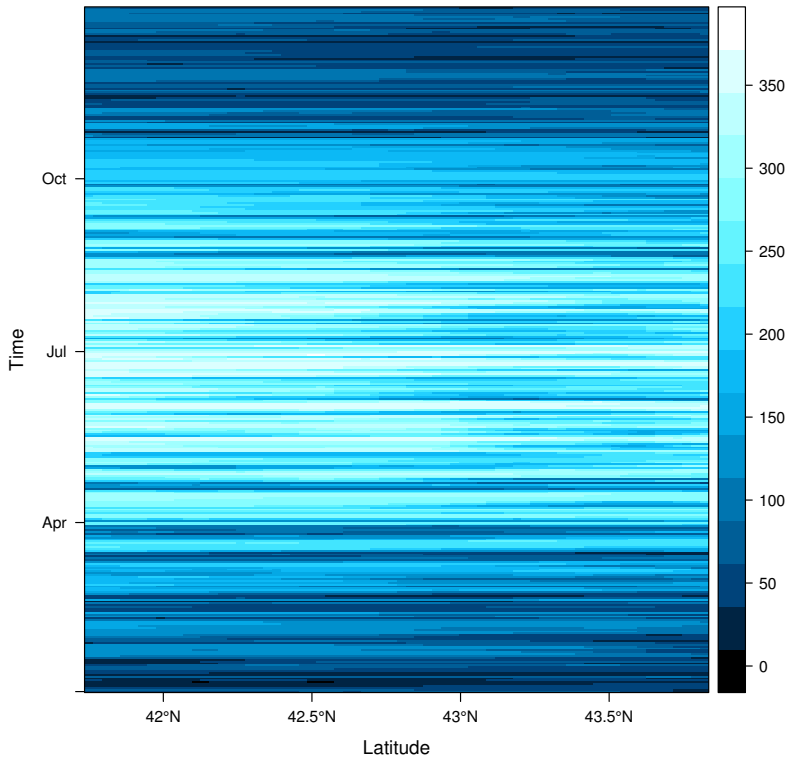

FIGURE 12.6: Hovmöller graphic displaying the time evolution of the average solar radiation for each latitude zone.

On the other hand, this 2D matrix can be conceived as a multivariate time series with each aggregated zone conforming to a different variable of the time series. This approach is followed by the `xyplot` (Figure 12.7) and `horizonplot` (Figure 12.8) methods, which reproduce the procedures described in Chapter 3 to display multivariate time series.

```
xyplot(SISdm, digits=1, col='black', lwd=0.2, alpha=0.6)
```

```
horizonplot(SISdm, digits=1,
          col.regions=rev(brewer.pal(n=6, 'PuOr')),
          xlab='', ylab='Latitude')
```

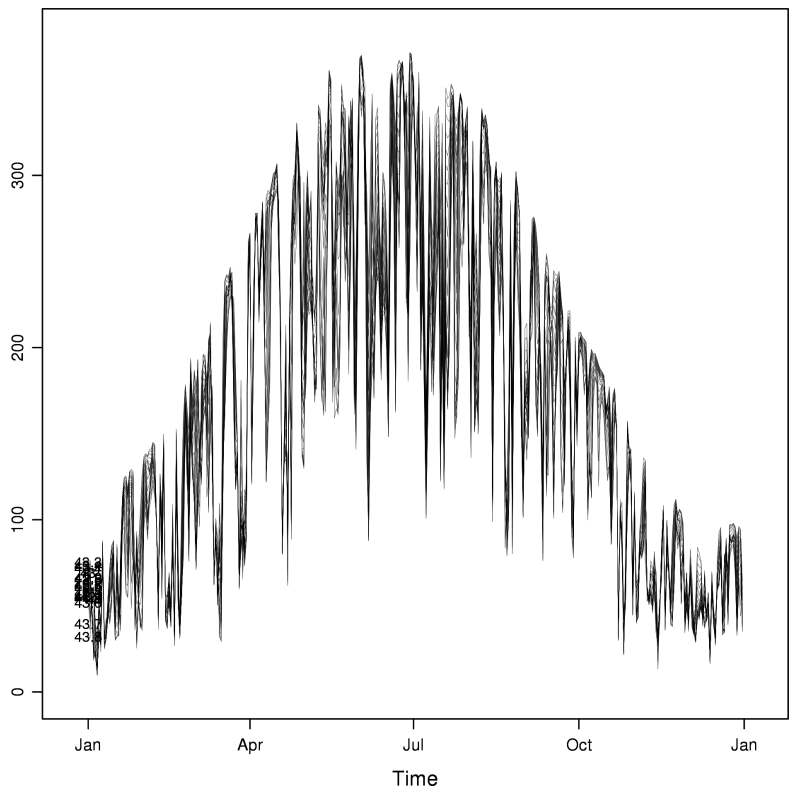

FIGURE 12.7: Time graph of the average solar radiation for each latitude zone. Each line represents a latitude band.

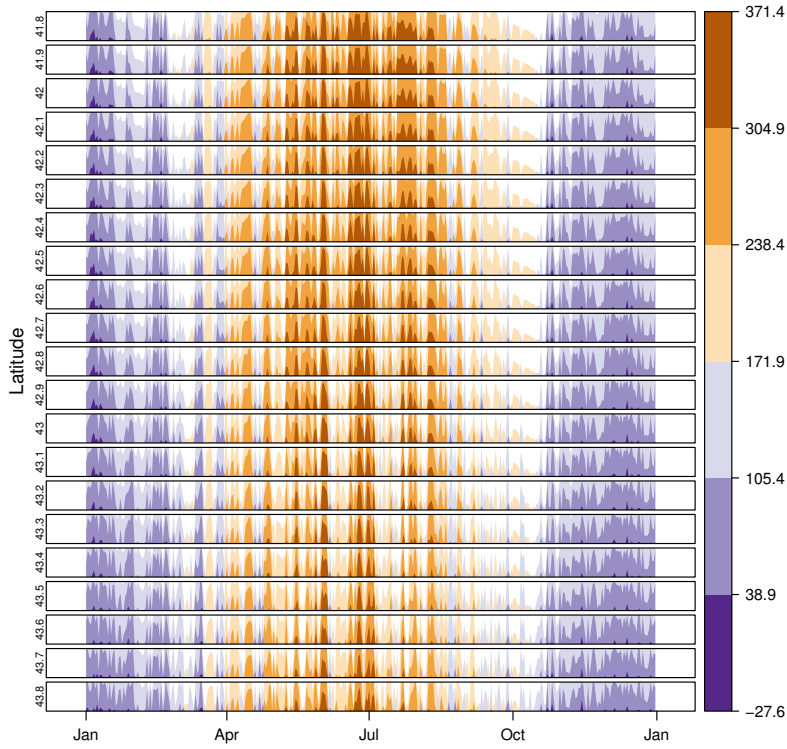

FIGURE 12.8: Horizon graph of the average solar radiation for each latitude zone.

These three figures highlight the stational behavior of the solar radiation, with higher values during the central months. It is interesting to note that (Figure 12.8) the radiation values around the equinoxes fluctuate near the yearly average value of each latitude region.

12.5 Animation

A different approach is to plot the individual layers of the space-time raster sequentially as movie frames to produce an animation. The procedure is quite simple:

- Plot each layer of the raster to produce a collection of graphic files.

- Join these files as a sequence of frames with a suitable tool (for example, `ffmpeg`) to create a movie file[1, 2].

The effectiveness of this visualization procedure is partly related to the similitude between consecutive frames. If the frames of the sequence diverge excessively from one to another, the user will experience difficulties to perceive any relationship between them. On the other hand, if the transitions between layers are smooth enough, the frames will be perceived as conforming to a whole story; and, moreover, the user will be able to spot both the stable patterns and the important variations.

12.5.1 Data

The daily solar radiation CM-SAF data do not meet the condition of a smooth transition between layers. The changes between the consecutive snapshots of daily radiation are too abrupt to be glued one after another. We will work with a different dataset in this section.

The THREDSS server[3] of Meteogalicia[4] provides access through different protocols to the output of a Weather Research and Forecasting (WRF) model, a mesoscale numerical weather prediction system. Among the set of available variables we will use the forecast of hourly cloud cover at low and mid levels. This space-time raster has a time horizon of 96 hours and a spatial resolution of 12 kilometers.

```
cft <- brick('data/cft_20130417_0000.nc')
## use memory instead of file
cft[] <- getValues(cft)
## set projection
projLCC2d <- "+proj=lcc␣+lon_0=-14.1␣+lat_0=34.823␣+lat_1=43␣+lat_
    2=43␣+x_0=536402.3␣+y_0=-18558.61␣+units=km␣+ellps=WGS84"
projection(cft) <- projLCC2d
#set time index
timeIndex <- seq(as.POSIXct('2013-04-17␣01:00:00', tz='UTC'), length
    =96, by='hour')
cft <- setZ(cft, timeIndex)
names(cft) <- format(timeIndex, 'D%d_H%H')
```

[1] The `animation` package (Xie 2013) defines several functions to wrap `ffmpeg` and `convert` from ImageMagick.

[2] An alternative method is the LaTeX animate package, which provides an interface to create portable JavaScript-driven PDF animations from rasterized image files.

[3] http://mandeo.meteogalicia.es/thredds/catalogos/WRF_2D/catalog.html

[4] http://www.meteogalicia.es

12.5.2 Spatial Context: Administrative Boundaries

Let's provide the spatial context with the countries boundaries, extracted from the `worldHires` database of the `maps` and `mapdata` packages.

```
library(maptools)
library(rgdal)
library(maps)
library(mapdata)


projLL <- CRS('+proj=longlat␣+datum=WGS84␣+ellps=WGS84␣+towgs84
    =0,0,0')
cftLL <- projectExtent(cft, projLL)
cftExt <- as.vector(bbox(cftLL))
boundaries <- map('worldHires',
              xlim=cftExt[c(1,3)], ylim=cftExt[c(2,4)],
              plot=FALSE)
boundaries <- map2SpatialLines(boundaries, proj4string=projLL)
boundaries <- spTransform(boundaries, CRS(projLCC2d))
```

12.5.3 Producing the Frames and the Movie

The next step is to produce the collection of frames. We will create a file with each layer of the `RasterBrick` using the `levelplot` function. This function provides the argument `layout` to control the arrangement of a multipanel display. If it is set to `c(1,1)`, a different page is created for each layer.

```
cloudTheme <- rasterTheme(region=brewer.pal(n=9, 'Blues'))
```

```
tmp <- tempdir()
trellis.device(png, file=paste0(tmp, '/Rplot%02d.png'),
                  res=300, width=1500, height=1500)
levelplot(cft, layout=c(1, 1), par.settings=cloudTheme) +
   layer(sp.lines(boundaries, lwd=0.6))
dev.off()
```

A suitable tool to concatenate these frames and create the movie is `ffmpeg`, a free cross-platform software to record, convert, and stream audio and video[5]. The resulting movie is available from the book website.

[5] http://www.ffmpeg.org/

```
old <- setwd(tmp)
## Create a movie with ffmpeg using 6 frames per second a bitrate of
     300kbs
movieCMD <- 'ffmpeg -r 6 -b 300k -i Rplot%02d.png output.mp4'
system(movieCMD)
file.remove(dir(pattern='Rplot'))
file.copy('output.mp4', paste0(old, '/figs/cft.mp4'), overwrite=TRUE
    )
setwd(old)
```

12.5.4 Static Image

Figure 12.9 shows a sequence of twenty-four snapshots (second day of the forecast series) of the movie. This graphic is also created with `levelplot` but now using the argument `layers` to choose a subset of the layers, and with a different value for `layout` to display a matrix of twenty-four panels.

```
levelplot(cft, layers=25:48, layout=c(6, 4),
        par.settings=cloudTheme,
        names.attr=paste0(sprintf('%02d', 1:24), 'h'),
        panel=panel.levelplot.raster) +
   layer(sp.lines(boundaries, lwd=0.6))
```

The movie and the static image are complementary tools and should be used together. Watching the movie you will perceive the cloud transit from Galicia to the Pyrenees gradually dissolving over the Cantabrian region. On the other hand, with Figure 12.9 you can locate the position of a group of clouds in a certain hour and simultaneously observe the relationship of that position with the evolution during that period. With the movie you will concentrate your attention on the movement. With small multiple pictures, your focus will be on positions and relations. You should use both graphical tools to grasp the entire 3D dataset.

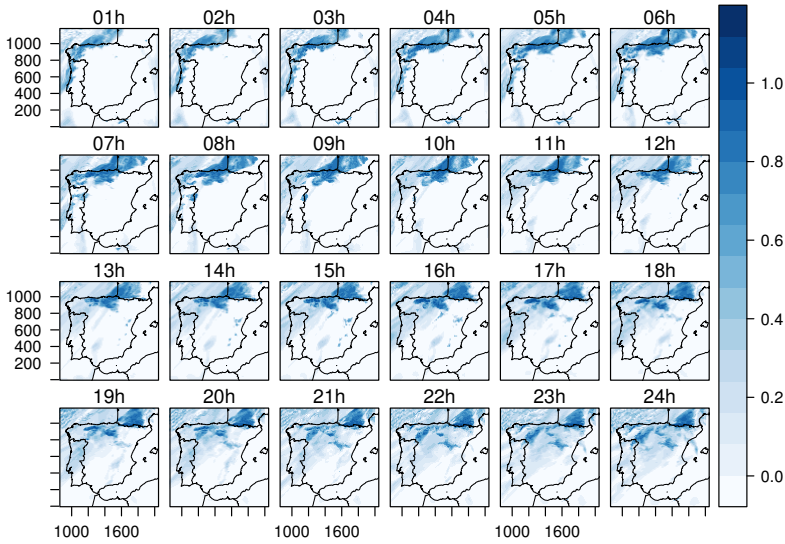

FIGURE 12.9: Forecast of hourly cloud cover at low and mid levels.

Chapter 13

Spatiotemporal Point Observations

13.1 Introduction

Throughout this chapter we will revisit the data from the Integrated Air Quality system of the Madrid City Council (section 10.1) to illustrate visualization methods applicable for point space-time data. This dataset comprises the time series of measurements acquired at each station of the network during 2011. In the section 8.1 the data were converted from spatiotemporal data to spatial data, where the time information was suppressed to display only the yearly average values. In this chapter we will work with the whole space-time dataset using the tools provided by the `spacetime` package (E. Pebesma 2012).

13.2 Data and Spatial Information

The starting point is to retrieve the data and combine it with the spatial and temporal information. The data are contained in the `airQuality` `data.frame`, and the locations are in `airStations`, a `data.frame` that is converted to a `SpatialPointsDataFrame` object with the `coordinates` method.

```
library(sp)

## Spatial location of stations
airStations <- read.csv2('data/airStations.csv')
## rownames are used as the ID of the Spatial object
rownames(airStations) <- substring(airStations$Codigo, 7)
coordinates(airStations) <- ~ long + lat
proj4string(airStations) <- CRS("+proj=longlat␣+ellps=WGS84")
## Measurements data
airQuality <- read.csv2('data/airQuality.csv')
## Only interested in NO2
NO2 <- airQuality[airQuality$codParam==8, ]
```

Each row of this `data.frame` corresponds to a measurement at one of the stations during a day of the year (long format, following the schema proposed in (E. Pebesma 2012)).

The `spacetime` package defines several classes for spatiotemporal data inheriting the classes defined by the `sp` and `xts` packages. In particular, the `STFDF`, a class for spatiotemporal data with full space-time grids with `n` spatial locations and `m` times, requires a `data.frame` with $n \cdot m$ rows, (spatial index moving fastest). Thus, we need to transform this structure to build a multivariate time series where each station is a different variable (space-wide under the schema of (E. Pebesma 2012)). The procedure is

- Add a column with the `POSIXct` time index.
- Reshape the `data.frame` from long to wide format with `reshape`.
- Define a multivariate time series with `zoo` (Figure 13.3).
- Coerce this time series to a vector with $n \cdot m$ rows.

```
library(zoo)
library(spacetime)

NO2$time <- with(NO2, ISOdate(year, month, day))
NO2wide <- reshape(NO2[,c('codEst', 'dat', 'time')],
                idvar='time', timevar='codEst',
                direction='wide')
NO2zoo <- zoo(NO2wide[,-1], NO2wide$time)

dats <- data.frame(vals=as.vector(t(NO2zoo)))
NO2st <- STFDF(airStations, index(NO2zoo), dats)
```

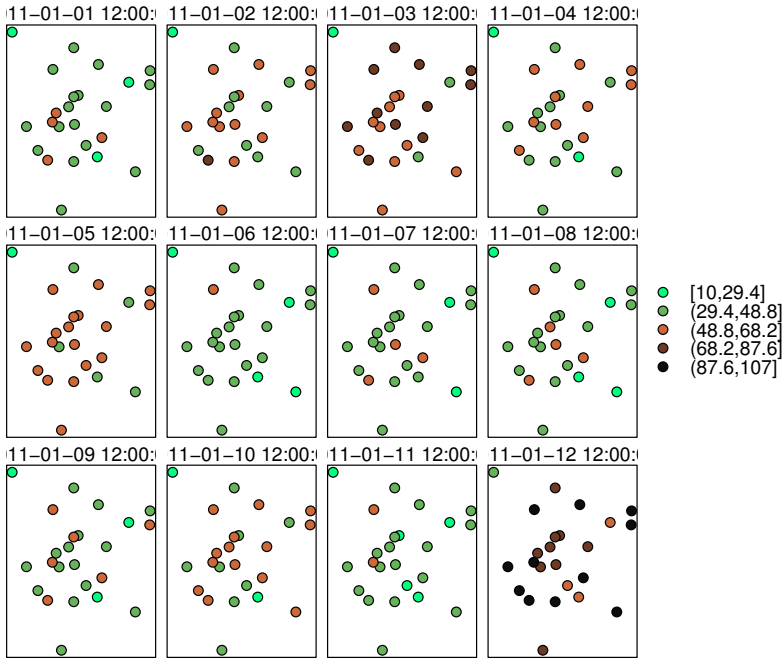

FIGURE 13.1: Scatterplots of the NO_2 values (2011) with a panel for each day of the time series. Each circle represents a different station.

13.3 Graphics with `spacetime`

The `stplot` function of the `spacetime` package supplies the main visualization methods for spatiotemporal data. When the mode `xy` is chosen (default) it is mainly a wrapper around `spplot` and displays a panel with the spatial data for each element of the time index (Figure 13.1). The problem with this approach is that only a limited number of panels can be correctly displayed on one page. In this example, we print the first 12 days of the sequence.

```
airPal <- colorRampPalette(c('springgreen1', 'sienna3', 'gray5'))(5)

stplot(NO2st[, 1:12], cuts=5, col.regions=airPal, edge.col='black')
```

With the mode `xt`, a space-time plot with space on the x-axis and time on the y-axis is plotted (Figure 13.2).

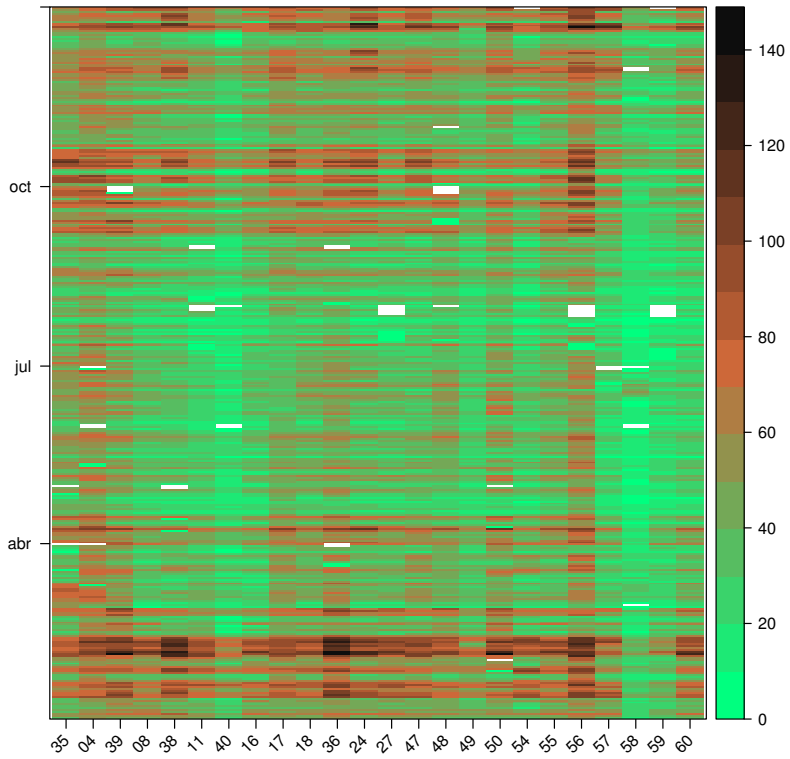

FIGURE 13.2: Space-time graphic of the NO_2 time series. Each column represents a different station (denoted with the last two digits of the code).

```
stplot(NO2st, mode='xt', col.regions=colorRampPalette(airPal)(15),
      scales=list(x=list(rot=45)), ylab='', xlab='')
```

Finally, with the mode `ts`, data are coerced to a multivariate time series that is displayed in a single plot (Figure 13.3).

```
stplot(NO2st, mode='ts', xlab='',
      lwd=0.1, col='black', alpha=0.6,
      auto.key=FALSE)
```

These three graphics complement each other and together provide a more complete view of the behavior of the data. For example in Figure 13.1, we can find stations whose levels remain almost constant throughout

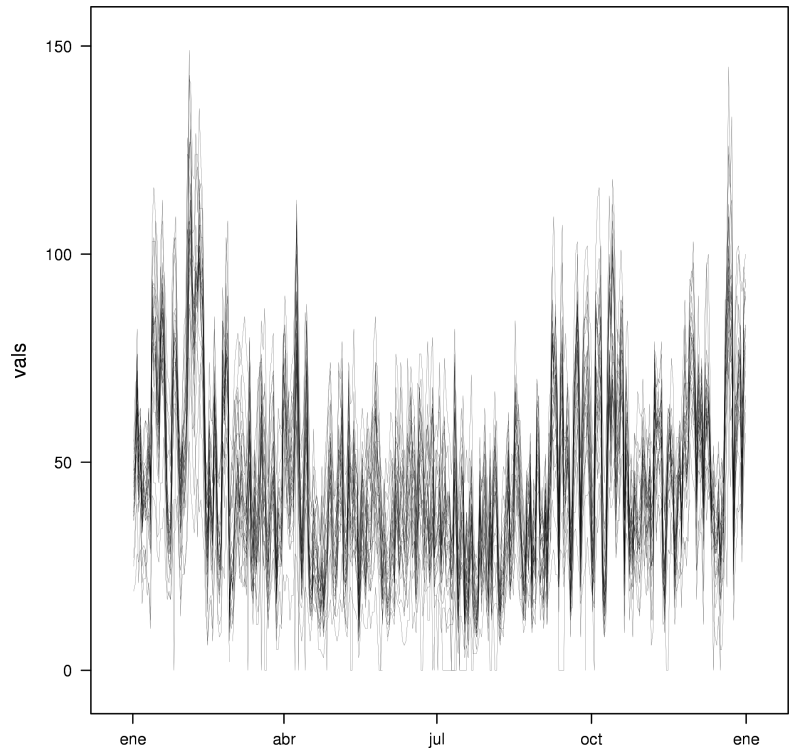

FIGURE 13.3: Time graph of the NO_2 time series (2011). Each line represents a different station.

the 12-day period (namely, El Pardo-28079058[1], the station at the top-left corner that is far from the city center), while others fluctuate notably during this same period (for example, Barajas-28079027 and Urb. Embajada-28079055, the two nearby stations at the right). On the other hand, Figure 13.2 loses the spatial information but gives a more comprehensive view of the evolution of the network. The station El Pardo-28079058 is significantly below the rest of the stations during the whole year, with the station Pza. Fdez Ladreda-28079056 being the opposite. In between, the stations could be divided into two or three groups according to their levels. Regardless, the group of stations reaches maximum values during the first

[1]Use Figure 8.5 as reference of the positions and codes of the stations.

days of autumn and at the end of winter. These maxima are clearly displayed in Figure 13.3.

13.4 Animation

Another approach for displaying this spatiotemporal data is using animation. Once again, we will take advantage of the functionalities of the `gridSVG` package.

13.4.1 Initial Snapshot

The first step is to define the initial parameters of the animation: starting values and duration.

```
library(gridSVG)
## Initial parameters
start <- NO2st[,1]
## values will be encoded as size of circles,
## so we need to scale them
startVals <- start$vals/5000

nStations <- nrow(airStations)
days <- index(NO2zoo)
nDays <- length(days)
## Duration in seconds of the animation
duration <- nDays*.3
```

The first snapshot of the data is produced with `spplot`. We define an auxiliary function, `panel.circlesplot`, to display the data encoding values with circles of variable size and color. This function uses `grid.circle` from the `grid` package.

The subsequent frames of the animation will modify the colors and sizes of the circles according to the `NO2st` object.

```
library(grid)

## Auxiliary panel function to display circles
panel.circlesplot <- function(x, y, cex, col='gray',
                        name='stationsCircles', ...){
grid.circle(x, y, r=cex,
          gp=gpar(fill=col, alpha=0.5),
          default.units='native', name=name)
}
```

```
pStart <- spplot(start, panel=panel.circlesplot,
             cex=startVals,
             scales=list(draw=TRUE), auto.key=FALSE)
pStart
```

13.4.2 Intermediate States to Create the Animation

From this initial state, `grid.animate` creates a collection of animated graphical objects with the intermediate states defined by `animUnit` and `animValue`. As previously stated, the NO_2 values will be encoded with the radius of each circle, and the color of the circles will distinguish between weekdays and weekend. The use of `rep=TRUE` ensures that the animation will be repeated indefinitely.

```
## Color to distinguish between weekdays ('green')
## and weekend ('blue')
isWeekend <- function(x) {format(x, '%w') %in% c(0, 6)}
color <- ifelse(isWeekend(days), 'blue', 'green')
colorAnim <- animValue(rep(color, each=nStations),
                  id=rep(seq_len(nStations), nDays))

## Intermediate sizes of the circles
vals <- NO2st$vals/5000
vals[is.na(vals)] <- 0
radius <- animUnit(unit(vals, 'native'),
                  id=rep(seq_len(nStations), nDays))

## Animation of circles including sizes and colors
grid.animate('stationsCircles',
          duration=duration,
          r=radius,
          fill=colorAnim,
          rep=TRUE)
```

13.4.3 Time Reference: Progress Bar

Information from an animation is better understood if a time reference is included, for example with a progress bar. The following code builds a progress bar with ticks at the first day of each month, and with color changing from gray (background) to blue as the time advances. On the other hand, it is convenient to provide a method so the user can stop and

restart the animation sequence if desired. This functionality is added with the definition of two events, `onmouseover` and `onmouseout`, included with the `grid.garnish` function.

```
## Progress bar
prettyDays <- pretty(days, 12)
## Width of the progress bar
pbWidth <- .95
## Background
grid.rect(.5, 0.01, width=pbWidth, height=.01,
        just=c('center', 'bottom'),
        name='bgbar', gp=gpar(fill='gray'))

## Width of the progress bar for each day
dayWidth <- pbWidth/nDays
ticks <- c(0, cumsum(as.numeric(diff(prettyDays)))*dayWidth) + .025
grid.segments(ticks, .01, ticks, .02)
grid.text(format(prettyDays, '%d-%b'),
             ticks, .03, gp=gpar(cex=.5))
## Initial display of the progress bar
grid.rect(.025, .01, width=0,
        height=.01, just=c('left', 'bottom'),
        name='pbar', gp=gpar(fill='blue', alpha='.3'))
## ...and its animation
grid.animate('pbar', duration=duration,
          width=seq(0, pbWidth, length=duration),
          rep=TRUE)
## Pause animations when mouse is over the progress bar
grid.garnish('bgbar',
          onmouseover='document.rootElement.pauseAnimations()',
          onmouseout='document.rootElement.unpauseAnimations()')
```

The SVG file is finally produced with `gridToSVG` (Figure 13.4)

```
grid.export('figs/NO2pb.svg')
```

13.4.4 Time Reference: A Time Series Plot

A different and more informative solution is to add a time series plot instead of a progress bar. This time series plot displays the average value of the set of stations, with a point and a vertical line to highlight the time position as the animation advances (Figure 13.5).

```
## Time series with average value of the set of stations
```

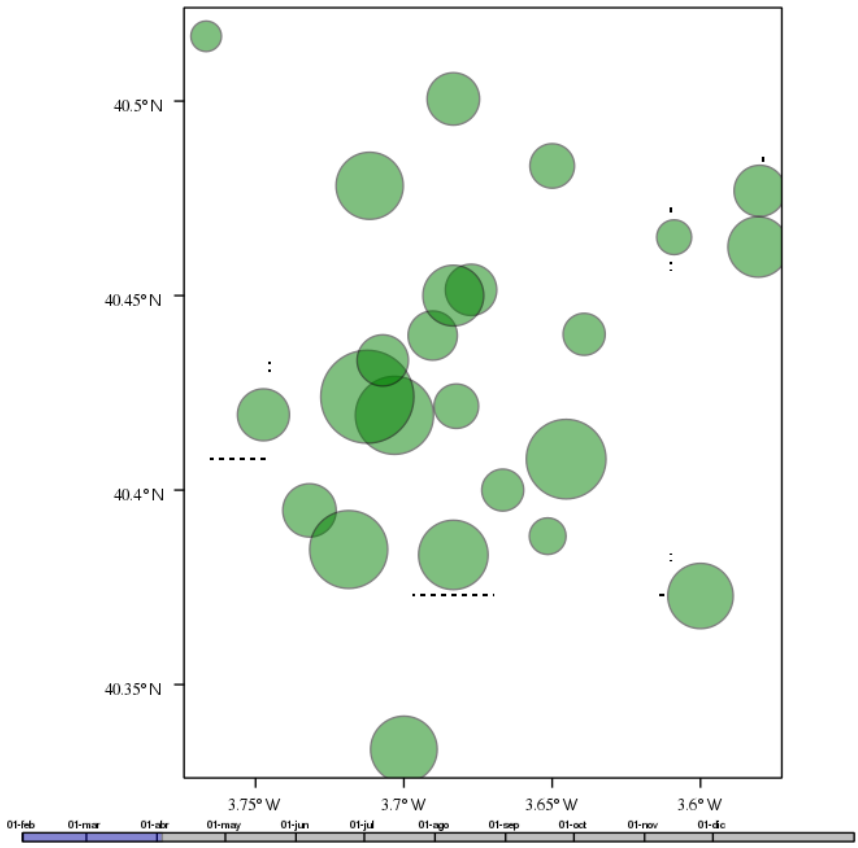

FIGURE 13.4: Animated circles of the NO_2 space-time data with a progress bar.

```
NO2mean <- zoo(rowMeans(NO2zoo, na.rm=TRUE), index(NO2zoo))
## Time series plot with position highlighted
pTimeSeries <- xyplot(NO2mean, xlab='', identifier='timePlot') +
   layer({
      grid.points(0, .5, size=unit(.5, 'char'),
                  default.units='npc',
                  gp=gpar(fill='gray'),
                  name='locator')
      grid.segments(0, 0, 0, 1, name='vLine')
   })

print(pStart, position=c(0, .2, 1, 1), more=TRUE)
print(pTimeSeries, position=c(.1, 0, .9, .25))
```

Once again, `grid.animate` creates a sequence of intermediate states for each object of the graphical scenes: The signaling point and vertical line follow the time evolution, while the sizes and colors of each station circle change as in the previous approach. Moreover, the `onmouseover` and `onmouseout` events are defined with `grid.garnish` so the user can pause and restart the animation by hovering the mouse over the time series plot.

```
grid.animate('locator',
          x=unit(as.numeric(index(NO2zoo)), 'native'),
          y=unit(as.numeric(NO2mean), 'native'),
          duration=duration, rep=TRUE)

xLine <- unit(index(NO2zoo), 'native')

grid.animate('vLine',
          x0=xLine, x1=xLine,
          duration=duration, rep=TRUE)

grid.animate('stationsCircles',
          duration=duration,
          r=radius,
          fill=colorAnim,
          rep=TRUE)

## Pause animations when mouse is over the time series plot
grid.garnish('timePlot', grep=TRUE,
          onmouseover='document.rootElement.pauseAnimations()',
          onmouseout='document.rootElement.unpauseAnimations()')

grid.export('figs/vLine.svg')
```

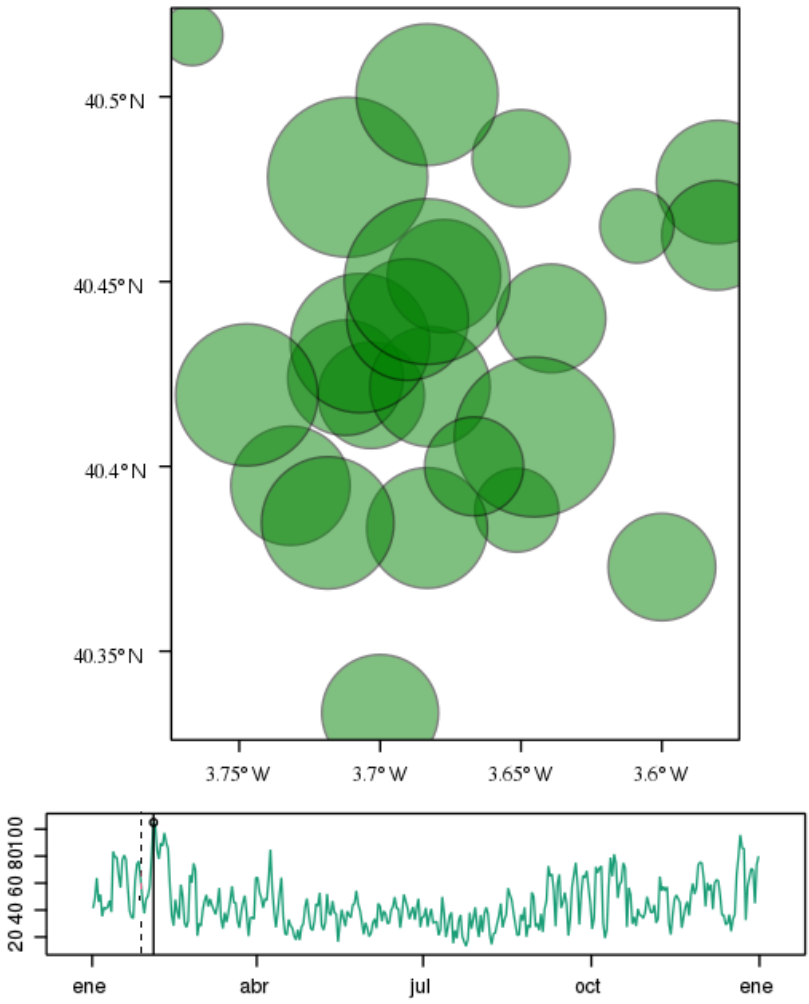

FIGURE 13.5: Animated circles of the NO_2 space-time data with a a time series as reference.

Bibliography

Becker, Richard A., Allan R. Wilks, and Ray Brownrigg (2013). *mapdata: Extra Map Databases*. R package version 2.2-2. URL: http://CRAN.R-project.org/package=mapdata.

Becker, Richard A., Allan R. Wilks, Ray Brownrigg, and Thomas P Minka (2013). *maps: Draw Geographical Maps*. R package version 2.3-3. URL: http://CRAN.R-project.org/package=maps.

Bivand, Roger (2013). *classInt: Choose Univariate Class Intervals*. R package version 0.1-21. URL: http://CRAN.R-project.org/package=classInt.

Bivand, Roger S., Edzer J. Pebesma, and Virgilio Gomez-Rubio (2008). *Applied Spatial Data Analysis with R*. Springer, New York. URL: http://www.asdar-book.org/.

Bivand, Roger, Tim Keitt, and Barry Rowlingson (2013). *rgdal: Bindings for the Geospatial Data Abstraction Library*. R package version 0.8-11. URL: http://CRAN.R-project.org/package=rgdal.

Bivand, Roger and Nicholas Lewin-Koh (2013). *maptools: Tools for Reading and Handling Spatial Objects*. R package version 0.8-26. URL: http://CRAN.R-project.org/package=maptools.

Byron, Lee and Martin Wattenberg (2008). *Stacked Graphs – Geometry & Aesthetics*. Tech. rep. URL: http://www.leebyron.com/else/streamgraph/download.php?file=stackedgraphs_byron_wattenberg.pdf.

Cairo, Alberto (2012). *The Functional Art: An Introduction to Information Graphics and Visualization*. New Riders Publishing, Aarhus, Denmark.

Carr, D. B. et al. (1987). "Scatterplot Matrix Techniques for Large N". English. In: *Journal of the American Statistical Association* 82.398, pp. 424–436. ISSN: 01621459. URL: http://www.jstor.org/stable/2289444.

Carr, Dan, Nicholas Lewin-Koh, and Martin Maechler (2013). *hexbin: Hexagonal Binning Routines*. R package version 1.26.2. URL: http://CRAN.R-project.org/package=hexbin.

Chambers, John M. (2008). *Software for Data Analysis: Programming with R*. ISBN 978-0-387-75935-7. Springer, New York. URL: http://stat.stanford.edu/~jmc4/Rbook/.

Chatfield, Chris (2003). *The Analysis of Time-series: An Introduction*. CRC Press, Boca Raton, FL., p. 333.

Cleveland, W. S. (1993). *Visualizing Data*. Summit, NJ: Hobart Press.

— (1994). *The Elements of Graphing Data*. Murray Hill, NJ.: AT&T, Bell Laboratories.

CM SAF (2013). *The Satellite Application Facility on Climate Monitoring*. http://www.cmsaf.eu.

Cressie, N. and C.K. Wikle (2011). *Statistics for Spatio-Temporal Data*. Wiley Series in Probability and Statistics. Wiley, New York. ISBN: 9780471692744.

Dent, B., J. Torguson, and T. Hodler (2008). *Cartography: Thematic Map Design*. McGraw-Hill Education, New York. ISBN: 9780072943825.

Eugster, Manuel J. A. and Thomas Schlesinger (2010). "osmar: OpenStreetMap and R". In: *R Journal*. URL: http://osmar.r-forge.r-project.org/RJpreprint.pdf.

Fellows, Ian and Jan Peter Stotz (2013). *OpenStreetMap: Access to Open Street Map Raster Images*. R package version 0.3.1. URL: http://CRAN.R-project.org/package=OpenStreetMap.

Few, S. (2007). *Visualizing Change: An Innovation in Time-Series Analysis*. Tech. rep. Perceptual Edge, Berkeley, CA. URL: http://www.perceptualedge.com/articles/visual_business_intelligence/visualizing_change.pdf.

— (2008). *Time on the Horizon*. Tech. rep. Perceptual Edge, Berkeley, CA. URL: http://www.perceptualedge.com/articles/visual_business_intelligence/time_on_the_horizon.pdf.

Friendly, Michael and Daniel Denis (2005). "The early origins and development of the scatterplot". In: *Journal of the History of the Behavioral Sciences* 41.2, pp. 103–130. ISSN: 1520-6696. DOI: 10.1002/jhbs.20078.

Gesmann, Markus and Diego de Castillo (2011). "googleVis: Interface between R and the Google Visualisation API". In: *The R Journal* 3.2, pp. 40–44. URL: http://journal.r-project.org/archive/2011-2/RJournal_2011-2_Gesmann+de~Castillo.pdf.

Grothendieck, Gabor and Thomas Petzoldt (2004). "R Help Desk: Date and Time Classes in R". In: *R News* 4.1, pp. 29–32. URL: http://CRAN.R-project.org/doc/Rnews/Rnews_2004-1.pdf.

Harrower, M. and S. I. Fabrikant (2008). "The Role of Map Animation in Geographic Visualization". In: *Geographic Visualization: Concepts, Tools and Applications*. Ed. by M. Dodge, M. McDerby, and Turner M. Chichester, UK: Wiley, pp. 49–65. URL: http://www.zora.uzh.ch/8979/.

Havre, S. et al. (2002). "ThemeRiver: Visualizing Thematic Changes in Large Document Collections". In: *IEEE Transactions on Visualization and Computer Graphics* 8.1, pp. 9–20. ISSN: 1077-2626. DOI: 10.1109/2945.981848.

Heer, J. and M. Agrawala (2006). "Multi-Scale Banking to 45 Degrees". In: *IEEE Transactions on Visualization and Computer Graphics* 12.5, pp. 701–708. ISSN: 1077-2626. DOI: 10.1109/TVCG.2006.163. URL: http://vis.berkeley.edu/papers/banking/2006-Banking-InfoVis.pdf.

Heer, J., N. Kong, and M. Agrawala (2009). "Sizing the Horizon: The Effects of Chart Size and Layering on the Graphical Perception of Time Series Visualizations". In: *ACM Human Factors in Computing Systems (CHI)*, pp. 1303–1312. URL: http://vis.berkeley.edu/papers/horizon/2009-TimeSeries-CHI.pdf.

Heer, Jeffrey, Michael Bostock, and Vadim Ogievetsky (2010). "A tour through the visualization zoo". In: *Communications of the ACM* 53.6, pp. 59–67. ISSN: 0001-0782. DOI: 10.1145/1743546.1743567. URL: http://doi.acm.org/10.1145/1743546.1743567.

Hengl, T. (2009). *A Practical Guide to Geostatistical Mapping*. University of Amsterdam, Amsterdam. URL: http://spatial-analyst.net/book/.

Hijmans, Robert J. (2013). *raster: Geographic Analysis and Modeling with Raster Data*. R package version 2.1-66. URL: http://CRAN.R-project.org/package=raster.

Hocking, Toby Dylan (2013). *directlabels: Direct labels for multicolor plots in lattice or ggplot2*. R package version 2013.6.15. URL: http://CRAN.R-project.org/package=directlabels.

Hovmöller, Ernest (1949). "The Trough-and-Ridge diagram". In: *Tellus* 1.2, pp. 62–66. ISSN: 2153-3490. DOI: 10.1111/j.2153-3490.1949.tb01260.x.

Ihaka, Ross et al. (2011). *colorspace: Color Space Manipulation*. R package version 1.1-0. URL: http://CRAN.R-project.org/package=colorspace.

Kahle, David and Hadley Wickham (2013). *ggmap: A package for spatial visualization with Google Maps and OpenStreetMap*. R package version 2.3. URL: http://CRAN.R-project.org/package=ggmap.

Kropotkin, Piotr (1906). *The Conquest of Bread*. G. P. Putnam's Sons. URL: http://en.wikisource.org/wiki/The_Conquest_of_Bread.

McIlroy, Doug et al. (2013). *mapproj: Map Projections*. R package version 1.2-1. URL: http://CRAN.R-project.org/package=mapproj.

Meihoefer, Hans-Joachim (1969). "The Utility of the Circle as an Effective Cartographic Symbol". In: *Cartographica: The International Journal for Geographic Information and Geovisualization* 6 (2), pp. 104–117. DOI: 10.3138/J04Q-1K34-26X1-7244.

Mumford, L. (1934). *Technics and Civilization*. San Diego, CA.: Harcourt, Brace & Company, Inc.

Murrell, Paul (2011). *R Graphics*. 2nd. The R Series. Boca Raton, FL.: Chapman & Hall/CRC, p. 546.

Murrell, Paul and Simon Potter (2013). *gridSVG: Export grid graphics as SVG*. R package version 1.3-1. URL: http://CRAN.R-project.org/package=gridSVG.

Neuwirth, Erich (2011). *RColorBrewer: ColorBrewer Palettes*. R package version 1.0-5. URL: http://CRAN.R-project.org/package=RColorBrewer.

Pebesma, Edzer (2012). "spacetime: Spatio-Temporal Data in R". In: *Journal of Statistical Software* 51.7, pp. 1–30. ISSN: 1548-7660. URL: http://www.jstatsoft.org/v51/i07.

Pebesma, Edzer J. (2004). "Multivariable Geostatistics in S: The gstat Package". In: *Computers and Geosciences* 30, pp. 683–691.

Pebesma, Edzer J. and Roger S. Bivand (2005). "Classes and methods for spatial data in R". In: *R News* 5.2, pp. 9–13. URL: http://CRAN.R-project.org/doc/Rnews/.

Perpiñán, Oscar (2012). "solaR: Solar Radiation and Photovoltaic Systems with R". In: *Journal of Statistical Software* 50.9, pp. 1–32. URL: http://www.jstatsoft.org/v50/i09/.

Perpiñán, Oscar and Robert Hijmans (2013). *rasterVis: Visualization Methods for the raster Package*. R package version 0.24. URL: http://CRAN.R-project.org/package=rasterVis.

Posselt, R., R.W. Mueller, et al. (2012). "Remote sensing of solar surface radiation for climate monitoring — the CM SAF retrieval in international comparison". In: *Remote Sensing of Environment* 118, pp. 186–198. ISSN: 0034-4257. DOI: 10.1016/j.rse.2011.11.016.

Posselt, R., R. Müller, et al. (2011). *CM SAF Surface Radiation MVIRI Data Set 1.0 - Monthly Means / Daily Means / Hourly Means*. DOI: 10.5676/EUM_SAF_CM/RAD_MVIRI/V001. URL: http://dx.doi.org/10.5676/EUM_SAF_CM/RAD_MVIRI/V001.

R Development Core Team (2013). *R: A Language and Environment for Statistical Computing*. ISBN 3-900051-07-0. R Foundation for Statistical Computing. Vienna, Austria. URL: http://www.R-project.org.

Ripley, Brian D. and Kurt Hornik (2001). "Date-Time Classes". In: *R News* 1.2, pp. 8–11. URL: http://CRAN.R-project.org/doc/Rnews/Rnews_2001-2.pdf.

Rossini, A. J. et al. (2004). "Emacs Speaks Statistics: A Multiplatform, Multipackage Development Environment for Statistical Analysis". In: *Journal of Computational and Graphical Statistics* 13.1, pp. 247–261. DOI: 10.1198/1061860042985. eprint: http://www.tandfonline.com/doi/pdf/10.1198/1061860042985. URL: http://ess.r-project.org/.

Ryan, Jeffrey A. and Joshua M. Ulrich (2013). *xts: eXtensible Time Series*. R package version 0.9-5. URL: http://CRAN.R-project.org/package=xts.

Sarkar, Deepayan (2008). *Lattice: Multivariate Data Visualization with R*. ISBN 978-0-387-75968-5. New York: Springer. URL: http://lmdvr.r-forge.r-project.org.

Sarkar, Deepayan and Felix Andrews (2012). *latticeExtra: Extra Graphical Utilities Based on lattice*. R package version 0.6-24. URL: http://CRAN.R-project.org/package=latticeExtra.

Schulte, Eric et al. (2012). "A Multi-Language Computing Environment for Literate Programming and Reproducible Research". In: *Journal of Statistical Software* 46.3, pp. 1–24. ISSN: 1548-7660. URL: http://www.jstatsoft.org/v46/i03.

Schulz, J. et al. (2009). "Operational Climate Monitoring from Space: The EUMETSAT Satellite Application Facility on Climate Monitoring". In: *Atmospheric Chemistry and Physics* 9, pp. 1687–1709. DOI: 10.5194/acp-9-1687-2009. URL: http://www.atmos-chem-phys.net/9/1687/2009/acp-9-1687-2009.pdf.

Slocum, Terry A. (2005). *Thematic Cartography and Geographic Visualization*. Englewood Cliffs, NJ.: Prentice Hall. ISBN: 9780130351234.

Tufte, E. R. (1990). *Envisioning information*. Cheshire, CT.: Graphic Press.

— (2001). *The Visual Display of Quantitative Information*. Cheshire, CT.: Graphic Press.

Ware, Colin (2008). *Visual Thinking for Design*. Burlington, MA.: Morgan Kaufmann Pub. ISBN: 9780123708960.

Wegenkittl, Rainer and Eduard Gröller (1997). "Fast Oriented Line Integral Convolution for Vector Field Visualization via the Internet". In: *Proceedings of the 8th Conference on Visualization'97*. IEEE Computer Society Press, pp. 309–316. URL: http://christl.cg.tuwien.ac.at/research/vis/dynsys/frolic/frolic_crc.pdf.

Wickham, Hadley (2009). *ggplot2: Elegant Graphics for Data Analysis*. New York: Springer. ISBN: 978-0-387-98140-6. URL: http://ggplot2.org/book/.

Wilkinson, Leland (1999). *The Grammar of Graphics*. New York: Springer. ISBN: 0387987746.

Wills, G. (2011). *Visualizing Time: Designing Graphical Representations for Statistical Data*. Statistics and Computing. New York: Springer. ISBN: 9780387779065.

Xie, Yihui (2013). "animation: An R Package for Creating Animations and Demonstrating Statistical Methods". In: *Journal of Statistical Software* 53.1, pp. 1–27. ISSN: 1548-7660. URL: http://www.jstatsoft.org/v53/i01.

Zeileis, Achim and Gabor Grothendieck (2005). "zoo: S3 Infrastructure for Regular and Irregular Time Series". In: *Journal of Statistical Software* 14.6, pp. 1–27. URL: http://www.jstatsoft.org/v14/i06/.

Zeileis, Achim, Kurt Hornik, and Paul Murrell (2009). "Escaping RGBland: Selecting Colors for Statistical Graphics". In: *Computational Statistics & Data Analysis* 53, pp. 3259–3270. DOI: 10.1016/j.csda.2008.11.033. URL: http://epub.wu.ac.at/1692/.

Index

`+.trellis`, 61, 95, 115
3D visualization, 116

`aggregate`, 59, 150, 151
`animUnit`, 72, 181
`animValue`, 181
`apply`, 41, 138, 151
`as_sp`, 142

`brewer.pal`, 59, 119, 132, 138, 139, 171
`bwplot`, 163

`cellStats`, 117
`char2dms`, 148
`classIntervals`, 65, 89, 122
`colorRamp`, 121
`colorRampPalette`, 177
`coordinates`, 138
`crop`, 124, 143
`CRS`, 135
`current.panel.limits`, 48
`cut`, 126

Data
  Air quality in Madrid, 150, 175
  CO_2, 56
  DIVA-GIS, 115, 136
  GADM, 136
  Geonetwork, 143
  Natural Earth Data, 136
  nomecalles, 95
  OpenStreetMap, 140
  Wind Speed, 130
  World Bank, 56
`diag.panel.splom`, 48
`direct.label`, 62
`do.call`, 149
`download.file`, 136

`Encoding`, 105, 136
`extent`, 124, 137

`ffmpeg`, 171
`find`, 141
`find_down`, 142
`findColours`, 89
`findCols`, 89
`findInterval`, 121

GeoJSON, 100
`get_osm`, 140

getNodeSet, 101
gIntersection, 137
glayer, 61
gregexpr, 149
grid.animate, 72, 181, 182
grid.circle, 180
grid.export, 35, 73, 102, 182
grid.garnish, 34, 102, 182
grid.hyperlink, 73, 102
grid.ls, 33
grid.rect, 182
grid.script, 34
grid.segments, 182
grid.text, 182
group.number, 62
group.value, 62
gsub, 149

hcl, 107
hclust, 59
hexbinplot, 50
Hill shading, 142, 143
hillShade, 115, 143
histogram, 127, 163
Horizon graph, 26
horizonplot, 28, 167
hovmoller, 165
htmlParse, 101

INE, 105

JavaScript, 34, 102
jQuery, 103
jQuery UI, 103

KML, 101

lapply, 148, 149
layer, 95, 98, 115, 139
levelplot, 113, 126, 139, 162, 171

map2SpatialLines, 113, 171
match, 142, 151
modifyList, 126

osmsource_api, 140

Package
  RColorBrewer, 119
  xts, 23
Packages
  classInt, 65, 89, 122
  colorspace, 107, 135, 144
  directlabels, 62
  ggmap, 91
  =ggplot2, 6
  googleVis, 56
  =grid, 4
  grid, 110, 180
  gridSVG, 33, 71, 102, 180
  gstat, 84, 98
  hexbin, 48, 117
  krige, 98
  =lattice, 5
  =latticeExtra, 5
  latticeExtra, 28, 52, 61, 105, 114, 144
  mapdata, 85, 113, 171
  mapproj, 85
  maps, 85, 105, 113, 171
  maptools, 83, 96, 105, 113, 135, 144, 151, 171
  OpenStreetMap, 91
  osmar, 140
  plotKML, 101
  raster, 81, 113, 124, 130, 135, 143, 153, 159, 161
  rasterVis, 83, 113, 126, 130, 135, 143, 159, 161
  RColorBrewer, 59
  rgdal, 84, 95, 100, 101, 171

rgeos, 105, 135
rgl, 116
sp, 80, 95, 105, 114, 135, 144, 148, 150, 175
spacetime, 158, 176
XML, 101, 148
xts, 15
zoo, 14, 161, 176
packGrob, 110
Panel function, 20, 41, 48, 50, 61
panel.hexbinplot, 48
panel.link.splom, 46
panel.loess, 48, 52
panel.number, 20
panel.points, 20
panel.polygon, 41
panel.rug, 52
panel.superpose, 61
panel.text, 20, 41, 61, 68, 139
panel.xblocks, 20
plinrain, 117

rainbow_hcl, 107
raster, 136
rasterTheme, 113, 117, 126, 138, 139
ratify, 126
read.csv2, 150, 175
readHTMLTable, 148
readLines, 149
readShapeLines, 95, 136
readShapePoly, 105, 136
Reduce, 109
reshape, 50, 176
rgb, 121

sapply, 41, 138
sequential_hcl, 108
setZ, 161
simpleTheme, 60
Small multiples, 28
sp.lineLabel, 139
sp.lines, 95, 98, 114, 139, 144
sp.pointLabel, 144
sp.points, 144
sp.pointLabel, 95, 96
sp.points, 98
sp.polygons, 95, 98, 139
spCbind, 151
splom, 46, 164
spplot, 98, 109, 180
sprintf, 151
spTransform, 95, 171
stack, 127, 153
STFDF, 176
STL, 116
stplot, 177
streamplot, 132
String manipulation, 149
strsplit, 138, 149
subscripts, 61
subset, 141
substr, 149
superpose.polygon, 41

tapply, 121
terrain, 115, 143
terrain_hcl, 139
textGrob, 110
trellis.focus, 46
trellis.par.get, 41

unionSpatialPolygons, 106
unstack, 41
useOuterStrips, 52

vectorplot, 131

way, 141
Web scraping, 148
WebGL, 116

writeOGR, 100

xyplot, 117, 167
xyplot.zoo, 25

yearmon, 14
yearqtr, 14

zApply, 162
zoo, 25