

Use R!

Series Editors:

Robert Gentleman Kurt Hornik Giovanni G. Parmigiani

For further volumes:

<http://www.springer.com/series/6991>

Roger S. Bivand • Edzer Pebesma
Virgilio Gómez-Rubio

Applied Spatial Data Analysis with R

Second Edition



Springer

Roger S. Bivand
Norwegian School of Economics
Bergen, Norway

Edzer Pebesma
Westfälische Wilhelms-Universität
Münster, Germany

Virgilio Gómez-Rubio
Department of Mathematics
Universidad de Castilla-La Mancha
Albacete, Spain

ISBN 978-1-4614-7617-7 ISBN 978-1-4614-7618-4 (eBook)
DOI 10.1007/978-1-4614-7618-4
Springer New York Heidelberg Dordrecht London

Library of Congress Control Number: 2013938605

© Springer Science+Business Media New York 2013

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed. Exempted from this legal reservation are brief excerpts in connection with reviews or scholarly analysis or material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the work. Duplication of this publication or parts thereof is permitted only under the provisions of the Copyright Law of the Publisher's location, in its current version, and permission for use must always be obtained from Springer. Permissions for use may be obtained through RightsLink at the Copyright Clearance Center. Violations are liable to prosecution under the respective Copyright Law.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

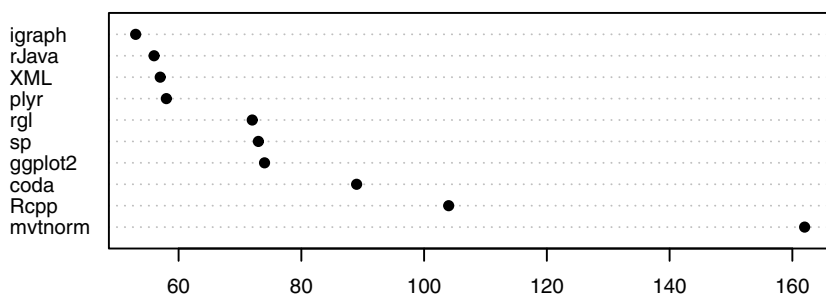
Ewie

Voor Ellen, Ulla en Mandus

A mis padres, Victorina y Virgilio Benigno

Preface (Second Edition)

Ten years ago, the small group of spatial data analysis enthusiasts who met in Vienna at the Distributed Statistical Computing conference mentioned in the preface to the first edition, considered that others might benefit from coordinating software development in our fields. We were in no way prepared for the dramatic and largely unexpected growth in use that software for spatial data analysis with R has seen (R Core Team, 2013). Some of this growth has come from the growth of R as a project, including growth in the use of R within disciplines analysing spatial data.



Analysis as of 10 Mar 2013 covering 4389 total CRAN packages
Limited to top 10 packages, excluding 'Recommended' ones shipped with R

Fig. 1 Direct dependency counts of CRAN packages

We do however feel humbled by the realisation that updating the **sp** package can potentially upset the work of many unsuspecting users and developers. In our first edition, we were proud to include a figure (Fig. 1.1, p. 5) showing the dependency tree of **sp**. In revising our book, we have been obliged to drop this figure, as it is illegible at less than poster size. Fig. 1 gives the current top ten ranking of counts of packages depending directly on CRAN

packages, using Dirk Eddelbuettel's code.¹ The number of direct dependencies for **sp** is 73, but if we include indirect dependencies and imports through the dependency tree, we reach 148, and the total number of unique CRAN packages directly or indirectly depending on, importing or suggesting **sp** is 507.²

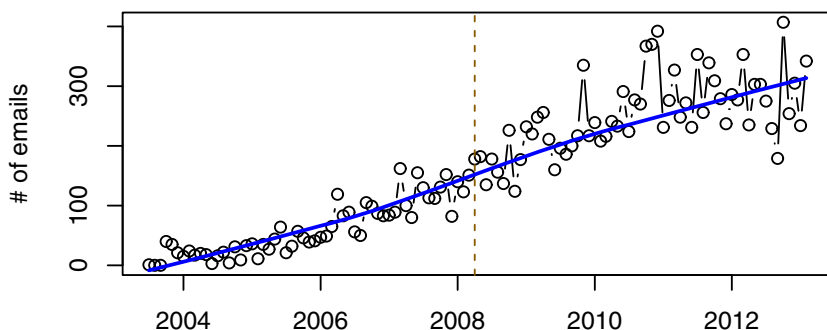


Fig. 2 Monthly numbers of postings to the R-sig-geo mailing list

Another expression for the vitality of the R spatial data analysis community is the development on the numbers of postings on the R-sig-geo mailing list, shown in Fig. 2. Importantly, the proportion of follow-up messages contributed by early participants has fallen, not because they have departed, but because the community has grown. The **raster** package is an important contribution that has taken raster processing and raster algebra to the next level and extended R as a raster GIS; a noticeable amount of list traffic now concerns use of this package. In addition, it seems that many courses have been organised bringing developers and users together, among these the GeoStat courses coordinated by Tom Hengl; we have benefitted from meeting users in the flesh, not only on the list.

Over the few years since the first edition was coming into being, we see clearly that spatial data, and the devices used for its collection, are becoming pervasive. In 2008, we could ask students whether they had access to a Global Positioning System (GPS) receiver. In 2013, we may ask how many GPS receivers students use in smartphones, tablets, vehicle navigation devices, etc. In 2008, Google Earth™ and Google Maps™ with others were seen as resources to be used on computers rather than mobile devices. Now, the failure of a smartphone manufacturer to handle spatial data satisfactorily is a top news story and can prejudice the careers of top managers. Use and handling of spatial data have grown greatly, but perhaps analysis is lagging,

¹ http://dirk.eddelbuettel.com/blog/2012/08/16#counting_cran_depends_followup

² Use is made of functions in the **pkgDepTools** package.

so work to make more and better analytical use of the positional data that is now potentially available should be moved up the agenda.

In revising our book, we have made the incremental changes needed to keep abreast of developments in the packages presented and discussed before. In addition, we have modified Chap. 5 to introduce the new **rgeos** package for handling operations on topologies. We have not attempted to cover the **raster** package in the detail that it deserves, hoping that a **raster** book will appear before long. We have replaced Chap. 6 with a new chapter on representing and handling spatio-temporal data, introducing the **spacetime** package; the chapter from the first edition is now a vignette in **sp**. Since the publication of Cressie and Wikle (2011) has provided new impetus to the analysis of spatio-temporal data, we expect these topics to grow in importance rapidly in coming years. We have also moved the detailed treatment of spatial neighbours to a vignette in the **spdep** package, making room for the presentation of new features now included in **spdep**. In Chap. 10, we have included worked examples of alternatives to WinBUGS for fitting Bayesian hierarchical models, including INLA (Rue et al., 2009) and BayesX and their R interface packages.

Finally, we are pleased that we can now present coloured figures,³ which we hope add to the value of the completed volume; thanks to Hannah Bracken for persevering with us in the revision process. The book website (<http://www.asdar-book.org>) will continue to provide code, data sets, and errata from this edition of our book; we will continue to run the code from the book, with suitable updates where required, nightly on the released version of R.

Bergen, Norway
Münster, Germany
Albacete, Spain

Roger S. Bivand
Edzer Pebesma
Virgilio Gómez-Rubio

³ Although reasonable care has been taken, the rendering of the colours may differ between the published figures and on-screen reproduction in an R session.

Preface (First Edition)

We began writing this book in parallel with developing software for handling and analysing spatial data with R (R Development Core Team, 2008). Although the book is now complete, software development will continue, in the R community fashion, of rich and satisfying interaction with users around the world, of rapid releases to resolve problems, and of the usual joys and frustrations of getting things done. There is little doubt that without pressure from users, the development of R would not have reached its present scale, and the same applies to analysing spatial data analysis with R.

It would, however, not be sufficient to describe the development of the R project mainly in terms of narrowly defined utility. In addition to being a community project concerned with the development of world-class data analysis software implementations, it promotes specific choices with regard to how data analysis is carried out. R is open source not only because open source software development, including the dynamics of broad and inclusive user and developer communities, is arguably an attractive and successful development model.

R is also, or perhaps chiefly, open source because the analysis of empirical and simulated data in science should be reproducible. As working researchers, we are all too aware of the possibility of reaching inappropriate conclusions in good faith because of user error or misjudgement. When the results of research really matter, as in public health, in climate change, and in many other fields involving spatial data, good research practice dictates that someone else should be, at least in principle, able to check the results. Open source software means that the methods used can, if required, be audited, and journaling working sessions can ensure that we have a record of what we actually did, not what we thought we did. Further, using **Sweave**⁴ – a tool that permits the embedding of R code for complete data analyses in documents – throughout this book has provided crucial support (Leisch, 2002; Leisch and Rossini, 2003).

⁴ <http://www.stat.uni-muenchen.de/~leisch/Sweave/>

We acknowledge our debt to the members of R-core for their continuing commitment to the R project. In particular, the leadership and example of Professor Brian Ripley has been important to us, although our admitted ‘muddling through’ contrasts with his peerless attention to detail. His interested support at the Distributed Statistical Computing conference in Vienna in 2003 helped us to see that encouraging spatial data analysis in R was a project worth pursuing. Kurt Hornik’s dedication to keep the Comprehensive R Archive Network running smoothly, providing package maintainers with superb, almost 24/7, service, and his dry humour when we blunder, have meant that the useR community is provided with contributed software in an unequalled fashion. We are also grateful to Martin Mächler for his help in setting up and hosting the R-sig-geo mailing list, without which we would have not had a channel for fostering the R spatial community.

We also owe a great debt to users participating in discussions on the mailing list, sometimes for specific suggestions, often for fruitful questions, and occasionally for perceptive bug reports or contributions. Other users contact us directly, again with valuable input that leads both to a better understanding on our part of their research realities and to the improvement of the software involved. Finally, participants at R spatial courses, workshops, and tutorials have been patient and constructive.

We are also indebted to colleagues who have contributed to improving the final manuscript by commenting on earlier drafts and pointing out better procedures to follow in some examples. In particular, we would like to mention Juanjo Abellán, Nicky Best, Peter J. Diggle, Paul Hiemstra, Rebeca Ramis, Paulo J. Ribeiro Jr., Barry Rowlingson, and Jon O. Skoien. We are also grateful to colleagues for agreeing to our use of their data sets. Support from Luc Anselin has been important over a long period, including a very fruitful CSISS workshop in Santa Barbara in 2002. Work by colleagues, such as the first book known to us on using R for spatial data analysis (Kopczewska, 2006), provided further incentives both to simplify the software and to complete its description. Without John Kimmel’s patient encouragement, it is unlikely that we would have finished this book.

Even though we have benefitted from the help and advice of so many people, there are bound to be things we have not yet grasped – so remaining mistakes and omissions remain our sole responsibility. We would be grateful for messages pointing out errors in this book; errata will be posted on the book website (<http://www.asdar-book.org>).

Bergen, Norway
Münster, Germany
London, UK

Roger S. Bivand
Edzer Pebesma
Virgilio Gómez-Rubio

Contents

Preface (Second Edition)	vii
Preface (First Edition)	xi
1 <i>Hello World: Introducing Spatial Data</i>	1
1.1 Applied Spatial Data Analysis	1
1.2 Why Do We Use R	2
1.2.1 ... In General?	2
1.2.2 ... for Spatial Data Analysis?	3
1.2.3 ... and for Reproducible Research?	4
1.3 R and GIS	5
1.3.1 What Is GIS?	5
1.3.2 Service-Oriented Architectures	6
1.3.3 Further Reading on GIS	6
1.4 Types of Spatial Data	8
1.5 Storage and Display	10
1.6 Applied Spatial Data Analysis	11
1.7 R Spatial Resources	14
1.8 Layout of the Book	15
 Part I Handling Spatial Data in R	
 2 <i>Classes for Spatial Data in R</i>	21
2.1 Introduction	21
2.2 Classes and Methods in R	23
2.3 Spatial Objects	28
2.4 SpatialPoints	30
2.4.1 Methods	31
2.4.2 Data Frames for Spatial Point Data	33

2.5	SpatialLines	37
2.6	SpatialPolygons	41
2.6.1	SpatialPolygonsDataFrame Objects	44
2.6.2	Holes and Ring Direction	46
2.7	SpatialGrid and SpatialPixel Objects	48
2.8	Raster Objects and the raster Package	54
3	Visualising Spatial Data	59
3.1	The Traditional Plot System	60
3.1.1	Plotting Points, Lines, Polygons, and Grids	60
3.1.2	Axes and Layout Elements	61
3.1.3	Degrees in Axes Labels and Reference Grid	65
3.1.4	Plot Size, Plotting Area, Map Scale, and Multiple Plots	66
3.1.5	Plotting Attributes and Map Legends	68
3.2	Trellis/Lattice Plots with spplot	69
3.2.1	A Straight Trellis Example	70
3.2.2	Plotting Points, Lines, Polygons, and Grids	70
3.2.3	Adding Reference and Layout Elements to Plots	73
3.2.4	Arranging Panel Layout	74
3.3	Alternatives Routes: ggplot , latticeExtra	75
3.4	Interactive Plots	76
3.4.1	Interacting with Base Graphics	77
3.4.2	Interacting with spplot and Lattice Plots	78
3.5	Colour Palettes and Class Intervals	79
3.5.1	Colour Palettes	79
3.5.2	Class Intervals	79
4	Spatial Data Import and Export	83
4.1	Coordinate Reference Systems	84
4.1.1	Using the EPSG List	85
4.1.2	PROJ.4 CRS Specification	86
4.1.3	Projection and Transformation	88
4.1.4	Degrees, Minutes, and Seconds	90
4.2	Vector File Formats	91
4.2.1	Using OGR Drivers in rgdal	92
4.2.2	Other Import/Export Functions	99
4.3	Raster File Formats	100
4.3.1	Using GDAL Drivers in rgdal	100
4.3.2	Other Import/Export Functions	107
4.4	Google Earth™, Google Maps™ and Other Formats	108
4.5	Geographical Resources Analysis Support System (GRASS)..	112
4.5.1	Broad Street Cholera Data	118
4.6	Other Import/Export Interfaces	122
4.6.1	Analysis and Visualisation Applications	122

4.6.2	TerraLib and aRT	123
4.6.3	Other GIS Systems	124
4.7	Installing rgdal	125
5	Further Methods for Handling Spatial Data	127
5.1	Support	127
5.2	Handling and Combining Features	130
5.2.1	The rgeos Package	130
5.2.2	Using rgeos	132
5.3	Map Overlay or Spatial Join	140
5.3.1	Spatial Aggregation	142
5.3.2	Using the raster Package for Extract Operations	145
5.3.3	Spatial Sampling	146
5.4	Auxiliary Functions	149
6	Spatio-Temporal Data	151
6.1	Introduction	151
6.2	Types of Spatio-Temporal Data	151
6.2.1	Spatial Point or Area, Time Instance or Interval	152
6.2.2	Are Space and Time of <i>Primary</i> Interest?	152
6.2.3	Regularity of Space-Time Layouts	152
6.2.4	Do Objects Change Location?	153
6.3	Classes in spacetime	154
6.4	Handling Time Series Data with xts	155
6.5	Construction of ST Objects	156
6.6	Selection, Addition, and Replacement of Attributes	158
6.7	Overlay and Aggregation	159
6.8	Visualisation	161
6.8.1	Multi-panel Plots	161
6.8.2	Space-Time Plots	162
6.8.3	Animated Plots	163
6.8.4	Time Series Plots	164
6.9	Further Packages	164
6.9.1	Handling Spatio-Temporal Data	165
6.9.2	Analysing Spatio-Temporal Data	165
6.10	Outlook	165

Part II Analysing Spatial Data

7	Spatial Point Pattern Analysis	173
7.1	Introduction	173
7.2	Packages for the Analysis of Spatial Point Patterns	174
7.3	Preliminary Analysis of a Point Pattern	178
7.3.1	Complete Spatial Randomness	179
7.3.2	<i>G</i> Function: Distance to the Nearest Event	179

7.3.3	F Function: Distance from a Point to the Nearest Event	181
7.4	Statistical Analysis of Spatial Point Processes	182
7.4.1	Homogeneous Poisson Processes	183
7.4.2	Inhomogeneous Poisson Processes	184
7.4.3	Estimation of the Intensity	184
7.4.4	Likelihood of an Inhomogeneous Poisson Process	187
7.4.5	Second-Order Properties	190
7.5	Some Applications in Spatial Epidemiology	192
7.5.1	Case-Control Studies	193
7.5.2	Binary Regression Estimator	198
7.5.3	Binary Regression Using Generalised Additive Models	199
7.5.4	Point Source Pollution	202
7.5.5	Accounting for Confounding and Covariates	206
7.6	Further Methods for the Analysis of Point Patterns	210
8	Interpolation and Geostatistics	213
8.1	Introduction	213
8.2	Exploratory Data Analysis	214
8.3	Non-geostatistical Interpolation Methods	215
8.3.1	Inverse Distance Weighted Interpolation	215
8.3.2	Linear Regression	216
8.4	Estimating Spatial Correlation: The Variogram	217
8.4.1	Exploratory Variogram Analysis	219
8.4.2	Cutoff, Lag Width, Direction Dependence	222
8.4.3	Variogram Modelling	224
8.4.4	Anisotropy	228
8.4.5	Multivariable Variogram Modelling	229
8.4.6	Residual Variogram Modelling	230
8.5	Spatial Prediction	232
8.5.1	Universal, Ordinary, and Simple Kriging	233
8.5.2	Multivariable Prediction: Cokriging	233
8.5.3	Collocated Cokriging	236
8.5.4	Cokriging Contrasts	237
8.5.5	Kriging in a Local Neighbourhood	237
8.5.6	Change of Support: Block Kriging	238
8.5.7	Stratifying the Domain	240
8.5.8	Trend Functions and Their Coefficients	241
8.5.9	Non-linear Transforms of the Response Variable	242
8.5.10	Singular Matrix Errors	243
8.6	Kriging, Filtering, Smoothing	245
8.7	Model Diagnostics	247
8.7.1	Cross Validation Residuals	247
8.7.2	Cross Validation z -Scores	249

8.7.3	Multivariable Cross Validation	250
8.7.4	Limitations to Cross Validation	250
8.8	Geostatistical Simulation	252
8.8.1	Sequential Simulation	252
8.8.2	Non-linear Spatial Aggregation and Block Averages...	254
8.8.3	Multivariable and Indicator Simulation.....	255
8.9	Model-Based Geostatistics and Bayesian Approaches	256
8.10	Monitoring Network Optimisation	256
8.11	Other R Packages for Interpolation and Geostatistics.....	258
8.11.1	Non-geostatistical Interpolation	258
8.11.2	Spatial	259
8.11.3	RandomFields	259
8.11.4	geoR and geoRglm	259
8.11.5	Fields	260
8.11.6	spBayes	260
8.12	Spatio-Temporal Prediction	260
9	Modelling Areal Data	263
9.1	Introduction	263
9.2	Spatial Neighbours and Spatial Weights	266
9.2.1	Neighbour Objects	266
9.2.2	Spatial Weights Objects	269
9.2.3	Handling Spatial Weights Objects	273
9.2.4	Using Weights to Simulate Spatial Autocorrelation ...	274
9.3	Testing for Spatial Autocorrelation	275
9.3.1	Global Tests	278
9.3.2	Local Tests	284
9.4	Fitting Models of Areal Data	288
9.4.1	Spatial Statistics Approaches	290
9.4.2	Spatial Econometrics Approaches.....	303
9.4.3	Other Methods.....	314
10	Disease Mapping	319
10.1	Introduction	320
10.2	Statistical Models	322
10.2.1	Poisson-Gamma Model.....	323
10.2.2	Log-Normal Model	325
10.2.3	Marshall's Global EB Estimator.....	326
10.3	Spatially Structured Statistical Models	328
10.4	Bayesian Hierarchical Models	330
10.4.1	The Poisson-Gamma Model Revisited.....	332
10.4.2	Spatial Models	336
10.5	Geoadditive Models.....	345
10.6	Detection of Clusters of Disease	347
10.6.1	Testing the Homogeneity of the Relative Risks	348
10.6.2	Moran's I Test of Spatial Autocorrelation	350
10.6.3	Tango's Test of General Clustering	351

10.6.4	Detection of the Location of a Cluster	352
10.6.5	Geographical Analysis Machine	353
10.6.6	Kulldorff's Statistic	353
10.6.7	Stone's Test for Localised Clusters	355
10.7	Spatio-Temporal Disease Mapping	356
10.7.1	Introduction	356
10.7.2	Spatio-Temporal Modelling of Disease	357
10.8	Other Topics in Disease Mapping	361
Afterword		363
	R and Package Versions Used	364
	Data Sets Used	364
References		367
Subject Index		387
Functions Index		401

Chapter 1

Hello World: Introducing Spatial Data

1.1 Applied Spatial Data Analysis

Spatial and spatio-temporal data are everywhere. Besides those we collect ourselves ('is it raining?'), they confront us on television, in newspapers, on route planners, on computer screens, on mobile devices, and on plain paper maps. Making a map that is suited to its purpose and does not distort the underlying data unnecessarily is however not easy. Beyond creating and viewing maps, spatial data *analysis* is concerned with questions not directly answered by looking at the data themselves. These questions refer to hypothetical processes that generate the observed data. Statistical inference for such spatial processes is often challenging, but is necessary when we try to draw conclusions about questions that interest us.

Possible questions that may arise include the following:

- Does the spatial patterning of disease incidences give rise to the conclusion that they are clustered, and if so, are the clusters found related to factors such as age, relative poverty, or pollution sources?
- Given a number of observed soil samples, which part of a study area is polluted?
- Given scattered air quality measurements, how many people are exposed to high levels of black smoke or particulate matter (e.g. PM₁₀),¹ and where do they live?
- Do governments tend to compare their policies with those of their neighbours, or do they behave independently?

In this book we will be concerned with *applied* spatial data analysis, meaning that we will deal with data sets, explain the problems they confront us with, and show how we can attempt to reach a conclusion. This book will refer to the theoretical background of methods and models for data analysis, but emphasise hands-on, do-it-yourself examples using R; readers needing

¹ Particulate matter smaller than about 10 μm .

this background should consult the references. All data sets used in this book and all examples given are available, and interested readers will be able to reproduce them.

In this chapter we discuss the following:

- (i) Why we use R for analysing spatial data
- (ii) The relation between R and geographical information systems (GIS)
- (iii) What spatial data are, and the types of spatial data we distinguish
- (iv) The challenges posed by their storage and display
- (v) The analysis of observed spatial data in relation to processes thought to have generated them
- (vi) Sources of information about the use of R for spatial data analysis and the structure of the book.

1.2 Why Do We Use R

1.2.1 ... *In General?*

The R system² (R Core Team, 2013) is a free software environment for statistical computing and graphics. It is an implementation of the S language for statistical computing and graphics (Becker et al., 1988). For data analysis, it can be highly efficient to use a special-purpose language like S, compared to using a general-purpose language.

For new R users without earlier scripting or programming experience, meeting a programming language may be unsettling, but the investment³ will quickly pay off. The user soon discovers how analysis components – written or copied from examples – can easily be stored, replayed, modified for another data set, or extended. R can be extended easily with new dedicated components, and can be used to develop and exchange data sets and data analysis approaches. It is often much harder to achieve this with programs that require long series of mouse clicks to operate.

R provides many standard and innovative statistical analysis methods. New users may find access to both well-tried and trusted methods, and speculative and novel approaches, worrying. This can, however, be a major strength, because if required, innovations can be tested in a robust environment against legacy techniques. Many methods for analysing spatial data are less frequently used than the most common statistical techniques, and thus benefit proportionally more from the nearness to both the data and the methods that R permits. R uses well-known libraries for numerical analysis, and can easily be extended by or linked to code written in S, C, C++, Fortran, or Java.

² <http://www.r-project.org>

³ A steep learning curve – the user learns a lot per unit time.

Links to various relational data base systems and geographical information systems exist, many well-known data formats can be read and/or written.

The level of voluntary support and the development speed of R are high, and experience has shown R to be environment suitable for developing professional, mission-critical software applications, both for the public and the private sector. The S language can not only be used for low-level computation on numbers, vectors, or matrices but can also be easily extended with classes for new data types and analysis methods for these classes, such as methods for summarising, plotting, printing, performing tests, or model fitting (Chambers, 1998).

In addition to the core R software system, R is also a social movement, with many participants on a continuum from useRs just beginning to analyse data with R to developRs contributing packages to the Comprehensive R Archive Network⁴ (CRAN) for others to download and employ.

Just as R itself benefits from the open source development model, contributed package authors benefit from a world-class infrastructure, allowing their work to be published and revised with improbable speed and reliability, including the publication of source packages and binary packages for many popular platforms. Contributed add-on packages are very much part of the R community, and most core developers also write and maintain contributed packages. A contributed package contains R functions, optional sample data sets, and documentation including examples of how to use the functions.

1.2.2 ...for Spatial Data Analysis?

For over 15 years, R has had an increasing number of contributed packages for handling and analysing spatial data. Up to 2003, these packages all used to make different assumptions about how spatial data were organised, and R itself had no capabilities for distinguishing coordinates from other numbers. In addition, methods for plotting spatial data and other tasks were scattered, made different assumptions on the organisation of the data, and were rudimentary. This was not unlike the situation for time series data at the time.

After some joint effort and wider discussion, a group⁵ of R developers have written the R package **sp** to extend R with classes and methods for spatial data (Pebesma and Bivand, 2005). Classes specify a structure and define how spatial data are organised and stored. Methods are instances of functions specialised for a particular data class. For example, the summary method for all spatial data classes may tell the range spanned by the spatial

⁴ CRAN mirrors are linked from <http://www.r-project.org/>

⁵ Mostly the authors of this book with help from Barry Rowlingson and Paulo J. Ribeiro Jr.

coordinates, and show which coordinate reference system is used (such as degrees longitude/latitude, or the UTM zone). It may in addition show some more details for objects of a specific spatial class. A plot method may, for example create a map of the spatial data.

The **sp** package provides classes and methods for points, lines, polygons, and grids (Sect. 1.4, Chap. 2). Adopting a single set of classes for spatial data offers a number of important advantages:

- (i) It is much easier to move data across spatial statistics packages. The classes are either supported directly by the packages, reading and writing data in the new spatial classes, or indirectly, for example by supplying data conversion between the **sp** classes and the package's classes in an interface package. This last option requires one-to-many links between the packages, which are easier to provide and maintain than many-to-many links.
- (ii) The new classes come with a well-tested set of methods and functions for plotting, printing, subsetting, and summarising spatial objects, or combining (overlying) spatial objects.
- (iii) Packages with interfaces to geographical information systems (GIS), for reading and writing GIS file formats, and for coordinate (re)projection code support the new classes.
- (iv) The new methods include Lattice plots, conditioning plots, plot methods that combine points, lines, polygons, and grids with map elements (reference grids, scale bars, north arrows), degree symbols (as in 52°N) in axis labels, etc.

Chapter 2 introduces the classes and methods provided by **sp**, and discusses some of the implementation details. Further chapters will show the degree of integration of **sp** classes and methods and the packages used for statistical analysis of spatial data. Interfacing other packages for handling and analysing spatial data is usually simple as we see in Part II.

In 2008 this book showed a graph depicting the R packages depending on (reusing classes from) **sp**, and packages depending on those. In 2013, this graph can no longer be shown on a book size paper: over 100 packages depend on **sp** directly, and many more indirectly. Of the over 4,400 packages on CRAN, **sp** has become one of the most reused packages, as noted in the preface to this edition.

1.2.3 ... and for *Reproducible Research*?

One of the joys that novel R users experience when they are at the start of the steep learning curve is the ability to communicate *complete* analyses by providing the script that reruns the analysis. This can be used to ask questions, share results, and trigger discussion, and not only demonstrates but

also documents research and communication skills. Not only at the personal level, but also at institutional and societal levels, scientific activity gains credibility when research is reproducible, and may *lose it when it is not*.

After decades, maybe centuries where scientific quality was determined only by the *journal article* – the one that we used to print on paper – we currently see a shift where both the data and software needed to create the published findings are seen as essential components. Publishing data and software has become rewarding:

- The European Commission has since a long time endorsed open source as a development and dissemination model for funded research projects.
- In 2012 the National Science Foundation (USA) changed its requirements for CVs from scientists applying for funding from listing only *publications* to listing *products*, which can also include published data and software; project proposals are required to describe a plan for publishing data and software.
- The *Journal of Statistical Software*, an open access journal on the subject of statistical software and algorithms publishes software alongside the journal paper. The journal has experienced exponential growth, and reached in 2011 the highest ISI impact factor in the category *probability and statistics*.
- The number of citations papers on freely reusable data and software receive motivates scientists to make data and software available, and to document this process in scientific publications.

In a *forum* contribution, Pebesma et al. (2012) argue why the R Software Environment is a good choice to carry out reproducible geoscientific research. The main argument is that the combination of (i) being a free, open source, cross-platform environment with clear software provenance, versioning and archiving with (ii) maintaining open, documented, accessible and sustained communication channels between users and developers, creates trust by individuals and organisations.

1.3 R and GIS

1.3.1 What Is GIS?

Storage and analysis of spatial data is traditionally done in Geographical Information Systems (GIS). According to the toolbox-based definition of Burrough and McDonnell (1998, p. 11), a GIS is ‘... a powerful set of tools for collecting, storing, retrieving at will, transforming, and displaying spatial data from the real world for a particular set of purposes’. Another definition mentioned in the same source refers to ‘... checking, manipulating, and analysing data, which are spatially referenced to the Earth’.

Its capacity to analyse and visualise data makes R a good choice for spatial data analysis. For many spatial analysis projects, using only R may be sufficient for the job. In other cases, R will be used in conjunction with GIS software and possibly a GIS data base as well. Chapter 4 will show how spatial data are imported from and exported to GIS file formats or data bases. As is often the case in applied data analysis, the real issue is not whether a given problem *can* be solved using an environment such as R, but whether it can be solved *efficiently and reproducibly* with R. In some cases, combining different software components in a workflow may be the most robust solution, for example scripting in languages such as Python.

1.3.2 Service-Oriented Architectures

Today, much of the practice and research in geographical information systems has moved from toolbox-centred architectures (think of the ‘classic’ Arc/Info™ or ArcGIS™ applications) towards *service-centred* architectures (such as Google Earth™). In toolbox-centred architectures, the GIS application and data are situated on the user’s computer or local area network. In service-centred architectures, the tools and data are situated on remote computers or virtual machines that may run a variety of operating systems, accessed through Internet connections.

Reasons for this change are the increasing availability and bandwidth of the Internet, and also ownership and maintenance of data and/or analysis methods. For instance, data themselves may not be freely distributable, but certain derived products (such as visualisations or generalisations) may be. A service can be kept and maintained by the provider without end users having to bother about updating their installed software or data bases. The R system operates well under both toolbox-centred and service-centred architectures. Several experiments integrating R in the sensor web and in web-based workflows, either on the server side or on the client side, have been reported (Pebesma et al., 2011; Nüst et al., 2011; Bastin et al., 2013).

1.3.3 Further Reading on GIS

It seems appropriate to give some recommendations for further reading concerning GIS, not least because a more systematic treatment would not be appropriate here. Chrisman (2002) gives a concise and conceptually elegant introduction to GIS, with weight on using the data stored in the system; the domain focus is on land planning. A slightly older text by Burrough and McDonnell (1998) remains thorough, comprehensive, and perhaps a shade closer to the earth sciences in domain terms than Chrisman.

Two newer comprehensive introductions to GIS cover much of the same ground, but are published in colour. Heywood et al. (2006) contains less extra material than Longley et al. (2005), but both provide very adequate coverage of GIS as it is seen from within the GIS community today. To supplement these, Wise (2002) provides a lot of very distilled experience on the technicalities of handling geometries in computers in a compact form, often without dwelling on the computer science foundations; these foundations are given by Worboys and Duckham (2004). Neteler and Mitasova (2008) provide an excellent analytical introduction to GIS in their book, which also shows how to use the open source GRASS GIS, and how it can be interfaced with R. Krivoruchko (2011) gives a comprehensive guide to spatial statistical data analysis, focussed chiefly on ESRITM ArcGISTM, and including a chapter and an appendix covering the use of R with ArcGISTM.

It is harder to provide guidance with regard to service-centred architectures for GIS. The book by Shekar and Xiong (2008) is a monumental, forward-looking collection with strong roots in computer and information science, and reflects the ongoing embedding of GIS technologies into database systems far more than the standard texts. Two hands-on alternatives show how service-centred architectures can be implemented at low cost by non-specialists, working, for example in environmental advocacy groups, or volunteer search and rescue teams (Mitchell, 2005; Erle et al., 2005); their approach is certainly not academic, but gets the job done quickly and effectively.

In books describing the handling of spatial data for data analysts (looking at GIS from the outside), Waller and Gotway (2004, pp. 38–67) cover most of the key topics, with a useful set of references to more detailed treatments; Banerjee et al. (2004, pp. 10–18) give a brief overview of cartography sufficient to get readers started in the right direction.

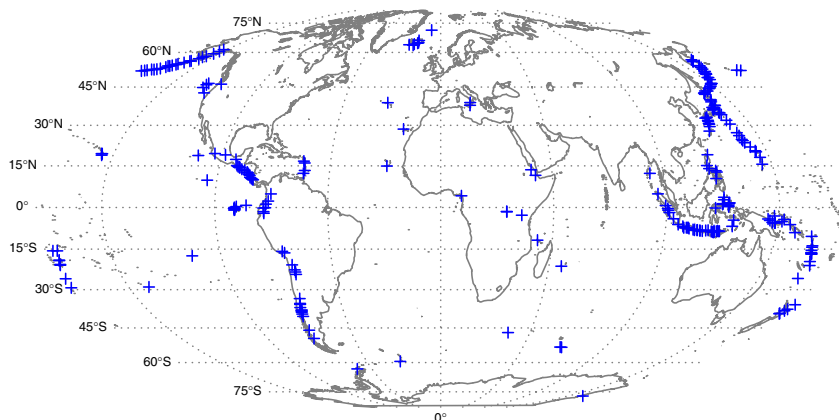


Fig. 1.1 Volcanoes of the world with last known eruption 1964 or later (+) (Source: National Geophysical Data Center)

1.4 Types of Spatial Data

Spatial data have spatial reference: they have coordinate values and a system of reference for these coordinates. As a fairly simple example, consider the locations of volcano peaks on the Earth. We could list the coordinates for all known volcanoes as pairs of longitude/latitude decimal degree values with respect to the prime meridian at Greenwich and zero latitude at the equator. The World Geodetic System (WGS84) is a frequently used representation of the Earth.

Suppose we are interested in the volcanoes that have shown activity between 1980 and 2000, according to some agreed seismic registration system. This data set consists of points only. When we want to draw these points on a (flat) map, we are faced with the problem of projection: we have to translate from the spherical longitude/latitude system to a new, non-spherical coordinate system, which inevitably changes their relative positions. In Fig. 1.1, these data are projected using a Mollweide projection, and, for reference purposes, coast lines have been added. Chapter 4 deals with coordinate reference systems, and with transformations between them.

If we also have the magnitude of the last observed eruption at the volcano, this information is called an *attribute*: it is non-spatial in itself, but this attribute information is believed to exist for each spatial entity (volcano).

Without explicit attributes, points usually carry implicit attributes, for example all points in this map have the constant implicit attribute – they mark a ‘volcano peak’, in contrast to other points that do not. We represent the *purely spatial* information of entities by data models. The different types of data models that we distinguish here include the following:

- Point*: a single point location, such as a GPS reading or a geocoded address
- Line*: a set of ordered points, connected by straight line segments
- Polygon*: an area, marked by one or more enclosing lines, possibly containing holes
- Grid*: a collection of points or rectangular cells, organised in a regular lattice

The first three are vector data models and represent entities as exactly as possible, while the final data model is a raster data model, representing continuous surfaces by using a regular tessellation. All spatial data consist of positional information, answering the question ‘where is it?’. In many applications these will be extended by attributes, answering the question ‘what is where?’; Chrisman (2002, pp. 37–69) distinguishes a range of spatial and spatio-temporal queries of this kind. Examples for these four basic data models and of types with attributes will now follow.

The location (x , y coordinates) of a volcano may be sufficient to establish its position relative to other volcanoes on the Earth, but for describing a single volcano we can use more information. Let us, for example try to describe the topography of a volcano. Figure 1.2 shows a number of different ways to represent a continuous surface (such as topography) in a computer.

First, we can use a large number of points on a dense regular *grid* and store the attribute *altitude* for each point to approximate the surface. Grey tones are used to specify classes of these points on Fig. 1.2a.

Second, we can form contour *lines* connecting ordered points with equal altitude; these are overlayed on the same figure, and separately shown on Fig. 1.2b. Note that in this case, the contour lines were derived from the point values on the regular grid.

A *polygon* is formed when a set of line segments forms a closed object with no lines intersecting. On Fig. 1.2a, the contour lines for higher altitudes are closed and form polygons.

Lines and polygons may have *attributes*, for example the 140 contour line of Fig. 1.2a may have the label ‘140 m above sea level’, or simply 140. Two closed contour lines have the attribute 160 m, but within the domain of this study area several non-closed contour lines have the attribute 110 m. The complete area inside the 140 m polygon (Fig. 1.2c) has the attribute ‘more than 140 m above sea level’, or >140. The area above the 160 m contour is represented by a polygon with a *hole* (Fig. 1.2d): its centre is part of the crater, which is below 160 m.

Polygons formed by contour lines of volcanoes usually have a more or less circular shape. In general, polygons can have arbitrary form, and may for certain cases even overlap. A special, but common case is when they represent the boundary of a single categorical variable, such as an administrative region. In that case, they cannot overlap and should divide up the entire study area: each point in the study area can and must be attributed to a single polygon, or lies on a boundary of one or more polygons.

A special form to represent spatial data is that of a grid: the values in each grid cell may represent an average over the area of the cell, or the value at the midpoint of the cell, or something more vague – think of image sensors. In the first case, we can see a grid as a special case of ordered points; in the second case, they are a collection of rectangular polygons. In any case, we can *derive* the position of each cell from the grid location, grid cell size, and the organisation of the grid cells. Grids are a common way to tessellate a plane. They are important because

- Devices such as digital cameras and remote sensing instruments register data on a regular grid
- Computer screens and projectors show data on a grid
- Many spatial or spatio-temporal models, such as climate models, discretise space by using a regular grid.

A comprehensive and up-to-date review of standards for representing geographical information is given by Kresse et al. (2012), including the relationship between the various standards.

Readers with a background in GI Science (Goodchild, 1992; Longley et al., 2005) may note that we have so far not distinguished discrete objects and events from continuous fields (Galton, 2004). Indeed, “point features with attributes” may represent equally well earth quakes with magnitudes (ob-

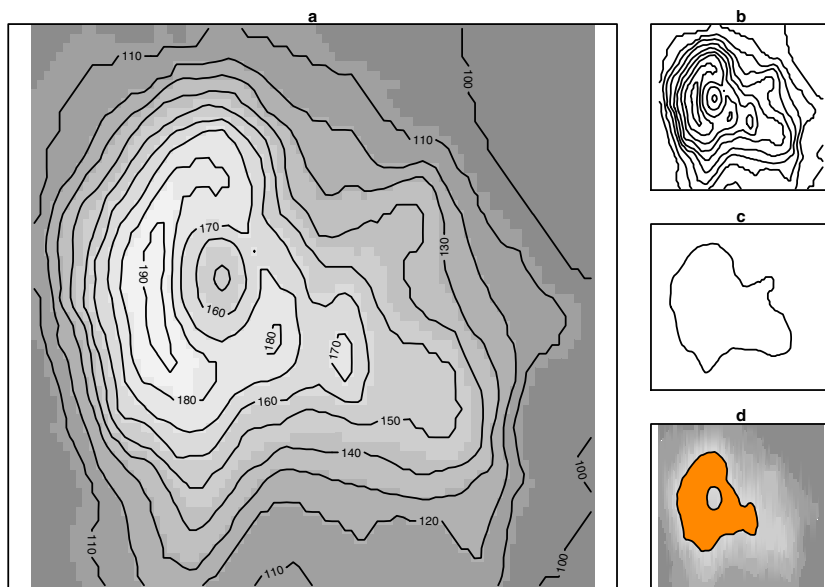


Fig. 1.2 Maunga Whau (Mt Eden) is one of about 50 volcanoes in the Auckland volcanic field. (a) Topographic information (altitude, m) for Maunga Whau on a 10×10 m grid, (b) contour lines, (c) 140 m contour line: a closed polygon, (d) area above 160 m (*hashed*): a polygon with a *hole*

jects/events, categorized as *point patterns* and dealt with in Chap. 7) as temperature readings from fixed sensors (fields, categorized as *geostatistical data* and dealt with in Chap. 8).

Representing these two very different types by one and the same class may give room to meaningless analyses such as spatially interpolating earthquake magnitudes, or summing temperatures over regions. As data coming from various external sources usually do not come with an indication whether a set of objects or a field is represented, automated mapping is not possible, which would leave this burden to the user. Not imposing such type restrictions may equally lead to error as to new findings.

1.5 Storage and Display

As R is open source, we can find out the meaning of every single bit and byte manipulated by the software if we need to do so. Most users will, however, be happy to find that this is unlikely to be required, and is left to a small group of developers and experts. They will rely on the fact that many users have seen, tested, or used the code before.

When running an R session, data are usually read or imported using explicit commands, after which all data are *kept in memory*; users may choose

to load a saved workspace or data objects. During an R session, the workspace can be saved to disk or chosen objects can be saved in a portable binary form for loading into the next session. When leaving an interactive R session, the question *Save workspace image?* may be answered positively to save results to disk. Saving the session history is a very useful way of documenting what has been done, and is recommended as normal practice – even better is to clean the file from mistyped commands, verify it is correct by rerunning it, and to give it an informative file name.

Despite the fact that computers have greater memory capacity than they used to, R may not be suitable for the analysis of massive data sets, because data being analysed is held in memory. Massive data sets may, for example come from satellite imagery, or detailed global coast line information. It is in such cases necessary to have some idea about data size and memory management and requirements. Under such circumstances it is often still possible to use R as an analysis engine on part of the data sets. Smaller useful data sets can be obtained by selecting a certain region or by sub-sampling, aggregating or generalising the original data. Chapters 2, 4 and 5 will give hints on how to do this.

Spatial data are usually displayed on maps, where the x - and y -axes show the coordinate values, with the aspect ratio chosen such that a unit in x equals a unit in y . Another property of maps is that elements are added for reference purposes, such as coast lines, rivers, administrative boundaries, or even satellite images.

Display of spatial data in R is a challenge on its own, and is dealt with in Chap. 3. For many users, the graphical display of statistical data is among the most compelling reasons to use R, as maps are traditionally amongst the strongest graphics we know.

The core R engine was not designed specifically for the display and analysis of maps, and the limited interactive facilities it offers have drawbacks in this area. Still, a large number of visualisations come naturally to R graphics, while they would take a substantial effort to accomplish in legacy GIS. For one thing, most GIS do not provide *conditioning plots*, where series of plots are organised in a regular lattice, share axes, and legends, and allow for systematic comparison across a large number of settings, scenarios, time, or other variables (e.g. Fig. 3.10). R provides on-screen graphics and has many graphics drivers, for example for vector graphics output to PostScript, Windows metafiles, PDF, and many bitmapped graphics formats. And, as mentioned, it works equally well as a front end or as a service providing back end for statistical analysis.

1.6 Applied Spatial Data Analysis

Statistical inference is concerned with drawing conclusions based on data and prior assumptions. The presence of a model of the data generating process may be more or less acknowledged in the analysis, but its reality will make

itself felt sooner or later. The model may be manifest in the design of data collection, in the distributional assumptions employed, and in many other ways. A key insight is that observations in space cannot in general be assumed to be mutually independent, and that observations that are close to each other are likely to be similar (*ceteris paribus*). This spatial patterning – spatial autocorrelation – may be treated as useful information about unobserved influences, but it does challenge the application of methods of statistical inference that assume the mutual independence of observations.

Not infrequently, the prior assumptions are not made explicit, but are rather taken for granted as part of the research tradition of a particular scientific subdiscipline. Too little attention typically is paid to the assumptions, and too much to superficial differences; for example Venables and Ripley (2002, p. 428) comment on the difference between the covariance function and the semi-variogram in geostatistics, that ‘[m]uch heat and little light emerges from discussions of their comparison’.

To illustrate the kinds of debates that rage in disparate scientific communities analysing spatial data, we sketch two current issues: red herrings in geographical ecology and the interpretation of spatial autocorrelation in urban economics.

The red herring debate in geographical ecology was ignited by Lennon (2000), who claimed that substantive conclusions about the impact of environmental factors on, for example species richness had been undermined by not taking spatial autocorrelation into account. Diniz-Filho et al. (2003) replied challenging not only the interpretation of the problem in statistical terms, but pointing out that geographical ecology also involves the scale problem, that the influence of environmental factors is moderated by spatial scale.

They followed this up in a study in which the data were sub-sampled to attempt to isolate the scale problem. But they begin: ‘It is important to note that we do not present a formal evaluation of this issue using statistical theory. . . , our goal is to illustrate heuristically that the often presumed bias due to spatial autocorrelation in OLS regression does not apply to real data sets’ (Hawkins et al., 2007, p. 376).

The debate continues with verve in Beale et al. (2007) and Diniz-Filho et al. (2007). This is quite natural, as doubts about the impacts of environmental drivers on species richness raise questions about, for example, the effects of climate change. How to analyse spatial data is obviously of importance within geographical ecology. However, Diniz-Filho et al. (2007, p. 850) conclude that ‘[w]hen multiple assumptions are not being met, as in the case of virtually all geographical analyses, can a result from any single method (whether spatial or non-spatial) be claimed to be better? . . . If different spatial methods themselves are unstable and generate conflicting results in real data, it makes no sense to claim that any particular method is always superior to any other’. Dray et al. (2012) review alternative views and suggest unifying approaches to community ecology in multivariate multiscale spatial analysis.

The urban economics debate is not as vigorous, but is of some practical interest, as it concerns the efficiency of services provided by local government. Revelli (2003) asks whether the spatial patterns observed in model residuals are a reaction to model misspecification, or do they signal the presence of substantive interaction between observations in space? In doing so, he reaches back to evocations of the same problem in the legacy literature of spatial statistics. As Cliff and Ord (1981, pp. 141–142) put it, ‘two adjacent supermarkets will compete for trade, and yet their turnover will be a function of general factors such as the distribution of population and accessibility’. They stress that ‘the presence of spatial autocorrelation may be attributable either to trends in the data or to interactions; ... [t]he choice of model must involve the scientific judgement of the investigator *and* careful testing of the assumptions’. When the fitted model is misspecified, it will be hard to draw meaningful conclusions, and the care advised by Cliff and Ord will be required.

One way of testing the assumptions is through changes in the policy context over time, where a behavioural model predicts changes in spatial autocorrelation – if the policy changes, the level of spatial interaction should change (Bivand and Szymanski, 1997; Revelli, 2003). Alternatives include using multiple levels in local government (Revelli, 2003), or different electoral settings, such as lame-duck administrations as controls (Bordignon et al., 2003). A recent careful study has used answers to a questionnaire survey to check whether interaction has occurred or not. It yields a clear finding that the observed spatial patterning in local government efficiency scores is related to the degree to which they compare their performance with that of other local government entities (Revelli and Tovmo, 2007).

This book will not provide explicit guidance on the choice of models, because the judgement of researchers in different scientific domains will vary. One aspect shared by both examples is that the participants stress the importance of familiarity with the core literature of spatial statistics. It turns out that many of the insights found there remain fundamental, despite the passage of time. Applied spatial data analysis seems to be an undertaking that, from time to time, requires the analyst to make use of this core literature.

Without attempting to be exhaustive in reviewing key books covering all the three acknowledged areas of spatial statistics – point processes, geostatistics, and areal data – we can make some choices. Bivand (2008, pp. 16–17) documents the enduring position of Ripley (1981)⁶ and Cliff and Ord (1981) in terms of paper citations. Ripley (1988) supplements and extends the earlier work, and is worth careful attention. The comprehensive text by Cressie (1993) is referred to very widely; careful reading of the often very short passages of relevance to a research problem can be highly rewarding. Schabenberger and Gotway (2005) cover much of the same material, incorporating advances made over the intervening period. Banerjee et al. (2004)

⁶ Reprinted in 2004.

show how the Bayesian approach to statistics can be used in applied spatial data analysis. A recent addition by Gaetan and Guyon (2010) shows clearly how the availability of R spatial statistics software is now interacting with the elaboration of statistical methods. The volume edited by Gelfand et al. (2010) provides a comprehensive overview of spatial statistics. Both Finkenstadt et al. (2006) and Cressie and Wikle (2011) provide recent material on spatio-temporal statistics.

Beyond the core statistical literature, many disciplines have their own traditions, often collated in widely used textbooks. Public health and disease mapping are well provided for by Waller and Gotway (2004), as is ecology by Fortin and Dale (2005). O’Sullivan and Unwin (2010) cover similar topics from the point of view of geography and GIS. Like Banerjee et al. (2004), the disciplinary texts differ from the core literature not only in the way theoretical material is presented, but also in the availability of the data sets used in the books for downloading and analysis. Haining (2003) is another book providing some data sets, and an interesting bridge to the use of Bayesian approaches in the geographies of health and crime. Despite its age, Bailey and Gatrell (1995) remains a good text, with support for its data sets in R packages.

In an *R News* summary, Ripley (2001) said that one of the reasons for the relatively limited availability of spatial statistics functions in R at that time was the success of the S-PLUSTM spatial statistics module (Kaluzny et al., 1998). Many of the methods for data handling and analysis are now available in R complement and extend those in the legacy S-PLUSTM module.

To summarise the approach to applied spatial data analysis adopted here, we can say that – as with the definition of geography as ‘what geographers do’ – applied spatial data analysis can best be understood by observing what practitioners do and how they do it. Since practitioners may choose to conduct analyses in different ways, it becomes vital to keep attention on ‘how they do it’, which R facilitates, with its unrivalled closeness to both data and the implementation of methods. It is equally important to create and maintain bridges between communities of practitioners, be they innovative statisticians or dedicated field scientists, or (rarely) both in the same person. The R Spatial community attempts to offer such opportunities, without necessarily prescribing or proscribing particular methods, and this approach will be reflected in this book.

1.7 R Spatial Resources

There are a range of resources for analysing spatial data with R, one being this book. In using the book, it is worth bearing in mind the close relationships between the increase in the availability of software for spatial data analysis on CRAN and the activities of the informal community of users interested in spatial data analysis. Indeed, without contributions, advice, bug reports, and

fruitful questions from users, very little would have been achieved. So before going on to present the structure of the book, we mention some of the more helpful online resources.

Since CRAN has grown to over 4,400 packages in March 2013, finding resources is not simple. One opportunity is to use the collection of ‘Task Views’ available on CRAN itself. One of these covers spatial data analysis, and another spatio-temporal data. Both are kept up to date. Other task views may also be relevant. These web pages are intended to be very concise, but because they are linked to the resources listed, including packages on CRAN, they can be considered as a kind of ‘shop window’. By installing the `ctv` package and executing the command `install.views("Spatial")`, you will install almost all the contributed packages needed to reproduce the examples in this book (which may be downloaded from the book website).

The spatial task view is available on all CRAN mirrors, but may be accessed directly⁷; it provides a very concise summary of available contributed packages. It also specifically links another resources, a mailing list dedicated to spatial data analysis with R. The R-sig-geo mailing list was started in 2003 after sessions on spatial statistics at the Distributed Statistical Computing conference organised in Vienna earlier the same year. By late 2007, the mailing list was being used by over 800 members; early 2013 this number has grown to over 2,750. The R-sig-geo list is meant to off-load the spatial topic traffic from the high-volume R-help mailing list (Fig. 2).

The archives of the mailing list are hosted in Zurich with the other R mailing list archives, and copies are held on Gmane and Nabble. This means that list traffic on an interesting thread can be accessed by general Internet search engines; a Google™ search on R `gstat` `kriging` picks up list traffic and relevant blogs easily.

1.8 Layout of the Book

This book is divided into two basic parts, the first presenting the shared R packages, functions, classes, and methods for handling spatial data. This part is of interest to users who need to access and visualise spatial data, but who are not initially concerned with drawing conclusions from analysing spatial data per se. The second part showcases more specialised kinds of spatial data analysis, in which the relative position of observations in space may contribute to understanding the data generation process. This part is not an introduction to spatial statistics in itself, and should be read with relevant textbooks and papers referred to in the chapters.

Chapters 2 through 6 introduce spatial data handling in R. Readers needing to get to work quickly may choose to read Chap. 4 first, and return

⁷ <http://CRAN.R-project.org/view=Spatial>

to other chapters later to see how things work. Those who prefer to see the naked structure first before using it will read the chapters in sequence, probably omitting technical subsections. The functions, classes, and methods are indexed, and so navigation from one section to another should be feasible.

Chapter 2 discusses in detail the classes for spatial data in R, as implemented in the **sp** package, and Chap. 3 discusses a number of ways of visualising for spatial data. Chapter 4 explains how coordinate reference systems work in the **sp** representation of spatial data in R, how they can be defined and how data can be transformed from one system to another, how spatial data can be imported into R or exported from R to GIS formats, and how R and the open source GRASS GIS are integrated. Chapter 5 covers methods for handling the classes defined in Chap. 2, especially for combining and integrating spatial data. Chapter 6 introduces the representation and handling of spatio-temporal data.

If we use the classification of Cressie (1993), we can introduce the applied spatial data analysis part of the book as follows: Chap. 7 covers the analysis of spatial point patterns, in which the relative position of points is compared with clustered, random, or regular generating processes. Chapter 8 presents the analysis of geostatistical data, with interpolation from values at observation points to prediction points. Chapter 9 deals with the statistical analysis of areal data, where the observed entities form a tessellation of the study area, and are often containers for data arising at other scales; Chap. 10 covers the special topic of disease mapping in R, and together they cover the analysis of lattice data, here termed areal data.

Data sets and code for reproducing the examples in this book are available from <http://www.asdar-book.org>; the website also includes support material and errata.

Part I

Handling Spatial Data in R

Handling Spatial Data

The key intuition underlying the development of the classes and methods in the **sp** package, and its closer dependent packages, is that users approaching R with experience of GIS will want to see ‘layers’, ‘coverages’, ‘rasters’, or ‘geometries’. Seen from this point of view, **sp** classes should be reasonably familiar, appearing to be well-known data models. On the other hand, for statistician users of R, ‘everything’ is a **data.frame**, a rectangular table with rows of observations on columns of variables. To permit the two disparate groups of users to play together happily, classes have grown that look like GIS data models to GIS and other spatial data people, and look and behave like data frames from the point of view of applied statisticians and other data analysts.

This part of the book describes the classes and methods of the **sp** package, and in doing so also provides a practical guide to the internal structure of many GIS data models, as R permits the user to get as close as desired to the data. However, users will not often need to know more than that of Chap. 4 to read in their data and start work. Visualisation is covered in Chap. 3, and so a statistician receiving a well-organised set of data from a collaborator may even be able to start making maps in two lines of code, one to read the data and one to plot the variable of interest using lattice graphics. Note that complete code examples, data sets, and other support material may be found on the book website.

If life was always so convenient, this part of the book could be much shorter than it is. But combining spatial data from different sources often means that much more insight is needed into the data models involved. The data models themselves are described in Chap. 2, and methods for handling and combining them are covered in Chap. 5, with substantial discussion of functions and operations provided in the **rgeos** package. Keeping track of which observation belongs to which geometry is also discussed here, seen from the GIS side as feature identifiers, and row names from the data frame side. In addition to

data import and export, Chap. 4 also describes the use and transformation of coordinate reference systems for **sp** classes, and integration of the open source GRASS GIS and R. Finally, Chap. 6 explains how the methods and classes introduced in Chap. 2 can be extended to spatio-temporal data.

Chapter 2

Classes for Spatial Data in R

2.1 Introduction

Many disciplines have influenced the representation of spatial data, both in analogue and digital forms. Surveyors, navigators, and military and civil engineers refined the fundamental concepts of mathematical geography, established often centuries ago by some of the founders of science, for example by al-Khwārizmī. Digital representations came into being for practical reasons in computational geometry, in computer graphics and hardware-supported gaming, and in computer-assisted design and virtual reality. The use of spatial data as a business vehicle has been spurred early in the present century by consumer wired and mobile broadband penetration and distributed server farms, with examples being Google Earth™, Google Maps™, and others. There are often interactions between the graphics hardware required and the services offered, in particular for the fast rendering of scene views.

In addition, space and other airborne technologies have vastly increased the volumes and kinds of spatial data available. Remote sensing satellites continue to make great contributions to earth observation, with multi-spectral images supplementing visible wavelengths. The Shuttle Radar Topography Mission (SRTM) in February 2000 has provided elevation data for much of the earth. Other satellite-borne sensor technologies are now vital for timely storm warnings, amongst other things. These complement terrestrial networks monitoring, for example, lightning strikes and the movement of precipitation systems by radar.

Surveying in the field has largely been replaced by aerial photogrammetry, mapping using air photographs usually exposed in pairs of stereo images. Legacy aerial photogrammetry worked with analogue images, and many research laboratories and mapping agencies have large archives of air photographs with coverage beginning from the 1930s. These images can be scanned to provide a digital representation at chosen resolutions.

While satellite imagery usually contains metadata giving the scene frame – the sensor direction in relation to the earth at scan time – air photographs need to be registered to known ground control points.

These ground control points were ‘known’ from terrestrial triangulation, but could be in error. The introduction of Global Positioning System (GPS) satellites has made it possible to correct the positions of existing networks of ground control points. The availability of GPS receivers has also made it possible for data capture in the field to include accurate positional information in a known coordinate reference system. This is conditioned by the requirement of direct line-of-sight to a sufficient number of satellites, not easy in mountain valleys or in city streets bounded by high buildings. Despite this limitation, around the world the introduction of earth observation satellites and revised ground control points have together caused breaks of series in published maps, to take advantage of the greater accuracy now available. This means that many older maps cannot be matched to freshly acquired position data without adjustment.

All of these sources of spatial data involve points, usually two real numbers representing position in a known coordinate reference system. It is possible to go beyond this simple basis by combining pairs of points to form line segments, combining line segments to form polylines, networks or polygons, or regular grid centres. Grids can be defined within a regular polygon, usually a rectangle, with given resolution – the size of the grid cells. All these definitions imply choices of what are known in geographical information systems (GIS) as data models, and these choices have most often been made for pragmatic reasons. All the choices also involve trade-offs between accuracy, feasibility, and cost. Standards for representing geographical information are reviewed by Kresse et al. (2012).

Artificial objects are easiest to represent, like roads, bridges, buildings, or similar structures. They are crisply defined, and are not subject to natural change – unlike placing political borders along the centre lines or deepest channels of meandering rivers. Shorelines are most often natural and cannot be measured accurately without specifying measurement scale. Boundaries between areas of differing natural land cover are frequently indeterminate, with gradations from one land cover category to another. Say that we want to examine the spatial distribution of a species by land cover category; our data model of how to define the boundary between categories will affect the outcome, possibly strongly. Something of the same affects remote sensing, because the reported values of the observed pixels will hide sub-pixel variation.

It is unusual for spatial data to be defined in three dimensions, because of the close links between cartography and data models for spatial data. When there are multiple observations on the same attribute at varying heights or depths, they are most often treated as separate layers. GIS-based data models do not fit time series data well either, even though some environmental monitoring data series are observed in three dimensions and time. Some GIS software can handle voxels, the 3D equivalent of pixels – 2D raster cells – but

the third dimension in spatial data is not handled satisfactorily, as is the case in computer-assisted design or medical imaging. On the other hand, many GIS packages do provide a 2.5D intermediate solution for viewing, by draping thematic layers, like land cover or a road network, over a digital elevation model. In this case, however, there is no ‘depth’ in the data model, as we can see when a road tunnel route is draped over the mountain it goes through.

2.2 Classes and Methods in R

In Chap. 1, we described R as a language and environment for data analysis. Although this is not the place to give an extended introduction to R,¹ it will be useful to highlight some of its features (see also Teetor, 2011, for an up-to-date introduction). In this book, we will be quoting R commands in the text, showing which commands a user could give, and how the non-graphical output might be represented when printed to the console.

Of course, R can be used as a calculator to carry out simple tasks, where no values are assigned to variables, and where the results are shown without being saved, such as the area of a circle of radius 10:

```
> pi * 10^2
[1] 314.1593
```

Luckily, π is a built-in constant in R called `pi`, and so entering a rounded version is not needed. So this looks like a calculator, but appearances mislead. The first misleading impression is that the arithmetic is simply being ‘done’, while in fact it is being translated (parsed) into functions (operators) with arguments first, and then evaluated:

```
> "*" (pi, "^" (10, 2))
[1] 314.1593
```

When the operators or functions permit, vectors of values may be used as readily as scalar values (which are vectors of unit length) – here the ‘:’ operator is used to generate an integer sequence of values:

```
> pi * (1:10)^2
[1] 3.141593 12.566371 28.274334 50.265482 78.539816 113.097336
[7] 153.938040 201.061930 254.469005 314.159265
```

The second misapprehension is that what is printed to the console is the ‘result’, when it is actually the outcome of applying the appropriate `print`

¹ Free documentation, including the very useful ‘An Introduction to R’ (Venables et al., 2013), may be downloaded from CRAN.

method for the class of the ‘result’, with default arguments. If we store the value returned for the area of our circle in variable `x` using the assignment operator `<-`, we can print `x` with the default number of digits, or with more if we so please. Just typing the variable name at the interactive prompt invokes the appropriate print method, but we can also pass it to the print method explicitly:

```
> x <- pi * 10^2
> x

[1] 314.1593

> print(x)

[1] 314.1593

> print(x, digits = 12)

[1] 314.159265359
```

We can say that the variable `x` contains an object of a particular class, in this case:

```
> class(x)

[1] "numeric"

> typeof(x)

[1] "double"
```

where `typeof` returns the storage mode of the object in variable `x`. It is the class of the object that determines the method that will be used to handle it; if there is no specific method for that class, it may be passed to a default method. These methods are also known as generic functions, often including at least `print`, `plot`, and `summary` methods. In the case of the `print` method, `numeric` is not provided for explicitly, and so the default method is used. The `plot` method, as its name suggests, will use the current graphics device to make a visual display of the object, dispatching to a specific method for the object class if provided. In comparison with the `print` method, the `summary` method provides a qualified view of the data, highlighting the key features of the object.

When the S language was first introduced, it did not use class/method mechanisms at all. They were introduced in Chambers and Hastie (1992) and S version 3, in a form that is known as **S3** classes or old-style classes. These classes were not formally defined, and ‘just grew’; the vast majority of objects returned by model fitting functions belong to old-style classes. Using a non-spatial example from the standard data set `cars`, we can see that it is an object of class `data.frame`, stored in a `list`, which is a vector whose components can be arbitrary objects; `data.frame` has both names and summary methods:

```

> class(cars)

[1] "data.frame"

> typeof(cars)

[1] "list"

> names(cars)

[1] "speed" "dist"

> summary(cars)

      speed      dist
Min.   : 4.0   Min.   : 2.00
1st Qu.:12.0   1st Qu.: 26.00
Median :15.0   Median : 36.00
Mean   :15.4   Mean    : 42.98
3rd Qu.:19.0   3rd Qu.: 56.00
Max.   :25.0   Max.    :120.00

```

The `data.frame` contains two variables, one recording the speed of the observed cars in mph, the other the stopping distance measured in feet – the observations were made in the 1920s. When uncertain about the structure of something in our R workspace, revealed for example by using the `ls` function for listing the contents of the workspace, the `str`² method often gives a clear digest, including the size and class:

```

> str(cars)

'data.frame':      50 obs. of  2 variables:
 $ speed: num 4 4 7 7 8 ...
 $ dist : num 2 10 4 22 16 ...

```

Data frames are containers for data used everywhere in S since their full introduction in Chambers and Hastie (1992, pp. 45–94). Recent and shorter introductions to data frames are given by Adler (2010, pp. 87–88), Teetor (2011, pp. 100–101, 122–143), and Dalgaard (2008, pp. 20–25) and in the online documentation (Venables et al., 2013, pp. 29–31 in the R 3.0.0 release). Data frames view the data as a rectangle of rows of observations on columns of values of variables of interest. The representation of the values of the variables of interest can include integer and floating point numeric types, logical, character, and derived classes. One very useful derived class is the factor, which is represented as integers pointing to character levels, such as ‘forest’ or ‘arable’. Printed, the values look like character values, but are not – when a data frame is created, all character variables included in it are converted to factor by default. Data frames also have unique row names, represented as an integer or character vector or as an internal mechanism to

² `str` can take additional arguments to control its output.

signal that the sequence from 1 to the number of rows in the data frame are used. The `row.names` function is used to access and assign data frame row names.

One of the fundamental abstractions used in R is the `formula` introduced in Chambers and Hastie (1992, pp. 13–44) – an online summary may be found in Venables et al. (2013, pp. 54–56 in the R 3.0.0 release). The abstraction is intended to make statistical modelling as natural and expressive as possible, permitting the analyst to focus on the substantive problem at hand. Because the `formula` abstraction is used in very many contexts, it is worth some attention. A `formula` is most often two-sided, with a response variable to the left of the `~` (tilde) operator, and in this case a determining variable on the right:

```
> class(dist ~ speed)
```

```
[1] "formula"
```

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

```
> lm(dist ~ speed, data = cars)
```

Call:

```
lm(formula = dist ~ speed, data = cars)
```

Coefficients:

(Intercept)	speed
-17.579	3.932

This is a simple example, but very much more can be done with the `formula` abstraction. If we create a factor for the `speed` variable by cutting it at its quartiles, we can contrast how the `plot` method displays the relationship between two numerical variables and a numerical variable and a factor (shown in Fig. 2.1):

```
> cars$qspeed <- cut(cars$speed, breaks = quantile(cars$speed),
+   include.lowest = TRUE)
> is.factor(cars$qspeed)
```

```
[1] TRUE
```

```
> plot(dist ~ speed, data = cars)
> plot(dist ~ qspeed, data = cars)
```

Finally, let us see how the `formula` with the right-hand side factor is handled by `lm` – it is converted into ‘dummy’ variable form automatically:

```
> lm(dist ~ qspeed, data = cars)
```

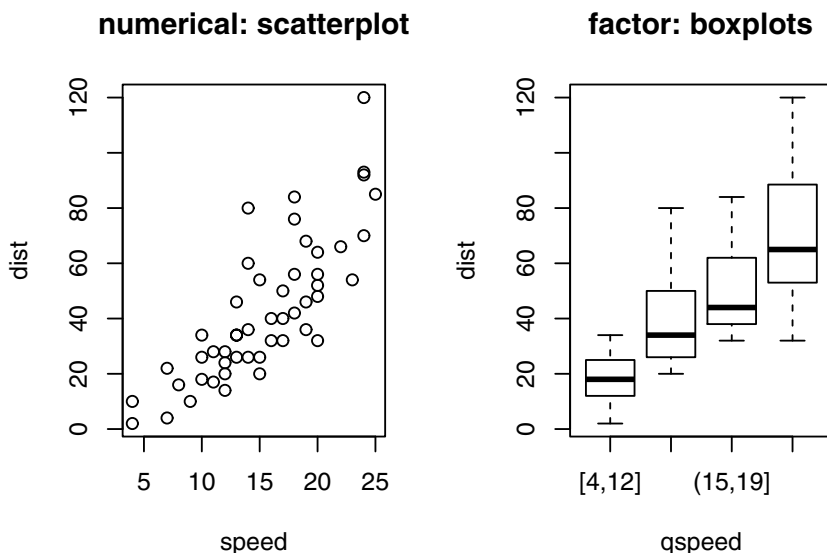


Fig. 2.1 Plot methods for a formula with numerical (*left panel*) and factor (*right panel*) right-hand side variables

Call:

```
lm(formula = dist ~ qspeed, data = cars)
```

Coefficients:

(Intercept)	qspeed(12,15]	qspeed(15,19]	qspeed(19,25]
18.20	21.98	31.97	51.13

Variables in the `formula` may also be transformed in different ways, for example using `log`. The `formula` is carried through into the object returned by model fitting functions to be used for prediction from new data provided in a `data.frame` with the same column names as the right-hand side variables, and the same level names if the variable is a factor.

New-style (S4) classes were introduced in the S language at release 4, and in Chambers (1998), and are described by Venables and Ripley (2000, pp. 75–121), by Chambers (2008, pp. 331–410), and in subsequent documentation installed with R. Old-style classes are most often simply lists with attributes; they are not defined formally. Although users usually do not change values inside old-style classes, there is nothing to stop them doing so, for example changing the representation of coordinates from floating point to integer numbers. This means that functions need to check, among other things, whether components of a class exist, and whether they are represented correctly, before they can be handled. The central advantage of new-style classes is that they have formal definitions that specify the name and type of the components, called *slots*, that they contain. This simplifies the writing, maintenance, and use of the classes, because their format is known from the definition.

Because the classes provided by the **sp** package are new-style classes, we will be seeing how such classes work in practice below. In particular, we will be referring to the slots in class definitions; slots are specified in the definition as the representation of what the class contains. Many methods are written for the classes to be introduced in the remainder of this chapter, in particular coercion methods for changing the way an object is represented from one class to another. New-style classes can also check the validity of objects being created, for example to stop the user from filling slots with data that do not conform to the definition.

2.3 Spatial Objects

The foundation class is the **Spatial** class, with just two slots. The first is a bounding box, a matrix of numerical coordinates with column names `c('min', 'max')`, and at least two rows, with the first row eastings (x -axis) and the second northings (y -axis). Most often the bounding box is generated automatically from the data in subclasses of **Spatial**. The second is a **CRS** class object defining the coordinate reference system, and may be set to 'missing', represented by `NA` in R, by `CRS(as.character(NA))`, its default value. Operations on **Spatial*** objects should update or copy these values to the new **Spatial*** objects being created. We can use `getClass` to return the complete definition of a class, including its slot names and the types of their contents:

```
> library(sp)
> getClass("Spatial")
Class "Spatial" [package "sp"]

Slots:

Name:      bbox proj4string
Class:     matrix      CRS

Known Subclasses:
Class "SpatialPoints", directly
Class "SpatialGrid", directly
Class "SpatialLines", directly
Class "SpatialPolygons", directly
Class "SpatialPointsDataFrame", by class "SpatialPoints", distance 2
Class "SpatialPixels", by class "SpatialPoints", distance 2
Class "SpatialGridDataFrame", by class "SpatialGrid", distance 2
Class "SpatialLinesDataFrame", by class "SpatialLines", distance 2
Class "SpatialPixelsDataFrame", by class "SpatialPoints", distance 3
Class "SpatialPolygonsDataFrame", by class "SpatialPolygons",
distance 2
```

As we see, `getClass` also returns known subclasses, showing the classes that include the **Spatial** class in their definitions. This also shows where we are

going in this chapter, moving from the foundation class to richer representations. But we should introduce the coordinate reference system (CRS) class very briefly; we will return to its description in Chap. 4.

```
> getClass("CRS")
Class "CRS" [package "sp"]
```

Slots:

```
Name:   projargs
Class:  character
```

The class has a character string as its only slot value, which may be a missing value. If it is not missing, it should be a PROJ.4-format string describing the projection (more details are given in Sect. 4.1.2). For geographical coordinates, the simplest such string is "+proj=longlat", using "longlat", which also shows that eastings always go before northings in **sp** classes. Let us build a simple **Spatial** object from a bounding box matrix, and a missing coordinate reference system:

```
> m <- matrix(c(0, 0, 1, 1), ncol = 2, dimnames = list(NULL,
+   c("min", "max")))
> crs <- CRS(projargs = as.character(NA))
> crs
```

CRS arguments: NA

```
> S <- Spatial(bbox = m, proj4string = crs)
> S
```

An object of class "Spatial"

Slot "bbox":

	min	max
[1,]	0	1
[2,]	0	1

Slot "proj4string":

CRS arguments: NA

We could have used **new** methods to create the objects, but prefer to use helper functions with the same names as the classes that they instantiate. If the object is known not to be projected, a sanity check is carried out on the coordinate range (which here exceeds the feasible range for geographical coordinates):

```
> bb <- matrix(c(350, 85, 370, 95), ncol = 2, dimnames = list(NULL,
+   c("min", "max")))
> Spatial(bb, proj4string = CRS("+proj=longlat"))
```

Error in validityMethod(object) :

Geographical CRS given to non-conformant data: 370 95

The definition of this class, and classes inheriting from it, does not include cartographic symbology, understood as specifications of symbols, their sizes, shapes or colours (Slocum et al., 2005). In **sp**, choices affecting the symbolic representation of objects are made when visualisation methods are used on objects, explicitly as arguments to those methods as described in Chap. 3.

2.4 SpatialPoints

The `SpatialPoints` class is the first subclass of `Spatial`, and a very important one. The extension of `SpatialPoints` to other subclasses means that explaining how this class works will yield benefits later on. In this section, we also look at methods for `Spatial*` objects, and at extending `Spatial*` objects to include attribute data, where each spatial entity, here a point, is linked to a row in a data frame. We take `Spatial*` objects to be subclasses of `Spatial`, and the best place to start is with `SpatialPoints`. Note also that we will refer to spatial objects with which data may be associated as *features*.

A two-dimensional point can be described by a pair of numbers (x, y) , defined over a known region. As Herring (2011, p. 20) puts it: “[a] point is a 0-dimensional geometric object and represents a single location in coordinate space” (see also Kresse et al., 2012, pp. 505–506). To represent geographical phenomena, the maximum known region is the earth, and the pair of numbers measured in degrees are a geographical coordinate, showing where our point is on the globe. The pair of numbers define the location on the sphere exactly, but if we represent the globe more accurately by an ellipsoid model, such as the World Geodetic System 1984 – introduced after satellite measurements corrected our understanding of the shape of the earth – that position shifts slightly. Geographical coordinates can extend from latitude 90° to -90° in the north–south direction, and from longitude 0° to 360° or equivalently from -180° to 180° in the east–west direction. The Poles are fixed, but where the longitudes fall depends on the choice of prime meridian, most often Greenwich just east of London. This means that geographical coordinates define a point on the earth’s surface unequivocally if we also know which ellipsoid model and prime meridian were used; the concept of datum, relating the ellipsoid to the distance from the centre of the earth, is introduced on p. 84.

Using the standard `read.table` function, we read in a data file with the positions of CRAN mirrors across the world in 2005. We extract the two columns with the longitude and latitude values into a matrix, and use `str` to view a digest:

```
> CRAN_df <- read.table("CRAN051001a.txt", header = TRUE)
> CRAN_mat <- cbind(CRAN_df$long, CRAN_df$lat)
> row.names(CRAN_mat) <- 1:nrow(CRAN_mat)
> str(CRAN_mat)

num [1:54, 1:2] 153 145 ...
- attr(*, "dimnames")=List of 2
..$ : chr [1:54] "1" "2" ...
..$ : NULL
```

The `SpatialPoints` class extends the `Spatial` class by adding a `coords` slot, into which a matrix of point coordinates can be inserted.

```
> getClass("SpatialPoints")
```

```
Class "SpatialPoints" [package "sp"]
```

```
Slots:
```

```
Name:      coords      bbox proj4string
Class:      matrix      matrix      CRS
```

```
Extends: "Spatial"
```

```
Known Subclasses:
```

```
Class "SpatialPointsDataFrame", directly
```

```
Class "SpatialPixels", directly
```

```
Class "SpatialPixelsDataFrame", by class "SpatialPixels", distance 2
```

It has a `summary` method that shows the bounding box, whether the object is projected (here `FALSE`, because the string `"longlat"` is included in the projection description), and the number of rows of coordinates. Classes in `sp` are not atomic: there is no `SpatialPoint` class that is extended by `SpatialPoints`. This is because R objects are vectorised by nature, not atomic. A `SpatialPoints` object may, however, consist of a single point. This class is in some ways similar to the `MultiPoint` definition given by Herring (2011, p. 20) and Kresse et al. (2012, pp. 505–506), but is not taken formally as a `GeometryCollection` in their terms.

```
> llCRS <- CRS("+proj=longlat +ellps=WGS84")
> CRAN_sp <- SpatialPoints(CRAN_mat, proj4string = llCRS)
> summary(CRAN_sp)
```

```
Object of class SpatialPoints
```

```
Coordinates:
```

```
              min      max
coords.x1 -122.95000 153.0333
coords.x2  -37.81667  57.0500
Is projected: FALSE
proj4string : [+proj=longlat +ellps=WGS84]
Number of points: 54
```

`SpatialPoints` objects may have more than two dimensions, but plot methods for the class use only the first two.

2.4.1 Methods

Methods are available to access the values of the slots of `Spatial` objects. The `bbox` method returns the bounding box of the object, and is used both for preparing plotting methods (see Chap. 3) and internally in handling data objects. The first row reports the west–east range and the second the south–north direction. If we want to take a subset of the points in a `SpatialPoints` object, the bounding box is reset, as we will see.

```
> bbox(CRAN_sp)
```

```

              min      max
coords.x1 -122.95000 153.0333
coords.x2  -37.81667  57.0500

```

First, the other generic method for all `Spatial` objects, `proj4string`, will be introduced. The basic method reports the projection string contained as a CRS object in the `proj4string` slot of the object, but it also has an assignment form, allowing the user to alter the current value, which can also be a CRS object containing a character NA value:

```

> proj4string(CRAN_sp)

[1] "+proj=longlat +ellps=WGS84"

> proj4string(CRAN_sp) <- CRS(as.character(NA))
> proj4string(CRAN_sp)

[1] NA

> proj4string(CRAN_sp) <- llCRS

```

Extracting the coordinates from a `SpatialPoints` object as a numeric matrix is as simple as using the `coordinates` method. Like all matrices, the indices can be used to choose subsets, for example CRAN mirrors located in Brazil in 2005:

```

> brazil <- which(CRAN_df$loc == "Brazil")
> brazil

[1] 4 5 6 7 8

> coordinates(CRAN_sp)[brazil, ]

  coords.x1 coords.x2
4 -49.26667 -25.41667
5 -42.86667 -20.75000
6 -43.20000 -22.90000
7 -47.63333 -22.71667
8 -46.63333 -23.53333

```

In addition, a `SpatialPoints` object can also be accessed by index, using the "[" operator, here on the coordinate values treated as an entity. The object returned is of the same class, and retains the projection information, but has a new bounding box:

```

> summary(CRAN_sp[brazil, ])

Object of class SpatialPoints
Coordinates:
              min      max
coords.x1 -49.26667 -42.86667
coords.x2 -25.41667 -20.75000
Is projected: FALSE
proj4string : [+proj=longlat +ellps=WGS84]
Number of points: 5

```

The "[" operator also works for negative indices, which remove those coordinates from the object, here by removing mirrors south of the Equator:

```
> south_of_equator <- which(coordinates(CRAN_sp)[, 2] <
+ 0)
> summary(CRAN_sp[-south_of_equator, ])
```

```
Object of class SpatialPoints
Coordinates:
      min      max
coords.x1 -122.95 140.10
coords.x2  24.15  57.05
Is projected: FALSE
proj4string : [+proj=longlat +ellps=WGS84]
Number of points: 45
```

Because `summary` and `print` methods are so common in R, we used them here without special mention. They are provided for `sp` classes, with `summary` reporting the number of spatial entities, the projection information, and the bounding box, and `print` gives a view of the data in the object. As usual in R, the actual underlying data and the output of the `print` method may differ, for example in the number of digits shown.

An important group of methods for visualisation of `Spatial*` objects are presented in detail in Chap. 3; each such object class has a `plot` method. Other methods will also be introduced in Chap. 5 for combining (overlying) different `Spatial*` objects – named `over` methods, and for sampling from `Spatial` objects.

2.4.2 Data Frames for Spatial Point Data

We described data frames on p. 25, and we now show how our `SpatialPoints` object can be taught to behave like a `data.frame`. Here we use numbers in sequence to index the points and the rows of our data frame, because neither the place names nor the countries are unique.

```
> str(row.names(CRAN_df))
chr [1:54] "1" "2" ...
```

What we would like to do is to associate the correct rows of our data frame object with ‘their’ point coordinates – it often happens that data are collected from different sources, and the two need to be merged. The `SpatialPointsDataFrame` class is the container for this kind of spatial point information, and can be constructed in a number of ways, for example from a data frame and a matrix of coordinates. If the matrix of point coordinates has row names and the `match.ID` argument is set to its default value of `TRUE`, then the matrix row names are checked against the row names of the data frame. If they match, but are not in the same order, the data frame rows are re-ordered to suit the

points. If they do not match, no `SpatialPointsDataFrame` is constructed. Note that the new object takes two indices, the first for the spatial object, the second, if given, for the column. Giving a single index number, or range of numbers, or column name or names returns a new `SpatialPointsDataFrame` with the requested columns. Using other extraction operators, especially the `$` operator, returns the data frame column referred to. These operators mimic the equivalent ones for other standard R classes as far as possible.

```
> CRAN_spdf1 <- SpatialPointsDataFrame(CRAN_mat, CRAN_df,
+   proj4string = llCRS, match.ID = TRUE)
> CRAN_spdf1[4, ]

      coordinates      place  north   east   loc    long
4 (-49.2667, -25.4167) Curitiba 25d25'S 49d16'W Brazil -49.26667
      lat
4 -25.41667

> str(CRAN_spdf1$loc)

Factor w/ 30 levels "Australia","Austria",...: 1 1 2 3 3 ...

> str(CRAN_spdf1[["loc"]])

Factor w/ 30 levels "Australia","Austria",...: 1 1 2 3 3 ...
```

If we re-order the data frame at random using `sample`, we still get the same result, because the data frame is re-ordered to match the row names of the points:

```
> s <- sample(nrow(CRAN_df))
> CRAN_spdf2 <- SpatialPointsDataFrame(CRAN_mat, CRAN_df[s,
+   ], proj4string = llCRS, match.ID = TRUE)
> all.equal(CRAN_spdf2, CRAN_spdf1)

[1] TRUE

> CRAN_spdf2[4, ]

      coordinates      place  north   east   loc    long
4 (-49.2667, -25.4167) Curitiba 25d25'S 49d16'W Brazil -49.26667
      lat
4 -25.41667
```

But if we have non-matching ID values, created by pasting pairs of letters together and sampling an appropriate number of them, the result is an error:

```
> CRAN_df1 <- CRAN_df
> row.names(CRAN_df1) <- sample(c(outer(letters, letters,
+   paste, sep = "")), nrow(CRAN_df1))

> CRAN_spdf3 <- SpatialPointsDataFrame(CRAN_mat, CRAN_df1,
+   proj4string = llCRS, match.ID = TRUE)

Error in SpatialPointsDataFrame(CRAN_mat, CRAN_df1,
proj4string = llCRS, : row.names of data and coords do not
match
```

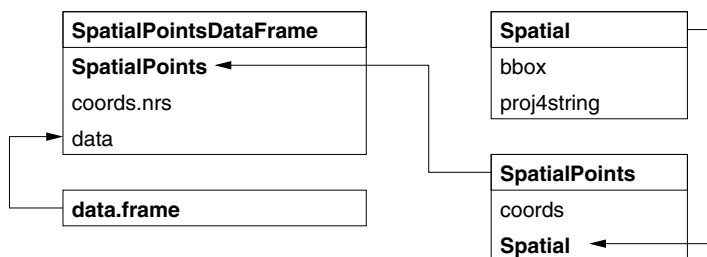


Fig. 2.2 Spatial points classes and their slots; *arrows* show subclass extensions

Let us examine the contents of objects of the `SpatialPointsDataFrame` class, shown in Fig. 2.2. Because the class extends `SpatialPoints`, it also inherits the information contained in the `Spatial` class object. The `data` slot is where the information from the data frame is kept, in a `data.frame` object.

```
> getClass("SpatialPointsDataFrame")

Class "SpatialPointsDataFrame" [package "sp"]

Slots:

Name:      data  coords.nrs      coords      bbox proj4string
Class:  data.frame      numeric      matrix      matrix      CRS

Extends:
Class "SpatialPoints", directly
Class "Spatial", by class "SpatialPoints", distance 2

Known Subclasses:
Class "SpatialPixelsDataFrame", directly, with explicit coerce
```

The `Spatial*DataFrame` classes have been designed to behave as far as possible like data frames, both with respect to standard methods such as `names`, and more demanding modelling functions like `model.frame` used in very many model fitting functions using `formula` and `data` arguments:

```
> names(CRAN_spdf1)

[1] "place" "north" "east"  "loc"   "long"  "lat"

> str(model.frame(lat ~ long, data = CRAN_spdf1), give.attr = FALSE)

'data.frame':      54 obs. of  2 variables:
 $ lat : num -27.5 -37.8 ...
 $ long: num 153 145 ...
```

Making our `SpatialPointsDataFrame` object from a matrix of coordinates and a data frame with or without ID checking is only one way to reach our goal, and others may be more convenient. We can construct the object by

giving the `SpatialPointsDataFrame` function a `SpatialPoints` object as its first argument:

```
> CRAN_spdf4 <- SpatialPointsDataFrame(CRAN_sp, CRAN_df)
> all.equal(CRAN_spdf4, CRAN_spdf2)
```

```
[1] TRUE
```

We can also assign coordinates to a data frame – this approach modifies the original data frame. The coordinate assignment function can take a matrix of coordinates with the same number of rows as the data frame on the right-hand side, or an integer vector of column numbers for the coordinates, or equivalently a character vector of column names, assuming that the required columns already belong to the data frame.

```
> CRAN_df0 <- CRAN_df
> coordinates(CRAN_df0) <- CRAN_mat
> proj4string(CRAN_df0) <- llCRS
> all.equal(CRAN_df0, CRAN_spdf2)
```

```
[1] TRUE
```

```
> str(CRAN_df0, max.level = 2)
```

```
Formal class 'SpatialPointsDataFrame' [package "sp"] with 5 slots
..@ data :'data.frame': 54 obs. of 6 variables:
..@ coords.nrs : num(0)
..@ coords : num [1:54, 1:2] 153 145 ...
.. ..- attr(*, "dimnames")=List of 2
..@ bbox : num [1:2, 1:2] -123 -37.8 ...
.. ..- attr(*, "dimnames")=List of 2
..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slots
```

Objects created in this way differ slightly from those we have seen before, because the `coords.nrs` slot is now used, and the coordinates are moved from the `data` slot to the `coords` slot, but the objects are otherwise the same:

```
> CRAN_df1 <- CRAN_df
> names(CRAN_df1)
```

```
[1] "place" "north" "east" "loc" "long" "lat"
```

```
> coordinates(CRAN_df1) <- c("long", "lat")
> proj4string(CRAN_df1) <- llCRS
> str(CRAN_df1, max.level = 2)
```

```
Formal class 'SpatialPointsDataFrame' [package "sp"] with 5 slots
..@ data :'data.frame': 54 obs. of 4 variables:
..@ coords.nrs : int [1:2] 5 6
..@ coords : num [1:54, 1:2] 153 145 ...
.. ..- attr(*, "dimnames")=List of 2
..@ bbox : num [1:2, 1:2] -123 -37.8 ...
.. ..- attr(*, "dimnames")=List of 2
..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slots
```

Transect and tracking data may also be represented as points, because the observation at each point contributes information that is associated with the point itself, rather than the line as a whole. Sequence numbers can be entered into the data frame to make it possible to trace the points in order, for example as part of a `SpatialLines` object as we see in the Sect. 2.5.

As an example, we use a data set³ from satellite telemetry of a single loggerhead turtle crossing the Pacific from Mexico to Japan (Nichols et al., 2000).

```
> turtle_df <- read.csv("seamap105_mod.csv")
> summary(turtle_df)
```

	id	lat	lon
Min.	: 1.00	Min. :21.57	Min. : -179.88
1st Qu.:	99.25	1st Qu.:24.36	1st Qu.: -147.38
Median	:197.50	Median :25.64	Median : -119.64
Mean	:197.50	Mean :27.21	Mean : -21.52
3rd Qu.:	295.75	3rd Qu.:27.41	3rd Qu.: 153.66
Max.	:394.00	Max. :39.84	Max. : 179.93


```
      obs_date
01/02/1997 04:16:53: 1
01/02/1997 05:56:25: 1
01/04/1997 17:41:54: 1
01/05/1997 17:20:07: 1
01/06/1997 04:31:13: 1
01/06/1997 06:12:56: 1
(Other)                :388
```

Before creating a `SpatialPointsDataFrame`, we will timestamp the observations, and re-order the input data frame by timestamp to make it easier to add months to Fig. 2.3, to show progress westwards across the Pacific (see Chap. 6 for a full treatment of spatio-temporal data objects):

```
> timestamp <- as.POSIXlt(strptime(as.character(turtle_df$obs_date),
+   "%m/%d/%Y %H:%M:%S"), "GMT")
> turtle_df1 <- data.frame(turtle_df, timestamp = timestamp)
> turtle_df1$lon <- ifelse(turtle_df1$lon < 0, turtle_df1$lon +
+   360, turtle_df1$lon)
> turtle_sp <- turtle_df1[order(turtle_df1$timestamp),
+   ]
> coordinates(turtle_sp) <- c("lon", "lat")
> proj4string(turtle_sp) <- CRS("+proj=longlat +ellps=WGS84")
```

The input data file is as downloaded, but without columns with identical values for all points, such as the number of the turtle (07667).

2.5 SpatialLines

Lines have been represented in S in a simple form as a sequence of points (see Becker et al., 1988; Murrell, 2011, pp. 79–84), based on lowering the

³ Data downloaded with permission from SEAMAP (Read et al., 2003), data set 105.

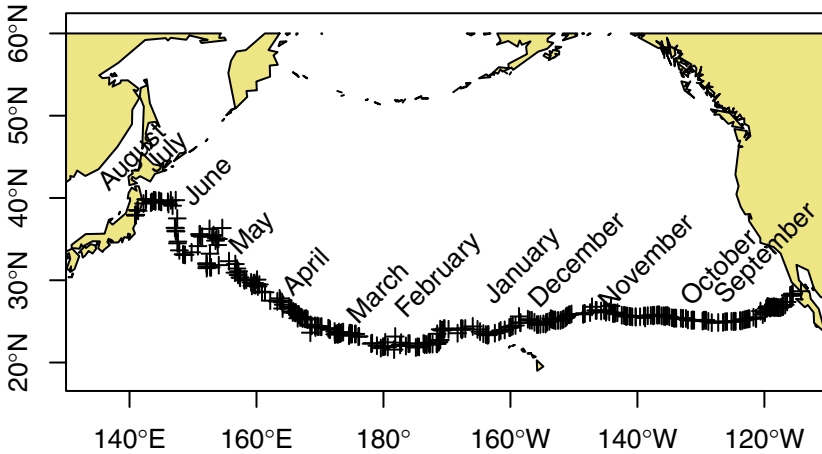


Fig. 2.3 Westward movements of a captive-raised adult loggerhead turtle (*Caretta caretta*) from 10 August 1996 to 12 August 1997

graphic ‘pen’ at the first point and drawing to the successive points until an NA is met. Then the pen is raised and moved to the next non-NA value, where it is lowered, until the end of the set of points. While this is convenient for graphics output, it is less so for associating lines with data values, because the line is not subsetted into data objects in any other way than by NA values.

The approach adopted here is to start with a `Line` object that is a matrix of 2D coordinates, without NA values. This corresponds to the line string defined by Herring (2011, pp. 21–23) as “a one-dimensional geometric object usually stored as a sequence of points ... with linear interpolation between points” (see also Kresse et al., 2012, pp. 506–507). Linear interpolation means that we assume that intermediate unobserved coordinates may be interpolated by using a straight line between the coordinates at either end of the line segment. This affects how we handle non-planar coordinates, usually geographical coordinates, for which another style of interpolation ought to be used. A list of `Line` objects forms the `Lines` slot of a `Lines` object, and formally corresponds to the `MultiLineString` definition of Herring (2011, p. 24). An identifying character tag is also required, and will be used for constructing `SpatialLines` objects using the same approach as was used above for matching ID values for spatial points.

```
> getClass("Line")
Class "Line" [package "sp"]
```

Slots:

```
Name: coords
Class: matrix
```

```
Known Subclasses: "Polygon"
```

```
> getClass("Lines")
```

```
Class "Lines" [package "sp"]
```

```
Slots:
```

```
Name:      Lines      ID
Class:      list character
```

Neither `Line` nor `Lines` objects inherit from the `Spatial` class. It is the `SpatialLines` object that contains the bounding box and projection information for the list of `Lines` objects stored in its `lines` slot. This degree of complexity is required to be able to add observation values in a data frame, creating `SpatialLinesDataFrame` objects, and to use a range of extraction methods on these objects.

```
> getClass("SpatialLines")
```

```
Class "SpatialLines" [package "sp"]
```

```
Slots:
```

```
Name:      lines      bbox proj4string
Class:      list      matrix      CRS
```

```
Extends: "Spatial", "SpatialLinesNULL"
```

```
Known Subclasses: "SpatialLinesDataFrame"
```

Let us examine an example of an object of this class, created from lines retrieved from the **maps** package world database, and converted to a `SpatialLines` object using the `map2SpatialLines` function in **maptools**. We can see that the `lines` slot of the object is a list of 51 components, each of which must be a `Lines` object in a valid `SpatialLines` object.

```
> library(maps)
> japan <- map("world", "japan", plot = FALSE)
> p4s <- CRS("+proj=longlat +ellps=WGS84")
> library(maptools)
> SLjapan <- map2SpatialLines(japan, proj4string = p4s)
> str(SLjapan, max.level = 2)
```

```
Formal class 'SpatialLines' [package "sp"] with 3 slots
..@ lines :List of 51
..@ bbox : num [1:2, 1:2] 123 24.3 ...
.. ..- attr(*, "dimnames")=List of 2
..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slots
```

`SpatialLines` and `SpatialPolygons` objects are very similar, as can be seen in Fig. 2.4 – the lists of component entities stack up in a hierarchical fashion. A very typical way of exploring the contents of these objects is to use `lapply` or `sapply` in combination with `slot`. The `lapply` and `sapply` functions apply their second argument, which is a function, to each of the elements of their first argument. The command used here can be read as

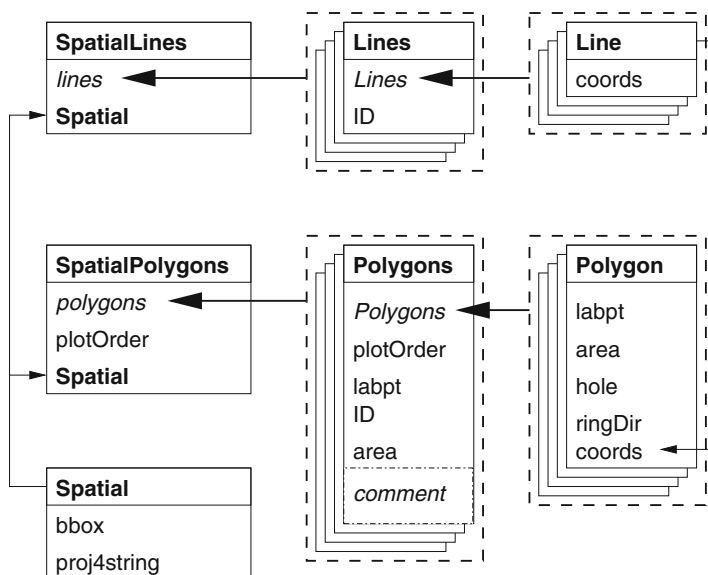


Fig. 2.4 SpatialLines and SpatialPolygons classes and slots; *thin arrows* show sub-class extensions, *thick arrows* the inclusion of lists of objects

follows: return the length of the **Lines** slot – how many **Line** objects it contains – of each **Lines** object in the list in the **lines** slot of **SLjapan**, simplifying the result to a numeric vector. If **lapply** was used, the result would have been a list. As we see, no **Lines** object contains more than one **Line** object:

```
> Lines_len <- sapply(slot(SLjapan, "lines"), function(x) length(slot(x,
+   "Lines")))
> table(Lines_len)

Lines_len
 1
51
```

We can use the **ContourLines2SLDF** function included in **maptools** in our next example, converting data returned by the base graphics function **contourLines** into a **SpatialLinesDataFrame** object; we used the volcano data set in Chap. 1, Fig. 1.2:

```
> volcano_sl <- ContourLines2SLDF(contourLines(volcano))
> t(slot(volcano_sl, "data"))

   C_1  C_2  C_3  C_4  C_5  C_6  C_7  C_8  C_9  C_10
level "100" "110" "120" "130" "140" "150" "160" "170" "180" "190"
```

We can see that there are ten separate contour level labels in the variable in the **data** slot, stored as a factor in the data frame in the object's **data** slot. As mentioned above, **sp** classes are new-style classes, and so the **slot** function can be used to look inside their slots.

To import data that we will be using shortly, we use another utility function in **maptools**, which reads shoreline data in ‘Mapgen’ format from the National Geophysical Data Center coastline extractor⁴ into a **SpatialLines** object directly, here selected for the window shown as the object bounding box:

```
> llCRS <- CRS("+proj=longlat +ellps=WGS84")
> auck_shore <- MapGen2SL("auckland_mapgen.dat", llCRS)
> summary(auck_shore)
```

Object of class SpatialLines

Coordinates:

min max

x 174.2 175.3

y -37.5 -36.5

Is projected: FALSE

proj4string : [+proj=longlat +ellps=WGS84]

The shorelines are still just represented by lines, shown in Fig. 2.5, and so colour filling of apparent polygons formed by line rings is not possible. For this we need a class of polygon objects, discussed in Sect. 2.6. Lines, however, can be generalised by removing detail that is not required for analysis or visualisation – the **maps**, **RArcInfo**, **maptools** and **rgeos** packages contain functions for line thinning. This operation can be performed successfully only on lines, because neighbouring polygons may have their shared boundary thinned differently. This leads to the creation of slivers, thin zones belonging to neither polygon or to both, but which may not be visible when plotted.

2.6 SpatialPolygons

The basic representation of a polygon in Sand consequently in R is a closed line, a sequence of point coordinates where the first point is the same as the last point. A set of polygons is made of closed lines separated by NA points. Like lines, it is not easy to work with polygons represented this way. To have a data set to use for polygons, we first identify the lines imported above representing the shoreline around Auckland. Many are islands, and so have identical first and last coordinates.

```
> lns <- slot(auck_shore, "lines")
> table(sapply(lns, function(x) length(slot(x, "Lines"))))
1
80
> islands_auck <- sapply(lns, function(x) {
+   crds <- slot(slot(x, "Lines")[[1]], "coords")
+   identical(crds[1, ], crds[nrow(crds), ])
+ })
> table(islands_auck)
```

⁴ <http://www.ngdc.noaa.gov/mgg/shorelines/shorelines.html>.

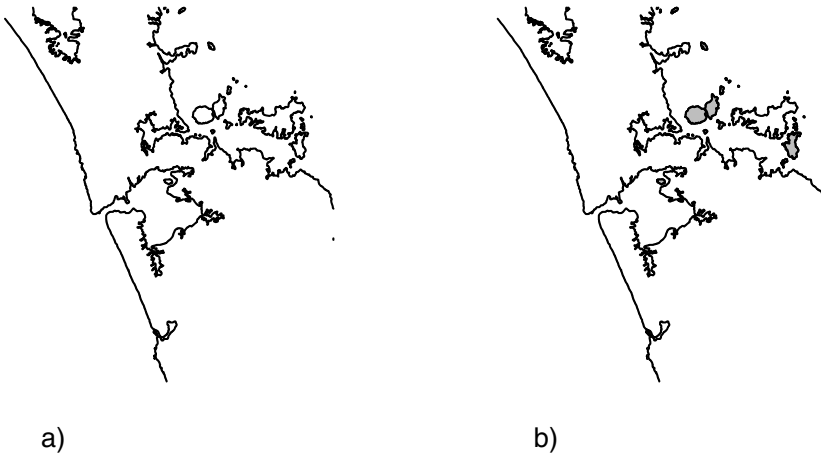


Fig. 2.5 Two maps of shorelines around Auckland: (a) line representation, (b) line representation over-plotted with islands converted to *polygons* and *shaded*. Note that Waiheke Island, the large island to the east, is not closed, and so not found as an island

```
islands_auck
FALSE TRUE
   16   64
```

Since all the `Lines` in the `auck_shore` object contain only single `Line` objects, checking the equality of the first and last coordinates of the first `Line` object in each `Lines` object tells us which sets of coordinates can validly be made into polygons. The nesting of classes for polygons is the same as that for lines, but the successive objects have more slots.

```
> getClass("Polygon")

Class "Polygon" [package "sp"]

Slots:

Name:    labpt    area    hole ringDir  coords
Class:  numeric  numeric logical integer  matrix

Extends: "Line"
```

The `Polygon` class extends the `Line` class by adding slots needed for polygons and checking that the first and last coordinates are identical. The extra slots are a label point, taken as the centroid of the polygon, the area of the polygon in the metric of the coordinates, whether the polygon is declared as a hole or not – the default value is a logical `NA`, and the ring direction of the polygon (discussed later in Sect. 2.6.2). No check is made of whether lines cross or polygons have ‘errors’, in other words whether features are simple in the

OpenGIS[®] (OpenGeoSpatial)⁵ context; these are discussed briefly later on p. 131. GIS should do this, and we assume that data read into R can be trusted and contain only simple features. `Polygon` objects are `LinearRing` objects as defined by Kresse et al. (2012, p. 506) and Herring (2011, p. 23), that is a closed `LineString`, but we assume but do not check that it is simple.

```
> getClass("Polygons")
```

```
Class "Polygons" [package "sp"]
```

```
Slots:
```

Name:	Polygons	plotOrder	labpt	ID	area
Class:	list	integer	numeric	character	numeric

The `Polygons` class contains a list of valid `Polygon` objects, an identifying character string, a label point taken as the label point of the constituent polygon with the largest area, and two slots used as helpers in plotting using R graphics functions, given this representation of sets of polygons. These set the order in which the polygons belonging to this object should be plotted, and the gross area of the polygon, equal to the sum of all the constituent polygons. A `Polygons` object may, for example, represent an administrative district located on two sides of a river, or archipelago. Each of the parts should be seen separately, but data are only available for the larger entity. In formal terms, `Polygons` objects are `MultiPolygon` objects as defined by Kresse et al. (2012, p. 507) and Herring (2011, p. 31).

```
> getClass("SpatialPolygons")
```

```
Class "SpatialPolygons" [package "sp"]
```

```
Slots:
```

Name:	polygons	plotOrder	bbox	proj4string
Class:	list	integer	matrix	CRS

```
Extends: "Spatial", "SpatialPolygonsNULL"
```

```
Known Subclasses: "SpatialPolygonsDataFrame"
```

The top level representation of polygons is as a `SpatialPolygons` object, a set of `Polygons` objects with the additional slots of a `Spatial` object to contain the bounding box and projection information of the set as a whole. Like the `Polygons` object, it has a plot order slot, defined by default to plot its member polygons, stored in the `polygons` as a list of `Polygons`, in order of gross area, from largest to smallest. Choosing only the lines in the Auckland shoreline data set which are closed polygons, we can build a `SpatialPolygons` object.

⁵ <http://www.opengeospatial.org/>.


```

> islands_sl <- auck_shore[islands_auck]
> list_of_Lines <- slot(islands_sl, "lines")
> islands_sp <- SpatialPolygons(lapply(list_of_Lines, function(x) {
+   Polygons(list(Polygon(slot(slot(x, "Lines"))[[1]],
+     "coords"))), ID = slot(x, "ID"))
+ }), proj4string = CRS("+proj=longlat +ellps=WGS84"))
> summary(islands_sp)

Object of class SpatialPolygons
Coordinates:
      min      max
x 174.30297 175.22791
y -37.43877 -36.50033
Is projected: FALSE
proj4string : [+proj=longlat +ellps=WGS84]

> slot(islands_sp, "plotOrder")

[1] 45 54 37 28 38 27 12 11 59 53  5 25 26 46  7 55 17 34 30 16  6 43
[23] 14 40 32 19 61 42 15 50 21 18 62 23 22 29 24 44 13  2 36  9 63 58
[45] 56 64 52 39 51  1  8  3  4 20 47 35 41 48 60 31 49 57 10 33

> order(sapply(slot(islands_sp, "polygons"), function(x) slot(x,
+   "area")), decreasing = TRUE)

[1] 45 54 37 28 38 27 12 11 59 53  5 25 26 46  7 55 17 34 30 16  6 43
[23] 14 40 32 19 61 42 15 50 21 18 62 23 22 29 24 44 13  2 36  9 63 58
[45] 56 64 52 39 51  1  8  3  4 20 47 35 41 48 60 31 49 57 10 33

```

As we saw with the construction of `SpatialLines` objects from raw coordinates, here we build a list of `Polygon` objects for each `Polygons` object, corresponding to a single identifying tag. A list of these `Polygons` objects is then passed to the `SpatialPolygons` function, with a coordinate reference system, to create the `SpatialPolygons` object. Again, like `SpatialLines` objects, `SpatialPolygons` objects are most often created by functions that import or manipulate such data objects, and seldom from scratch.

2.6.1 SpatialPolygonsDataFrame *Objects*

As with other spatial data objects, `SpatialPolygonsDataFrame` objects bring together the spatial representations of the polygons with data. The identifying tags of the `Polygons` in the `polygon` slot of a `SpatialPolygons` object are matched with the row names of the data frame to make sure that the correct data rows are associated with the correct spatial objects. The data frame is re-ordered by row to match the spatial objects if need be, provided all the objects can be matched to row names. If any differences are found, an error results. Both identifying tags and data frame row names are character strings, and so their sort order is also character, meaning that "2" follows "11" and "111".⁶

⁶ Some `maptools` functions use Gregory R. Warnes' `mixedorder` sort from `gtools` to sort integer-like strings in integer order.

As an example, we take a set of scores by US state of 1999 Scholastic Aptitude Test (SAT) used for spatial data analysis by Melanie Wall.⁷ In the data source, there are also results for Alaska, Hawaii, and for the US as a whole. If we would like to associate the data with state boundary polygons provided in the **maps** package, it is convenient to convert the boundaries to a **SpatialPolygons** object – see also Chap. 4.

```
> library(maps)
> state.map <- map("state", plot = FALSE, fill = TRUE)
> IDs <- sapply(strsplit(state.map$names, ":"), function(x) x[1])
> library(maptools)
> state.sp <- map2SpatialPolygons(state.map, IDs = IDs,
+   proj4string = CRS("+proj=longlat +ellps=WGS84"))
```

Then we can use identifying tag matching to suit the rows of the data frame to the **SpatialPolygons**. Here, we subset to the matched rows of the data frame, to ensure that one row corresponds to each Polygons object, to achieve one-to-one matching:

```
> sat <- read.table("state.sat.data_mod.txt", row.names = 5,
+   header = TRUE)
> str(sat)

'data.frame':      52 obs. of  4 variables:
 $ oname : Factor w/ 52 levels "ala","alaska",...: 1 2 3 4 5 ...
 $ vscore: int 561 516 524 563 497 ...
 $ mscore: int 555 514 525 556 514 ...
 $ pc : int 9 50 34 6 49 ...

> id <- match(row.names(sat), row.names(state.sp))
> row.names(sat)[is.na(id)]

[1] "alaska" "hawaii" "usa"

> sat1 <- sat[!is.na(id), ]
> state.spdf <- SpatialPolygonsDataFrame(state.sp, sat1)
> str(slot(state.spdf, "data"))

'data.frame':      49 obs. of  4 variables:
 $ oname : Factor w/ 52 levels "ala","alaska",...: 1 3 4 5 6 ...
 $ vscore: int 561 524 563 497 536 ...
 $ mscore: int 555 525 556 514 540 ...
 $ pc : int 9 34 6 49 32 ...

> str(state.spdf, max.level = 2)
```

```
Formal class 'SpatialPolygonsDataFrame' [package "sp"] with 5 slots
..@ data :'data.frame': 49 obs. of 4 variables:
..@ polygons :List of 49
..@ plotOrder : int [1:49] 42 25 4 30 27 ...
..@ bbox : num [1:2, 1:2] -124.7 25.1 ...
.. ..- attr(*, "dimnames")=List of 2
..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slots
```

⁷ <http://www.biostat.umn.edu/~brad/data/state-sat.dat>, data here supplemented with variable names and state names as used in **maps**.

If we modify the row name of ‘arizona’ in the data frame to ‘Arizona’, there is no longer a match with a polygon identifying tag, and an error is signalled.

```
> rownames(sat1)[2] <- "Arizona"
> SpatialPolygonsDataFrame(state.sp, sat1)

Error in SpatialPolygonsDataFrame(state.sp, sat1) :
  row.names of data and Polygons IDs do not match
```

In subsequent analysis, Wall (2004) also drops District of Columbia. Rather than having to manipulate polygons and their data separately, when using a `SpatialPolygonsDataFrame` object, we can say:

```
> DC <- "district of columbia"
> not_dc <- !(row.names(state.spdf) == DC)
> state.spdf1 <- state.spdf[not_dc, ]
> dim(state.spdf1)
```

```
[1] 48  4
```

```
> summary(state.spdf1)
```

Object of class `SpatialPolygonsDataFrame`

Coordinates:

```
      min      max
x -124.68134 -67.00742
y  25.12993  49.38323
```

Is projected: FALSE

proj4string : [+proj=longlat +ellps=WGS84]

Data attributes:

	oname		vscore		mscore		pc
ala	: 1	Min.	:479.0	Min.	:475.0	Min.	: 4.00
ariz	: 1	1st Qu.:	:506.2	1st Qu.:	:505.2	1st Qu.:	: 9.00
ark	: 1	Median	:530.5	Median	:532.0	Median	:28.50
calif	: 1	Mean	:534.6	Mean	:534.9	Mean	:35.58
colo	: 1	3rd Qu.:	:563.0	3rd Qu.:	:558.5	3rd Qu.:	:63.50
conn	: 1	Max.	:594.0	Max.	:605.0	Max.	:80.00
(Other):	42						

2.6.2 Holes and Ring Direction

The hole and ring direction slots are included in `Polygon` objects as heuristics to address some of the difficulties arising from R not being a GIS. In a traditional vector GIS, and in the underlying structure of the data stored in **maps**, boundaries between polygons are stored only once as arcs between nodes (shared vertices between three or more polygons, possibly including the external space), and the polygons are constructed on the fly from lists of directed boundary arcs, including boundaries with the external space – void – not included in any polygon. This is known as the topological representation of polygons, and is appropriate for GIS software, but arguably not for

other software using spatial data. It was mentioned above that it is the user's responsibility to provide line coordinates such that the coordinates represent the line object the user requires. If the user requires, for example, that a river channel does not cross itself, the user has to impose that limitation. Other users will not need such a limitation, as for example tracking data may very well involve an animal crossing its tracks.

The approach that has been chosen in **sp** is to use two markers commonly encountered in practice, marking polygons as holes with a logical (TRUE/FALSE) flag, the `hole` slot, and using ring direction – clockwise rings are taken as not being holes, anti-clockwise as being holes. This is needed because the non-topological representation of polygons has no easy way of knowing that a polygon represents an internal boundary of an enclosing polygon, a hole, or lake.

An approach that works when the relative status of polygons is known is to set the hole slot directly. This is done in reading GSHHS shoreline data, already used in Fig. 2.3 and described in Chap. 4. The data source includes a variable for each polygon, where the levels are land: 1, lake: 2, island in lake: 3, and lake on island in lake: 4. The following example takes a region of interest on the northern, Canadian shore of Lake Huron, including Manitoulin Island, and a number of lakes on the island, including Kongawong Lake.

```
> length(slot(manitoulin_sp, "polygons"))
[1] 1

> sapply(slot(slot(manitoulin_sp, "polygons")[[1]], "Polygons"),
+       function(x) slot(x, "hole"))

[1] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[12] FALSE FALSE FALSE FALSE FALSE TRUE TRUE TRUE

> sapply(slot(slot(manitoulin_sp, "polygons")[[1]], "Polygons"),
+       function(x) slot(x, "ringDir"))

[1] 1 -1 1 1 1 1 1 1 1 1 1 1 1 1 1 -1 -1 -1
```

The class definitions used for polygons in **sp** do not accord with those of the simple features specification of the Open Geospatial Consortium (also discussed on p. 131). The **rgeos** package, an interface to Geometry Engine – Open Source (GEOS) to be presented in Sect. 5.2.1, uses this specification, in which each hole (interior ring) must be associated with its containing exterior ring. In order to avoid introducing incompatible changes into the class definition of **Polygons** objects, a comment has been added as a single character string to each such object.

Here we can trust the data source to assign the hole status correctly, and use the simple function `createSPComment` to add such comments to each **Polygons** member of the `polygons` slot of this **SpatialPolygons** object. Exterior rings are coded zero, while interior rings are coded with the 1-based index of the exterior ring to which they belong. The right panel of Fig. 2.6

shows the comment tags for all but the first two `Polygon` objects in the single `Polygons` object in the data set. Almost all the `Polygon` objects are islands in Lake Huron coded zero, but there are three lakes on Manitoulin Island coded with its 1-based index; 1 is the mainland shore, and 2 is Lake Huron, which has an indeterminate zero code, because here it is not fully contained by the mainland. The use of such comments permits us to assign the `LinearRings` in a `MultiPolygon` object to the correct member `Polygon` object in the terminology of Kresse et al. (2012, p. 507) and Herring (2011, pp. 26–28).

```
> library(rgeos)

> manitoulin_sp <- createSPComment(manitoulin_sp)
> sapply(slot(manitoulin_sp, "polygons"), comment)

[1] "0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 3 3"
```

In the left panel of Fig. 2.6, there is only one `Polygons` object in the `polygons` slot of `manitoulin_sp`, representing the continental landmass, exposed along the northern edge, and containing the complete set of polygons. Within this is a large section covered by Lake Huron, which in turn is covered by islands and lakes on islands. Not having a full topological representation means that for plotting, we paint the land first, then paint the lake, then the islands, and finally the lakes on islands. Because the default plotting colour for holes is ‘transparent’, they can appear to be merged into the surrounding land – the same problem arises where the hole slot is wrongly assigned. The `plotOrder` slots in `Polygons` and `SpatialPolygons` objects attempt to get around this problem, but care is usually sensible if the spatial objects being handled are complicated. Since R version 2.12.0, polygon plotting functions have been able to choose to detect holes themselves, so that objects with wrongly declared hole status may still be displayed correctly (Murrell, 2012).

2.7 SpatialGrid and SpatialPixel Objects

The point, line, and polygon objects we have considered until now have been handled one-by-one. Grids are regular objects requiring much less information to define their structure. Once the single point of origin is known, the extent of the grid can be given by the cell resolution and the numbers of rows and columns present in the full grid. This representation is typical for remote sensing and raster GIS, and is used widely for storing data in regular rectangular cells, such as digital elevation models, satellite imagery, and interpolated data from point measurements, as well as image processing.



Fig. 2.6 The northern, Canadian shore of Lake Huron, including Manitoulin Island and lakes on the island; islands (*light blue*) and lakes on islands (*khaki*); in the *left panel*, some Polygon objects are marked with their GSHHS levels, in the *right panel*, they are marked with their comment tags

```
> getClass("GridTopology")
Class "GridTopology" [package "sp"]
```

Slots:

Name:	cellcentre.offset	cellsize	cells.dim
Class:	numeric	numeric	integer

As an example, we make a `GridTopology` object from the bounding box of the Manitoulin Island vector data set. If we choose a cell size of 0.01° in each direction, we can offset the south-west cell centre to make sure that at least the whole area is covered, and find a suitable number of cells in each dimension.

```
> bb <- bbox(manitoulin_sp)
> bb
      min      max
x 277.0 278.0
y  45.7  46.2
> cs <- c(0.01, 0.01)
> cc <- bb[, 1] + (cs/2)
> cd <- ceiling(diff(t(bb))/cs)
> manitoulin_grd <- GridTopology(cellcentre.offset = cc,
+   cellsize = cs, cells.dim = cd)
> manitoulin_grd
              x      y
cellcentre.offset 277.005 45.705
cellsize          0.010  0.010
cells.dim         100.000 50.000
```

The object describes the grid completely, and can be used to construct a `SpatialGrid` object. A `SpatialGrid` object contains `GridTopology` and `Spatial` objects.

```
> getClass("SpatialGrid")
```

```
Class "SpatialGrid" [package "sp"]
```

```
Slots:
```

```
Name:          grid          bbox proj4string
Class: GridTopology      matrix      CRS
```

```
Extends: "Spatial"
```

```
Known Subclasses: "SpatialGridDataFrame"
```

Using the `GridTopology` object created above, and passing through the coordinate reference system of the original GSHHS data, the bounding box is created automatically, as we see from the summary of the object:

```
> p4s <- CRS(proj4string(manitoulin_sp))
> manitoulin_SG <- SpatialGrid(manitoulin_grd, proj4string = p4s)
> summary(manitoulin_SG)
```

```
Object of class SpatialGrid
```

```
Coordinates:
```

```
      min      max
x 277.0 278.0
y  45.7  46.2
```

```
Is projected: FALSE
```

```
proj4string :
```

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

```
Grid attributes:
```

```
      cellcentre.offset cellsize cells.dim
x           277.005      0.01      100
y           45.705      0.01      50
```

As an example of using these classes with imported data, we use an excerpt from the Shuttle Radar Topography Mission (SRTM) flown in 2000, for the Auckland area⁸ (Fig. 2.8). The data have been read from a Geotiff file into a `SpatialGridDataFrame` object – a `SpatialGrid` object extended with a `data` slot occupied by a `data.frame` object, filled with a single band of data representing elevation in metres. After checking the class of the data object, we examine in turn its slots. The `grid` slot contains the underlying `GridTopology` object, with the lower left cell centre coordinates, the pair of cell size resolution values, here both equal to 3 arcsec, and the numbers of columns and rows:

```
> class(auck_el1)

[1] "SpatialGridDataFrame"
attr(,"package")
[1] "sp"

> slot(auck_el1, "grid")
```

⁸ Downloaded from the seamless data distribution system for 3 arcsec 'Finished' (90 m) data, <http://earthexplorer.usgs.gov/>; the data can be downloaded as 1° square tiles.

```

              x              y
cellcentre.offset 1.742004e+02 -3.749958e+01
cellsize          8.333333e-04  8.333333e-04
cells.dim         1.320000e+03  1.200000e+03

```

```
> slot(auck_el1, "bbox")
```

```

      min      max
x 174.2 175.3
y -37.5 -36.5

```

```
> object.size(auck_el1)
```

```
12677176 bytes
```

```
> object.size(slot(auck_el1, "data"))
```

```
12672672 bytes
```

The total size of the `SpatialGridDataFrame` object is just over 12 MB, almost all of which is made up of the `data` slot.

```
> is.na(auck_el1$band1) <- auck_el1$band1 <= 0
> summary(auck_el1$band1)
```

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	NA's
1		23	53	78	106	686	791938

Almost half of the data are at or below sea level, and should be set to `NA`. Once this is done, about half of the data are missing. In other cases, even larger proportions of raster grids are missing, suggesting that an alternative representation of the same data might be attractive. One candidate from Terralib, discussed further in Chap. 4, is the cell representation of rasters, where the raster cells with data are represented by the coordinates of the cell centre, and by the sequence number of the cell among all the cells in the raster. In this representation, missing data are discarded, and savings in space and processing time can be large. It also permits cells to be stored, like points, in an external database. The class is here termed `SpatialPixels`, and has the same slots as `SpatialGrid` objects, but differently filled (Fig. 2.7). The `SpatialPixelsDataFrame` class is analogous.

```
> auck_el2 <- as(auck_el1, "SpatialPixelsDataFrame")
```

```
> object.size(auck_el2)
```

```
25351272 bytes
```

```
> object.size(slot(auck_el2, "grid.index"))
```

```
3168288 bytes
```

```
> object.size(slot(auck_el2, "coords"))
```

```
12673512 bytes
```

```
> sum(is.na(auck_el1$band1)) + nrow(slot(auck_el2, "coords"))
```

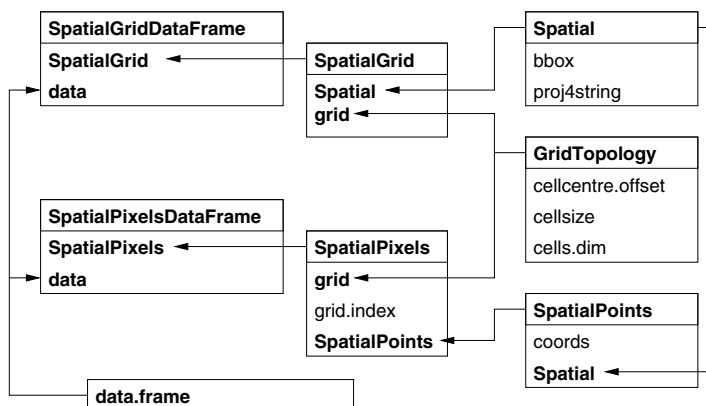



Fig. 2.7 `SpatialGrid` and `SpatialPixel` classes and their slots; arrows show subclass extensions

```
[1] 1584000
```

```
> prod(slot(slot(auck_el2, "grid"), "cells.dim"))
```

```
[1] 1584000
```

Returning to our example, we can coerce our `SpatialGridDataFrame` object to a `SpatialPixelsDataFrame` object. In this case, the proportion of missing to occupied cells is unfavourable, and when the `grid.index` and `coords` slots are populated with cell indices and coordinates, the output object is almost twice as large as its `SpatialGridDataFrame` equivalent. We can also see that the total number of cells – the product of the row and column dimensions – is equal to the number of coordinates in the output object plus the number of missing data values deleted by coercion. Had the number of attributes been 10, then the space saving relative to storing the full grid would have been 37%; with 100 attributes it would have been 48% for this particular case.

```
> auck_el_500 <- auck_el2[auck_el2$band1 > 500, ]
> summary(auck_el_500)
```

```
Object of class SpatialPixelsDataFrame
```

```
Coordinates:
```

```
      min      max
```

```
x 175.18917 175.24333
```

```
y -37.10333 -37.01833
```

```
Is projected: FALSE
```

```
proj4string :
```

```
[+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84
```

```
+towgs84=0,0,0]
```

```
Number of points: 1114
```

```
Grid attributes:
```

```
  cellcentre.offset    cellsize cells.dim
```

```
x          174.20042 0.0008333333      1320
```

```
y          -37.49958 0.0008333333      1200
Data attributes:
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
501.0  523.0   552.0   559.4  591.0   686.0
```

```
> object.size(auck_el_500)
```

```
40936 bytes
```

Taking just the raster cells over 500 m, of which there are very few, less than 1 % of the total, yields a much smaller object. In this case it has a smaller bounding box, and gaps between the pixels present.

We can also create a **SpatialPixels** object directly from a **SpatialPoints** object. As our example, we use the Meuse bank data set provided with **sp**. We can pass a **SpatialPoints** object to the **SpatialPixels** function, where the **Spatial** object components are copied across, and the points checked to see whether they lie on a regular grid. If they do, the function will return a **SpatialPixels** object:

```
> data(meuse.grid)
> mg_SP <- SpatialPoints(cbind(meuse.grid$x, meuse.grid$y))
> summary(mg_SP)
```

```
Object of class SpatialPoints
```

```
Coordinates:
```

```
      min      max
coords.x1 178460 181540
coords.x2 329620 333740
```

```
Is projected: NA
```

```
proj4string : [NA]
```

```
Number of points: 3103
```

```
> mg_SPix0 <- SpatialPixels(mg_SP)
```

```
> summary(mg_SPix0)
```

```
Object of class SpatialPixels
```

```
Coordinates:
```

```
      min      max
coords.x1 178440 181560
coords.x2 329600 333760
```

```
Is projected: NA
```

```
proj4string : [NA]
```

```
Number of points: 3103
```

```
Grid attributes:
```

```
      cellcentre.offset cellsize cells.dim
coords.x1           178460         40         78
coords.x2           329620         40        104
```

```
> prod(slot(slot(mg_SPix0, "grid"), "cells.dim"))
```

```
[1] 8112
```

As we can see from the product of the cell dimensions of the underlying grid, over half of the full grid is not present in the **SpatialPixels** representation, because many grid cells lie outside the study area. Alternatively, we can coerce a **SpatialPoints** object to a **SpatialPixels** object:

```
> mg_SPix1 <- as(mg_SP, "SpatialPixels")
> summary(mg_SPix1)

Object of class SpatialPixels
Coordinates:
      min      max
coords.x1 178440 181560
coords.x2 329600 333760
Is projected: NA
proj4string : [NA]
Number of points: 3103
Grid attributes:
      cellcentre.offset cellsize cells.dim
coords.x1      178460      40      78
coords.x2      329620      40     104
```

2.8 Raster Objects and the raster Package

The **raster** was published on CRAN in 2010, and its users now generate a good deal of traffic on the R-sig-geo mailing list. The package is documented in a number of vignettes, which are available with the package, and online from CRAN (Hijmans, 2012b).⁹ It uses **sp** classes for vector data, and adds new classes for raster data, together with many methods and functions for data handling and analysis. A key advance made possible by this package is that raster objects may be held on disk rather than in memory; options may be used to control where the data are held (Hijmans, 2012c). Returning to the Shuttle Radar Topography Mission data for the Auckland area, we can create a **RasterLayer** object by passing the name of a raster file to the **raster** function; the file is accessed using functions in the **rgdal** package discussed in detail in Chap. 4:

```
> library(raster)

> r <- raster("70042108.tif")

> class(r)

[1] "RasterLayer"
attr(,"package")
[1] "raster"

> inMemory(r)

[1] FALSE

> object.size(r)

11512 bytes
```

⁹ <http://cran.r-project.org/web/packages/raster/index.html>.

```
> cellStats(r, max)

[1] 686

> cellStats(r, min)

[1] -3.402823e+38

> inMemory(r)

[1] FALSE
```

The package provides a richer range of classes for collections of rasters called *stacks* and *bricks*, and we return to them, and functions and methods for handling this representation of raster data in Sects. 5.3.2 and 6.9.1. We need to remove values of less than or equal to zero, since elevations are only recorded on land. We can do this as described by Hijmans (2012c, p. 4) in blocks of rows, storing the data of the new object in a temporary file:

```
> out <- raster(r)
> bs <- blockSize(out)
> out <- writeStart(out, filename = tempfile(), overwrite = TRUE)
> for (i in 1:bs$n) {
+   v <- getValues(r, row = bs$row[i], nrow = bs$nrow[i])
+   v[v <= 0] <- NA
+   writeValues(out, v, bs$row[i])
+ }
> out <- writeStop(out)
> cellStats(out, min)

[1] 1

> cellStats(out, max)

[1] 686

> inMemory(out)

[1] FALSE
```

The **raster** package also provides convenient plot methods, which we use to create Fig. 2.8:

```
> plot(out, col = terrain.colors(100))
```

Coercion methods are available from and to **SpatialGridDataFrame** objects, and between other classes defined in **sp** and **raster**. The **RasterLayer** object can be coerced to **SpatialGridDataFrame** and back again:

```
> r1 <- as(out, "SpatialGridDataFrame")
> summary(r1)
```

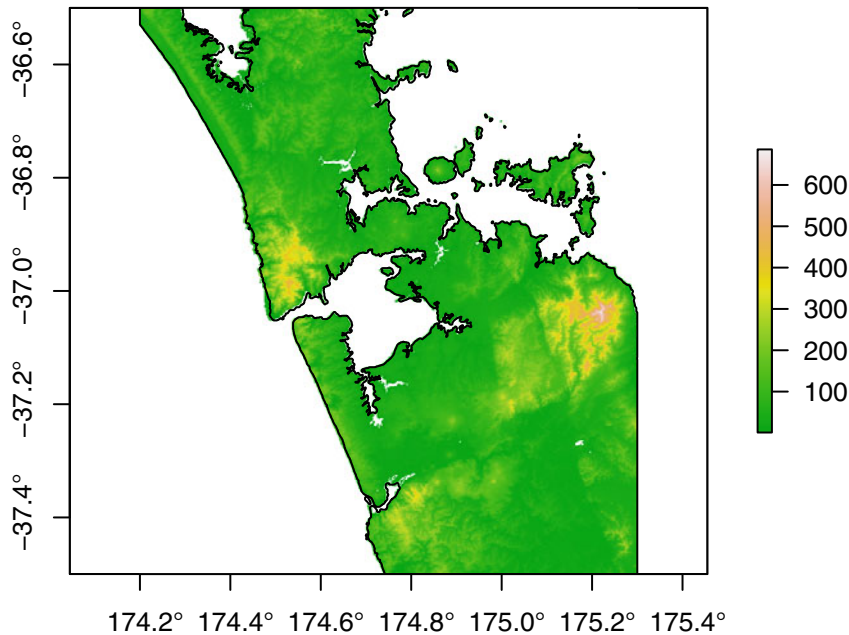


Fig. 2.8 SRTM elevation data in metres for the Auckland isthmus over-plotted with an excerpt from the GSHHS full resolution shoreline – there are detailed differences stemming from the very different technologies underlying the two data sources

```
Object of class SpatialGridDataFrame
Coordinates:
      min  max
s1 174.2 175.3
s2 -37.5 -36.5
Is projected: FALSE
proj4string :
[+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84
+towgs84=0,0,0]
Grid attributes:
      cellcentre.offset  cellsize  cells.dim
s1      174.20042  0.0008333333      1320
s2      -37.49958  0.0008333333      1200
Data attributes:
      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.   NA's
1      23     53     78    106    686  791938

> r2 <- as(r1, "RasterLayer")
> summary(r2)

      layer
Min.      1
1st Qu.   23
Median    53
3rd Qu.  106
```

Max.	686
NA's	791938

We have now described a coherent and consistent set of classes for spatial data. Other representations are also used by R packages, and we show further ways of converting between these representations and external formats in Chap. 4. Before treating data import and export, we discuss graphical methods for **sp** classes, to show that the effort of putting the data in formal classes may be justified by the relative ease with which we can make maps.

Chapter 3

Visualising Spatial Data

A major pleasure in working with spatial data is their visualisation. Maps are amongst the most compelling graphics, because the space they map is the space we think we live in, and maps may show things we cannot see otherwise. Although one can work with all R plotting functions on the raw data, for example extracted from **Spatial** classes by methods like `coordinates` or `as.data.frame`, this chapter introduces the plotting methods for objects inheriting from class **Spatial** that are provided by package **sp**.

R plots can be seen as following one of two systems. First, the ‘traditional’ plotting system, which allows incremental addition. Second, the systems that use or build upon **grid** (Murrell 2011), and does not allow simple incremental addition. The main example of the second system is the Trellis Graphics system, provided by package **lattice** (Sarkar 2008), present in default R installations. The **ggplot2** package is increasingly popular (Wickham 2009, Fig. 1), which builds on **grid**, and provides similar capabilities as **lattice** with different defaults and a different user interface (Wilkinson 2005).

Traditional graphics are typically built incrementally: graphic elements are added in several consecutive function calls. Trellis graphics allow plotting of high-dimensional data by providing *conditioning plots*: organised lattices of plots with shared axes (Cleveland 1993, 1994). This feature is particularly useful when multiple maps need to be compared, for example in case of a spatial time series, comparison across a number of species or variables, or comparison of different modelling scenarios or approaches. Trellis graphs are designed to avoid wasting space by repetition of identical information. The value of this feature, rarely found in other software, is hard to overestimate. Waller and Gotway (2004, pp. 68–86) provide an introduction to statistical mapping, which may be deepened with reference to Slocum et al. (2005).

Package **sp** provides `plot` methods that build on the traditional R plotting system (`plot`, `image`, `lines`, `points`, etc.), as well as a ‘new’ generic method called `spplot` that uses the Trellis system (notably `xyplot` or `levelplot` from the **lattice** package) and can be used for conditioning plots.

The `spplot` methods are introduced in a later sub-section, first we deal with the traditional plot system.

3.1 The Traditional Plot System

3.1.1 Plotting Points, Lines, Polygons, and Grids

In the following example session, we create points, lines, polygons, and a grid object, from `data.frame` objects, retrieved from the `sp` package by function `data`, and plot them. The four plots obtained by the `plot` and `image` commands are shown in Fig. 3.1.

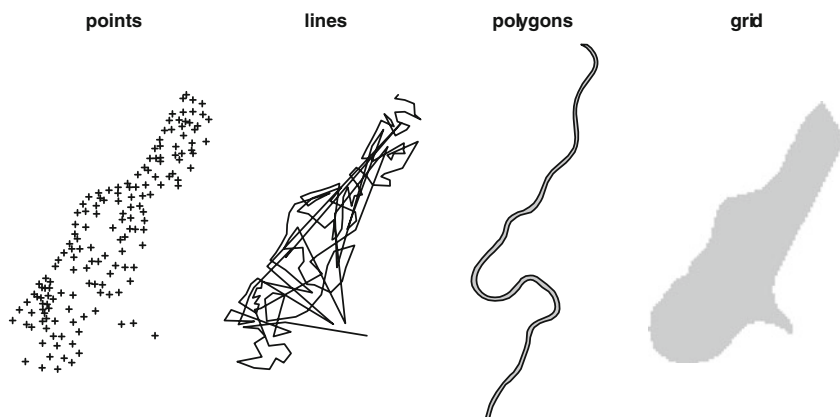


Fig. 3.1 The meuse data set: sample points, the sample path (*line*), the Meuse river (*ring*) and the *gridded* study area

```
> library(sp)
> data(meuse)
> coordinates(meuse) <- c("x", "y")
> plot(meuse)
> title("points")
```

The `SpatialPointsDataFrame` object used is created from a `data.frame` provided with `sp`, and the `plot` method shows the points with the default symbol.

```
> cc <- coordinates(meuse)
> m.sl <- SpatialLines(list(Lines(list(Line(cc)), "line1")))
> plot(m.sl)
> title("lines")
```


A **SpatialLines** object is made by joining up the points in sequence, and **plot** draws the resulting zig-zags.

```
> data(meuse.riv)
> meuse.lst <- list(Polygons(list(Polygon(meuse.riv)),
+   "meuse.riv"))
> meuse.pol <- SpatialPolygons(meuse.lst)
> plot(meuse.pol, col = "grey")
> title("polygons")
```

We make a **SpatialPolygons** object from data provided with **sp** outlining the banks of the River Meuse.

```
> data(meuse.grid)
> coordinates(meuse.grid) <- c("x", "y")
> meuse.grid <- as(meuse.grid, "SpatialPixels")
> image(meuse.grid, col = "grey")
> title("grid")
```

Finally, we convert grid data for the same Meuse bank study area into a **SpatialPixels** object and display it using the **image** method, with all cells set to "grey".

On each map, one unit in the *x*-direction equals one unit in the *y*-direction. This is the default when the coordinate reference system is not **longlat** or is unknown. For unprojected data in geographical coordinates (longitude/latitude), the default aspect ratio depends on the (mean) latitude of the area plotted. The default aspect can be adjusted by passing the **asp** argument.

A map becomes more readable when we combine several elements. We can display elements from those created above by using the **add = TRUE** argument in function calls:

```
> image(meuse.grid, col = "lightgrey")
> plot(meuse.pol, col = "grey", add = TRUE)
> plot(meuse, add = TRUE)
```

the result of which is shown in Fig. 3.2.

The over-plotting of polygons by points is the consequence of the order of plot commands. Up to now, the plots only show the geometry (topology, shapes) of the objects; we start plotting attributes (e.g. what has actually been measured at the sample points) in Sect. 3.1.5.

As an alternative to **plot(x, add=TRUE)**, one can use the commands **lines** for objects of class **SpatialLines** and **points** for **SpatialPoints**; text elements can be added by **text**.

3.1.2 Axes and Layout Elements

Maps often do not have axes, as the information carried in map axes can often be omitted. Especially, projected coordinates are usually long, hard to read

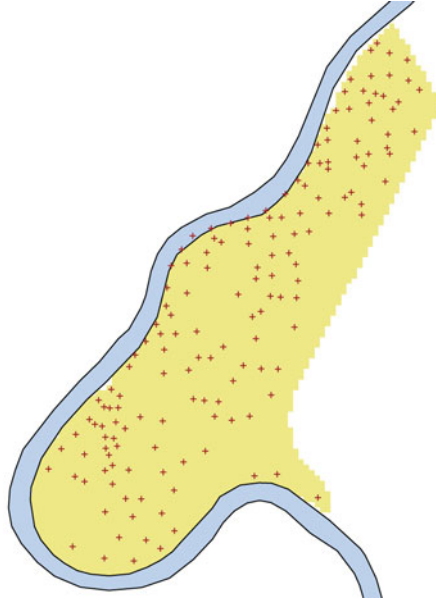


Fig. 3.2 Map elements combined into a single map

and geographical reference is much easier when recognisable features such as administrative boundaries, rivers, coast lines, etc. are present. In the standard `plot` functions, the Boolean argument `axes` can be set to control axis plotting, and the function `axis` can be called to add axes, fine-tuning their appearance (tic placement, tic labels, and font size). The following commands result in Fig. 3.3:

```
> layout(matrix(c(1, 2), 1, 2))
> plot(meuse.pol, axes = TRUE)
> plot(meuse.pol, axes = FALSE)
> axis(1, at = c(178000 + 0:2 * 2000), cex.axis = 0.7)
> axis(2, at = c(326000 + 0:3 * 4000), cex.axis = 0.7)
> box()
```

Not plotting axes does not increase the amount of space R used for plotting the data.¹ R still reserves the necessary space for adding axes and titles later on. We can, however, explicitly instruct R not to reserve this space by using function `par`, which is intended to have side effects on the next plot on the current device. The `par`-settable arguments that are useful for controlling the physical size of the plot are listed in Table 3.1.

In Fig. 3.4, generated by

```
> oldpar = par(no.readonly = TRUE)
> layout(matrix(c(1, 2), 1, 2))
```

¹ This is not true for Trellis plots; see Sect. 3.2.

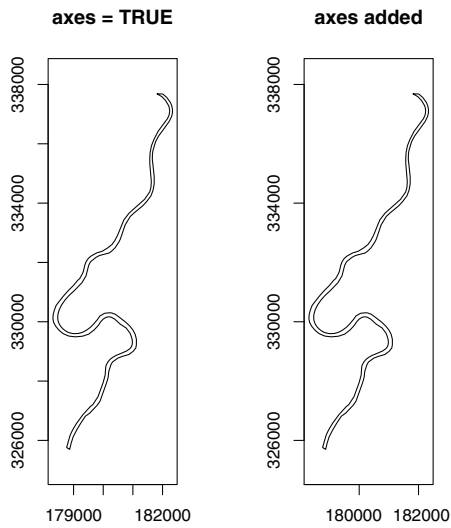


Fig. 3.3 Default axes (*left*) and custom axes (*right*) for the `meuse.riv` data

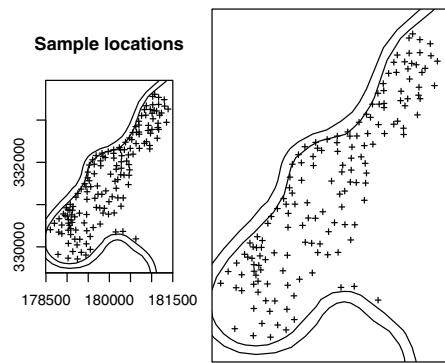


Fig. 3.4 Plots covering the same amount of paper, with (*left*), and without (*right*) the default space R reserves for axes and title(s)

Table 3.1 Graphic arguments useful for controlling figure and plotting region

Argument	Meaning	Unit	Length
<code>fin</code>	Figure region	Inch	2
<code>pin</code>	Plotting region	Inch	2
<code>mai</code>	Plotting margins	Inch	4
<code>mar</code>	Plotting margins	Lines of text	4

see `?par` for more information

```
> plot(meuse, axes = TRUE, cex = 0.6)
> plot(meuse.pol, add = TRUE)
> title("Sample locations")
> par(mar = c(0, 0, 0, 0) + 0.1)
```

```
> plot(meuse, axes = FALSE, cex = 0.6)
> plot(meuse.pol, add = TRUE)
> box()
> par(oldpar)
```

the same data set is plotted twice within the same amount of space, at the left-hand side with R's default margins leaving space for axes, and on the right-hand side with maximised plotting space and no axes drawn.

Modifying the margins by setting `mar` in the `par` command, for example to `par(mar=c(3,3,2,1))` further optimises space usage when axes are drawn, leaving (little) space for a title. It should be noted that the margin sizes are absolute, expressed in units the height of a line of text, and so their effect on map scale decreases when the plotting region is enlarged.

The `plot` methods provided by package `sp` do not allow the printing of axis labels, such as 'Easting' and 'Northing', or '*x*-coordinate' and '*y*-coordinate'. The reason for this is technical, but mentioning axis names is usually obsolete once the graph is referred to as a map. The units of the coordinate reference system (such as metres) should be equal for both axes and do not need mentioning twice. Geographical coordinates are perhaps an exception, but this is made explicit by axis tic labels such as 52°N, or by adding a reference grid.

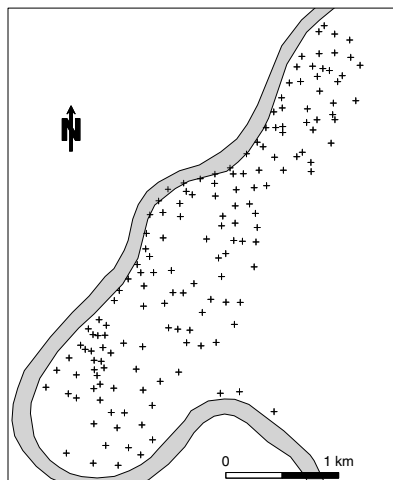


Fig. 3.5 *Scale bar and north arrow as map elements*

When we decide not to draw axes on a map, in addition to reference boundaries, we can provide the reader of a map with a guidance for distance and direction by plotting a scale bar and a north arrow, which can be placed interactively using `locator` followed by a few well-chosen clicks in the map (Fig. 3.5):

```

> plot(meuse)
> plot(meuse.pol, add = TRUE)
> plot(meuse)
> SpatialPolygonsRescale(layout.scale.bar(), offset = locator(1),
+   scale = 1000, fill = c("transparent", "black"), plot.grid = FALSE)
> text(locator(1), "0")
> text(locator(1), "1 km")
> SpatialPolygonsRescale(layout.north.arrow(), offset = locator(1),
+   scale = 400, plot.grid = FALSE)

```

When large numbers of maps for identical areas have to be produced with identical layout elements, it pays off to write a function that draws all layout elements. As an alternative, one may use conditioning plots; see the `spplot` method in Sect. 3.2.

3.1.3 Degrees in Axes Labels and Reference Grid

Unprojected data have coordinates in latitude and longitude degrees, with negative degrees referring to degrees west (of the prime meridian) and south (of the Equator). When unprojected spatial data are plotted using `sp` methods (`plot` or `spplot`), the axis label marks will give units in decimal degrees N/S/E/W, for example 50.5°N. An example is shown in Fig. 3.6, using `degAxis` to control which tick-marks are to be drawn.

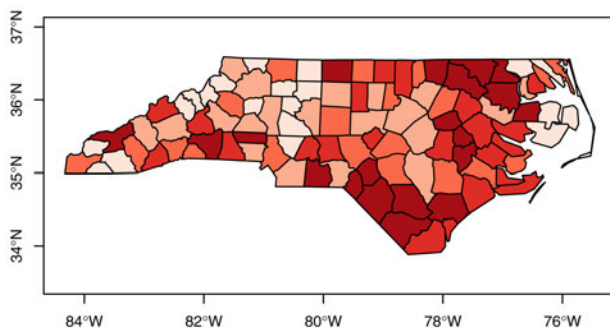


Fig. 3.6 Decimal degrees in axis labels: the North Carolina SIDS data

When, for reference purposes, a grid needs to be added to a map, the function `gridlines` can be used to generate an object of class `SpatialLines`. By default it draws lines within the bounding box of the object at values where the default axes labels are drawn; other values can be specified. Grid lines may be latitude/longitude grids, and these are non-straight lines. This is accomplished by generating a grid for unprojected data, projecting it, and plotting it over the map shown. An example is given in Fig. 1.1. This is the

code used to define and draw projected latitude/longitude grid lines and grid line labels for this figure, which uses the world map from package **maps**:

```
> library(maptools)
> library(maps)
> wrld <- map("world", interior = FALSE, xlim = c(-179,
+ 179), ylim = c(-89, 89), plot = FALSE)
> wrld_p <- pruneMap(wrld, xlim = c(-179, 179))
> llCRS <- CRS("+proj=longlat +ellps=WGS84")
> wrld_sp <- map2SpatialLines(wrld_p, proj4string = llCRS)
> prj_new <- CRS("+proj=moll")
> library(rgdal)
> wrld_proj <- spTransform(wrld_sp, prj_new)
> wrld_grd <- gridlines(wrld_sp, easts = c(-179, seq(-150,
+ 150, 50), 179.5), norths = seq(-75, 75, 15), ndiscr = 100)
> wrld_grd_proj <- spTransform(wrld_grd, prj_new)
> at_sp <- gridat(wrld_sp, easts = 0, norths = seq(-75,
+ 75, 15), offset = 0.3)
> at_proj <- spTransform(at_sp, prj_new)
> plot(wrld_proj, col = "grey60")
> plot(wrld_grd_proj, add = TRUE, lty = 3, col = "grey70")
> text(coordinates(at_proj), pos = at_proj$pos, offset = at_proj$offset,
+ labels = parse(text = as.character(at_proj$labels)),
+ cex = 0.6)
```

Here, function `gridat` returns an object to draw the labels for these ‘gridded curves’.

3.1.4 Plot Size, Plotting Area, Map Scale, and Multiple Plots

R distinguishes between figure region, which is the size of the total figure including axes, title, etc., and plotting region, which is the area where the actual data are plotted. To control the total size of the figure, we can get and set the figure size in inches:

```
> par("pin")
[1] 5.76 5.16
```

```
> par(pin = c(4, 4))
```

If we want to enlarge the plotting window, we may have to close the current plotting device and re-open it specifying size, for example

```
> dev.off()
> X11(width = 10, height = 10)
```

on Unix machines; replace `X11` with `windows` on MS-Windows computers and with `quartz` on Mac OS X. When graphic output is written to files, we can use, for example

```
> pdf("file.pdf", width = 5, height = 7)
```

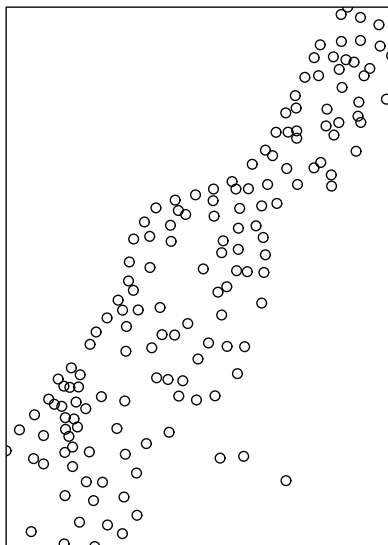


Fig. 3.7 Plotting region *exactly* equal to sample location ranges: border point symbols are clipped

The geographical (data) area that is shown on a plot is by default that of the data, extended with a 4% margin on each side. Because the plot size is fixed before plotting, only one of the axes will cover the entire plotting region, the other will be centred and have larger margins. We can control the data area plotted by passing `xlim` and `ylim` in a plot command, but by default they will still be extended with 4% on each side. To prevent this extension, we can set `par(xaxs="i")` and `par(yaxs="i")`. In the following example

```
> pin <- par("pin")
> dxy <- apply(bbox(meuse), 1, diff)
> ratio <- dxy[1]/dxy[2]
> par(pin = c(ratio * pin[2], pin[2]), xaxs = "i", yaxs = "i")
> plot(meuse, pch = 1)
> box()
```

we first set the aspect of the plotting region equal to that of the data points, and then we plot the points without allowing for the 4% extension of the range in all directions. The result (Fig. 3.7) is that in all four sides one plotting symbol is clipped by the plot border.

If we want to create more than one map in a single figure, as was done in Fig. 3.1, we can sub-divide the figure region into a number of sub-regions. We can split the figure into two rows and three columns either by

```
> par(mfrow = c(2, 3))
or
> layout(matrix(1:6, 2, 3, byrow = TRUE))
```

Table 3.2 Useful annotation arguments to be passed to `plot` or `image` methods

Class(es)	Argument	Meaning	Further help
SpatialLinesDataFrame	<code>col</code>	Colour	?lines
	<code>lwd</code>	Line width	?lines
	<code>lty</code>	Line type	?lines
SpatialPolygonsDataFrame	<code>border</code>	Border colour	?polygon
	<code>density</code>	Hashing density	?polygon
	<code>angle</code>	Hashing angle	?polygon
	<code>lty</code>	Line type	?polygon
	<code>pbg</code>	Hole colour	
SpatialPointsDataFrame	<code>pch</code>	Symbol	?points
	<code>col</code>	Colour	?points
	<code>bg</code>	Fill colour	?points
	<code>cex</code>	Symbol size	?points
SpatialPixelsDataFrame ^a and	<code>zlim</code>	Attribute value limits	?image.default
	<code>col</code>	Colours	?image.default
SpatialGridDataFrame	<code>breaks</code>	Break points	?image.default

^aUse `image` to plot gridded data

Each time a plot command that would normally create a new plot is called (i.e. without `add = TRUE`), a plot is drawn in a new sub-area; this is done row-wise for this example, or column-wise when `byrow = FALSE`. Function `layout` also allows us to vary the height and width of the sub-areas.

Map scale is the ratio between the length of one unit on the map and one unit in the real world. It can only be controlled ahead of time when both the size of the plotting region, which is by default only a part of the figure size unless all margins are set to zero, and the plotting area are defined, or otherwise exactly known.

3.1.5 Plotting Attributes and Map Legends

Up to now we have only plotted the geometry or topology of the spatial objects. If in addition we want to show feature characteristics or *attributes* of the objects, we need to use type, size, or colour of the symbols, lines, or polygons. Grid cells are usually plotted as small adjacent squares, so their plotting is in some sense a special case of plotting polygons. Table 3.2 lists the graphic arguments that can be passed to the `plot` methods for the `Spatial` classes with attributes. When a specific colour, size, or symbol type refers to a specific numeric value or category label of an attribute, a map legend is needed to communicate this information. Example code for function `legend` is given below and shown in Fig. 3.8.

We provide `image` methods for objects of class `SpatialPixelsDataFrame` and `SpatialGridDataFrame`. As an example, we can plot interpolated (see

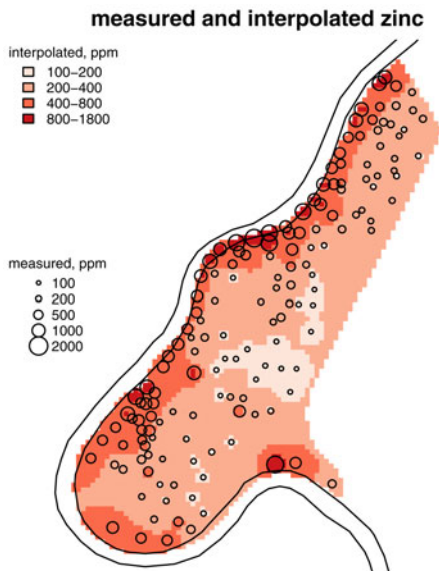


Fig. 3.8 Sample data points for zinc (ppm) plotted over an interpolated image, with symbol area proportional to measured concentration

Chap.8) zinc concentration (`zinc.idw`) as a background image along with the data:

```
> grays = gray.colors(4, 0.55, 0.95)
> image(zn.idw, col = grays, breaks = log(c(100, 200, 400,
+      800, 1800)))
> plot(meuse.pol, add = TRUE)
> plot(meuse, pch = 1, cex = sqrt(meuse$zinc)/20, add = TRUE)
> legVals <- c(100, 200, 500, 1000, 2000)
> legend("left", legend = legVals, pch = 1, pt.cex = sqrt(legVals)/20,
+       bty = "n", title = "measured")
> legend("topleft", legend = c("100-200", "200-400", "400-800",
+       "800-1800"), fill = grays, bty = "n", title = "interpolated")
```

the result of which is shown in Fig. 3.8. This example shows how the `legend` command is used to place two legends, one for symbols and one for colours. In this example, rather light grey tones are used in order not to mask the black symbols drawn.

3.2 Trellis/Lattice Plots with `splot`

Apart from the traditional `plot` methods provided by package `sp`, a second method, called `splot`, provides plotting of spatial data with attributes through the Trellis graphics system (Cleveland 1993, 1994), which is for R

provided (and extended) by package **lattice** (Sarkar 2008). Trellis plots are a bit harder to deal with initially because plot annotation, the addition of information like **legend**, **lines**, **text**, etc., is handled differently and needs to be thought out first. The advantage they offer is that many maps can be composed into single (sets of) graphs, easily and efficiently.

3.2.1 A Straight Trellis Example

Consider the plotting of two interpolation scenarios for the zinc variable in the **meuse** data set, obtained on the direct scale and on the log scale. We can do this either by the **levelplot** function from **lattice**, as in

```
> library(lattice)
> levelplot(z ~ x + y | name, spmap.to.lev(zn[c("direct",
+      "log")])), asp = "iso")
```

or equivalently by using **spplot**, which for grids is a simple wrapper around **levelplot** and simplifies to

```
> spplot(zn[c("direct", "log")])
```

The results of this are shown in Fig. 3.9. Function **levelplot** needs a **data.frame** as second argument with the grid values for both maps in a single column (**z**) and a factor (**name**) to distinguish between them. Helper function **spmap.to.lev** converts the **SpatialPixelsDataFrame** object to this format by replicating the coordinates, stacking the attribute variables, and adding a factor to distinguish the two maps. Function **spplot** plots each attribute passed in a single panel, which results in this case in two panels.

The **spplot** method does all this too, but hides many details. It provides a simple access to the functions provided by package **lattice** for plotting objects deriving from class **Spatial**, while retaining the flexibility offered by **lattice**. It also allows for adding geographic reference elements to maps.

Note that the plot shows four dimensions: the geographic space spanning *x*- and *y*-coordinates, the attribute values displayed in colour or grey tone, and the *panel* identifier, here the interpolation scenario but which may be used to denote, for example attribute variable or time.

3.2.2 Plotting Points, Lines, Polygons, and Grids

Function **spplot** plots spatial objects using colour (or grey tone) to denote attribute values. The first argument therefore has to be a spatial object with attributes.

Figure 3.10 shows a typical plot with four variables. If the goal is to compare the absolute levels in ppm across the four heavy metal variables, it

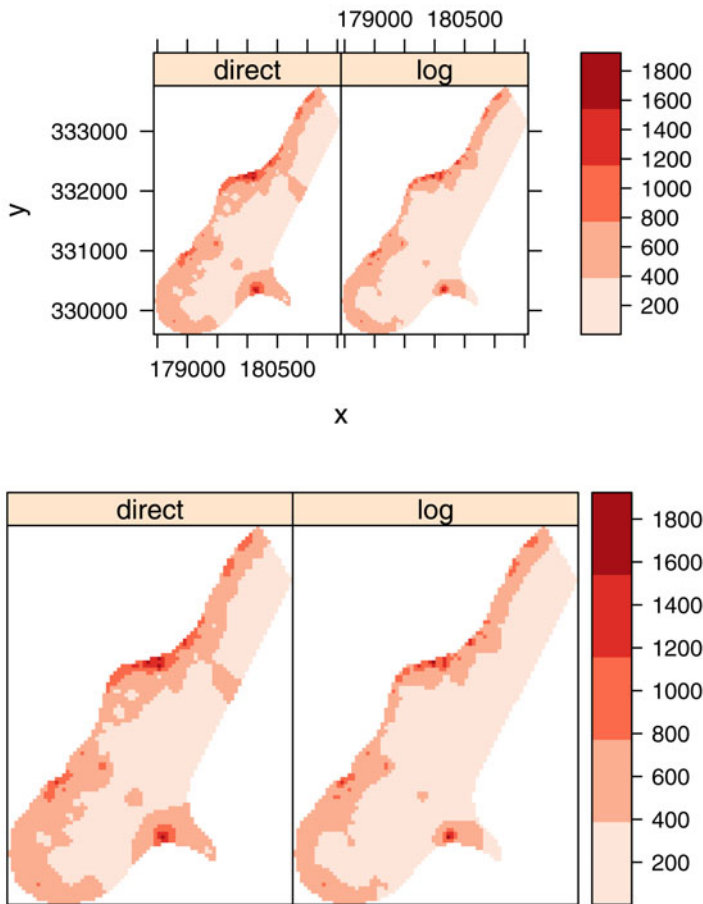


Fig. 3.9 Two interpolation scenarios for the `meuse` data set, plotted on the same total size. *Top*: plotted by `levelplot`, *bottom*: plotted by `splot`, which turns off drawing of axes scales

makes sense to plot them in a single figure with one legend. For such cases, the conditioning plots of `splot` are ideal. Other cases in which multiple sub-maps are useful are, for example when different moments of time or different modelling scenarios are used to define the factor that splits the data over sub-plots (panels).

The first argument to `splot` is a `Spatial*DataFrame` object with points, lines, polygons, or a grid. The second argument tells which attributes (column names or numbers) should be used; if omitted, all attributes are plotted. Further attributes control the plotting: colours, symbols, legend classes, size, axes, and geographical reference items to be added.

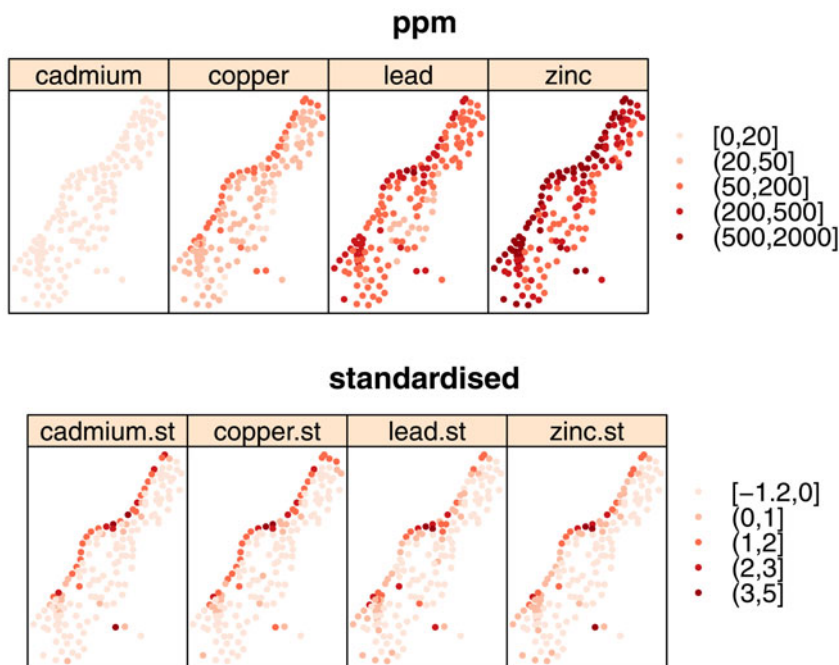


Fig. 3.10 Soil measurements for four heavy metals in the Meuse data set; *upper*: in ppm units, *lower*: each variable scaled to mean zero and unit standard variance

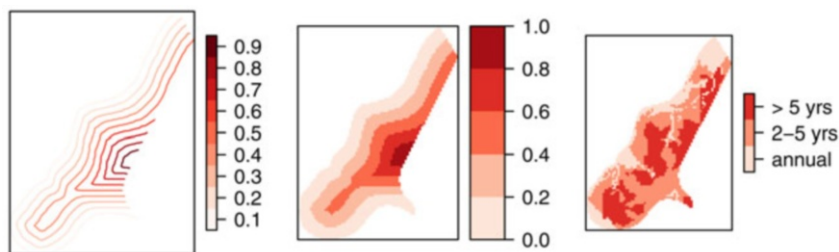


Fig. 3.11 *Left*: contour lines for distance to river Meuse, levels represented by grey tones; *middle*: grid plot of a numerical variable; *right*: plot of the factor variable flood frequency; note the different color key

An example of a `SpatialLinesDataFrame` plot is shown in Fig. 3.11 (left), where the R function `contourLines` is used to calculate the contourlines:

```
> library(maptools)
> data(meuse.grid)
> coordinates(meuse.grid) <- c("x", "y")
> meuse.grid <- as(meuse.grid, "SpatialPixelsDataFrame")
> im <- as.image.SpatialGridDataFrame(meuse.grid["dist"])
> cl <- ContourLines2SLDF(contourLines(im))
> splot(cl)
```

3.2.3 Adding Reference and Layout Elements to Plots

Method `splot` takes a single argument, `sp.layout`, to annotate plots with lines, points, grids, polygons, text, or combinations of these. This argument contains either a single layout item or a list of layout items. A single layout item is a list object. Its first component is the name of the layout function to be called, followed by the object to be plotted and then optional arguments to adjust colour, symbol, size, etc. The layout functions provided are the following:

sp layout function	Object class	Useful arguments ^a
<code>sp.points</code>	<code>SpatialPoints</code>	<code>pch</code> , <code>cex</code> , <code>col</code>
<code>sp.polygons</code>	<code>SpatialPolygons</code>	<code>lty</code> , <code>lwd</code> , <code>col</code>
<code>sp.lines</code>	<code>SpatialLines</code>	<code>lty</code> , <code>lwd</code> , <code>col</code>
<code>sp.text</code>	<code>text</code>	(see <code>panel.text</code>)

^aFor help, see `?par`

An example of building an `sp.layout` structure is as follows:

```
> river <- list("sp.polygons", meuse.pol)
> north <- list("SpatialPolygonsRescale", layout.north.arrow(),
+   offset = c(178750, 332500), scale = 400)
> scale <- list("SpatialPolygonsRescale", layout.scale.bar(),
+   offset = c(180200, 329800), scale = 1000, fill = c("transparent",
+   "black"))
> txt1 <- list("sp.text", c(180200, 329950), "0")
> txt2 <- list("sp.text", c(181200, 329950), "1 km")
> pts <- list("sp.points", meuse, pch = 3, col = "black")
> meuse.layout <- list(river, north, scale, txt1, txt2,
+   pts)

> splot(zn["log"], sp.layout = meuse.layout)
```

the result of which is shown in Fig. 3.12. Although the construction of this is more elaborate than annotating base plots, as was done for Fig. 3.5, this method seems better for the larger number of graphs as shown in Fig. 3.10.

A special layout element is `which` (integer), to control to which panel a layout item should be added. If `which` is present in the top-level list it applies to all layout items; in sub-lists with layout items it denotes the panel or set of panels in which the layout item should be drawn. Without `which`, layout items are drawn in each panel.

The order of items in the `sp.layout` argument matters; in principle objects are drawn in the order they appear. By default, when the object of `splot` has points or lines, `sp.layout` items are drawn before the points to allow grids and polygons drawn as a background. For grids and polygons, `sp.layout` items are drawn afterwards (so the item will not be overdrawn by the grid and/or polygon). For grids, adding a list element `first = TRUE` ensures that the item is drawn *before* the grid is drawn (e.g. when filled polygons are

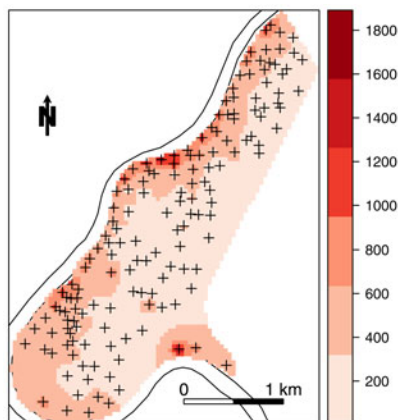


Fig. 3.12 Interpolated `spplot` image with layout elements

added). Transparency may help when combining layers; it is available for the PDF device and several other devices.

Function `sp.theme` returns a `lattice` theme that can be useful for plots made by `spplot`; use `trellis.par.set(sp.theme())` after a device is opened or changed to make this effective. Currently, this only sets the colours to `bpy.colors`.

3.2.4 Arranging Panel Layout

The default layout of `spplot` plots is computed by (i) the dimensions of the graphics device and the size of each panel and (ii) a row-wise ordering, starting top-left. The row-wise ordering can be started bottom-left if `as.table = FALSE` is passed to the `spplot` call. Note that `FALSE` is the default value for functions in `lattice`.

Besides `as.table`, panel layout can be modified with the `layout` and `skip` arguments. Argument `layout` is a numeric vector with the number of columns and the number of rows, for example `layout = c(3,4)` will result in three columns and four rows. Argument `skip` can be used to leave certain panels blank, in plotting order: `layout = c(3,3)`, `skip = c(F,T,T,F,F,T,F,F,F)` will plot six panels in a lower triangular 3×3 panel matrix. Figure 8.10 gives an example of this. More information about `layout`, `skip`, and `as.table` can be found in the help for `lattice` function `xyplot`.

3.3 Alternatives Routes: **ggplot**, **latticeExtra**

Package **ggplot2** (Wickham 2009) provides a suite of plotting methods that is not very dissimilar to those in **lattice**, but have by default a notably different *look*, and a very different interface. The main function, **ggplot**, takes an object and will try to convert it to a **data.frame** using method **fortify**. Several methods are provided, and we can see from

```
> library(ggplot2)
> methods(fortify)
```

that **fortify** methods exist for **Line**, **Lines**, **Polygon**, **Polygons**, **SpatialPolygons**, **SpatialLinesDataFrame**, **SpatialPolygonsDataFrame**, and for **map** objects. For **SpatialPointsDataFrame** objects such as in **meuse**, we need to do the conversion ourselves, and

```
> m = as(meuse, "data.frame")
> ggplot(m, aes(x, y)) + geom_point() + coord_equal()
```

gives the map shown in Fig. 3.13. In this command,

- **aes** is used to specify which variables need to be plotted on the *x* and *y*-axis,
- **geom_point()** dictates that the plot should be a scatter plot, and
- **coord_equal()** makes sure that units along the *x*-axis equal those along the *y*-axis (data are projected).

An overview of all functions provided, with graphical examples, is found on <http://ggplot2.org/>. The grey background with white lines is the default in **ggplot2**.

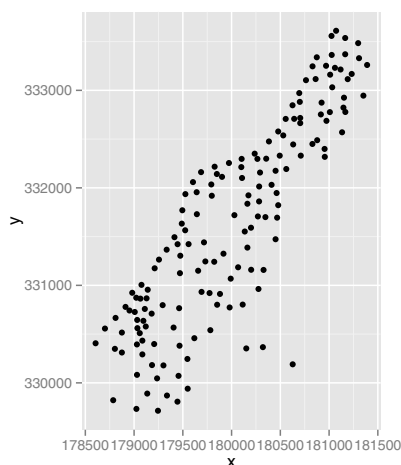


Fig. 3.13 **ggplot** of the **meuse** data points

The *gg* in **ggplot** refers to *grammar of graphics* (Wilkinson 2005). We see in the above expression that components are connected with a `+`, which may be seen as a grammatical construct. More intuitively, the `+` can also be used to add, as of putting one thing on top of another.

Another effort to combine different graphic elements with the `+` operator is provided by **latticeExtra**. As objects returned by **splot** are **trellis** objects, they can be combined as follows:

```
> library(latticeExtra)
> p = splot(meuse["zinc"])
> m = SpatialPolygonsDataFrame(meuse.pol, data.frame(col = 1),
+   match.ID = FALSE)
> l = splot(m)
> l + p
> p + l
```

the result of which is shown in Fig. 3.14. Package **rasterVis** further uses **latticeExtra** methods to improve visualisation of spatial data, in particular raster maps based on **raster**.

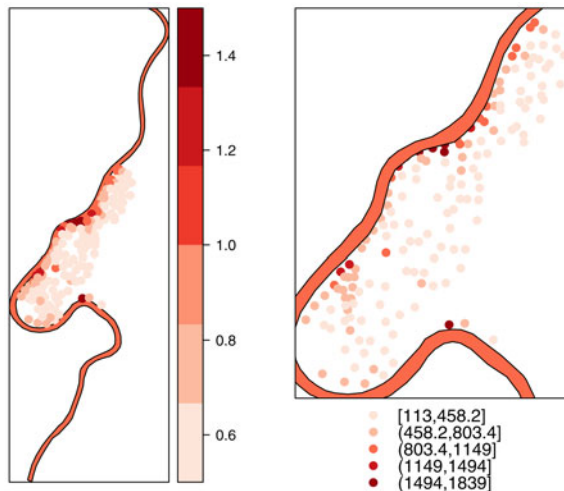


Fig. 3.14 Two plots produced by `+` from **latticeExtra**. *Left*: color key and extent are taken from the river polygon; *right*: color key and extent are taken from the **zinc** point measurements

3.4 Interactive Plots

The interaction R allows with plots in the traditional and **lattice** plot systems is rather limited, compared with stand-alone software written for interacting with data, or GIS. The main functionality is centred around which information is present at the location where a mouse is clicked.

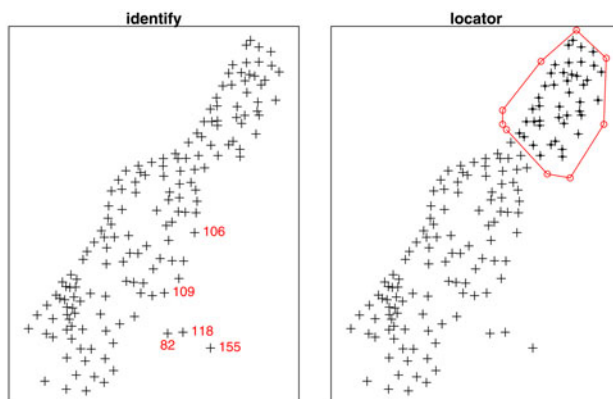


Fig. 3.15 Interaction with point plots. *Left*: individual identification of points; *right*: digitising a region, highlighted points included in the region

3.4.1 Interacting with Base Graphics

Base graphics has two functions to interact with interactive (i.e. screen) graphic devices:

`locator` returns the locations of points clicked, in coordinates of the x - and y -axis

`identify` plots and returns the labels (by default: row number) of the items nearest to the location clicked, within a specified maximum distance (0.25 in. in plot units, by default).

Both functions wait for user input; left mouse clicks are registered; a right mouse click ends the input. An example session for `identify` may look like this:

```
> plot(meuse)
> meuse.id <- identify(coordinates(meuse))
```

and the result may look like the left side of Fig. 3.15. An example digitise session, followed by selection and re-plotting of points within the area digitised may be as follows:

```
> plot(meuse)
> region <- locator(type = "o")
> n <- length(region$x)
> p <- Polygon(cbind(region$x, region$y)[c(1:n, 1), ],
+             hole = FALSE)
> ps <- Polygons(list(p), ID = "region")
> sps <- SpatialPolygons(list(ps))
> plot(meuse[sps, ], pch = 16, cex = 0.5, add = TRUE)
```

with results in the right-hand side of Fig. 3.15. Note that we ‘manually’ close the polygon by adding the first point to the set of points digitised.

To identify particular polygons, we can use `locator` and overlay the points with the polygon layer shown in Fig. 3.6:

```
> library(maptools)
> prj <- CRS("+proj=longlat +datum=NAD27")
> nc_shp <- system.file("shapes/sids.shp", package = "maptools")[1]
> nc <- readShapePoly(nc_shp, proj4string = prj)

> plot(nc)
> pt <- locator(type = "p")
> print(pt)
$x
[1] -78.69484

$y
[1] 35.8044

> pt.sp = SpatialPoints(cbind(pt$x, pt$y), proj4string = prj)
> over(pt.sp, nc)

      AREA PERIMETER CNTY_ CNTY_ID NAME  FIPS FIPSNO CRESS_ID BIR74
1 0.219      2.13  1938    1938 Wake 37183  37183      92 14484
SID74 NWBIR74 BIR79 SID79 NWBIR79
1    16    4397 20857    31    6221
```

3.4.2 Interacting with `spplot` and Lattice Plots

In R, Trellis (**lattice**) plots have the same interaction functionality as base plots. However, the process is a bit more elaborate because multiple panels may be present. To select points with `spplot`, use

```
> ids <- spplot(meuse, "zinc", identify = TRUE)
```

This will show the points selected and return the selected points' row numbers.

In essence, and what the above function hides, we first select a panel, then identify within this panel, and finally unselect it, which is accomplished by the **lattice** functions

```
> library(lattice)
> trellis.focus("panel", column = 1, row = 1)
> ids <- panel.identify()
> trellis.unfocus()
```

Digitising can be done by the function `grid.locator` from package **grid**, which underlies the functionality in **lattice**. A single point is selected by

```
> library(grid)
> trellis.focus("panel", column = 1, row = 1)
> as.numeric(grid.locator())
> trellis.unfocus()
```

Package **sp** contains a simple function `spplot.locator` to return a digitised area, simulating the base plot `locator` behaviour. It returns a two-column matrix with spatial coordinates.

3.5 Colour Palettes and Class Intervals

3.5.1 Colour Palettes

R provides a number of colour palettes, and the functions providing them are self-descriptive: `rainbow`, `grey.colors`, `heat.colors`, `terrain.colors`, `topo.colors`, and `cm.colors` (cm for cyan-magenta) – `cm.colors` are the default palette in `spplot` and diverge from white. For quantitative data, shades in a single colour are usually preferred. These can be created by `colorRampPalette`, which creates a color interpolating function taking the required number of shades as argument, as in

```
> rw.colors <- colorRampPalette(c("red", "white"))
> image(meuse.grid["dist"], col = rw.colors(10))
```

Package **RColorBrewer** provides the palettes described (and printed) in Brewer et al. (2003) for continuous, diverging, and categorical variables. An interface for exploring how these palettes look on maps is found in the `colorbrewer` applet.²

It also has information on suitability of each of the palettes for colour-blind people, black-and-white photo-copying, projecting by LCD projectors, use on LCD or CRT screens, and for colour printing. Another, non-interactive, overview is obtained by

```
> library(RColorBrewer)
> example(brewer.pal)
```

Package **sp** provides the ramp `bpy.colors` (blue-pink-yellow), which has the advantage that it has many colors and that it prints well both on color and black-and-white printers.

3.5.2 Class Intervals

Although we can mimic continuous variation by choosing many (e.g. 100 or more) colours, matching map colours to individual colours in the legend is approximate. If we want to communicate changes connected to certain fixed levels, for example levels related to regulation, or if we for other reasons want differentiable or identifiable class intervals, we should limit the number of classes to, for example six or less.

Class intervals can be chosen in many ways, and some have been collected for convenience in the **classInt** package. The first problem is to assign class boundaries to values in a single dimension, for which many classification

² See <http://www.colorbrewer.org/>.

techniques may be used, including pretty, quantile, and natural breaks among others, or even simple fixed values. From there, the intervals can be used to generate colours from a colour palette as discussed earlier. Because there are potentially many alternative class memberships even for a given number of classes (by default from `nclass.Sturges`), choosing a communicative set matters.

We try just two styles, quantiles and Fisher-Jenks natural breaks for five classes (Slocum et al. 2005, pp. 85–86), among the many available – for further documentation see the help page of the `classIntervals` function. They yield quite different impressions, as we see:

```
> library(RColorBrewer)
> library(classInt)
> pal <- brewer.pal(5, "Reds")
> q5 <- classIntervals(meuse$zinc, n = 5, style = "quantile")
> q5

style: quantile
  one of 14,891,626 possible partitions of this variable into 5 classes
    [113,186.8) [186.8,246.4) [246.4,439.6) [439.6,737.2) [737.2,1839]
           31             31             31             31             31

> diff(q5$brks)

[1] 73.8 59.6 193.2 297.6 1101.8

> plot(q5, pal = pal)
```

The empirical cumulative distribution function, used in the plot method for the `classIntervals` object returned, suggests that using quantiles is not necessarily a good idea. While of course the number of sites in each class is equal by definition, the observed values are far from uniformly distributed. Examining the widths of the classes using `diff` on the class breaks shows that many sites with moderate zinc values will be assigned to the darkest colour class. Figure 3.16 shows the plot of this class interval set compared with that for a five-class Fisher-Jenks classification. There are two implementations of this style, one named ‘fisher’, the other ‘jenks’. This ‘natural breaks’ set of class intervals is based on minimising the within-class variance, like many of the other styles available.

```
> fj5 <- classIntervals(meuse$zinc, n = 5, style = "fisher")
> fj5

style: fisher
  one of 14,891,626 possible partitions of this variable into 5 classes
    [113,307.5) [307.5,573) [573,869.5) [869.5,1286.5)
           75             32             29             12
[1286.5,1839]
           7
```

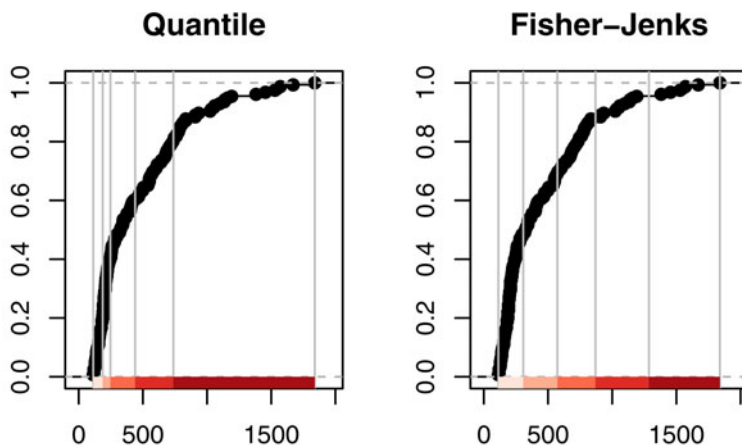


Fig. 3.16 Comparison of quantile and natural breaks methods for setting class intervals, Meuse bank zinc ppm

```
> diff(fj5$brks)
[1] 194.5 265.5 296.5 417.0 552.5
> plot(fj5, pal = pal)
```

Once we are satisfied with the chosen class intervals and palette, we can go on to plot the data, using the `findColours` function to build a vector of colours and attributes, which can be used in constructing a legend:

```
> q5Colours <- findColours(q5, pal)
> plot(meuse, col = q5Colours, pch = 19)
> legend("topleft", fill = attr(q5Colours, "palette"),
+       legend = names(attr(q5Colours, "table")), bty = "n")
```

The output for these two classifications is shown in Fig. 3.17, and does show that choice of representation matters. Using quantile-based class intervals, it appears that almost all the river bank sites are equally polluted, while the natural breaks intervals discriminate better.

For image, we can specify the `breaks` argument, as was done in Fig. 3.8. While the `classIntervals` function can be used with raster data, it may be prudent to search for class intervals using a sample of the input data, including the extremities to save time; this heuristic is used by many GIS. The default class interval style used by image is to divide the range into a number of classes of equal width (equivalent to the equal or pretty styles in `classIntervals`). With very skewed data, for example 2D density plots, this may give the impression of the data having disappeared, because almost all the cells will be in one extreme class, and only a few in other classes. Changing the class intervals will ‘magically’ reveal the data.

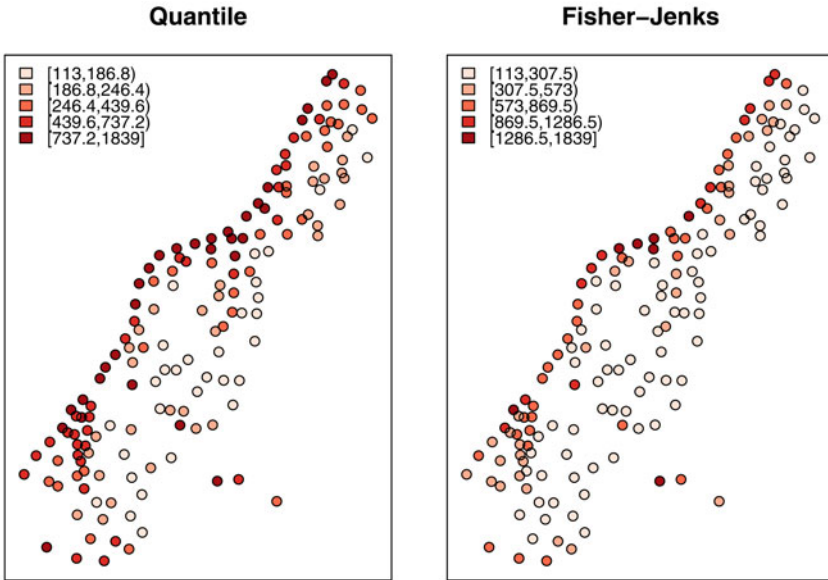


Fig. 3.17 Comparison of output maps made with quantile and natural breaks class intervals, Meuse bank zinc ppm

For the `splot` methods for lines, polygons, and grids, we can pass the argument `pretty = TRUE`, which ensures that colour breaks coincide with legend values (see right-hand side of Fig. 3.11). To specify class intervals with `splot`, for points data we can pass the `cuts` argument, and for lines, polygons, or grids we can pass the `at` argument. To also control the key tic marks and labels, we need to specify `colorkey` as well. For example, the middle plot of Fig. 3.11 was created by:

```
> cuts = (0:10)/10
> splot(meuse.grid, "dist", colorkey = list(labels = list(at = cuts)),
+       at = cuts)
```

Having provided a framework for handling and visualising spatial data in R, we now move to demonstrate how user data may be imported into R, and the results of analysis exported.

Chapter 4

Spatial Data Import and Export

Geographical information systems (GIS) and the types of spatial data they handle were introduced in Chap. 1. We now show how spatial data can be moved between **sp** objects in R and external formats, including the ones typically used by GIS. In this chapter, we first show how coordinate reference systems can be handled portably for import and export, going on to transfer vector and raster data, and finally consider ways of linking R and GIS more closely.

Before we begin, it is worth noting the importance of open source projects in making it possible to offer spatial data import and export functions in R. Many of these projects are now gathered in the Open Source Geospatial Foundation.¹ There are some projects which form the basis for the others, in particular the Geospatial Data Abstraction Library² (GDAL, pronounced Góodal, coordinated by Frank Warmerdam). Many of the projects also use the PROJ.4 Cartographic Projections library,³ originally written by Gerald Evenden then of the United States Geological Survey, and modified and maintained by Frank Warmerdam. Without access to such libraries and their communities, it would not be possible to provide import or export facilities for spatial data in R. Many of the open source toolkits are also introduced in depth in Mitchell (2005) and Hall and Leahy (2008). As we proceed, further links to relevant sources of information, such as mailing list archives, will be given.

In this chapter, we consider the representation of coordinate reference systems in a robust and portable way. Next, we show how spatial data may be read into R, and be written from R, using the most popular formats. The interface with GRASS GIS will be covered in detail, and finally the export of data for visualisation will be described.

¹ <http://www.osgeo.org/>.

² <http://www.gdal.org/>.

³ <http://trac.osgeo.org/proj/>.

First, we show how loading the package providing most of the interfaces to the software of these open source projects, **rgdal**, reports their status:

```
> library(rgdal)

rgdal: version: 0.8-5, (SVN revision 449)
Geospatial Data Abstraction Library extensions to R successfully loaded
Loaded GDAL runtime: GDAL 1.9.2, released 2012/10/08
Path to GDAL shared files: /usr/local/share/gdal
Loaded PROJ.4 runtime: Rel. 4.8.0, 6 March 2012, [PJ_VERSION: 480]
Path to PROJ.4 shared files: (autodetected)
```

We see that the release version numbers and dates of the external dynamically loaded libraries are reported. In addition, the values of the package version, code revision number, and paths to GDAL and PROJ.4 metadata directories are reported.⁴

4.1 Coordinate Reference Systems

Spatial data vary a great deal both in the ways in which their position attributes are recorded and in the adequacy of documentation of how position has been determined. This applies both to data acquired from secondary sources and to Global Positioning System input, or data capture from analogue maps by digitising. This also constitutes a specific difference from the analysis say of medical imagery, which in general requires only a local coordinate system; astronomy and the mapping of other planets also constitute a separate but linked field. Knowledge about the coordinate reference system is needed to establish the positional coordinates' units of measurement, obviously needed for calculating distances between observations and for describing the network topology of their relative positions. This knowledge is essential for integrating spatial data for the same study area, but coming from different sources. Waller and Gotway (2004, pp. 40–47) describe some of the key concepts and features to be dealt with here in an accessible fashion.

Coordinate reference systems (CRS) are at the heart of geodetics and cartography: how to represent a bumpy ellipsoid on the plane. We can speak of geographical CRS expressed in degrees and associated with an ellipse – a model of the shape of the earth, a prime meridian defining the origin in longitude, and a datum. The concept of a datum is arbitrary and anchors a specific geographical CRS to an origin point in three dimensions, including an assumed height above the assumed centre of the earth or above a standard measure of sea level. Since most of these quantities have only been subject to accurate measurement since the use of satellites for surveying became common, changes in ellipse and datum characteristics between legacy maps and newly collected data are not unusual.

⁴ The report returned when loading **rgdal** may be suppressed by wrapping the call in `suppressPackageStartupMessages`.

In contrast, projected CRS are expressed by a specific geometric model projecting to the plane and measures of length, as well as the underlying ellipse, prime meridian, and datum. Most countries have multiple CRS, often for very good reasons. Surveyors in cities have needed to establish a local datum and a local triangulation network, and frequently these archaic systems continue to be used, forming the basis for property boundaries and other legal documents.

Cartography and surveying has seen the development of national triangulations and of stipulated national projections, or sub-national or zoned projections for larger countries. Typically, problems arise where these regimes meet. The choices of ellipse, prime meridian, and datum may differ, and the chosen projection and metric may also differ, or have different key parameters or origin offsets. On land, national borders tend to be described adequately with reference to the topography, but at sea, things change. It was because the coastal states around the North Sea basin had incompatible and not fully defined CRS that the European Petroleum Survey Group (EPSG; now Oil & Gas Producers (OGP) Surveying & Positioning Committee) began collecting a geodetic parameter data set⁵ starting in 1986, based on earlier work in member companies.

The PROJ.4 library does not report the version of the EPSG list distributed with releases of PROJ.4, but this may be discovered by reading the NEWS file and extracting lines reporting the PROJ.4 release from newest to oldest:

```
> NEWS <- "http://svn.osgeo.org/metacrs/proj/trunk/proj/NEWS"
> PROJ4_NEWS <- readLines(url(NEWS))

> lns <- grep("Release Notes|EPSG", PROJ4_NEWS)
> head(PROJ4_NEWS[lns])

[1] "4.8.0 Release Notes"
[2] "  o Upgrade to EPSG 7.9.  Some changes in ideal datum selection."
[3] "4.7.0 Release Notes"
[4] "  o Regenerated nad/epsg init file with EPSG 7.1 database,
    including new"
[5] "    support for Google Mercator (EPSG:3857)."
```

```
[6] "4.6.1 Release Notes"
```

4.1.1 Using the EPSG List

The EPSG list is under continuous development, with corrections being made to existing entries, and new entries being added as required. Copies of the list are provided in GDAL and PROJ.4, and in Windows and OSX binary

⁵ <http://www.epsg.org/>.

rgdal packages,⁶ because the list permits the conversion of a large number of CRS into the PROJ.4 style description used here. Since it allows for datum transformation as well as projection, the number of different coordinate reference systems is very much larger than that in the **mapproj** package. Datum transformation is based on transformation to the World Geodetic System of 1984 (WGS84), or inverse transformation from it to an alternative specified datum. WGS84 was introduced after measurements of earth from space had become very accurate, and forms a framework into which local and national systems may be fitted.

The **rgdal** package copy of the EPSG list can be read into a data frame and searched using **grep**, for example. We try to reproduce the example formerly given by the Royal Netherlands Navy entitled ‘From ED50 towards WGS84, or does your GPS receiver tell you the truth?’ A position has been read from a chart in the ED50 datum about a nautical mile west of the jetties of IJmuiden, but needs to be converted to the WGS84 datum for comparison with readings from a GPS satellite navigation instrument. We need to transform the chart coordinates in ED50 – ED50 is the European Datum 1950 – to coordinates in the WGS84 datum (the concept of a datum is described on p. 84). In this case to save space, the search string has been chosen to match exactly the row needed; entering just ED50 gives 35 hits:

```
> EPSG <- make_EPSG()
> EPSG[grep("^# ED50$", EPSG$note), ]

      code  note
159 4230 # ED50

                                prj4
159 +proj=longlat +ellps=intl +towgs84=-87,-98,-121,0,0,0,0 +no_defs
```

The EPSG code is in the first column of the data frame and the PROJ.4 specification in the third column, with the known set of tags and values.

4.1.2 PROJ.4 CRS Specification

The PROJ.4 library uses a ‘tag=value’ representation of coordinate reference systems, with the tag and value pairs enclosed in a single character string. This is parsed into the required parameters within the library itself. The only values used autonomously in **CRS** class objects are whether the string is a character **NA** (missing) value for an unknown CRS, and whether it contains the string **longlat**, in which case the CRS contains geographical coordinates.⁷ There are a number of different tags, always beginning with **+**, and separated from the value with **=**, using white space to divide the tag/value pairs from

⁶ See installation note at chapter end, p. 125.

⁷ The value **latlong** is not used, although valid, because coordinates in **sp** class objects are ordered with eastings first followed by northings.

each other.⁸ If we use the special tag `+init` with value `epsg:4230`, where 4230 is the EPSG code found above, the coordinate reference system will be populated from the tables supplied with the libraries (PROJ.4 and GDAL) and included in **rgdal**.

```
> CRS("+init=epsg:4230")
```

CRS arguments:

```
+init=epsg:4230 +proj=longlat +ellps=intl
+towgs84=-87,-98,-121,0,0,0,0 +no_defs
```

The three tags that are known in this version of the EPSG list are `+proj` – projection, which takes the value `longlat` for geographical coordinates – `+ellps` – ellipsoid, with value `intl` for the International Ellipsoid of 1909 (Hayford), and `+towgs84`, with a vector of three non-zero parameters for spatial translation (in geocentric space, ΔX , ΔY , ΔZ). Had seven parameters been given, they would permit shifting by translation + rotation + scaling; note that sources may vary in the signs of the parameters. There was no `+towgs84` tag given for this EPSG code in the first edition of this book, using EPSG 6.13,⁹ and so without further investigation it was then not possible to make the datum transformation. Lots of information about CRS in general can be found in *Grids & Datums*,¹⁰ a regular column in Photogrammetric Engineering & Remote Sensing. The February 2003 number covers the Netherlands and gives a three-parameter transformation; adding $\Delta X = 87 \pm 3$ m, $\Delta Y = -96 \pm 3$ m, $\Delta Z = -120 \pm 3$ m, with a sign change on ΔX , gives an alternative specification (note that the EPSG `+towgs84` values are within the tolerances of the *Grids & Datums* values):

```
> ED50 <- CRS("+init=epsg:4230 +towgs84=-87,-96,-120,0,0,0,0")
> ED50
```

CRS arguments:

```
+init=epsg:4230 +towgs84=-87,-96,-120,0,0,0,0 +proj=longlat
+ellps=intl +no_defs
```

When **rgdal** is loaded in the running R session, the proposed tags are verified against the valid set, and additions, as here, override those drawn from the EPSG list. Datum transformation shifts coordinates between differently specified ellipsoids in all three dimensions, even if the data appear to be only 2D, because 2D data are assumed to be on the surface of the ellipsoid. It may seem unreasonable that the user is confronted with the complexities of coordinate reference system specification in this way. The EPSG list provides a good deal of help, but assumes that wrong help is worse than no help. Modern specifications are designed to avoid ambiguity, and so this issue will become less troublesome with time, although old maps are going to be a source of data for centuries to come.

⁸ In addition to the EPSG list, there are many examples at the PROJ.4 website, for example: http://geotiff.maptools.org/proj_list/.

⁹ The version shipped with PROJ.4 4.8.0 is EPSG 7.9 as we saw above.

¹⁰ <http://www.asprs.org/Grids-Datums.html>.

4.1.3 Projection and Transformation

In the Dutch navy case, we do not need to project because the input and output coordinates are geographical:

```
> IJ.east <- as(char2dms("4d31'00\"E"), "numeric")
> IJ.north <- as(char2dms("52d28'00\"N"), "numeric")
> IJ.ED50 <- SpatialPoints(cbind(x = IJ.east, y = IJ.north),
+   proj4string = ED50)
> res <- spTransform(IJ.ED50, CRS("+proj=longlat +datum=WGS84"))
> x <- as(dd2dms(coordinates(res)[1]), "character")
> y <- as(dd2dms(coordinates(res)[2], TRUE), "character")
> cat(x, y, "\n")

4d30'55.294"E 52d27'57.195"N

> spDistsN1(coordinates(IJ.ED50), coordinates(res), longlat = TRUE) *
+   1000

[1] 124.0994

> library(maptools)
> gzAzimuth(coordinates(IJ.ED50), coordinates(res))

      x
-134.3674
```

Using correctly specified coordinate reference systems, we can reproduce the example successfully, with a 124 m shift between a point plotted in the inappropriate WGS84 datum and the correct ED50 datum for the chart:

For example: one who has read his position 52d28'00"N/ 4d31'00"E (ED50) from an ED50-chart, right in front of the jetties of IJmuiden, has to adjust this co-ordinate about 125 m to the Southwest The corresponding co-ordinate in WGS84 is 52d27'57"N/ 4d30'55"E.

The work is done by the `spTransform` method, taking any `Spatial*` object, and returning an object with coordinates transformed to the target CRS. There is no way of warping regular grid objects, because for arbitrary transformations, the new positions will not form a regular grid. The solution in this case is to convert the object to point locations, transform them to the new CRS, and interpolate to a suitably specified grid in the new CRS.

Two helper functions are also used here to calculate the difference between the points in ED50 and WGS84: `spDistsN1` and `gzAzimuth`. Function `spDistsN1` measures distances between a matrix of points and a single point, and uses Great Circle distances on the WGS84 ellipsoid if the `longlat` argument is `TRUE`. It returns values in kilometres, and so we multiply by 1,000 here to obtain metres. `gzAzimuth` gives azimuths calculated on the sphere between a matrix of points and a single point, which must be geographical coordinates, with north zero, and negative azimuths west of north. If we use the three translation parameters provided by the current EPSG list instead of those given in Grids and Datums, we find that the distance is almost 2 m greater, and the azimuth is slightly changed:

```

> proj4string(IJ.ED50) <- CRS("+init=epsg:4230")
> res <- spTransform(IJ.ED50, CRS("+proj=longlat +datum=WGS84"))
> spDistsN1(coordinates(IJ.ED50), coordinates(res), longlat = TRUE) *
+   1000

[1] 125.8692

> gzAzimuth(coordinates(IJ.ED50), coordinates(res))

      x
-133.8915

```

So far in this section we have used an example with geographical coordinates. There are many different projections to the plane, often chosen to give an acceptable representation of the area being displayed. There exist no all-purpose projections, all involve distortion when far from the centre of the specified frame, and often the choice of projection is made by a public mapping agency.

```

> EPSG[grep("Atlas", EPSG$note), 1:2]

      code      note
626  2163      # US National Atlas Equal Area
2328 3978      # NAD83 / Canada Atlas Lambert
2329 3979 # NAD83(CSRs) / Canada Atlas Lambert

> CRS("+init=epsg:2163")

+init=epsg:2163 +proj=laea +lat_0=45 +lon_0=-100 +x_0=0 +y_0=0
+a=6370997 +b=6370997 +units=m +no_defs

```

For example, the US National Atlas has chosen a particular CRS for its view of the continental US, with a particular set of tags and values to suit. The projection chosen has the value `laea`, which, like many other values used to represent CRS in PROJ.4 and elsewhere, is rather cryptic. Provision is made to access descriptions within the PROJ.4 library to make it easier to interpret the values in the CRS. The `projInfo` function can return several kinds of information in tabular form, and those tables can be examined to shed a little more light on the tag values.

```

> proj <- projInfo("proj")
> proj[proj$name == "laea", ]

      name      description
52 laea Lambert Azimuthal Equal Area

> ellps <- projInfo("ellps")
> ellps[grep("a=6370997", ellps$major), ]

      name      major      ell      description
42 sphere a=6370997.0 b=6370997.0 Normal Sphere (r=6370997)

```

It turns out that this CRS is in the Lambert Azimuthal Equal Area projection, using the sphere rather than a more complex ellipsoid, with its centre at 100° west and 45° north. This choice is well-suited to the needs of the Atlas, a compromise between coverage, visual communication, and positional accuracy.

All this detail may seem unnecessary, until the analysis we need to complete turns out to depend on data in different coordinate reference systems. At that point, spending time establishing as clearly as possible the CRS for our data will turn out to have been a wise investment. The same consideration applies to importing and exporting data – if their CRS specifications are known, transferring positional data correctly becomes much easier. Fortunately, for any study region the number of different CRS used in archived maps is not large, growing only when the study region takes in several jurisdictions. Even better, all modern data sources are much more standardised (most use the WGS84 datum), and certainly much better at documenting their CRS specifications.

4.1.4 Degrees, Minutes, and Seconds

In common use, the sign of the coordinate values may be removed and the value given a suffix of E or N for positive values of longitude or latitude and W or S for negative values. In addition, values are often recorded traditionally not as decimal degrees, but as degrees, minutes, and decimal seconds, or some truncation of this. These representations raise exactly the same questions as for time series, although time can be mapped onto the infinite real line, while geographical coordinates are cyclical – move 360° and you return to your point of departure. For practical purposes, geographical coordinates should be converted to decimal degree form; this example uses the Netherlands point that we have already met:

```
> IJ.dms.E <- "4d31'00\"E"
> IJ.dms.N <- "52d28'00\"N"
```

We convert these character strings to class ‘DMS’ objects, using function `char2dms`:

```
> IJ_east <- char2dms(IJ.dms.E)
> IJ_north <- char2dms(IJ.dms.N)
> IJ_east

[1] 4d31'E

> IJ_north

[1] 52d28'N

> getSlots("DMS")

      WS      deg      min      sec      NS
"logical" "numeric" "numeric" "numeric" "logical"
```

The `DMS` class has slots to store representations of geographical coordinates, however, they might arise, but the `char2dms()` function expects the character input format to be as placed, permitting the degree, minute, and second symbols to be given as arguments. We get decimal degrees by coercing from class ‘`DMS`’ to class ‘`numeric`’ with the `as()` function:

```
> c(as(IJ_east, "numeric"), as(IJ_north, "numeric"))
[1] 4.516667 52.466667
```

4.2 Vector File Formats

Spatial vector data are points, lines, polygons, and fit the equivalent `sp` classes. There are a number of commonly used file formats, most of them proprietary, and some newer ones which are adequately documented. GIS are also more and more handing off data storage to database management systems, and some database systems now support spatial data formats. Vector formats can also be converted outside R to formats for which import is feasible.

GIS vector data can be either topological or simple. Legacy GIS were topological, desktop GIS were simple (sometimes known as spaghetti). The `sp` vector classes are simple, meaning that for each polygon all coordinates are stored without checking that boundaries have corresponding points. A topological representation in principal stores each point only once, and builds arcs (lines between nodes) from points, polygons from arcs – the GRASS open source GIS (GRASS Development Team, 2012) from version 6.0 and subsequent releases has such a topological representation of vector features. Only the **R**`ArcInfo` package tries to keep some traces of topology in importing legacy ESRI™ ArcInfo™ binary vector coverage data (or e00 format data) – **maps** uses topology because that was how things were done when the underlying code was written. The import of ArcGIS™ coverages is described fully in Gómez-Rubio and López-Quílez (2005); conversion of imported features into `sp` classes is handled by the `pal2SpatialPolygons` function in **maptools**.

It is often attractive to make use of the spatial databases in the **maps** package. They can be converted to `sp` class objects using functions such as `map2SpatialPolygons` in the **maptools** package. An alternative source of coastlines is the `Rgshhs` function in **maptools**, interfacing binary databases of varying resolution distributed by the ‘Global Self-consistent, Hierarchical, High-resolution Shoreline Database’ project.¹¹

The best resolution databases are rather large, and so **maptools** ships only with the coarse resolution one; users can install and use higher resolution databases locally. Figures 2.3 and 2.8, among others in earlier chapters, have been made using these sources.

¹¹ <http://www.soest.hawaii.edu/wessel/gshhs/>.

A format that is commonly used for exchanging vector data is the shapefile. This file format has been specified by ESRI™, the publisher of ArcView™ and ArcGIS™, which introduced it initially to support desktop mapping using ArcView™.¹² This format uses at least three files to represent the data, a file of geometries with an `*.shp` extension, an index file to the geometries `*.shx`, and a legacy `*.dbf` DBF III file for storing attribute data. Note that there is no standard mechanism for specifying missing attribute values in this format. If a `*.prj` file is present, it will contain an ESRI™ well-known text CRS specification. The shapefile format is not fully compatible with the OpenGIS® Simple Features Specification (see p. 131 for a discussion of this specification). Its incompatibility is, however, the same as that of the `SpatialPolygons` class, using a collection of polygons, both islands and holes, to represent a single observation in terms of attribute data.

4.2.1 Using OGR Drivers in rgdal

Using the OGR vector functions of the Geospatial Data Abstraction Library, interfaced in `rgdal`,¹³ lets us read spatial vector data for which drivers are available. A driver is a software component plugged-in on demand – here the OGR library tries to read the data using all the formats that it knows, using the appropriate driver if available. OGR also supports the handling of coordinate reference systems directly, so that if the imported data have a specification, it will be read.

The availability of OGR drivers differs from platform to platform, and can be listed using the `ogrDrivers` function. The function also lists whether the driver supports the creation of output files. Because the drivers often depend on external software, the choices available will depend on the local computer installation. It is frequently convenient to convert from one external file format to another using utility programs such as `ogr2ogr` in binary OSGeo4W Windows releases, which typically include a wide range of drivers.¹⁴ On the system used for building this book, the first 10 drivers listed by `ogrDrivers` are:

```
> head(ogrDrivers(), n = 10)

      name write
1  AeronavFAA FALSE
2    ARCGEN FALSE
3   AVCBin FALSE
4  AVCEOO FALSE
5     BNA  TRUE
```

¹² The format is fully described in this white paper:

<http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf>.

¹³ See installation note at chapter end.

¹⁴ <http://trac.osgeo.org/osgeo4w/>.


```

6      CouchDB  TRUE
7          CSV  TRUE
8          DGN  TRUE
9          DXF  TRUE
10     EDIGEO FALSE

```

Functions using OGR are based on the concepts of *data source name* and *layer*, both required to access data and metadata. In some circumstances, we need to find out which layers are offered by a data source, using the function `ogrListLayers` taking a data source name argument; an example is given on p. 97. The ways in which the data source name and layer arguments are specified and may differ forms for different drivers and it is worth reading the relevant web pages¹⁵ for the format being imported. In some cases, a data source name contains only one layer, but in other cases many layers may be present.

Recent releases of OGR have included facilities for handling the encoding of strings, both the values of string fields and of field names. These are diffusing to drivers for different file formats slowly, but have already caused difficulties for Windows users of CP1252 (a codepage specifying encoding) and the ESRI™ Shapefile driver. An analysis of this issue is provided in a vignette:

```
> vignette("OGR_shape_encoding", package = "rgdal")
```

which explains how to prevent the OGR driver trying to modify the encoding, thus making it possible to keep the data representation the same in R and ArcGIS™.

The `readOGR` and `ogrInfo` functions take at least two arguments – the data source name (`dsn`) and the layer (`layer`). For ESRI™ shapefiles, `dsn` is usually the name of the directory containing the three (or more) files to be imported (given as "." if the working directory), and `layer` is the name of the shapefile without the ".shp" extension. Additional examples are given on the function help page for file formats, but it is worth noting that the same functions can also be used where the data source name is a database connection, and the layer is a table, for example using PostGIS in a PostgreSQL database.

We can use the classic Scottish lip cancer data set by district downloaded from the additional materials page for Chap. 9 in Waller and Gotway (2004).¹⁶ There are three files making up the shapefile for Scottish district boundaries at the time the data were collected – the original study and extra data in a separate text file are taken from Clayton and Kaldor (1987):

```

> scot_dat <- read.table("scotland.dat", skip = 1)
> names(scot_dat) <- c("District", "Observed", "Expected",
+   "PcAFF", "Latitude", "Longitude")

```

¹⁵ http://www.gdal.org/ogr/ogr_formats.html.

¹⁶ <http://www.sph.emory.edu/~lwaller/WGindex.htm>.

We can use the `ogrInfo` function to show a summary of the layer before we read it. The function returns an object containing information, shown with a print method, and reporting the data source name, the layer name, the driver, number of features, and featype type. If a coordinate reference system is associated with the layer, it is shown, but here, none is available. The shapefile appears to be in geographical coordinates, as we see from its extent, showing the coordinates of the bounding box of the vector features. The LDID value for ESRI™ Shapefiles is explaining in the OGR encoding vignette mentioned on p. 93.

```
> ogrInfo(".", "scot")

Source: ".", layer: "scot"
Driver: ESRI Shapefile number of rows 56
Feature type: wkbPolygon with 2 dimensions
Extent: (-8.621389 54.62722) - (-0.7530556 60.84444)
LDID: 0
Number of fields: 2
  name type length typeName
1 NAME   4      16   String
2  ID    2      16    Real
```

We also see that `ogrInfo` retrieves information on the fields of attribute data provided in the layer. The name of the field will be used in the "data" slot of the imported object, and its type will be "integer" or "numeric" for matching numeric input data, although many drivers return "numeric" where "integer" would be more appropriate. In `readOGR`, input strings (including all date and time fields read as strings) are converted to factors if the `stringsAsFactors` argument is TRUE; the argument defaults to the value returned by `default.stringsAsFactors`.

No *.prj file is present, so, after importing from the working directory with a missing CRS value, we assign a suitable coordinate reference system.

```
> scot_LL <- readOGR(dsn = ".", layer = "scot")

OGR data source with driver: ESRI Shapefile
Source: ".", layer: "scot"
with 56 features and 2 fields
Feature type: wkbPolygon with 2 dimensions

> proj4string(scot_LL)

[1] NA

> proj4string(scot_LL) <- CRS("+proj=longlat +ellps=WGS84")
```

As indicated above, when we get the classes of variables in the "data" slot of `scot_LL`, we see that the input string field has been converted to **factor** representation, and the **numeric** ID field could just as well been an integer:

```
> sapply(slot(scot_LL, "data"), class)
```

```

      NAME      ID
"factor" "numeric"

> scot_LL$ID

[1] 12 13 19  2 17 16 21 50 15 25 26 29 43 39 40 52 42 51 34 54 36 46
[23] 41 53 49 38 44 30 45 48 47 35 28  4 20 33 31 24 55 18 56 14 32 27
[45] 10 22  6  8  9  3  5 11  1  7 23 37

```

The Clayton and Kaldor data are for the same districts, but with the rows ordered differently, so that before combining the data with the imported polygons, they need to be matched first:

```

> scot_dat$District

[1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22
[23] 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
[45] 45 46 47 48 49 50 51 52 53 54 55 56

> ID_D <- match(scot_LL$ID, scot_dat$District)
> scot_dat1 <- scot_dat[ID_D, ]
> row.names(scot_dat1) <- row.names(scot_LL)
> library(maptools)
> scot_LLa <- spCbind(scot_LL, scot_dat1)
> all.equal(scot_LLa$ID, scot_LLa$District)

[1] TRUE

> names(scot_LLa)

[1] "NAME"      "ID"        "District"  "Observed"  "Expected"
[6] "PcAFF"     "Latitude"  "Longitude"

```

Figure 4.1 compares the relative risk by district with the Empirical Bayes smooth values – we return to the actual techniques involved in Chap. 10, here the variables are being added to indicate how results may be exported from R below. The relative risk does not take into account the possible uncertainty associated with unusual incidence rates in counties with relatively small populations at risk, while Empirical Bayes smoothing shrinks such values towards the rate for all the counties taken together.

```

> library(spdep)
> O <- scot_LLa$Observed
> E <- scot_LLa$Expected
> scot_LLa$SMR <- probmap(O, E)$relRisk/100
> library(DCluster)
> scot_LLa$smth <- empbaysmooth(O, E)$smthrr

```

Finally, we project the district boundaries to the British National Grid as described by Waller and Gotway (2004):

```

> scot_BNG <- spTransform(scot_LLa, CRS("+init=epsg:27700"))

```

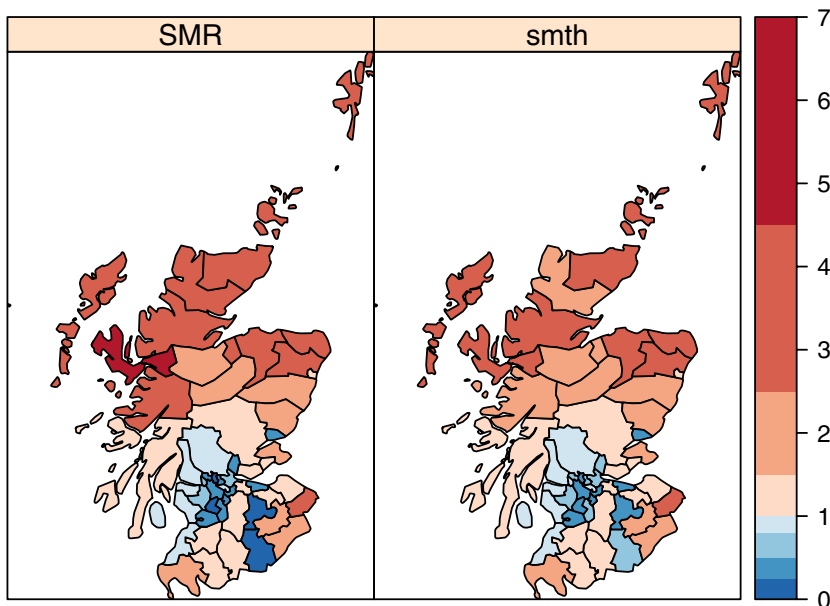


Fig. 4.1 Comparison of relative risk (SMR) and EB smoothed relative risk (smth) for Scottish lip cancer

We may export `SpatialPointsDataFrame`, `SpatialLinesDataFrame` and `SpatialPolygonsDataFrame` objects using the `writeOGR` function in `rgdal` to file formats for which output drivers are implemented. As an example, we can export the projected Scottish data set, including our added results, to a shapefile, using `driver="ESRI Shapefile"`, or to other file formats:

```
> drv <- "ESRI Shapefile"
> writeOGR(scot_BNG, dsn = ".", layer = "scot_BNG", driver = drv)

> list.files(pattern = "^scot_BNG")

[1] "scot_BNG.dbf" "scot_BNG.prj" "scot_BNG.shp" "scot_BNG.shx"
```

The output now contains a `*.prj` file with the fully specified coordinate reference system for the British National Grid, to which we projected the data object. As mentioned above, the `ogrDrivers` function can be used to see which drivers are available. Some, like `driver="ESRI Shapefile"`, can represent points, lines or polygons, and are intended for general vector data exchange. The OpenGIS® specifications include several XML-based formats, initially the Geography Markup Language (GML), but more recently followed by the Keyhole Markup Language (KML) brought forward by Google™. The GML format is also used within the OpenGIS® Web Feature Service (WFS), in which the server is the data source name, and the layer is a table of geographical data. Let us use the OGR WFS driver to access European Forest

Fire Information System data from the Joint Research Centre of the European Commission, adding a driver-specific prefix to the URL of the service; we can use `ogrListLayers` to check for available layers:

```
> dsn <- "WFS:http://geohub.jrc.ec.europa.eu/effis/ows"
> ogrListLayers(dsn)

[1] "EFFIS:FireNews"      "EFFIS:Fires30Days"  "EFFIS:Fires7Days"
[4] "EFFIS:FiresAll"     "EFFIS:Hotspots1Day" "EFFIS:Hotspots7Days"
[7] "EFFIS:HotspotsAll"

> Fires <- readOGR(dsn, "EFFIS:FiresAll")

OGR data source with driver: WFS
Source: "WFS:http://geohub.jrc.ec.europa.eu/effis/ows", layer:
"EFFIS:FiresAll"
with 1770 features and 11 fields
Feature type: wkbPoint with 2 dimensions

> names(Fires)

[1] "gml_id"      "FireDate"    "Country"     "Province"    "Commune"
[6] "Area_HA"    "CountryFul"  "Class"       "X"           "Y"
[11] "LastUpdate"
```

The data downloaded are those in the current database,¹⁷ and will change as new incidents accrue. First we create a bounding box covering the relevant parts of Europe, and use it to make a spatial selection of only the coastlines and national boundaries taken from the `wrld_simpl` data set included in **maptools**, falling within the box with `gIntersection`, a binary topological operator in the **rgeos** package. Subsetting the coastlines reduces plotting times, because the plotting method does not need to discard data far outside its data window.

```
> x <- c(-15, -15, 38, 38, -15)
> y <- c(28, 62, 62, 28, 28)
> crds <- cbind(x = x, y = y)
> bb <- SpatialPolygons(list(Polygons(list(Polygon(coords = crds)),
+   "1")))
> library(maptools)
> data(wrld_simpl)
> proj4string(bb) <- CRS(proj4string(wrld_simpl))
> library(rgeos)
> slbb <- gIntersection(bb, as(wrld_simpl, "SpatialLines"))
> spl <- list("sp.lines", slbb, lwd = 0.7, col = "khaki4")
```

Next we convert the input fire date to a `Date` object, then discard any incidents on Réunion. Using the new space-time classes introduced in Chap. 6 and specified in the **spacetime** package, we can plot three incident maps using the `stplot` method, conditioned by time quantiles, as shown in Fig. 4.2:

¹⁷ Those used here were accessed on 2012-01-04.

```

> Fires$dt <- as.Date(as.character(Fires$FireDate), format = "%d-%m-%Y")
> Fires0 <- Fires[-which(coordinates(Fires)[, 2] < 0),
+ ]
> Fires1 <- Fires0[order(Fires0$dt), ]
> library(spacetime)
> Fires2 <- STIDF(as(Fires1, "SpatialPoints"), Fires1$dt,
+ as(Fires1, "data.frame"))
> stplot(Fires2, number = 3, sp.layout = spl, cex = 0.5)

```

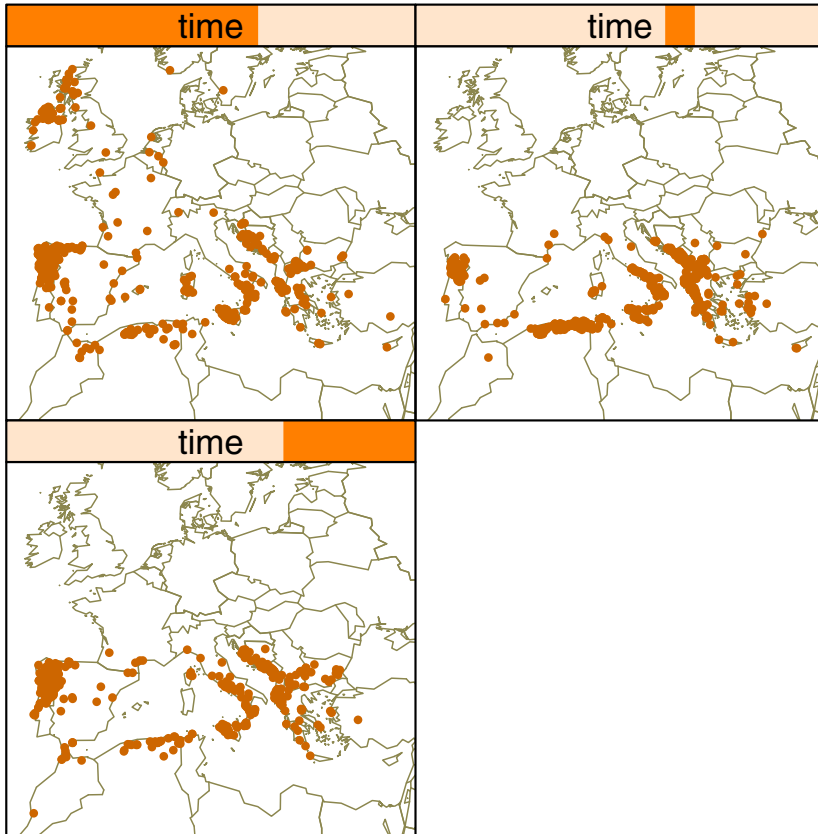


Fig. 4.2 Space-time plot of forest fire incidents, conditioned on time by quantiles: *upper left panel* – first third of incidents, *upper right panel* – second third, *lower left panel* – final third

The OGR GPX driver may be used to exchange data with GPS devices, using a simple XML representation. We may here prepare a file for uploading the fire locations recorded for Greece, for example to permit sites to be surveyed at fixed post-fire intervals; we must re-name the ID column to the GPX mandatory "name":

```
> names(Fires1)[1] <- "name"
> GR_Fires <- Fires1[Fires1$Country == "GR", ]
> writeOGR(GR_Fires, "EFFIS.gpx", "waypoints", driver = "GPX",
+         dataset_options = "GPX_USE_EXTENSIONS=YES")
```

The use of the `dataset_options` argument permits the inclusion of identifying data in the GPX file, which may be accessed on suitable GPS devices. In this case, the retrieved values for the first incident may be shown as:

```
> GR <- readOGR("EFFIS.gpx", "waypoints")

OGR data source with driver: GPX
Source: "EFFIS.gpx", layer: "waypoints"
with 89 features and 34 fields
Feature type: wkbPoint with 2 dimensions

> GR[1, c(5, 24:28)]
      coordinates      name ogr_FireDate ogr_Country
1 (22.261, 39.4258) FiresAll.1552 26-06-2011          GR
  ogr_Province   ogr_Commune ogr_Area_HA
1      Larisa Dimos Krannonos      376
```

The Keyhole Markup Language (KML) is a further XML-based OGR driver, to which we return in Sect. 4.4 below.

4.2.2 Other Import/Export Functions

If the **rgdal** package is not available, there are two other packages that can be used for reading and writing shapefiles. The **shapefiles** package is written without external libraries, using file connections. It can be very useful when a shapefile is malformed, because it gives access to the raw numbers. The **maptools** package contains a local copy of the library used in OGR for reading shapefiles (the DBF reader is in the **foreign** package), and provides a helper function `getinfo.shape` to identify whether the shapefile contains points, lines, or polygons.

```
> getinfo.shape("scot_BNG.shp")

Shapefile type: Polygon, (5), # of Shapes: 56
```

There is a function to read vector data from shapefiles: `readShapeSpatial`. It is matched by an equivalent exporting function: `writeSpatialShape`, using local copies of shapelib functions otherwise available in **rgdal** in the OGR framework. The **RArcInfo** package also provides local access to OGR functionality, for reading ArcGIS™ binary vector coverages, but with the addition of a utility function for converting e00 format files into binary coverages; full details are given in Gómez-Rubio and López-Quílez (2005).

4.3 Raster File Formats

There are very many raster and image formats; some allow only one band of data, others assume that data bands are Red-Green-Blue (RGB), while yet others are flexible and self-documenting. The simplest formats are just rectangular blocks of uncompressed data, like a matrix, but sometimes with row indexing reversed. Others are compressed, with multiple bands, and may be interleaved so that subscenes can be retrieved without unpacking the whole image. There are now a number of R packages that support image import and export, such as the **ReadImages** and **biOps** packages and the **EBImage** package in the Bioconductor project. The requirements for spatial raster data handling include respecting the coordinate reference system of the image, so that specific solutions are needed. There is, however, no direct support for the transformation or ‘warping’ of raster data from one coordinate reference system to another.

4.3.1 Using GDAL Drivers in rgdal

Many drivers are available in **rgdal** in the **readGDAL** function, which – like **readOGR** – finds a usable driver if available and proceeds from there. Using arguments to **readGDAL**, subregions or bands may be selected, and the data may be decimated, which helps handle large rasters. The simplest approach is just to read all the data into the R workspace – here we will use the same excerpt from the Shuttle Radar Topography Mission (SRTM) flown in 2000, for the Auckland area as in Chap. 2.

```
> auck_e11 <- readGDAL("70042108.tif")
70042108.tif has GDAL driver GTiff
and has 1200 rows and 1320 columns
> summary(auck_e11)
Object of class SpatialGridDataFrame
Coordinates:
      min      max
x 174.2 175.3
y -37.5 -36.5
Is projected: FALSE
proj4string :
[+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84
+towgs84=0,0,0]
Grid attributes:
      cellcentre.offset      cellsize cells.dim
x      174.20042 0.0008333333      1320
y      -37.49958 0.0008333333      1200
Data attributes:
      Min.      1st Qu.      Median      Mean      3rd Qu.      Max.
-3.403e+38  0.000e+00  1.000e+00 -1.869e+34  5.300e+01  6.860e+02
```



```
> is.na(auck_el1$band1) <- auck_el1$band1 <= 0 | auck_el1$band1 >
+ 10000
```

The `readGDAL` function is actually a wrapper for substantially more powerful R bindings for GDAL written by Timothy Keitt. The bindings allow us to handle very large data sets by choosing sub-scenes and re-sampling, using the `offset`, `region.dim`, and `output.dim` arguments. The bindings work by opening a data set known by GDAL using a `GDALDriver` class object, but only reading the required parts into the workspace.

```
> x <- GDAL.open("70042108.tif")
> xx <- getDriver(x)
> xx
```

```
An object of class "GDALDriver"
Slot "handle":
<pointer: 0x3145730>
```

```
> getDriverLongName(xx)
```

```
[1] "GeoTIFF"
```

```
> x
```

```
An object of class "GDALReadOnlyDataset"
Slot "handle":
<pointer: 0x85ad850>
```

```
> dim(x)
```

```
[1] 1200 1320
```

```
> GDAL.close(x)
```

Here, `x` is a derivative of a `GDALDataset` object, and is the GDAL data set handle; the data are not in the R workspace, but all their features are there to be read on demand. An open GDAL handle can be read into a `SpatialGridDataFrame`, so that `readGDAL` may be done in pieces if needed. Information about the file to be accessed may also be shown without the file being read, using the GDAL bindings packaged in the utility function `GDALInfo`:

```
> GDALInfo("70042108.tif")
```

```
rows      1200
columns   1320
bands      1
lower left origin.x      174.2
lower left origin.y     -37.5
res.x      0.0008333333
res.y      0.0008333333
ysign      -1
oblique.x   0
oblique.y   0
driver      GTiff
projection  +proj=longlat +datum=WGS84 +no_defs
```

```

file          70042108.tif
apparent band summary:
  GDType hasNoDataValue NoDataValue blockSize1 blockSize2
1 Float32      FALSE           0           1       1320
apparent band statistics:
      Bmin      Bmax Bmean Bstd
1 -4294967295 4294967295    NA    NA
Metadata:
AREA_OR_POINT=Area
TIFFTAG_RESOLUTIONUNIT=1 (unitless)
TIFFTAG_SOFTWARE=IMAGINE TIFF Support
Copyright 1991 - 1999 by ERDAS, Inc. All Rights Reserved
@(#)RCSfile: etif.c $ $Revision: 1.11 $ $Date$
TIFFTAG_XRESOLUTION=1
TIFFTAG_YRESOLUTION=1

```

While `Spatial` objects do not contain a record of their symbolic representation (see p. 29), it is possible to quantise numerical bands and associate them with colour tables when exporting raster data using some drivers. We can see here that the same colour table is retrieved from a `GTiff` file; the ways in which colour tables are handled varies considerably from driver to driver. The colour table is placed in a `list` because they are associated with raster bands, possibly one for each band in a multi-band raster, so the `colorTable=` argument to `writeGDAL` must be `NULL` or a list of length equal to the number of bands. Note that the integer raster values pointing to the colour table are zero-based, that is, the index for looking up values in the colour table starts at zero for the first colour, and so on. Care is needed to move the missing value beyond values pointing to the colour table; note that the colours have lost their alpha-channel settings. More examples are given in the `writeGDAL` help page.

```

> brks <- c(0, 10, 20, 50, 100, 150, 200, 300, 400, 500,
+          600, 700)
> pal <- terrain.colors(11)
> pal

[1] "#00A600FF" "#2DB600FF" "#63C600FF" "#A0D600FF" "#E6E600FF"
[6] "#E8C727FF" "#EAB64EFF" "#ECB176FF" "#EEB99FFF" "#F0CFC8FF"
[11] "#F2F2F2FF"

> length(pal) == length(brks) - 1

[1] TRUE

> auck_el1$band1 <- findInterval(auck_el1$band1, vec = brks,
+   all.inside = TRUE) - 1
> writeGDAL(auck_el1, "demIndex.tif", drivername = "GTiff",
+   type = "Byte", colorTable = list(pal), mvFlag = length(brks) -
+   1)
> Gi <- GDALInfo("demIndex.tif", returnColorTable = TRUE)
> CT <- attr(Gi, "ColorTable")[[1]]
> CT[CT > "#000000"]

```

```
[1] "#00A600" "#2DB600" "#63C600" "#A0D600" "#E6E600" "#E8C727"
[7] "#EAB64E" "#ECB176" "#EEB99F" "#F0CFC8" "#F2F2F2"
```

We use the Meuse grid data set to see how data may be written out using GDAL.¹⁸ The `writeGDAL` function can be used directly for drivers that support file creation. For other file formats, which can be made as copies of a prototype, we need to create an intermediate GDAL data set using `create2GDAL`, and then use functions operating on the GDAL data set handle to complete. First we simply output inverse distance weighted interpolated values of Meuse Bank logarithms of zinc ppm as a GeoTiff file.

```
> library(gstat)
> log_zinc <- idw(log(zinc) ~ 1, meuse, meuse.grid)["var1.pred"]

> summary(log_zinc)

Object of class SpatialPixelsDataFrame
Coordinates:
      min      max
x 178440 181560
y 329600 333760
Is projected: TRUE
proj4string :
[+init=epsg:28992 +proj=sterea +lat_0=52.15616055555555
+lon_0=5.387638888888889 +k=0.9999079 +x_0=155000 +y_0=463000
+ellps=bessel
+towgs84=565.417,50.3319,465.552,-0.398957,0.343988,-1.8774,4.0725
+units=m +no_defs]
Number of points: 3103
Grid attributes:
  cellcentre.offset cellsize cells.dim
x              178460         40         78
y              329620         40        104
Data attributes:
      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
4.791   5.484   5.694   5.777   6.041   7.482

> writeGDAL(log_zinc, fname = "log_zinc.tif", drivervname = "GTiff",
+   type = "Float32", options = "INTERLEAVE=PIXEL")

> GDALInfo("log_zinc.tif")

rows           104
columns         78
bands           1
lower left origin.x      178440
lower left origin.y      329600
res.x              40
res.y              40
ysign             -1
oblique.x          0
```

¹⁸ The current EPSG list provides `+towgs84` parameter values, which were not present in earlier versions of that list.

```

oblique.y    0
driver       GTiff
projection   +proj=sterea +lat_0=52.15616055555555
+lon_0=5.38763888888889 +k=0.9999079 +x_0=155000
+y_0=463000 +ellps=bessel
+towgs84=565.417,50.3319,465.552,-0.398957,0.343988,-1.8774,4.0725
+units=m +no_defs
file         log_zinc.tif
apparent band summary:
      GDType hasNoDataValue NoDataValue blockSize1
1 Float32      FALSE           0           26
  blockSize2
1          78
apparent band statistics:
      Bmin      Bmax Bmean Bsd
1 -4294967295 4294967295   NA  NA
Metadata:
AREA_OR_POINT=Area

```

The output file can for example be read into ENVI™ directly, or into ArcGIS™ possibly via the ‘Calculate statistics’ tool in the Raster section of the Toolbox, and displayed by adjusting the symbology classification.

In much the same way that `writeGDAL` can write per-band colour tables to exported files for some drivers (see p. 102), the same can be done with category names. The same remarks with respect to zero-based indexing also apply, so that zero in the raster points to the first category name. On import using `readGDAL` with the default value of the `as.is=` argument of `FALSE`, the integer band will be associated with the category names in the file and converted to "factor":

```

> Soil <- meuse.grid["soil"]
> table(Soil$soil)

 1    2    3
1665 1084 354

> Soil$soil <- as.integer(Soil$soil) - 1
> Cn <- c("Rd10A", "Rd90C/VII", "Bkd26/VII")
> writeGDAL(Soil, "Soil.tif", drivername = "GTiff", type = "Byte",
+   catNames = list(Cn), mvFlag = length(Cn))
> Gi <- GDALInfo("Soil.tif", returnCategoryNames = TRUE)
> attr(Gi, "CATlist")[[1]]

[1] "Rd10A"      "Rd90C/VII" "Bkd26/VII"

> summary(readGDAL("Soil.tif"))

Soil.tif has GDAL driver GTiff
and has 104 rows and 78 columns
Input level values and names
0 Rd10A
1 Rd90C/VII
2 Bkd26/VII
Object of class SpatialGridDataFrame

```

```

Coordinates:
      min      max
x 178440 181560
y 329600 333760
Is projected: TRUE
proj4string :
[+proj=sterea +lat_0=52.15616055555555 +lon_0=5.38763888888889
+k=0.9999079 +x_0=155000 +y_0=463000 +ellps=bessel
+towgs84=565.417,50.3319,465.552,-0.398957,0.343988,-1.8774,4.0725
+units=m +no_defs]
Grid attributes:
      cellcentre.offset cellsize cells.dim
x              178460         40         78
y              329620         40        104
Data attributes:
      Rd10A Rd90C/VII Bkd26/VII      NA's
      1665      1084      354      5009

```

The range of drivers available for raster data is vast, and steadily increasing. The `gdalDrivers` function shows those available, including chosen properties; here are the first 10 on the book production platform:

```

> head(gdalDrivers(), n = 10)

```

	name	long_name	create	copy
1	AAIGrid	Arc/Info ASCII Grid	FALSE	TRUE
2	ACE2	ACE2	FALSE	FALSE
3	ADRG	ARC Digitized Raster Graphics	TRUE	FALSE
4	AIG	Arc/Info Binary Grid	FALSE	FALSE
5	AirSAR	AirSAR Polarimetric Image	FALSE	FALSE
6	BAG	Bathymetry Attributed Grid	FALSE	FALSE
7	BIGGIF	Graphics Interchange Format (.gif)	FALSE	FALSE
8	BLX	Magellan topo (.blx)	FALSE	TRUE
9	BMP	MS Windows Device Independent Bitmap	TRUE	FALSE
10	BSB	Maptech BSB Nautical Charts	FALSE	FALSE

For example, there is now an R driver for storing portable `SpatialGridDataFrame` objects:

```

> writeGDAL(log_zinc, fname = "log_zinc.rda", drivername = "R")
> GDALinfo("log_zinc.rda")

```

rows	104
columns	78
bands	1
lower left origin.x	178440
lower left origin.y	329600
res.x	40
res.y	40
ysign	-1
oblique.x	0
oblique.y	0
driver	R
projection	+proj=sterea +lat_0=52.15616055555555

```
+lon_0=5.38763888888889 +k=0.9999079 +x_0=155000
+y_0=463000 +ellps=bessel
+towgs84=565.417,50.3319,465.552,-0.398957,0.343988,-1.8774,4.0725
+units=m +no_defs
file          log_zinc.rda
apparent band summary:
      GDType hasNoDataValue NoDataValue blockSize1
1 Float64          FALSE           0           1
  blockSize2
1          78
apparent band statistics:
      Bmin      Bmax Bmean Bsd
1 -4294967295 4294967295   NA   NA
Metadata:
R_OBJECT_NAME=gg
```

As in the vector case, GDAL now supports a range of web services. The OpenGIS® Web Map Service (WMS) driver can be used with a local XML file describing the service, and customised offset, region size (both in raster cells, ordered: northings, eastings), and output size. This example reads a raster version of OpenStreetMap¹⁹ data for the centre of Bergen, Norway:

```
> service_xml <- "frmt_wms_openstreetmap_tms.xml"
> offset <- c(19339000, 34546000)
> osm <- readGDAL(service_xml, offset = offset, region.dim = c(2000,
+ 2000), output.dim = c(1000, 1000))
```

```
frmt_wms_openstreetmap_tms.xml has GDAL driver WMS
and has 67108864 rows and 67108864 columns
```

```
> summary(osm)
```

```
Object of class SpatialGridDataFrame
```

```
Coordinates:
```

```
      min      max
```

```
x 592129 593323.3
```

```
y 8487754 8488948.3
```

```
Is projected: TRUE
```

```
proj4string :
```

```
[+proj=merc +lon_0=0 +k=1 +x_0=0 +y_0=0 +a=6378137 +b=6378137
```

```
+units=m +no_defs]
```

```
Grid attributes:
```

```
      cellcentre.offset cellsize cells.dim
```

```
x          592129.6 1.194329      1000
```

```
y          8487754.5 1.194329      1000
```

```
Data attributes:
```

	band1	band2	band3
Min.	: 0.0	Min. : 0.0	Min. : 0.0
1st Qu.:	202.0	1st Qu.:208.0	1st Qu.:208.0
Median :	240.0	Median :232.0	Median :227.0
Mean :	223.5	Mean :219.3	Mean :215.1
3rd Qu.:	241.0	3rd Qu.:238.0	3rd Qu.:232.0
Max. :	255.0	Max. :255.0	Max. :255.0

¹⁹ <http://www.openstreetmap.org/>, this selection corresponds to <http://www.openstreetmap.org/?lat=60.39542&lon=5.32233&zoom=16&layers=C>.

Figure 4.3 shows the retrieved data, which has been extracted from the tiled OpenStreetMap global raster database.



Fig. 4.3 Use of the WMS GDAL driver to retrieve OpenStreetMap raster data for the centre of Bergen, Norway (©OpenStreetMap contributors, CC-BY-SA)

4.3.2 Other Import/Export Functions

There is a simple `readAsciiGrid` function in **maptools** that reads ESRI™ Arc ASCII grids into `SpatialGridDataFrame` objects; it does not handle CRS and has a single band. The companion `writeAsciiGrid` is for writing Arc ASCII grids. It is also possible to use connections to read and write arbitrary binary files, provided that the content is not compressed. Functions in the R image analysis packages referred to above may also be used to read and write a

number of image formats. If the grid registration slots in objects of classes defined in the **pixmap** package are entered manually, these objects may also be used to hold raster data.

4.4 Google Earth™, Google Maps™ and Other Formats

As we have seen above, web-based services are becoming ever more important channels for exchanging spatial data. We will examine two directions for transfer, first the import of background maps into R, and next the export of data for display on web-based mapping platforms.

The **RgoogleMaps** package provides tools to access Google Maps™ data in image form using the Google Static Maps API, in order to permit background maps to be used in R. The HTTP request issued specifies the image required, which is then composed and downloaded for display in R graphics devices. The object uses screen coordinates internally, but may be reshaped as a **SpatialGridDataFrame**, permitting standard **sp** methods to be used for overplotting. The package has been extended to issue HTTP requests to OpenStreetMap, but as yet without geographical registration, so that we can retrieve images for the street layout of the centre of Bergen, Norway in this way, in addition to using WMS from OpenStreetMap; the results are shown in Fig. 4.4:

```
> library(RgoogleMaps)
> myMap <- GetMap(center = c(60.395, 5.322), zoom = 16,
+   destfile = "MyTile2.png", matype = "mobile")

> BB <- do.call("rbind", myMap$BBOX)
> dBB <- rev(diff(BB))
> DIM12 <- dim(myMap$myTile)[1:2]
> cs <- dBB/DIM12
> cc <- c(BB[1, 2] + cs[1]/2, BB[1, 1] + cs[2]/2)
> GT <- GridTopology(cc, cs, DIM12)
> p4s <- CRS("+proj=longlat +datum=WGS84")
> SG_myMap <- SpatialGridDataFrame(GT, proj4string = p4s,
+   data = data.frame(r = c(t(myMap$myTile[, , 1])) *
+     255, g = c(t(myMap$myTile[, , 2])) * 255, b = c(t(myMap$myTile[,
+     , 3])) * 255))

> myMap1 <- GetMap.OSM(lonR = c(5.319, 5.328), latR = c(60.392,
+   60.398), scale = 4000, destfile = "MyTile.png")
```

Vector data from OpenStreetMap is also available for download from the recently contributed package **osmar**; the package is under active development. If we retrieve a similar area to that sourced from Google Maps™, we should be able to overlay the vector data after conversion to an **sp** class; we can also see who had made most contributions of lines to OSM at the time this snapshot was downloaded:


```

> library(osmar)
> api <- osmsource_api()
> box <- corner_bbox(5.319, 60.392, 5.328, 60.398)
> torget <- get_osm(box, source = api)
> torget1 <- as_sp(torget, "lines")
> sort(table(torget1$user), decreasing = TRUE)[1:3]

```

Karl Ove Hufthammer
47

M E Menk
39

Bård Aase
29

The package also provides methods for selection of particular kinds of objects, so that we can find and select the city terminus of the light rail route for overplotting in this way (using data downloaded as this chapter was being revised, but not necessarily valid before or after, for example, the tag "light_rail" appears to have been changed to "railway" at some stage, and an inoperative museum tramway added):

```

> bybane <- find(torget, way(tags(k == "light_rail")))
> bybane <- find_down(torget, way(bybane))
> bybane <- subset(torget, ids = bybane)
> bybane <- as_sp(bybane, "lines")

```

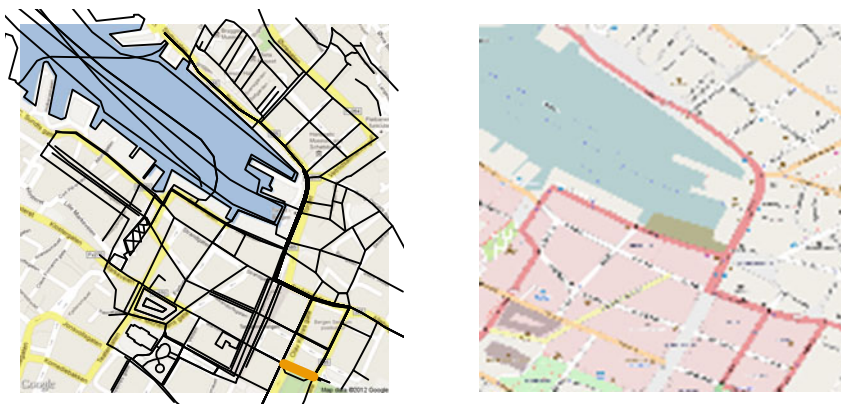


Fig. 4.4 Background maps imported with functions in **RgoogleMaps** from Google Maps™, overplotted with line data imported with functions in **osmar** with the light rail terminus shown as an *orange line*; and OpenStreetMap for the centre of Bergen, Norway (©Google™ and ©OpenStreetMap contributors, CC-BY-SA)

Data may be exported for display with Google Earth™ and other systems in the Keyhole Markup Language (KML) format. Vector data can be exported directly in a number of ways, given that it is in geographical coordinates and in the WGS84 datum. Point data may use the KML driver in OGR through **writeOGR**, with the values of attributes shown in bubbles when displayed points are clicked:



Fig. 4.5 Forest fires in Europe from JRC database shown in Google Earth™

```
> writeOGR(Fires[, c("gml_id", "FireDate", "Area_HA")],
+   dsn = "fires.kml", layer = "fires", driver = "KML")
```

Figure 4.5 shows the KML file displayed in Google Earth™, and the reader may try out the attribute query mechanism, as well as zooming in to regions of interest in a running Google Earth application. Lines and Polygons may also be exported with the KML driver and `writeOGR`, but there is only limited support for styles and attribute data. There are also three functions in **maptools**, `kmlPoints`, `kmlLine` and `kmlPolygon`, that permit more control over style but at the cost of more work setting arguments. The **plotKML** package is now available on CRAN, providing unified methods for setting styles and other display qualities, and handling space-time data.

Our next attempt, to export a raster, will be more ambitious; in fact we can use this technique to export anything that can be plotted on a PNG graphics device. We export a coloured raster of interpolated log zinc ppm values to a PNG file with an alpha channel for viewing in Google Earth™. Since the target software requires geographical coordinates, a number of steps will be needed. First we make a polygon to bound the study area and project it to geographical coordinates:

```
> library(maptools)
> grd <- as(meuse.grid, "SpatialPolygons")
> proj4string(grd) <- CRS(proj4string(meuse))
> grd.union <- unionSpatialPolygons(grd, rep("x", length(slot(grd,
+   "polygons"))))
> ll <- CRS("+proj=longlat +datum=WGS84")
> grd.union.ll <- spTransform(grd.union, ll)
```



Fig. 4.6 Interpolated log zinc ppm for the Meuse Bank data set shown in Google Earth™

Next we construct a suitable grid in geographical coordinates, as our target object for export, using the `GE_SpatialGrid` wrapper function. This grid is also the container for the output PNG graphics file, so `GE_SpatialGrid` also returns auxiliary values that will be used in setting up the `png` graphics device within R. We use the `over` method to set grid cells outside the river bank area to NA, and then discard them by coercion to a `SpatialPixelsDataFrame`:

```
> llGRD <- GE_SpatialGrid(grd.union.ll)
> llGRD_in <- over(llGRD$SG, grd.union.ll)
> llSGDF <- SpatialGridDataFrame(grid = slot(llGRD$SG,
+   "grid"), proj4string = CRS(proj4string(llGRD$SG)),
+   data = data.frame(in0 = llGRD_in))
> llSPix <- as(llSGDF, "SpatialPixelsDataFrame")
```

We use `idw` from the `gstat` package to make an inverse distance weighted interpolation of zinc ppm values from the soil samples available, also, as here, when the points are in geographical coordinates; interpolation will be fully presented in Chap. 8 (the need for the `::` notation is explained on Sect. 8.3.1):

```
> meuse_ll <- spTransform(meuse, CRS("+proj=longlat +datum=WGS84"))
> llSPix$pred <- gstat::idw(log(zinc) ~ 1, meuse_ll, llSPix)$var1.pred
```

Since we have used `GE_SpatialGrid` to set up the size of an `Rpng` graphics device, we can now use it as usual, here with `image`. In practice, any base graphics methods and functions can be used to create an image overlay. Finally, after closing the graphics device, we use `kmlOverlay` to write a `*.kml` file giving the location of the overlay and which will load the image at that position when opened in Google Earth™, as shown in Fig. 4.6:

```
> png(file = "zinc_IDW.png", width = llGRD$width, height = llGRD$height,
+      bg = "transparent")
> par(mar = c(0, 0, 0, 0), xaxs = "i", yaxs = "i")
> image(llSPix, "pred", col = bpy.colors(20))
> dev.off()
> kmlOverlay(llGRD, "zinc_IDW.kml", "zinc_IDW.png")
```

4.5 Geographical Resources Analysis Support System (GRASS)

GRASS²⁰ is a major open source GIS, originally developed as the Geographic Resources Analysis Support System by the U.S. Army Construction Engineering Research Laboratories (CERL, 1982–1995), and subsequently taken over by its user community. GRASS has traditional strengths in raster data handling, but two advances (floating point rasters and support for missing values) were not completed when development by CERL was stopped. These were added for many modules in the GRASS 5.0 release; from GRASS 6.0, new vector support has been added. GRASS is a very large but very simple system – it is run as a collection of separate programs built using shared libraries of core functions. There is then no GRASS ‘program’, just a script setting environment variables needed by the component programs. GRASS does interact with the OSGeo stack of applications; for further reviews, see Neteler et al. (2008, 2012) and Jolma et al. (2012).

An R package to interface with GRASS has been available on CRAN – **GRASS** – since the release of GRASS 5.0. It provided a compiled interface to raster and sites data, but not vector data, and included a frozen copy of the core GRASS GIS C library, modified to suit the fact that its functions were being used in an interactive, longer-running program like R. The **GRASS** package is no longer being developed, but continues to work for users of GRASS 5. The GRASS 5 interface is documented in Neteler and Mitasova (2004, pp. 333–354) and Bivand (2000).

The current GRASS releases, from GRASS 6.0, with GRASS 6.4.2 released in early 2012, have a different interface, using the **sp** classes presented in Chap. 2. Neteler and Mitasova (2008) describe GRASS 6 fully, and present this interface on pp. 353–364. The **spgrass6** package depends on **rgdal** for moving data, because GRASS also uses GDAL and OGR as its main import/export mechanisms. The interface works by exchanging temporary files in formats that both GRASS and **rgdal** know; a custom binary interface using **r.in.bin** and **r.out.bin** is also available, and may be faster than using a GDAL file format. This kind of loose coupling is less of a burden than it was before, with smaller, slower machines. This is why the GRASS 5 interface was tight-coupled, with R functions reading from and writing to the GRASS

²⁰ <http://grass.osgeo.org/>.

database directly. Using GRASS plug-in drivers in GDAL/OGR is another possibility for reading GRASS data directly into R through **rgdal**, without needing **spgrass6**; **spgrass6** can use these plug-in drivers if present for reading GRASS data.

GRASS uses the concept of a working region or window, specifying both the viewing rectangle and – for raster data – the resolution. The data in the GRASS database can be from a larger or smaller region and can have a different resolution, and are re-sampled to match the working region for analysis. This current window should determine the way in which raster data are retrieved and transferred.

GRASS also uses the concepts of a location, with a fixed and uniform coordinate reference system, and of mapsets within the location. The location is typically chosen at the start of a work session, and with the location, the user will have read access to possibly several mapsets, and write access to some, probably fewer, to avoid overwriting the work of other users of the location.

Intermediate temporary files are the chosen solution for interaction between GRASS and R in **spgrass6**, and drivers may be chosen by the user. Note that missing values are defined and supported for GRASS raster data, but that missing values for vector data are not uniformly defined or supported. It does not yet seem to be possible to use mechanisms in **rgdal** to interface colour tables or category names for rasters. Native Windows GRASS is now firmly established for GRASS 6.4.*, and the interface functions well on that platform; and **spgrass6** binaries for Windows and Mac OSX are on CRAN.

Each GRASS program takes a `--interface-description` flag, which when run returns an XML description of its flags and parameters. These descriptions are used by the GRASS GUI to populate its menus, and are also used in **spgrass6** to check that GRASS programs are used correctly. This also means that the **parseGRASS** function can set up an object in a searchable list on the R side of the interface, to avoid re-parsing interface descriptions that have already been encountered in a session. The middle function is **doGRASS**, which takes the flags and parameters chosen, checks their validity, and constructs a command string. Finally, **execGRASS** uses the **system** function to execute the GRASS program with the chosen flag and parameter values.

The package may be used in two ways, either in an R session started from within a GRASS session from the command line, or with the **initGRASS** function. The function may be used with an existing GRASS location and mapset, or with a one-time throw-away location, and takes the GRASS installation directory as its first argument; an example is given on p.136. It then starts a GRASS session within the R session, and is convenient for scripting GRASS in R, rather than Python, which is the GRASS scripting language in development version GRASS 7.

R is started here from within a GRASS session from the command line, and the **spgrass6** loaded with its dependencies:


```
> library(spgrass6)
> execGRASS("g.region", flags = "p")

projection: 1 (UTM)
zone:      13
datum:     nad27
ellipsoid: clark66
north:     4928000
south:     4914020
west:      590010
east:      609000
nsres:     30
ewres:     30
rows:      466
cols:      633
cells:     294978
```

The examples used here are taken from the ‘Spearfish’ sample data location (South Dakota, USA, 103.86W, 44.49N), perhaps the most typical for GRASS demonstrations. Data moved from GRASS over the interface will be given category labels if present. The interface does not support the transfer of factor level labels from R to GRASS, nor does it set colours or quantisation rules. The `readRAST6` command here reads elevation values into a `SpatialGridDataFrame` object, treating the values returned as floating point and the geology categorical layer into a factor:

```
> spear <- readRAST6(c("elevation.dem", "geology"), cat = c(FALSE,
+ TRUE))
> summary(spear)

Object of class SpatialGridDataFrame
Coordinates:
      min      max
[1,] 590010 609000
[2,] 4914020 4928000
Is projected: TRUE
proj4string :
[+proj=utm +zone=13 +a=6378206.4 +rf=294.9786982 +no_defs
+nadgrids=/home/rsb/topics/grass/g642/grass-6.4.2/etc/nad/conus
+to_meter=1.0]
Grid attributes:
  cellcentre.offset cellsize cells.dim
1           590025         30         633
2           4914035         30         466
Data attributes:
  elevation.dem      geology
Min.      :1066   sandstone:75062
1st Qu.:1200   limestone:61278
Median :1316    shale   :46207
Mean    :1354    sand    :36706
3rd Qu.:1488   igneous  :36394
Max.    :1840   (Other) :37561
NA's    :2661   NA's    : 1770
```

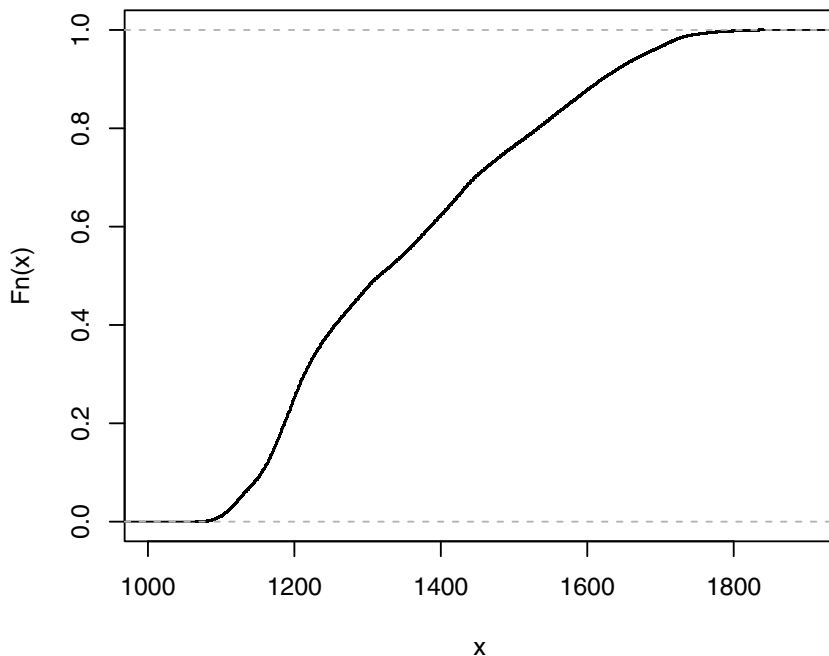


Fig. 4.7 Empirical cumulative distribution function of elevation for the Spearfish location

When the `cat` argument is set to `TRUE`, the GRASS category labels are imported and used as factor levels; checking back, we can see that they agree:

```
> table(spear$geology)

metamorphic  transition      igneous  sandstone  limestone
    11556         142      36394     75062     61278
shale sandy shale  claysand      sand
  46207     11340     14523     36706

> execGRASS("r.stats", input = "geology", flags = c("quiet",
+   "c", "l"))

1 metamorphic 11556
2 transition 142
3 igneous 36394
4 sandstone 75062
5 limestone 61278
6 shale 46207
7 sandy shale 11340
8 claysand 14523
9 sand 36706
* no data 1770
```

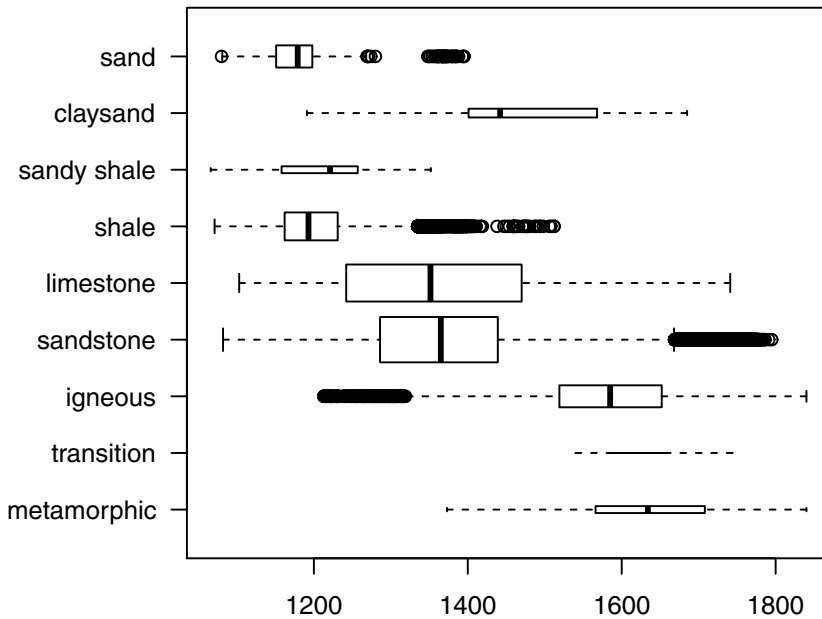


Fig. 4.8 Boxplots of elevation by geology category, Spearfish location

Figure 4.7 shows an empirical cumulative distribution plot of the elevation values, giving readings of the proportion of the study area under chosen elevations. In turn Fig. 4.8 shows a simple boxplot of elevation by geology category, with widths proportional to the share of the geology category in the total area. We have used the `readRAST6` function to read from GRASS rasters into R; the `writeRAST6` function allows a single named column of a `SpatialGridDataFrame` object to be exported to GRASS.

The `spgrass6` package also provides functions to move vector features and associated attribute data to R and back again; unlike raster data, there is no standard mechanism for handling missing values. The `readVECT6` function is used for importing vector data into R, and `writeVECT6` for exporting to GRASS. The first data set to be imported from GRASS contains the point locations of sites where insects have been monitored, the second is a set of stream channel centre-lines:

```
> bugsDF <- readVECT6("bugsites")
> vInfo("streams")
      nodes    points    lines boundaries centroids    areas
      139         0     104         12          4         4
islands    faces kernels primitives    map3d
      4         0         0        120         0

> streams <- readVECT6("streams", type = "line,boundary",
+   remove.duplicates = FALSE)
```


The `remove.duplicates` argument is set to `TRUE` when there are only, for example lines or areas, and the number present is greater than the data count (the number of rows in the attribute data table). The `type` argument is used to override type detection when multiple types are non-zero, as here, where we choose lines and boundaries, but the function guesses areas, returning just filled water bodies.

Because different mechanisms are used for passing information concerning the GRASS location coordinate reference system for raster and vector data, the PROJ.4 strings often differ slightly, even though the actual CRS is the same. We can see that the representation for the point locations of beetle sites does differ here; the vector representation is more in accord with standard PROJ.4 notation than that for the raster layers, even though they are the same. In the summary of the `spear` object above, the ellipsoid was represented by `+a` and `+rf` tags instead of the `+ellps` tag using the `clrk66` value:

```
> summary(bugsDF)

Object of class SpatialPointsDataFrame
Coordinates:
      min      max
coords.x1 590232 608471
coords.x2 4914096 4920512
Is projected: TRUE
proj4string :
[+proj=utm +zone=13 +datum=NAD27 +units=m +no_defs
+ellps=clrk66
+nadgrids=@conus,@alaska,@ntv2_0.gsb,@ntv1_can.dat]
Number of points: 90
Data attributes:
      cat      str1
Min.   : 1.00   Beetle site:90
1st Qu.:23.25
Median :45.50
Mean   :45.50
3rd Qu.:67.75
Max.   :90.00
```

This necessitates manual assignment from one representation to the other in some occasions, and is due to GRASS using non-standard but equivalent extensions to PROJ.4.

There are a number of helper functions in the `spgrass6` package, one `gmeta2grd` to generate a `GridTopology` object from the current GRASS region settings. This is typically used for interpolation from point data to a raster grid, and may be masked by coercion from a `SpatialGrid` to a `SpatialPixels` object having set cells outside the study area to NA. A second utility function for vector data uses the fact that GRASS 6 uses a topological vector data model. The `vect2neigh` function returns a data frame with the left and right neighbours of arcs on polygon boundaries, together with the length of the arcs. This can be used to modify the weighting of polygon contiguities based on the length of shared boundaries. Like GRASS,

GDAL/OGR, PROJ.4, and other OSGeo projects, the functions offered by **spgrass6** are changing, and current help pages should be consulted to check correct usage.

The interface between GRASS 6 and R has been used in research in a number of fields, for example by Carrera-Hernández and Gaskin (2008) in implementing the Basin of Mexico hydrogeological database, and by Grohmann and Steiner (2008) in SRTM resampling using short distance kriging. The work by Haywood and Stone (2011) is interesting in that it uses the interface to apply the Weka machine learning software suite, itself interfaced to R through the **RWeka** package, to GIS data in GRASS; R then becomes a convenient bridge between applications, with the GRASS–R interface opening up other possibilities beyond R.

4.5.1 Broad Street Cholera Data

Even though we know that John Snow already had a working hypothesis about cholera epidemics, his data remain interesting, especially if we use a GIS to find the street distances from mortality dwellings to the Broad Street pump in Soho in central London. Brody et al. (2000) point out that John Snow did not use maps to ‘find’ the Broad Street pump, the polluted water source behind the 1854 cholera epidemic, because he associated cholera with water contaminated with sewage, based on earlier experience. The accepted opinion of the time was that cholera was most probably caused by a ‘concentrated noxious atmospheric influence’, and maps could just as easily have been interpreted in support of such a point source.

The specific difference between the two approaches is that the atmospheric cause would examine straight-line aerial distances between the homes of the deceased and an unknown point source, while a contaminated water source would rather look at the walking distance along the street network to a pump or pumps. The basic data to be used here were made available by Jim Detwiler, who had collated them for David O’Sullivan for use on the cover of O’Sullivan and Unwin (2003), based on earlier work by Waldo Tobler and others. The files were a shapefile with counts of deaths at front doors of houses and a georeferenced copy of the Snow map as an image; the files were registered in the British National Grid CRS. The steps taken in GRASS were to set up a suitable location in the CRS, to import the image file, the file of mortalities, and the file of pump locations.

To measure street distances, the building contours were first digitised as a vector layer, cleaned, converted to raster leaving the buildings outside the street mask, buffered out 4m to include all the front door points within the street mask, and finally distances measured from each raster cell in the buffered street network to the Broad Street pump and to the nearest other pump. These operations in summary were as follows:

```
v.digit -n map=vsnow4 bgcmd="d.rast map=snow"
v.to.rast input=vsnow4 output=rfsnow use=val value=1
r.buffer input=rfsnow output=buff2 distances=4
r.cost -v input=buff2 output=snowcost_not_broad \
  start_points=vpump_not_broad
r.cost -v input=buff2 output=snowcost_broad start_points=vpump_broad
```

The main operation here is `r.cost`, which uses the value of 2.5 m stored in each cell of `buff2`, which has a resolution of 2.5 m, to cumulate distances from the start points in the output rasters. The operation is carried out for the other pumps and for the Broad Street pump. This is equivalent to finding the line of equal distances shown on the extracts from John Snow's map shown in Brody et al. (2000, p. 65). It is possible that there are passages through buildings not captured by digitising, so the distances are only as accurate as can now be reconstructed.

```
> sohoSG <- readRAST6(c("snowcost_broad", "snowcost_not_broad"))
> buildings <- readVECT6("vsnow4")
> proj4string(sohoSG) <- CRS(proj4string(buildings))
```

For visualisation, we import the building outlines, and the two distance rasters. Next we import the death coordinates and counts, and overlay the deaths on the distances, to extract the distances for each house with mortalities – these are added to the `deaths` object, together with a logical variable indicating whether the Broad Street pump was closer (for this distance measure) or not:

```
> deaths <- readVECT6("deaths3")
> o <- over(deaths, sohoSG)
> library(maptools)
> deaths <- spCbind(deaths, o)
> deaths$b_nearer <- deaths$snowcost_broad < deaths$snowcost_not_broad

> by(deaths$Num_Cases, deaths$b_nearer, sum)

deaths$b_nearer: FALSE
[1] 221
-----
deaths$b_nearer: TRUE
[1] 357

> nb_pump <- readVECT6("vpump_not_broad")
> b_pump <- readVECT6("vpump_broad")
```

There are not only more mortalities in houses closer to the Broad Street pump, but the distributions of distances are such that their inter-quartile ranges do not overlap. This can be seen in Fig. 4.9, from which a remaining question is why some of the cases appear to have used the Broad Street pump in spite of having a shorter distance to an alternative. Finally, we import the locations of the pumps to assemble a view of the situation, shown in Fig. 4.10.

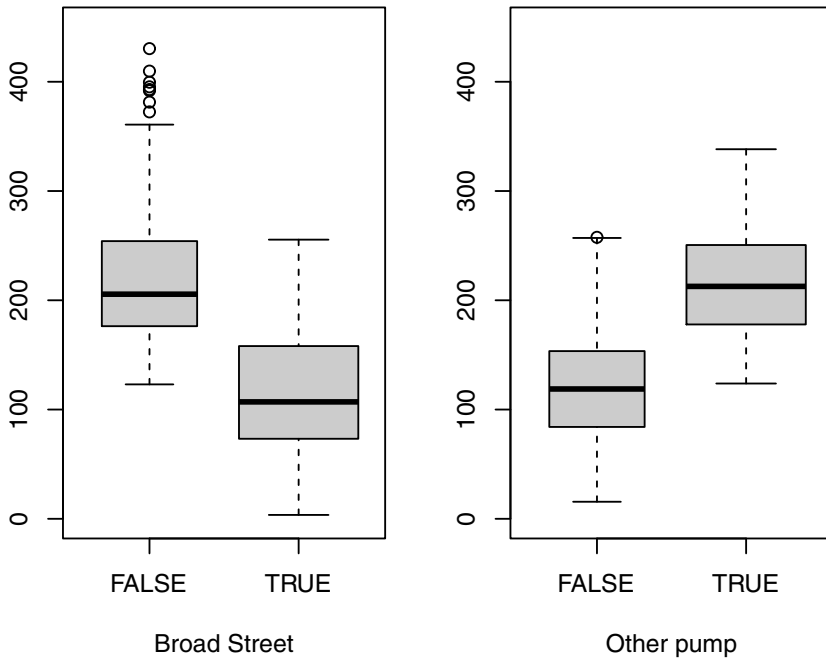


Fig. 4.9 Comparison of walking distances from homes of fatalities to the Broad Street pump or another pump by whether the Broad Street pump was closer or not

The colour scaled streets indicate the distance of each 2.5 m raster cell from the Broad Street pump along the street network. The buildings are overlaid on the raster, followed by proportional symbols for the number of mortalities per affected house, coded for whether they are closer to the Broad Street pump or not, and finally the pumps themselves.

It is possible to reproduce some of the analysis on the R side using **rgeos** and **gdistance**, by importing the digitised building outlines from GRASS into R, using **gBuffer** to buffer them in from the street so that the house points with mortalities fall in the street, then an **over** method to define a street raster.

```
> library(rgeos)
> vsnow4buf <- gBuffer(buildings, width = -4)
> GRD <- gmeta2grd()
> SG <- SpatialGrid(GRD, proj4string = CRS(proj4string(vsnow4buf)))
> o <- over(SG, vsnow4buf)
> crs <- CRS(proj4string(vsnow4buf))
> SGDF <- SpatialGridDataFrame(GRD, proj4string = crs),
+   data = data.frame(o = o))
> SGDF$o[is.na(SGDF$o)] <- 2.5
> SGDF$o[SGDF$o == 1] <- NA
```

Finally, **rSPDistance** may be used to find cost distances from houses with mortalities to pumps:

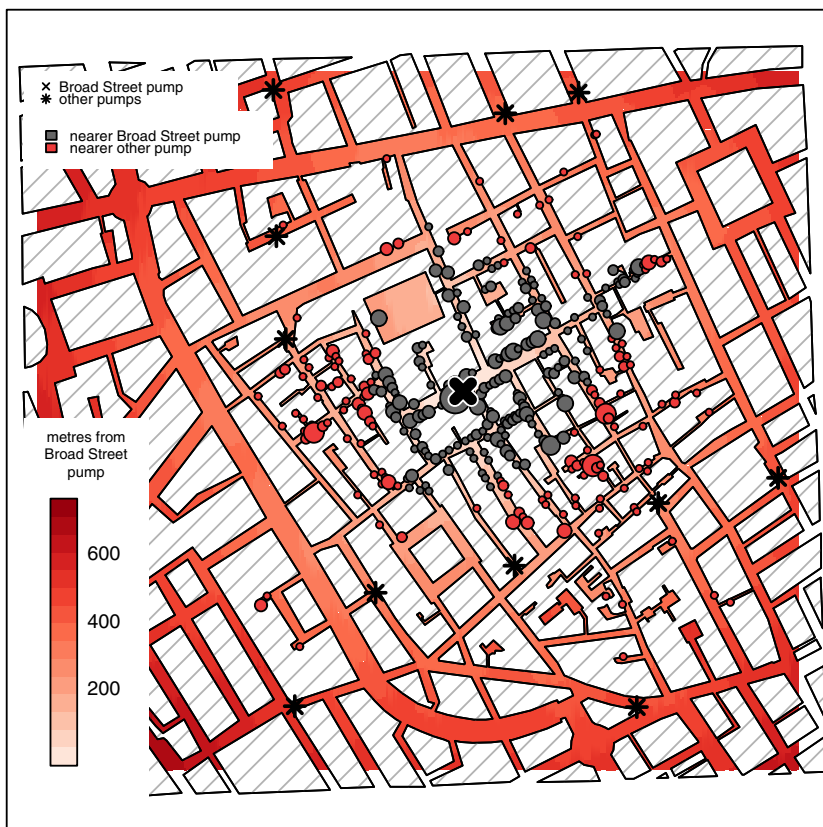


Fig. 4.10 The 1854 London cholera outbreak near Golden Square

```
> library(gdistance)
> r <- as(SGDF, "RasterLayer")
> tr <- transition(r, mean, 8)
> d_b_pump <- rSPDistance(tr, deaths, b_pump, theta = 1e-12)
> d_nb_pump <- rSPDistance(tr, deaths, nb_pump, theta = 1e-12)

> deaths$g_snowcost_broad <- d_b_pump[, 1]
> deaths$g_snowcost_not_broad <- apply(d_nb_pump, 1, min)
> deaths$g_b_nearer <- deaths$g_snowcost_broad < deaths
  $g_snowcost_not_broad
> by(deaths$Num_Cases, deaths$g_b_nearer, sum)

deaths$g_b_nearer: FALSE
[1] 272
-----
deaths$g_b_nearer: TRUE
[1] 306
```

Neither the values nor the distances are the same as those yielded by the GRASS module `r.cost`, but the conclusion is the same despite the differences in implementations of cost distances.

4.6 Other Import/Export Interfaces

The classes for spatial data introduced in **sp** have made it easier to implement and maintain the import and export functions described earlier in this chapter. In addition, they have created opportunities for writing other interfaces, because the structure of the objects in R is better documented. In this section, a number of such interfaces will be presented, with others to come in the future, hosted in **maptools** or other packages. Before going on to discuss interfaces with external applications, conversion wrappers for R packages will be mentioned.

The **maptools** package contains interface functions to convert selected **sp** class objects to classes used in the **spatstat** for point pattern analysis – these are written as coercion methods to and from **spatstat** `ppp`, `owin`, `im` and `psp` classes. **maptools** also contains the `SpatialLines2PolySet` and `SpatialPolygons2PolySet` functions to convert **sp** class objects to `PolySet` class objects as defined in the **PBSmapping** package, and a pair of matching functions in the other direction. This package provides a number of GIS procedures needed in fisheries research (PBS is the name of the Pacific Biological Station in Nanaimo, British Columbia, Canada).

The four successor packages to the **adehabitat** package: **adehabitatHR**, **adehabitatHS**, **adehabitatLT**, and **adehabitatMA**, all depend on **sp** and use **sp** classes directly. The original package was documented in Calenge (2006), and includes many tools for the analysis of space and habitat use by animals.

4.6.1 Analysis and Visualisation Applications

While many kinds of data analysis can be carried out within the R environment, it is often very useful to be able to write out files for use in other applications or for sharing with collaborators not using R. These functions live in **maptools** and will be extended as required. The `sp2tmap` function converts a `SpatialPolygons` object for use with the StataTM `tmap` contributed command,²¹ by creating a data frame with the required columns. The data frame returned by the function is exported using `write.dta` from the **foreign** package, which should also be used to export the attribute data with the

²¹ <http://www.stata.com/search.cgi?query=tmap>.

polygon tagging key. The `sp2WB` function exports a `SpatialPolygons` object as a text file in `S-PLUS™` map format to be imported by WinBUGS.

The **GeoXp** package provides some possibilities for interactive statistical data visualisation within R, including mapping (Laurent et al., 2012). The R graphics facilities are perhaps better suited to non-interactive use, however, especially as it is easy to write data out to Mondrian (Theus, 2002; Theus and Urbanek, 2009).²² Mondrian provides fully linked multiple plots, and although the screen can become quite ‘busy’, users find it easy to explore their data in this environment. The function `sp2Mondrian` in **maptools** writes out two files, one with the data, the other with the spatial objects from a `SpatialPolygonsDataFrame` object for Mondrian to read; the polygon format before Mondrian 1.0 used a single file and may still be used, controlled by an additional argument.

4.6.2 *TerraLib and aRT*

The **aRT** package²³ provides an advanced modular interface to TerraLib.²⁴ TerraLib is a GIS classes and functions library intended for the development of multiple GIS tools. Its main aim is to enable the development of a new generation of GIS applications, based on the technological advances on spatial databases. TerraLib defines the way that spatial data are stored in a database system, and can use MySQL, PostgreSQL, Oracle, or Access as a back-end. The library itself can undertake a wide range of GIS operations on the data stored in the database, as well as storing and retrieving the data as spatial objects from the database system.

The **aRT** package interfaces **sp** classes with TerraLib classes, permitting data to flow between R, used as a front-end system interacting with the user, through TerraLib and the back-end database system. One of the main objectives of **aRT** is to do spatial queries and operations in R. Because these operations are written to work efficiently in TerraLib, a wide range of overlay and buffering operations can be carried out, without them being implemented in R itself. Operations on the geometries, such as whether they touch, how far apart they are, whether they contain holes, polygon unions, and many others, can be handed off to TerraLib.

A further innovation is the provision of a wrapper for the R compute engine, allowing R with **aRT** to be configured with TerraLib between the back-end database system and a front-end application interacting with the user. This application, for example TerraView, can provide access through menus to spatial data analysis functionality coded in R using **aRT**.²⁵ All of

²² <http://rosuda.org/Mondrian/>.

²³ <http://leg.ufpr.br/aRT/>.

²⁴ <http://www.terralib.org/>.

²⁵ Andrade Neto and Ribeiro Jr. (2005).

this software is released under open source licences, and offers considerable opportunities for building non-proprietary customised systems for medium and larger organisations able to commit resources to C++ programming. Organisations running larger database systems are likely to have such resources anyway, so **aRT** and TerraLib provide a real alternative for fresh spatial data handling projects.

4.6.3 Other GIS Systems

An interface package – **RSAGA** – has been provided for SAGA GIS²⁶; like the GRASS 6 interface, it uses **system** to pass commands to external software. The R interface with SAGA has been used by Brenning (2009) for integrating terrain analysis and multispectral remote sensing in automatic rock glacier detection, using modern regression techniques – the availability of many varied techniques in R permitted them to be evaluated rapidly. Goetz et al. (2011) follow this up in integrating physical and empirical landslide models. In a paper on geostatistical modelling of topography, Hengl et al. (2008) use the interface between R and SAGA to benefit from the strengths of both software components. Hengl et al. (2010) address the associated problem of stream network uncertainty, when the stream networks are derived from interpolated elevation data, again using the interface between SAGA and R; R is also used extensively for scripting SAGA.

In connection with a comprehensive book on spatial statistical data analysis based on ArcGIS™, Krivoruchko (2011, pp. 767–801) devotes Appendix 2 to the use of R with ArcGIS™, using both file transfer, and (D)COM and Python interfaces. In addition, Chap. 16, pp. 676–715 covers a range of useful examples of R/ArcGIS use for spatial data analysis, with many code examples. A link to updated code may be found on the website of this book.

In the discussion above, integration between R and GIS has principally taken the form of file transfer. It is possible to use other mechanisms, similar in nature to the embedding of R in TerraView using **aRT**. One example is given by Tait et al. (2004), using the **RStatConnector** (D)COM mechanism to use R as a back-end from ArcGIS™. The specific context is the need to provide epidemiologists using ArcGIS™ for animal disease control and detection with point pattern analysis tools, using a GIS interface. The prototype was a system using **splancs** running in R to calculate results from data passed from ArcGIS™, with output passed back to ArcGIS™ for display. A practical difficulty of embedding both R and **splancs** on multiple workstations is that of software installation and maintenance.

²⁶ <http://www.saga-gis.org>

The marine geospatial ecology tools project²⁷ follows up the work begun in the concluded ArcRstats project, providing for execution in many environments (Roberts et al., 2010). It is not hard to write small Python scripts to interface R and ArcGIS™ through temporary files and the `system` function. This is illustrated by the **RPyGeo** package, which uses R to write Python scripts for the ArcGIS™ geoprocessor.

4.7 Installing **rgdal**

Because **rgdal** depends on external libraries, on GDAL and PROJ.4, and particular GDAL drivers may depend on further libraries, installation is not as easy as with self-contained R packages. Thanks to sustained contributions by Brian Ripley and Uwe Ligges, CRAN publishes a self-contained Windows binary **rgdal** package for 32-bit and 64-bit architectures, with a substantial range of drivers available. A similar range of drivers is available for Intel architectures for Mac OSX in binary **rgdal** packages published on CRAN thanks to continuing help from Simon Urbanek.

For Linux/Unix, it is necessary to install **rgdal** from source, after first having installed the external dependencies. Users of open source GIS applications such as GRASS will already have GDAL and PROJ.4 installed anyway, because they are required for such applications.

In general, GDAL and PROJ.4 will install from source without difficulty, but care may be required to make sure that libraries needed for drivers are available and function correctly. If the programs `proj`, `gdalinfo`, and `ogrinfo` work correctly for data sources of interest after GDAL and PROJ.4 have been installed, then **rgdal** will also work correctly. Mac OSX users may find William Kyngesburye's frameworks²⁸ a useful place to start should additional drivers be required. More information is available by searching the archives of the R-sig-geo mailing list, but frequent changes in **rgdal** may render older postings less relevant.

Windows users needing other drivers, and for whom conversion using programs in the OSGeo4W²⁹ binary for Windows is not useful, may choose to install **rgdal** from source, compiling the **rgdal** DLL with VC++ and linking against the OSGeo4W DLLs – see the `inst/README.windows` file in the source package for details.

²⁷ <http://code.env.duke.edu/projects/mget>.

²⁸ <http://www.kyngchaos.com/software/frameworks>.

²⁹ <http://trac.osgeo.org/osgeo4w/>.

Chapter 5

Further Methods for Handling Spatial Data

This chapter is concerned with a more detailed explanation of some of the methods that are provided for working with the spatial classes described in Chap. 2. We first consider the question of the spatial support of observations, going on to cover the handling and combination of features using in particular the **rgeos** package. Next we consider map overlay, also known as spatial join operations, including aggregation, extract operations in the **raster** package, and spatial sampling.

5.1 Support

In data analysis in general, the relationship between the abstract constructs we try to measure and the operational procedures used to make the measurements is always important. Very often substantial metadata volumes are generated to document the performance of the instruments used to gather data. Naturally, the same applies to spatial data. Positional data need as much care in documenting their collection as other kinds of data. When approximations are used, they need to be recorded as such. Part of the issue is the correct recording of projection and datum, which are covered in Chap. 4. The time-stamping of observations is typically useful, for example when administrative boundaries change over time.

The recording of position for surveying, for example for a power company, involves the inventorying of cables, pylons, transformer substations, and other infrastructure. Much GIS software was developed to cater for such needs, inventory rather than analysis. For inventory, arbitrary decisions, such as placing the point coordinate locating a building by the right-hand doorpost facing the door from the outside, have no further consequences. When, however, spatial data are to be used for analysis and inference, the differences between arbitrary assumptions made during observation and other possible

spatial representations of the phenomena of interest will feed through to the conclusions. The adopted representation is known as its support, and is discussed by Waller and Gotway (2004, pp. 38–39). The point support of a dwelling may be taken as the point location of its right-hand doorpost, a soil sample may have point support of a coordinate surveyed traditionally or by GPS. But the dwelling perhaps should have polygonal support, and in collecting soil samples, most often the point represents a central position in the circle or square used to gather a number of different samples, which are then bagged together for measurement.

An example of the effects of support is the impact of changes in voting district boundaries in election systems, which are not strictly proportional. The underlying voting behaviour is fixed, but different electoral results can be achieved by tallying results in different configurations or aggregations of the voters' dwellings. The **BARD** package for automated redistricting and heuristic exploration of redistricter revealed preference is an example of the use of R for studying this problem (Altman and McDonald, 2011); the package is archived on CRAN. When carried out to benefit particular candidates or parties, this is known as gerrymandering. The aggregations are arbitrary polygons, because they do not reflect a political entity as such. This is an example of change of support, moving from the position of the dwelling of the voter to some aggregation. Change of support is a significant issue in spatial data analysis, and is introduced in Schabenberger and Gotway (2005, pp. 284–285). A much more thorough treatment is given by Gotway and Young (2002), who show how statistical methods can be used to carry through error associated with change of support to further steps in analysis. In a very similar vein, it can be argued that researchers in particular subject domains should consider involving statisticians from the very beginning of their projects, to allow sources of potential uncertainty to be instrumented if possible. One would seek to control error propagation when trying to infer from the data collected later during the analysis and reporting phase (Guttorp, 2003; Wikle, 2003). An example might be marine biologists and oceanographers not collecting data at the same place and time, and hoping that data from different places and times could be readily combined without introducing systematic error.

One of the consequences of surveying as a profession being overtaken by computers, and of surveying data spreading out to non-surveyor users, is that understanding of the imprecision of positional data has been diluted. Some of the imprecision comes from measurement error, which surveyors know from their training and field experience. But a digital representation of a coordinate looks very crisp and precise, deceptively so. Surveying and cartographic representations are just a summary from available data. Where no data were collected, the actual values are guesswork and can go badly wrong, as users of maritime charts routinely find. Further, support is routinely changed for purposes of visualisation: contours or filled contours representing a grid make the data look much less 'chunky' than an image plot of the same

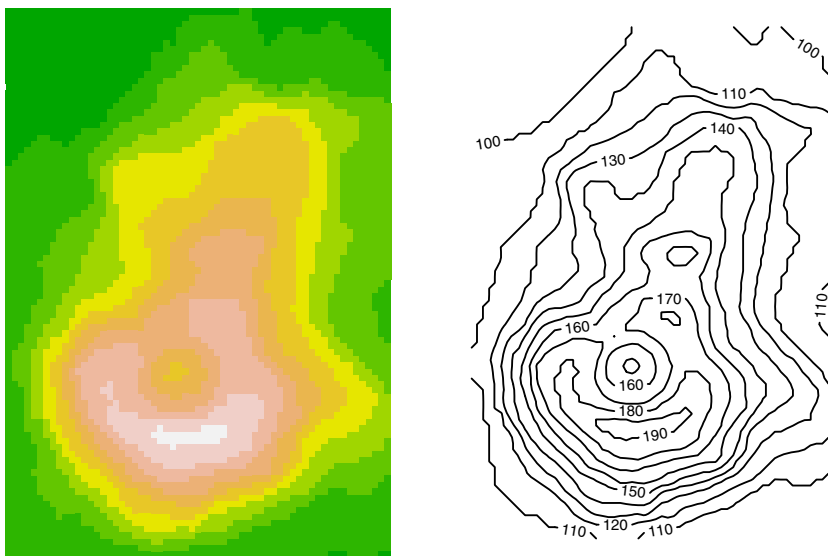


Fig. 5.1 Image plot and contour plot representations of Maunga Whau from the standard R `volcano` data set, for the same elevation class intervals (rotated to put north at the top)

data, as Fig. 5.1 shows. In fact, the data were digitised from a paper map by Ross Ihaka, as much other digital elevation data have been, and the paper map was itself a representation of available data, not an exact reproduction of the terrain. Even SRTM data can realistically be used only after cleaning; the 3arcsec data used in Sect. 2.7 were re-sampled from noisier 1 arcsec data using a specific re-sampling and cleaning algorithm. A different algorithm would yield a slightly different digital elevation model.

While we perhaps expect researchers wanting to use R to analyse spatial data to be applied domain scientists, it is worth noting that geographical information science, the field of study attempting to provide GIS with more consistent foundations, is now actively incorporating error models into position measurement, and into spatial queries based on error in observed values. Say we are modelling crop yield based on soil type and other variables, and our spatial query at point i returns `"sand"`, when in fact the correct value at that location is `"clay"`, our conclusions will be affected. The general application of uncertainty to spatial data in a GIS context is reviewed by Worboys and Duckham (2004, pp. 328–358), and attribute error propagation is discussed by Heuvelink (1998). In an open computing environment like R, it is quite possible to think of ‘uncertain’ versions of the ‘crisp’ classes dealt with so far, in which, for example point position could be represented as a value drawn from a statistical model, allowing the impact of positional uncertainty on other methods of analysis to be assessed (see for example Leung et al., 2004).

5.2 Handling and Combining Features

Moving from one spatial representation to another, and combining data with different representations are typical operations performed with spatial data. These operations involve combining congruent or non-congruent spatial data objects, and only some are provided directly, chiefly for non-congruent objects. Overlay operations, for example, are mentioned by Burrough and McDonnell (1998, pp. 52–53) and covered in much more detail by O’Sullivan and Unwin (2010, pp. 315–340) and Unwin (1996), who show how many of the apparently deterministic inferential problems in overlay are actually statistical in nature, as noted earlier.

The introduction of the **raster** and **rgeos** packages has provided R users with a much broader range of GIS operations than earlier, many for handling, combining and querying features. The **raster** package overloads arithmetic operators to provide map algebra for congruent rasters, and supplements these with focal operators. There are functions for rasterising vector objects to permit operations to be carried out across types of representations; we will return to these below. Details are given in the package documentation and in Hijmans (2012b).

The **rgeos** package assumes that the features may be treated as planar, so that distances are measured in map units. When **raster** is working with congruent features, it is not important whether the coordinates are geographical or projected. When no coordinate reference system is declared, **raster** assumes that the coordinates are geographical (see Sect. 4.1 for a discussion of coordinate reference systems).

Two packages associated with **raster** try to address the problems that arise in measuring distances on grids specified in geographical coordinates. The **gdistance** package provides functions for calculating distances and routes on such grids (van Etten, 2012). The **geosphere** package in turn contains functions for spherical trigonometry for geographical applications, including the calculation of the area, perimeter, and other characteristics of polygons specified with geographical coordinates (Hijmans, 2012a). Auxilliary functions are described in Sect. 5.4.

5.2.1 The *rgeos* Package

The **rgeos** package was developed as a Google™ Summer of Coding project in 2010 by Colin Rundel, and, like **rgdal**, is released on CRAN as a source package to be installed using the GEOS (Geometry Engine – Open Source) library¹ already present on the computer, and as self-contained Windows and

¹ <http://trac.osgeo.org/geos/>.

Mac OSX binary packages including a recent GEOS version by static linking. On loading, the package shows its package version, and the version of the underlying GEOS library.

```
> library(rgeos)

rgeos: version: 0.2-13, (SVN revision 376)
GEOS runtime version: 3.3.8-CAPI-1.7.8
Polygon checking: TRUE
```

It also reports the status of polygon checking, which will be explained briefly below. First, we'll present a summary of the basis for the GEOS library, which is a C++ port of the Java Topology Suite (JTS).² Revisions to JTS pass to GEOS fairly quickly, and GEOS releases occur quite often. Writing software for processing linear geometry on the 2-dimensional Cartesian plane is hard, because the predicates and operations for regular cases are only a small subset of the cases that are met in practice. JTS, and GEOS, only handle planar geometries, so spatial objects should be projected. Two design choices are described in Davis and Aquino (2003) and Aquino (2003), the original JTS technical specifications and developer guide.³ The design choices are related to the numerical issues raised by computational geometry, and to the standard used to represent features. One key issue is the identification of equal coordinates – doing computational geometry on the real plane leads to the minimum distance between coordinates permitted if they are to be seen as identical. JTS and GEOS place coordinates on a very fine grid, scaling and rounding them, so that sufficiently similar coordinates are placed on the same grid node. The scaling factor can be returned by using `getScale`; we will see below how it can be used:

```
> getScale()

[1] 1e+08
```

The second design choice taken in JTS and GEOS was to follow the OpenGIS[®]⁴ Simple Features Specification (Herring, 2011), in which polygons may have one and only one exterior boundary ring, and an unlimited number of interior boundaries – holes. As our `Polygons` objects are `Multi-Polygon` objects as defined by Kresse et al. (2012, p. 507) and Herring (2011, p. 31), we have no direct match to their `Polygon` object, defined in Kresse et al. (2012, p. 507) and Herring (2011, pp. 26–28). This means that multiple exterior boundaries – such as a county made up of several islands – are represented as multiple polygons. In the specification, they are linked to attribute data through a look-up table pointing to the appropriate attribute data row.

² <http://tsusiatsoftware.net/jts/main.html>.

³ These are still available at <http://www.vividsolutions.com/jts/bin/JTSTechnicalSpecs.pdf>, <http://www.vividsolutions.com/jts/bin/JTSDeveloperGuide.pdf>.

⁴ See <http://www.opengeospatial.org/standards/sfa>.

This differs from the `Polygons` class defined above (p. 43), and makes it necessary for `Polygons` to have appropriately formatted `comment` attributes to encode any holes with their enclosing exterior rings (see Sect. 2.6.2 for a further discussion).

This design choice only affects polygon representation, so points and lines can be moved between the form used in `sp` and GEOS without change. The `rgeos` package is provided with a snapshot of the tests published by JTS and GEOS, and may be run when the package is loaded by running them, turning off polygon checking because the input data are already in GEOS well-known text (WKT) format:

```
> library(testthat)
> set_do_poly_check(FALSE)
> test_package("rgeos")
> set_do_poly_check(TRUE)
```

Warnings appear because the tests try to find out how the software handles invalid features.

5.2.2 Using rgeos

In order to explore the use of functions in the `rgeos` package, we introduce a new data set combining vector data from the Brazilian 2010 population census with remotely sensed raster data. The data are from Olinda, which is a historic city in the state of Pernambuco, on Brazil's Atlantic coast. The census data used here is from preliminary reports, and the sources are given in detail on p. 366. The data have been subsetted and pre-processed to simplify discussions of `rgeos` functionality, to an ESRI Shapefile for the vector data, and to three GeoTiff files for the raster data. The vector data contains the boundaries of the 2010 census enumeration districts (ED), ED identification codes and a single residential population variable (Pessoas residentes).

```
> olinda <- readOGR(".", "olinda1")
> proj4string(olinda) <- CRS("+init=epsg:4674")
> olinda_utm <- spTransform(olinda, CRS("+init=epsg:31985"))
```

The input vector data are in SIRGAS 2000 (effectively WGS84) geographical coordinates, and are projected to UTM 25S, as commonly used in this part of Brazil. Using `readOGR` (see p. 92), to read the vector data provides information for the allocation of interior rings to exterior rings described in Sect. 2.6.2. This means that polygonal data read with `readOGR` should be `rgeos`-ready, conformant with OGC SFS, without further processing. Polygonal data from other sources may need to be pre-processed with the `rgeos` function `createSPComment`, or the `maptools` function `checkPolygonsHoles` applied listwise to the `Polygons` objects in the "polygons" slot of a `SpatialPolygons` object. If polygonal data read with `readOGR` still fail in `rgeos`,

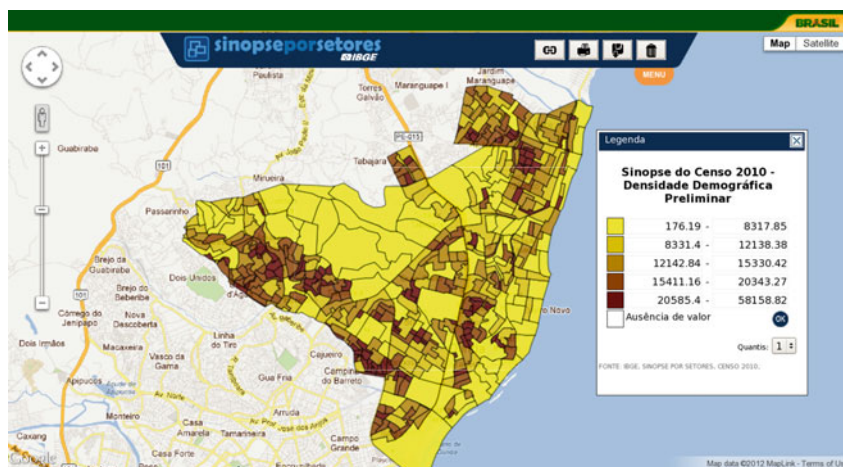


Fig. 5.2 Screen capture of population density display for Olinda, 2010, IBGE – Instituto Brasileiro de Geografia e Estatística

it is worth trying the re-creation of the comments of the `Polygons` objects. If that does not help, changing the GEOS scaling factor may help, as we will see below, but if the input data are not clean, and contain slivers, dangles, or other artefacts, it may be necessary to clean the data before use.

First, we will try to reproduce the online display of population density shown in Fig. 5.2 using the downloaded data. Having projected the polygons to UTM, we have a metre metric, but the display is in km^2 . Of course, `Polygons` objects have an "area", so we can extract the areas by ED from the `SpatialPolygons` object. We can also calculate them using the `rgeos` function `gArea`, using the `byid` argument to return values by ED, rather than for the whole city:

```
> Area <- gArea(olinda_utm, byid = TRUE)
> olinda_utm$area <- sapply(slot(olinda_utm, "polygons"),
+   slot, "area")
> all.equal(unname(Area), olinda_utm$area)

[1] TRUE

> olinda_utm$dens <- olinda_utm$V014/(olinda_utm$area/1e+06)
```

It is comforting to see that the areas by ED are the same, despite being calculated with two different implementations of underlying computational geometry methods. We can tally the ED densities by dividing the ED resident populations by the area after conversion from m^2 to km^2 . Figure 5.3 may be compared with the screen dump, and appears to show the same spatial pattern, even though we have had to reconstruct the areas of the enumeration districts.

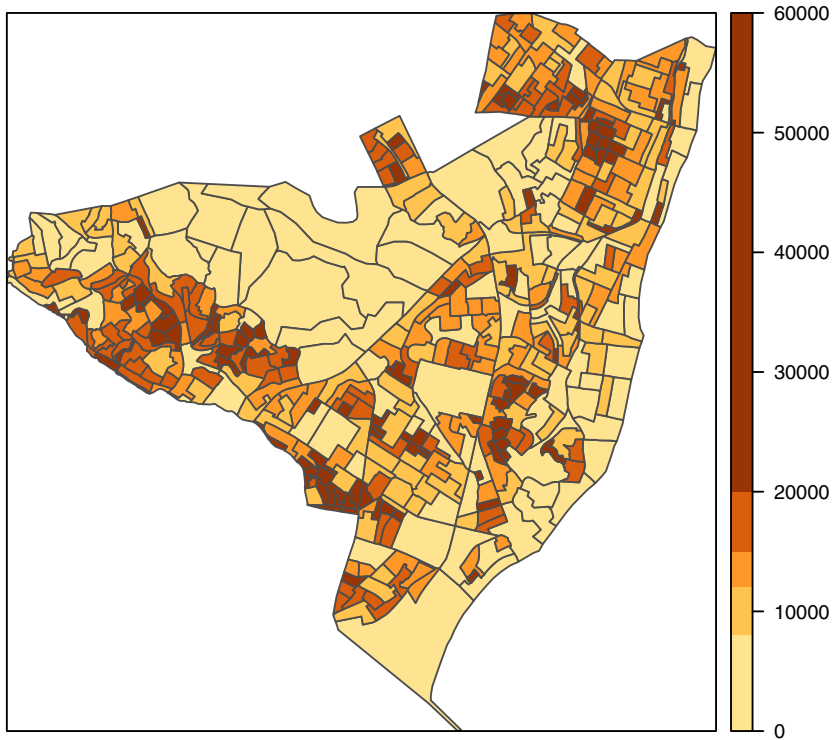


Fig. 5.3 Population density, Olinda, 2010

The functions provided in JTS, GEOS, and **rgeos** fall into three main classes: miscellaneous functions like **gArea**, topological predicates, and topological operations. Some predicates and operations are unary, taking one vector object, while others are binary, taking two objects. Because there are also numerous exceptions from this tidy scheme, short-cut functions are often provided. If we want to merge all the ED polygons to create a new **SpatialPolygons** object for the city limits, we need a unary version of the binary operation **gUnion**, because the two objects to which the union operation is applied are the same. So we use **gUnaryUnion**, one of a number of short-cut functions, to carry out this merge operation, also known as dissolving polygons. The operation can also take an **id** argument, a character vector defining id labels for the resulting features, which must be of the same length as the number of **Polygons** objects in the input object.

```
> bounds <- gUnaryUnion(olinda_utm)
> gArea(olinda_utm)
[1] 41691721
> sapply(slot(slot(bounds, "polygons")[[1]], "Polygons"),
+       slot, "area")
```

```
[1] 4.169172e+07 2.137516e-07 3.721028e-06 4.407639e-10 1.466113e-04
[6] 4.849594e-05 9.587388e-08
```

After merging the polygons, we find that we have a single `Polygons` object with multiple `Polygon` object members, one with the area we expect, but several other slivers. If we try the `gOverlaps` unary predicate, which tests the `Polygons` objects in `olinda_utm` to see if they overlap with each other excluding comparisons with the same `Polygons` object. As we see, they do overlap, even though the input shapefile was downloaded from a reliable source.

```
> polys_overlap <- gOverlaps(olinda_utm, byid = TRUE)
> any(polys_overlap)
[1] TRUE
```

The overlaps were not introduced during projection either – the reader may check as an exercise. Recalling the design choices made in JTS and GEOS, we can try to change the scaling factor, here by four orders of magnitude. Since at this scaling we see no remaining overlaps, we may merge the polygons, create the desired city limits object, and reset to scaling factor to its original value:

```
> oScale <- getScale()
> setScale(10000)
> polys_overlap <- gOverlaps(olinda_utm, byid = TRUE)
> any(polys_overlap)
[1] FALSE
> bounds <- gUnaryUnion(olinda_utm)
> setScale(oScale)
> sapply(slot(slot(bounds, "polygons")[[1]], "Polygons"),
+        slot, "area")
[1] 41691721
```

Next, let us read rasters containing Landsat 7 data, one with the 14.25m resolution panchromatic band, and a second with 28.5m Enhanced Thematic Mapper bands 1–5 and 7. Finally, the 90m resolution SRTM digital elevation model raster is also available; all have been cropped to a bounding box including Olinda, and are all in the same projection. The coordinate reference system definitions are overwritten using that of the city limits object, because identical projections are checked as strings rather than the equivalence of arguments.

```
> pan <- readGDAL("L7_ETM8s.tif")
> proj4string(pan) <- CRS(proj4string(bounds))
> TM <- readGDAL("L7_ETMs.tif")
> proj4string(TM) <- CRS(proj4string(bounds))
> names(TM) <- c("TM1", "TM2", "TM3", "TM4", "TM5", "TM7")
> dem <- readGDAL("olinda_dem_utm25s.tif")
> proj4string(dem) <- CRS(proj4string(bounds))
> is.na(dem$band1) <- dem$band1 <= 0
```

We do not have a map layer of water channels, but may be interested in finding out which enumeration districts are crossed by such channels, and the proportion of the area of EDs within a buffer of channels. We can use GRASS GIS to analyse the watersheds found in the digital elevation model, so first create a throw-away location, and set its projection to that of the city limits object:

```
> library(spgrass6)
> myGRASS <- "/home/rsb/topics/grass/g642/grass-6.4.2"
> loc <- initGRASS(myGRASS, tempdir(), SG = dem, override = TRUE)
> execGRASS("g.mapset", mapset = "PERMANENT")
> execGRASS("g.proj", flag = "c", proj4 = proj4string(bounds))
> execGRASS("g.mapset", mapset = loc$MAPSET)
> execGRASS("g.region", flag = "d")
```

First, however, we resample the digital elevation model from the input 90 m to the 14.25 m resolution of the Landsat panchromatic band, using a regularized spline with tension method (Neteler and Mitasova, 2008, pp. 121–122, 247–249):

```
> writeRAST6(dem, "dem", flags = "o")
> execGRASS("g.region", rast = "dem")
> respan <- gridparameters(pan)$cellsize
> execGRASS("r.resamp.rst", input = "dem", ew_res = respan[1],
+   ns_res = respan[2], elev = "DEM_resamp")
> execGRASS("g.region", rast = "DEM_resamp")
```

While this may introduce smoothing in the digital elevation model that differs from the real land surface in Olinda, it permits us to carry out the watershed analysis as the next step (Neteler and Mitasova, 2008, pp. 143–147). Here we choose only to return the raster stream lines with parameter values that appear to work well enough for this data set. Next we thin the raster streams, and complete by converting them to a GRASS line vector:

```
> execGRASS("r.watershed", elevation = "DEM_resamp", stream = "stream",
+   threshold = 1000L, convergence = 5L, memory = 300L)
> execGRASS("r.thin", input = "stream", output = "stream1",
+   iterations = 200L)
> execGRASS("r.to.vect", input = "stream1", output = "stream",
+   feature = "line")
```

When we read the stream lines object back into the running R session, we find that there are relatively many line segments. Examining the summary of lengths of these lines, we see using `gLength` that many are 14.25 m long, equal to the resolution of the resampled digital elevation model:

```
> stream_utm <- readVECT6("stream")
> proj4string(stream_utm) <- CRS("+init=epsg:31985")
> nrow(stream_utm)
```

```
[1] 286
```

```
> summary(gLength(stream_utm, byid = TRUE))

  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 14.25  14.25  358.80  469.50  712.90 2644.00
```

Using the unary predicate `gTouches`, we can generate a matrix of touching line segments; the points of contact are end nodes of all 14.25 m segments:

```
> t0 <- gTouches(stream_utm, byid = TRUE)
> any(t0)
```

```
[1] TRUE
```

Using a utility function `n.comp.nb` in **spdep** (the spatial neighbour object **nb** is discussed in Chap. 9), we can identify the connected graph components after converting the matrix of touches to a graph neighbour object, showing the number of components found:

```
> library(spdep)

> lw <- mat2listw(t0)
> nComp <- n.comp.nb(lw$neighbours)
> nComp$nc
```

```
[1] 21
```

The `gLineMerge` function lets us join the stream channel segments into continuous objects based on their membership of connected graphs. We see that the distribution of lengths is now more plausible, and can confirm that `gLength` yields the same values as `SpatialLinesLengths`:

```
> lns <- gLineMerge(stream_utm, id = as.character(nComp$comp.id))
> length(row.names(lns))
```

```
[1] 21
```

```
> summary(gLength(lns, byid = TRUE))

  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 14.25  326.30 1230.00  6394.00 2336.00 57750.00
```

```
> all.equal(SpatialLinesLengths(lns), unname(gLength(lns,
+   byid = TRUE)))
```

```
[1] TRUE
```

The stream channel lines will now be used with the enumeration district boundaries to add a variable tallying the total length in metres of channels by ED. We use the binary topological operation `gIntersection`, which here returns a `SpatialLines` object with `Lines` objects for each of the EDs that intersect with the water channels:

```
> GI <- gIntersection(lns, olinda_utm, byid = TRUE)
> class(GI)
```

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

> length(row.names(GI))

[1] 238
```

As not all EDs intersect with water channels, the IDs (row names) of the `SpatialLines` object will be used to ensure that the lengths per ED are assigned correctly. Pre-assigning a vector with zero in all elements, we split the ID strings into two, choosing the second value, and incrementing it by one. The incrementation is needed because the default IDs returned `readOGR` are zero-based. After calculating the lengths with `gLength` by ED containing channels, we sum the channel lengths per ED using `tapply`, aggregating on the ED IDs, and assign to the pre-assigned vector:

```
> res <- numeric(nrow(olinda_utm))
> head(row.names(GI))

[1] "10 1" "10 9" "10 11" "10 14" "10 21" "10 24"

> range(as.integer(row.names(olinda_utm)))

[1] 0 469

> rnGI <- as.integer(sapply(strsplit(row.names(GI), " "),
+   "[" , 2)) + 1
> length(rnGI) == length(unique(rnGI))

[1] FALSE

> lens <- gLength(GI, byid = TRUE)
> tlens <- tapply(lens, rnGI, sum)
> res[as.integer(names(tlens))] <- unname(tlens)
> olinda_utm$stream_len <- res
> summary(olinda_utm$stream_len)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    0.0     0.0     0.0  144.1  206.2 4205.0
```

In this case, the numbers of objects being handled by the binary topological operation is not very large, so little time is saved by building and querying a Sort-Tile-Recursive (STR) tree first (Leutenegger et al., 1997). The tree returned is a vector of the same length of the second argument, with integer vector components listing the IDs of stream `Lines` object bounding boxes (known as envelopes in GEOS) intersecting each `Polygons` object bounding box:

```
> tree <- gBinarySTRtreeQuery(lns, olinda_utm)
> table(sapply(tree, length))

0  1  2  3
1 223 232 14
```

In this case, the bounding boxes of the `Lines` objects span much of the city area, so all but one ED bounding box intersects one or more streams. Here, knowing which candidate objects to which to apply `gIntersection` saves little time; savings increase in proportion to the reduction of the degree of overlap between bounding boxes. Looping over the `Polygons` objects, we find their intersects with the subset of `Lines` objects found when querying the tree, but no longer using `byid=TRUE`, to aggregate the line segments in one step. As we see, the lengths found by ED are the same:

```
> res1 <- numeric(length = length(tree))
> for (i in seq(along = res1)) {
+   if (!is.null(tree[[i]])) {
+     gi <- gIntersection(lns[tree[[i]]], olinda_utm[i,
+     ])
+     res1[i] <- ifelse(is.null(gi), 0, gLength(gi))
+   }
+ }
> all.equal(olinda_utm$stream_len, res1)

[1] TRUE
```

Next we construct a 50 m buffer on either side the stream channels using the `gBuffer` function, with default round cap and join styles where features end or join; these defaults affect the shape of the output object, as do the numbers of points used to approximate quarter circles. The buffer object is formed as a single `Polygons` object, simply including all areas within 50 m of a stream channel:

```
> buf50m <- gBuffer(lns, width = 50)
> length(slot(buf50m, "polygons"))

[1] 1
```

Using the procedure described above, we can assign the areas within the buffer by ID to the enumeration districts by intersection with `gIntersection` and `gArea`, and then calculate the proportion of the district area within 50 m of a water channel:

```
> GI1 <- gIntersection(buf50m, olinda_utm, byid = TRUE)
> res <- numeric(length(slot(olinda_utm, "polygons")))
> head(row.names(GI1))

[1] "buffer 1" "buffer 6" "buffer 9" "buffer 10" "buffer 11"
[6] "buffer 14"

> rnGI <- as.integer(sapply(strsplit(row.names(GI1), " "),
+   "[", 2)) + 1
> length(rnGI) == length(unique(rnGI))

[1] TRUE

> res[rnGI] <- gArea(GI1, byid = TRUE)
> olinda_utm$buf_area <- res
> olinda_utm$prop_50m <- olinda_utm$buf_area/olinda_utm$area
```

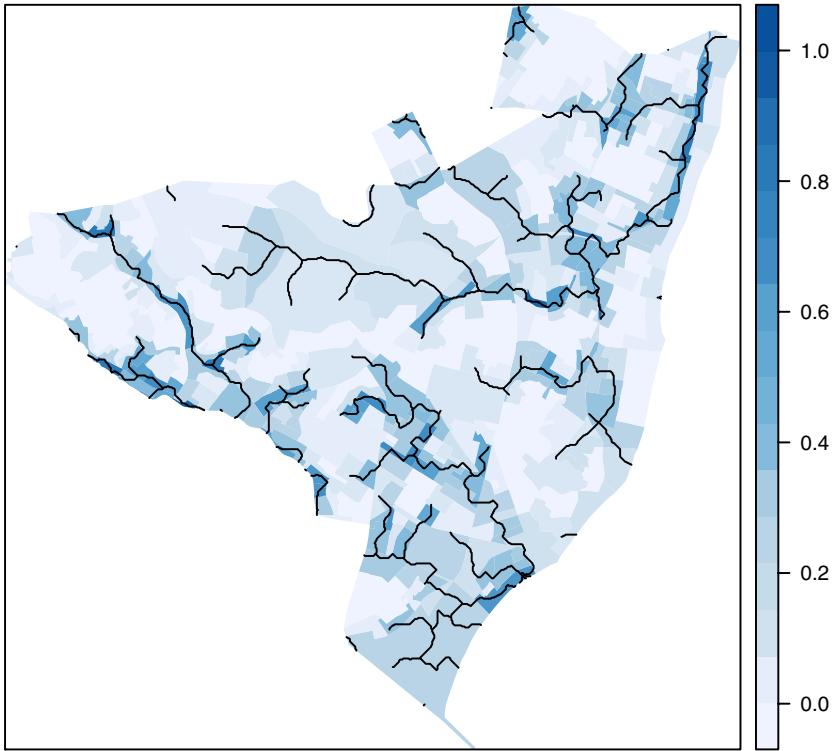


Fig. 5.4 Proportions of enumeration district areas within 50 m of stream channels, and channel locations within Olinda

To complete data handling before visualization, we may clip the stream channels to the city bounds of Olinda, matching the extent of the enumeration districts:

```
> stream_inside <- gIntersection(lns, bounds)
```

Fig. 5.4 shows the proportions of enumeration district areas within 50 m of stream channels, and the channel locations within the city. It should be recalled that the channels we have been using are derived from a smoothed digital elevation model, so do not necessarily reflect the actual channels that could be found by surveying or photogrammetry.

5.3 Map Overlay or Spatial Join

Numerical overlay or spatial join retrieves the indexes or attributes of one spatial object at the locations of another. In particular, the overlay method

```
> over(x, y)
```

or equivalently,

```
> x %over% y
```

retrieves,

- I In case `y` has no attributes, the indexes of the `y` corresponding to each feature of `x`, or `NA` in case of no correspondence;
- II In case `y` has attributes, a `data.frame` with the attributes of `y` corresponding to the locations of `x`, or an `NA` record in case of no correspondence.

Correspondence means that two features spatially intersect (touch, overlap, cover, includes, etc.). The length of the vector returned in case I and the number of rows returned in case II equals `length(x)`. This is identical to the *left outer join* in SQL, where the matching is based on spatial intersection.

The implementation is provided partly by `sp`, partly by `rgeos`. Table 5.1 (Pebesma, 2012a) provides some details. In particular, in some cases `SpatialPixels` or `SpatialGrid` objects are converted to points for a spatial match, in other cases they are treated as cells (points in grid cells), in some cases they are converted to polygons (lines intersecting grid cells). To override these defaults, objects can be explicitly converted to points or polygons prior to calling `over`.

	y: Points	y: Lines	y: Polygons	y: Pixels	y: Grid
x: Points	s	r	s	s	s
x: Lines	r	r	r	r:y	r:y
x: Polygons	s	r	r	s:y	s:y
x: Pixels	s:x	r:x	s:x	s:x	s:x
x: Grid	s:x	r:x	s:x	s:x	s:x

Table 5.1 `over` methods implemented for different `x` and `y` arguments. `s`: provided by `sp`; `r`: provided by `rgeos`. `s:x` or `s:y` indicates that the `x` or `y` argument is converted to grid cell center *points*; `r:x` or `r:y` indicates grids or pixels are converted to polygons

A typical use of `over` is to select features of one `Spatial` object that are on or inside another. Suppose we want to select all the points in `meuse` falling in the `meuse.grid` grid, we could do this by:

```
> sel = over(meuse, as(meuse.grid, "SpatialPixels"))
> meuse = meuse[!is.na(sel), ]
```

as this happens really frequently, the selection syntax for features was extended such that it understands:

```
> meuse = meuse[meuse.grid, ]
```


as doing exactly this: select those points of `meuse` that coincide with (fall within) grid cells of `meuse.grid`.

In the case where a feature (e.g. a polygon) of `x` matches multiple features (e.g. points) in `y`, `over(x, y)` returns an arbitrary (the first) match. To obtain *all* matches, one can specify `over(x, y, returnList=TRUE)` which then returns a `list` of length `length(x)`, with each list element an integer vector with matching `y` indexes or a `data.frame` with all matching attribute values. As the goal will typically be to merge these attribute values in some way, one could pass a function, e.g. `mean`, as in `over(x, y, fn = mean)`, in which case all attribute records matching to each feature in `x` are aggregated using this function. Binding these aggregated attributes to the original `x` object is called *spatial aggregation*.

5.3.1 Spatial Aggregation

Spatial aggregation involves two elements: a grouping predicate and an aggregation function. As we have seen in the previous section, grouping predicates and grouped sets of attributes are obtained by using `over`. Aggregation functions can be any R function returning a single value. Consider the case where we want to obtain maximum measured `zinc` concentrations from the `meuse` data set over 400 m grid cells:

```
> gt <- GridTopology(c(178480, 329640), c(400, 400), c(8,
+   11))
> coarseGrid <- SpatialGrid(gt, proj4string(meuse))
> agg <- aggregate(meuse[c("zinc", "lead")], coarseGrid,
+   max)
```

the results of which are shown in Fig. 5.5. It should be noted that these methods for aggregation match do not account for partial overlapping of polygons or grid cells: matching is true or false, and no weighting takes place.

Further aggregation methods are provided by **raster**, in particular the function `rasterize` converts points, lines and polygons values to raster cells, and methods `aggregate`, `disaggregate` and `resample` allow one to go from one grid cell resolution to another.

In the following examples, we will be querying `SpatialPixel` objects with `SpatialPolygons` objects representing the Olinda enumeration districts.

Returning to the Olinda use case and illustrating how to combine polygon and grid data using the `over` methods, we will construct some variables to which it may be applied. The Normalized Difference Vegetation Index (NDVI) is a ratio of sums and differences of Thematic Mapper visible red and near-infrared bands, and can be interpreted to indicate the relative presence of live green plant canopies, as usual qualified by sub-pixel variability.

```
> TM$ndvi <- (TM$TM4 - TM$TM3)/(TM$TM4 + TM$TM3)
```

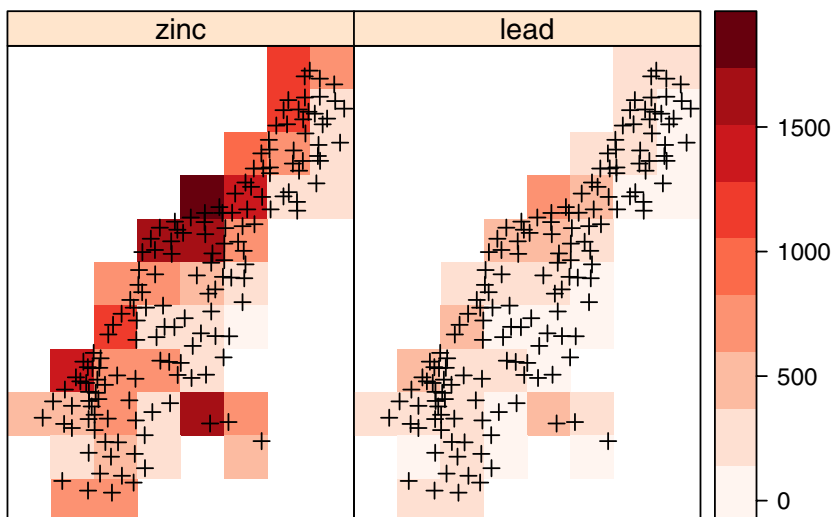


Fig. 5.5 Maximum zinc and lead concentrations measured within 400 m grid cells

addition In addition, we can calculate the principal components of the six Thematic Mapper bands available at 28.5m resolution, subsetting spatially to the city limits. Figure 5.6 shows the spatial distribution of the first two principal components.

```
> TM0 <- as(TM, "SpatialPixelsDataFrame")
> TM1 <- TM0[bounds, ]
> PC <- prcomp(as(TM1, "data.frame"),[, 1:6], center = TRUE,
+   scale. = TRUE)
> PCout <- predict(PC)
> TM1$PC1 <- PCout[, 1]
> TM1$PC2 <- PCout[, 2]
> TM1$PC3 <- PCout[, 3]
```

Recalling that when the second argument object has attributes, `over` methods return a `data.frame` object, we can add aggregated principal components values taken from the cell values falling into each enumeration district, then binding that `data.frame` to the input `SpatialPolygonsDataFrame`:

```
> o_mean <- over(olinda_utm, TM1[, c("PC1", "PC2", "PC3")])
> str(o_mean)

'data.frame':      470 obs. of  3 variables:
 $ PC1: num 1.36 0.28 ...
 $ PC2: num -0.432 0.699 ...
 $ PC3: num 0.0755 0.1326 ...

> row.names(o_mean) <- row.names(olinda_utm)
> olinda_utmA <- spCbind(olinda_utm, o_mean)
```

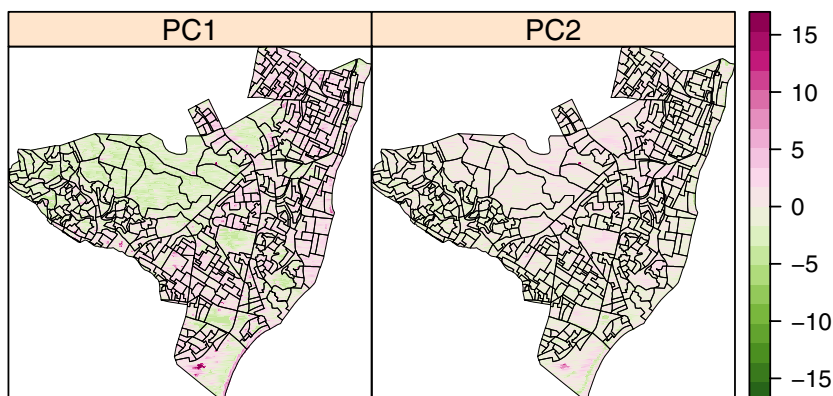


Fig. 5.6 Spatial distribution of the first two principal components, TM bands 1–5, 7, Olinda

The default function used to reduce the set of cell values to a scalar is `mean`, but others may be passed through the `fn` argument, here `median`:

```
> o_median <- over(olinda_utm, TM1[, c("PC1", "PC2", "PC3")],
+   fn = median)
> row.names(o_median) <- row.names(olinda_utmA)
> names(o_median) <- paste(names(o_median), "med", sep = "_")
> olinda_utmB <- spCbind(olinda_utmA, o_median)
> TM1$count <- 1
> o_count <- over(olinda_utm, TM1[, "count"], fn = sum)
> olinda_utmB$count <- o_count$count
```

Choosing relevant functions to aggregate the values by query feature is important, as the distributional shapes and centres in each aggregate do vary, and also reflect support issues, as we may see by comparing means and medians, and the counts of cells:

```
> summary(olinda_utmB[, grep("^PC/count", names(olinda_utmB))])
```

Object of class `SpatialPolygonsDataFrame`

Coordinates:

min max

x 288712.2 298526

y 9110320.2 9120257

Is projected: TRUE

proj4string :

[+init=epsg:31985 +proj=utm +zone=25 +south +ellps=GRS80

+towgs84=0,0,0,0,0,0,0 +units=m +no_defs]

Data attributes:

PC1	PC2	PC3
Min. : -3.7479	Min. : -3.5494	Min. : -3.28964
1st Qu.: -0.7436	1st Qu.: -0.7722	1st Qu.: -0.26779
Median : 0.7213	Median : -0.3086	Median : 0.13062
Mean : 0.4817	Mean : -0.2122	Mean : 0.08731

3rd Qu.: 1.6019	3rd Qu.: 0.2772	3rd Qu.: 0.52835
Max. : 6.3413	Max. : 3.1913	Max. : 2.11267
PC1_med	PC2_med	PC3_med
Min. : -2.7084	Min. : -1.5478	Min. : -1.20463
1st Qu.: -0.2182	1st Qu.: -0.6229	1st Qu.: -0.03047
Median : 0.7264	Median : -0.2799	Median : 0.18651
Mean : 0.5387	Mean : -0.2419	Mean : 0.18797
3rd Qu.: 1.5250	3rd Qu.: 0.1260	3rd Qu.: 0.44002
Max. : 3.5291	Max. : 1.5978	Max. : 1.15698
count		
Min. : 4.0		
1st Qu.: 48.0		
Median : 73.0		
Mean : 109.1		
3rd Qu.: 114.8		
Max. : 1851.0		

Finally, we may add median elevation and median NDVI values by enumeration district from `SpatialGridDataFrame` objects in the same way, where one median elevation is NA, because at 90 m resolution, no cell centre point falls into that enumeration district:

```
> o_dem_median <- over(olinda_utm, dem, fn = median)
> olinda_utmB$dem_median <- o_dem_median$band1
> summary(olinda_utmB$dem_median)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.   NA's
   4.00   8.50   11.50   19.11  27.00   73.00     1

> o_ndvi_median <- over(olinda_utm, TM1["ndvi"], fn = median)
> olinda_utmB$ndvi_median <- o_ndvi_median$ndvi
```

5.3.2 Using the raster Package for Extract Operations

It is possible to use `extract` methods from the **raster** package to aggregate queried values; note that the arguments are reversed in order compared to `over` methods in **sp**.

```
> library(raster)

raster version 2.1-16 (14-March-2013)
```

We convert our `SpatialPixelsDataFrame` object to a `RasterStack` object, and obtain when querying with a `SpatialPolygons` object, a list of numeric matrices with the values of all variables in rasters in the `RasterStack` object, for all the cells with centres falling in each `Polygons` object:

```
> TMrS <- stack(TM1)
> e1 <- extract(TMrS, as(olinda_utm, "SpatialPolygons"))
```

If we convert the single variable `SpatialGridDataFrame` object containing the 90 m digital elevation model to a `RasterLayer` object, we obtain a list of numeric vectors of different lengths, but, as we saw above, one enumeration district contains no cell centres at this resolution:

```
> e2 <- extract(raster(dem), as(olinda_utm, "SpatialPolygons"))
> table(sapply(e2, is.null))
FALSE  TRUE
  469     1
```

As we can see by comparison, `extract` methods from the **raster** package yield the same results as `over` methods in **sp** for 28.5 m resolution counts of cell centres, and NDVI medians, and 90 m resolution elevations by `Polygons` object, once again with care needed to handle the `Polygons` object not including any 90 m cell centre:

```
> all.equal(sapply(e1, nrow), olinda_utmB$count)
[1] TRUE

> all.equal(sapply(e1, function(x) median(x[, "ndvi"])),
+           olinda_utmB$ndvi_median)
[1] TRUE

> med <- sapply(e2, function(x) ifelse(is.null(x), as.numeric(NA),
+           median(x, na.rm = TRUE)))
> all.equal(med, olinda_utmB$dem_median)
[1] TRUE
```

5.3.3 Spatial Sampling

One way of trying to get control over data in a research setting like the one described might be to sample points from the total study area, to be able to examine whether the observed phenomena seem to be associated with particular ranges of values of the supposed environmental ‘drivers’, or to survey ‘ground truth’ values. Sample design is not typically paid much attention in applied spatial data analysis, very often for practical reasons, for example the need to handle incoming data as they flow in, rather than being able to choose which data to use. In the case of veterinary epidemiology, it is not easy to impose clear control because of the time pressure to offer policy advice. Schemes for spatial sampling have been given in the literature, for example by Ripley (1981, pp. 19–27), and they are available in **sp** using generic method `spsample`. Five sampling schemes are available: “**random**”, which places the points at random within the sampling area; “**regular**”, termed a *centric systematic sample* by Ripley and for which the grid offset

can be set, and "stratified" and "nonaligned", which are implemented as variations on the "regular" scheme – "stratified" samples one point at random in each cell, and "nonaligned" is a systematic masked scheme using combinations of random x and y to yield a single coordinate in each cell. The fifth scheme samples on a hexagonal lattice. The spatial data object passed to the `spsample` method can be simply a `Spatial` object, in which case sampling is carried out within its bounding box. It can be a line object, when samples are taken along the line or lines. More typically, it is a polygon object or a grid object, providing an observation window defining the study area or areas.

Returning again to the Olinda data set, we take three samples of about 1,000 at random within the city bounds represented as a polygon, then subsetting the digital elevation raster to the city limits and converting to `SpatialPixelsDataFrame` representation, a random sample and a regular sample within that object:

```
> set.seed(9876)
> p_r <- spsample(bounds, 1000, type = "random")
> length(p_r)

[1] 1000

> dem <- dem[bounds, ]
> dem_sp <- as(dem, "SpatialPixelsDataFrame")
> g_r <- spsample(dem_sp, 1000, type = "random")
> length(g_r)

[1] 979

> g_rg <- spsample(dem_sp, 1000, type = "regular")
> length(g_rg)

[1] 1003
```

As we see, the numbers of samples output may not match exactly the number requested. We extract the elevation values at the sample points for display as empirical cumulative distribution functions in Fig. 5.7; the figure also shows the distributions of the sample points.

```
> p_r_dem <- over(p_r, dem)
> g_r_dem <- over(g_r, dem)
> g_rg_dem <- over(g_rg, dem)
```

We can also tabulate the values read from the elevation raster at the sample points:

```
> tab <- rbind(polygon_random = c(fivenum(p_r_dem$band1),
+   nrow(coordinates(p_r_dem))), grid_random = c(fivenum(g_r_dem$band1),
+   nrow(coordinates(g_r_dem))), grid_regular = c(fivenum(g_rg_dem$band1),
+   nrow(coordinates(g_rg_dem))))
> colnames(tab) <- c("minimum", "lower-hinge", "median",
+   "upper-hinge", "maximum", "n")
> tab
```

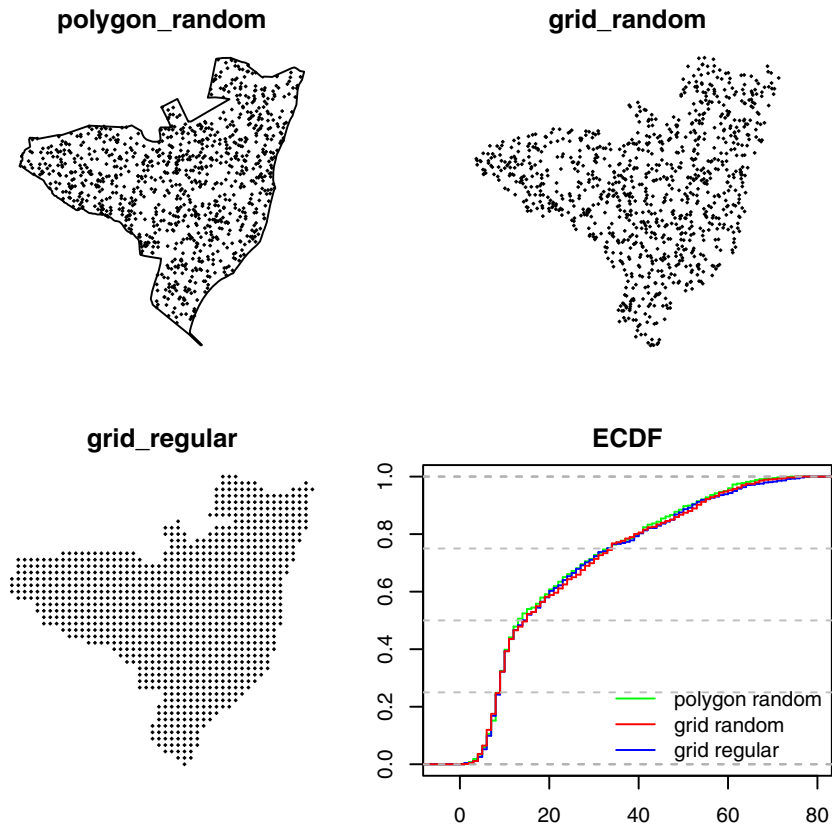


Fig. 5.7 Spatial distributions of sample points for three sampling methods, Olinda; Empirical cumulative distribution functions of elevations measured at sample points from the 90 m resolution elevation raster

	minimum	lower-hinge	median	upper-hinge	maximum	n
<code>polygon_random</code>	1	9	13	33	74	1000
<code>grid_random</code>	1	9	14	33	77	979
<code>grid_regular</code>	1	9	15	34	75	1003

Finally, we may cross-check using the random sample within the city limits polygon to ensure that the point-in-polygon outcomes using the `over` method and the `rgeos` binary predicate `gContains` agree. The `over` method returns an integer vector, showing which `Polygons` object each sample point belongs to. The `gContains` function with `byid=TRUE` returns a logical matrix showing the same, again provided that no point falls into two `Polygons` objects. Tabulation of the output from collapsing this matrix shows that there are no such exceptions (which may occur when input features overlap), and the two approaches yield the same result.

```

> o_sp <- as(olinda_utm, "SpatialPolygons")
> whichPoly <- over(p_r, o_sp)
> whichPoly1 <- gContains(o_sp, p_r, byid = TRUE)
> whichPoly1a <- apply(unname(whichPoly1), 1, which)
> table(sapply(whichPoly1a, length))

 1
1000

> all.equal(whichPoly, whichPoly1a)

[1] TRUE

```

5.4 Auxiliary Functions

New functions and methods are added to **maptools** quite frequently, often following suggestions and discussions on the R-sig-geo mailing list mentioned in Chap. 1. When positions are represented by geographical coordinates, it is often useful to be able to find the azimuth between them. The **gzAzimuth** function is a simple implementation of standard algorithms for this purpose, and gives azimuths calculated on the sphere between a matrix of points and a single point.⁵ The **gcDestination** function returns the geographical coordinates of points at a given distance and bearing from given starting points.

A set of methods for matrices or **SpatialPoints** objects in geographical coordinates has been contributed to give timings for sunrise, sunset, and other solar measures for dates given as **POSIXct** objects:

```

> hels <- matrix(c(24.97, 60.17), nrow = 1)
> p4s <- CRS("+proj=longlat +datum=WGS84")
> Hels <- SpatialPoints(hels, proj4string = p4s)
> d041224 <- as.POSIXct("2004-12-24", tz = "EET")
> sunrise(Hels, d041224, direction = "sunrise", POSIXct.out = TRUE)

      day_frac      time
newlon 0.3924249 2004-12-24 09:25:05

```

Finally, **elide** methods have been provided for translating, rotating, and disguising coordinate positions in **sp** vector classes such as **SpatialPoints**. The features can be shifted in two dimensions, scaled such that the longest dimension is scaled $[0, 1]$, flipped, reflected, and rotated, if desired in relation to the bounding box of a different **Spatial** object. The methods can be used for standardising displays, for example in point pattern analysis, or for obfuscating position to meet in part privacy considerations. Since obscuring position was a reason for providing the methods, they have been given a suitably obscure name.

⁵ The function is based with permission on work by S. Abdali: The Correct Qibla, <http://patriot.net/users/abdali/ftp/qibla.pdf>.

The methods discussed in this chapter are intended to provide ways for manipulating spatial objects to help in structuring analytical approaches to support problems amongst others. These are not the only ways to organise spatial data, do try to make it easier to concentrate on exploring and analysing the data, rather than dealing with the intricacies of particular representations peculiar to specific software or data providers.

Chapter 6

Spatio-Temporal Data

6.1 Introduction

Observations refer to properties or qualities at particular locations in space and moments in time. In many cases, locations and/or times are not taken into account explicitly, because they are not relevant. In other cases, they are. Most of this book addresses the case where spatial location matters, and temporal variation is not present or ignored. Texts on time series analysis mostly do the reverse. This chapter will address first steps in handling spatio-temporal data, and analysing them.

The fact that the areas of spatial statistics and time series analysis have individually developed much further than that of spatio-temporal statistics may have several reasons. Schabenberger and Gotway (2005) argue that analysis of spatio-temporal data often happens *conditionally*, meaning that either first the spatial aspect is analysed, after which the temporal aspects are analysed, or reversed, but not in a joint, integral modelling approach, where space and time are not separated. As a possible reason they mention the lack of good software to handle, import, export, display and analyse such data.

The R package **spacetime** (Pebesma, 2012b), on CRAN since 2010, is a start to fill this gap, and similar to **sp** has the ambition to provide easy means for handling and communicating spatio-temporal data. Special purpose packages for e.g. visualisation (**plotKML**) or geostatistical analysis (**gstat**) are then relieved from some burden of handling data.

6.2 Types of Spatio-Temporal Data

Before we try to represent spatio-temporal data, we will spend some time on defining properties of spatio-temporal data that are relevant.

6.2.1 *Spatial Point or Area, Time Instance or Interval*

Both in space and time, it makes sense to think about whether an observation is related to an atom, a point in space and instance in time, or to a larger entity (a line, area, or volume in space; a period in time or time interval). Although possibly *all* observations are related to a larger volume and non-zero period because the act of observation takes space and time, in many cases this volume and/or period is so small compared to the extent of the data, that it is convenient to consider it to be atomic. Examples where “zero” volume or *point* data are assumed are soil samples, rain gauge measurements, or earth quake epicentres. Examples where “zero” time interval width or *moment* data are assumed include lightning, accidents, birth and death.

6.2.2 *Are Space and Time of Primary Interest?*

Locations (points, areas) or times (moments, intervals) at which something is recorded can be of primary interest, or rather a matter of convenience. Examples where points and/or times are of primary interest are: disease cases, earth quakes, and lightning. An example where they are not of primary interest are measurements of water temperature, or air quality.

The first type concerns objects and/or events, and their registered locations and times indicate where and when they were or occurred, but also that at all other, remaining locations in the observation window they were not. Spatio-temporal point pattern analysis methods are used to analyse such data.

The second type concerns *fields*, which can in principle be measured at arbitrary locations, but happen to be measured at a limited number of locations. Here, lack of measurement at some time/location does not indicate anything about the field value. Typically, for the first type the *count* or *sum* operation leads to a meaningful value, where for the second type it is not.

6.2.3 *Regularity of Space-Time Layouts*

Both spatial data as well as time series often have some regularity: spatial grids or images have a regular layout of grid cells or pixels, many time series data have a regular time discretisation.

Spatio-temporal data can have a regular space-time layout, meaning that for each of a fixed set of n locations, the same time series of m observations is available. In that case, the space-time locations form a regular $n \times m$ layout. A regular space-time layout neither implies spatial locations are regular – they can be of any type (points, lines, polygons, grids) – nor does it imply that times are regular.

When locations and times of the data set are of primary interest, they will not be regular. For many reasons, such data are often aggregated to a regular space-time layout before they are analysed. Examples may be the earth quakes with magnitude above a certain threshold, counted over cells on a regular grid and over a particular time period. In other cases, data have been aggregated for privacy reasons, for instance in case of disease data (e.g. disease cases aggregated to administrative regions, per year).

When locations and times are not of primary interest, i.e. where field properties such as air temperature are observed, the typical case is that sensors are kept at a fixed location and measure over predefined time intervals (e.g. an air quality parameter is recorded every half hour, temperature is recorded as average value per minute). A reason to deviate from this may be that network operation costs increase with measurement frequency (e.g. wireless networks, or networks that communicate through SMS). If this is not the case, the constant frequency is the common case, leading to regular space-time layouts.

Aggregating spatio-temporal data with an irregular layout to a regular layout, or aggregating data with a high resolution to a lower resolution often helps us to understand these data or is required to visualise it.

6.2.4 Do Objects Change Location?

A second property is whether observations in space and time are connected, because an object persists, and may for instance be moving. Movement may be continuous (persons, cars), or sudden (capital of a country).

Güting and Schneider (2005) distinguish a useful set of different data types for spatio-temporal data. In particular, they define for point features

- a** An event *without* temporal duration (time is an instant), e.g., an accident, a lightning, a birth, a death;
- b** An events *with* a temporal duration but no movement, e.g., a tree, a (point in the) capital of a country, someone's home address;
- c** A *moving* point, e.g., the trajectory of a persons, a car or a bird.

Güting and Schneider (2005) obtain 10 different types by adding the *set of* form for **a** and **b**, and doubling the resulting set of 5 by distinguishing area from point features. It is not clear why **c** did not get a plural form, which would have resulted in 12 types. Also, a capital may move (sudden), but a tree does not (by itself). Further, more complicated types could be formed by introducing relations and evolution: rivers meander and branch, countries can split or merge, persons meet other persons, and even multiply.

6.3 Classes in **spacetime**

A set of classes for spatio-temporal data is implemented in **spacetime** (Fig.6.1). In particular, this set supports:

- Time represented by time *intervals* or time *instances*;
- Objects containing single instances, or sets of these instances;
- Regular and irregular layouts of space-time combinations;
- A sparse regular layout for efficiency reasons;
- Simple (sets of) trajectories;
- Objects consisting of space/time points only, or with additional properties (attributes).
- Attributes varying over space and time, or only over space, or only over time.

The set of classes is shown in Fig. 6.1. All classes derive from an abstract class

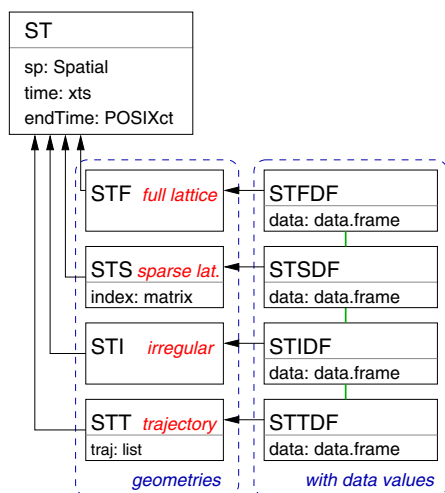


Fig. 6.1 Classes for spatio-temporal data in package **spacetime**. Arrows denote inheritance, lower side of boxes list slot name and type, vertical lines indicate possible coercion (both ways)

ST, a template for specific classes holding data. Spatial locations derive from **Spatial**, and hence may reflect points, pixels, lines or polygons, but not a mixture.¹

¹ **SpatialGrid** objects will be converted to **SpatialPixels**, because only **SpatialGrid** objects follow a different subsetting convention where **x[i,]** does not refer to grid cell **i**, but to grid row **i**.

The `sp` data slot may be an object of any `Spatial*DataFrame` class,² to hold attribute data that does not vary over time. Slot `time` holds the (start) times, and possibly attribute data that do not vary over space. It should be of class `xts`, which is briefly discussed in the next section. Slot `endTime` of class `POSIXct` indicates end times, and must be a vector having length identical to the number of records in slot `time`. If all end times equal the times in `time`, time is considered as instant time, otherwise time is considered to consist of time intervals.

The simplest structure is the `STI` for irregular spatio-temporal layouts, which is simply an unordered sequence of space/time locations. For this object, the number of records in the `sp` slot and the `time` slot needs to be identical, and identical to the number of records in the `data` slot of an `STIDF` object.

Regular space-time layouts are represented by objects of class `STF`. If an `STF` object has n locations in slot `sp` and m times in slot `time`, it means that we have mn observations, with all time instances (or intervals) replicated for each spatial location, and vice versa. The attribute corresponding to location i and time j is record (row) $(j - 1) * n + i$, spatial locations cycling first. This means that the full space-time lattice is filled.

Incomplete, or sparse spatio-temporal objects with regular layout can be represented by class `STS`. Here, the `index` slot is a $n \times 2$ matrix with in row i the indices to the spatial and to the temporal reference in the first and second column of the i -th observation.

Objects of class `STT` are meant to hold sets of trajectories. The actual trajectories are held in a list `traj`, where each list element is an object of class `STI`, or `STIDF` if attributes are present. The `sp` and `time` slot only contain the extreme values to be able to compute a space-time bounding box for the whole object. The implementation for the trajectories was inspired by the `ltraj` class of package `adehabitatLT`.

Besides `STT` objects, which represent objects that move over time, `STF`, `STS` and `STI` objects do not record whether they represent field variables or objects/events. In particular the irregular `STIDF` could equally well reflect a marked point pattern (earth quake location/times with magnitudes) or a field variable (temperature readings over space and time).

A set of methods for these classes is shown in Table 6.1. The next sections will discuss and illustrate those that are non-obvious.

6.4 Handling Time Series Data with `xts`

Two very powerful R packages are available for the handling and analysis of time series data: `zoo` and `xts`. The `time` slot of `ST` objects is of class `xts`, from package `xts`, which extends `zoo` objects from package `zoo`. Objects of class `xts` can be constructed using any of the following time classes in R: `Date`,

² Except for `SpatialGridDataFrame`.

Method	What it does
<code>stConstruct</code>	Creates STFDF or STIDF objects from single or multiple tables, time series, and spatial objects
<code>[[, \$, \$<-</code>	Select or replace data values
<code>[</code>	Select spatial and/or temporal subsets, and/or data variables
<code>as</code>	Coerce to other spatio-temporal objects, <code>xts</code> , <code>Spatial</code> , <code>matrix</code> , or <code>data.frame</code>
<code>stplot</code>	Create spatio-temporal plots, see Sect. 6.8
<code>over</code>	Overlay: retrieve index or data values of one object at the locations and times of another; see Sect. 6.7
<code>aggregate</code>	Aggregate data values over particular spatial, temporal, or spatio-temporal domains; see Sect. 6.7
<code>na.locf</code>	Fill NA value using last measured in time series
<code>na.spline,</code> <code>na.approx</code>	Replace NA with interpolated values

Table 6.1 Methods for spatio-temporal data in package `spacetime`

`POSIXct`, `chron`, `yearmon`, and `yearqtr`. Internally it represents time by a `POSIXct` index.

Objects of class `xts` have powerful selection mechanisms, e.g. by using ISO 8601³ notation of time instance or intervals. This allows simple constructions as `x["2012"]` to select observations from the full year of 2012, or `"2012-05"` to denote the full month of May in 2012. As these objects extend `zoo` objects, powerful aggregation methods for these are reused, e.g. to go from hourly to monthly, or from daily to quarterly data using arbitrary aggregation functions.

Neither `zoo` nor `xts` objects have a notion of the difference between time instances or time intervals: they seem to serve their needs by keeping this information implicit. For spatio-temporal data, it turned out to be easier to make this explicit, and hence a `POSIXct` slot `endTime` is present in all ST objects.

For regular space-time layouts (STF, STS), the default assumption is that time reflects time intervals extending to the next observation in time. For all other layouts time is assumed to reflect instances when no `endTime` is specified.

6.5 Construction of ST Objects

The data used here are from the North American Breeding Bird Survey, for the Eurasian Collared Dove, *Streptopelia decaocto*, for the years 1986–2003, published by Cressie and Wikle (2011).⁴

³ http://en.wikipedia.org/wiki/ISO_8601

⁴ Downloaded from the book web site, ftp://ftp.wiley.com/public/sci_tech_med/spatio_temporal_data/

First we will import the longitude and latitudes of the observation locations.

```
> ecd.ll <- as.matrix(read.table("ECDovelatlon.dat", header = FALSE))
> library(sp)
> ecd.ll <- SpatialPoints(ecd.ll[, c(2, 1)])
> proj4string(ecd.ll) <- CRS("+proj=longlat +datum=WGS84")
```

Next, we will specify the times, and convert them into `Date` format:

```
> library(xts)
> library(spacetime)
> ecd.years <- 1986:2003
> ecd.y <- as.Date(paste(ecd.years, "-01-01", sep = ""),
+   "%Y-%m-%d")
```

Finally, we read the full data set, set missing values, and create the `STFDF` object:

```
> ecd <- read.table("ECDoveBBS1986_2003.dat", header = FALSE)
> ecd[ecd == -1] <- NA
> ecd.st <- STFDF(ecd.ll, ecd.y, data.frame(counts = as.vector
+   (as.matrix(ecd))))
> dim(ecd.st)
```

space	time variables	
253	18	1

which indicates that this data set has 253 spatial locations, 18 time replicates at each location, and one attribute variable.

Functions `STF`, `STFDF`, `STIDF` etc. are used to create the objects with the same name, from their components. An alternative way to construct the objects from `data.frame` tables is by using `stConstruct`, as in

```
> ecd.st2 <- stConstruct(ecd, ecd.ll, list(counts = names(ecd)),
+   TimeObj = ecd.y, interval = TRUE)
> all.equal(ecd.st2, ecd.st)
```

```
[1] TRUE
```

where the first argument can be the data table either

- In *long form*, where all attribute values are in one column,
- In *space-wide form*, where different columns represent time series for different locations, or
- In *time-wide form*, where different columns represent different moments or periods in time.

The help page of `stConstruct` contains examples for each of these.

6.6 Selection, Addition, and Replacement of Attributes

Subsets of STFDF, STSDF and STIDF objects are obtained by the syntax `obj[space, time, attr, drop = TRUE]` where `space`, `time` and `attr` can be numeric or logical vectors that indicate which locations, which times, and which attribute variables need to be selected. In addition,

- `space` can be a character vector, to select the spatial locations with matching `row.names`,
- `space` can be a `Spatial` object, to select the spatial locations that intersect with this object,
- `time` can be a character string denoting time instance or time interval in ISO 8601 notation, to select a time instance or interval,
- In case `drop = FALSE`, selection returns `ST*DF` objects, if `TRUE` a `Spatial` object is returned in case of a single time step and an `xts` object is returned in case of a single spatial location,
- Trajectories are selected or deselected as complete entities when part of them matches the criteria.

As an example, we create a `SpatialPolygons` object with the state of Florida, from the data available in `maps`:

```
> library(maps)
> m <- map("state", "florida", fill = TRUE, plot = FALSE)
> library(maptools)
> FL <- map2SpatialPolygons(m, "FL")
> proj4string(FL) <- proj4string(ecd.st)
```

we can use this object to select all the data points inside the state Florida, as well as within the time period 1998–2003, and optionally select a variable (of which we have only one):

```
> dim(ecd.st[FL, ])

      space      time variables
      58        18           1

> dim(ecd.st[, "1998::2003"])

      space      time variables
      253        6           1

> dim(ecd.st[, , "counts"])

      space      time variables
      253        18           1

> dim(ecd.st[FL, "1998::2003", "counts"])

      space      time variables
      58         6           1
```

where the commas need to be placed as in the examples here. The vector of data values is obtained by using double braces or `$`:

```

> mode(ecd.st[[1]])
[1] "numeric"
> length(ecd.st[[1]])
[1] 4554
> length(ecd.st[["counts"]])
[1] 4554
> length(ecd.st$counts)
[1] 4554

```

and variables can be added (or replaced) by assigning them, as in

```

> ecd.st$sqrtcounts <- sqrt(ecd.st$counts)

```

6.7 Overlay and Aggregation

Similar to the `over` method for `Spatial` objects (Sect. 5.3), spatio-temporal objects have an `over` method to find matching space-time geometries, or retrieve attribute values at these matching geometries. In

```

> over(x, y)

```

at the spatio-temporal locations of `x`, the index of `y` is retrieved if `y` has no attribute values, or else the `data.frame` with the attribute values of `y` at these locations is retrieved.

As an example, we want total Eurasian Collared Dove counts over the state of Florida, for 2-year periods. We can construct a target space-time geometry by

```

> bb <- STF(FL, ecd.y[c(4, 6, 8, 10, 12)])

```

Next, the naive approach would be to carry out

```

> over(bb, ecd.st)
  counts sqrtcounts
1      0          0
2      0          0
3      0          0
4      0          0
5      0          0

```

but as `bb` contains many of the points in `ecd.st`, only the first one is returned, which is little helpful. The full set would be obtained in the form of a list of `data.frames` when argument `returnList = TRUE` were added, but we can process this list in the same step, as that is usually wanted anyway:

```
> over(bb, ecd.st, fn = sum, na.rm = TRUE)
```

	counts	sqrtcounts
1	3	1.732051
2	5	4.414214
3	92	37.404228
4	176	64.018950
5	860	202.209044

Then, we can assign this attribute table to create a new STFDF:

```
> bb.counts <- new("STFDF", bb, data = over(bb, ecd.st,
+     fn = sum, na.rm = TRUE))
```

We have now aggregated the data, and we can do the same thing using the simpler syntax of method `aggregate`:

```
> aggregate(ecd.st, bb, sum, na.rm = TRUE)
```

	counts	sqrtcounts	timeIndex
1989-01-01	3	1.732051	1
1991-01-01	5	4.414214	2
1993-01-01	92	37.404228	3
1995-01-01	176	64.018950	4
1997-01-01	860	202.209044	5

Here, the first argument `ecd.st` is aggregated using the aggregation (or grouping) predicate of the second argument `bb`, and the aggregation function `sum` of the third argument. Further arguments (`na.rm = TRUE`) are passed on to this aggregation function.

The `aggregate` command returns a time series object of class `xts` in this case, because we aggregate over one a single spatial geometry. By default it returns a simplified structure (time series, or `Spatial` object) if possible, unless `simplify = FALSE` is specified.

For the `aggregate` methods for `ST*DF` objects, the aggregation function takes a vector as argument. The aggregation predicate can be

- An object deriving from `ST`, in which case the two spatio-temporal geometries are matched,
- An object deriving from `Spatial`, in which case the spatial geometries are matched,
- Character, indicating an aggregation period such as `"1 week"` or `"2 years"`,
- A function, which, when applied to the time index of the object returns the time classes.

For instance, the bird data aggregated (averaged) over 5-year periods when omitting missing counts, which is shown in Fig. 6.2, is obtained by

```
> ecd.5y <- aggregate(ecd.st, "5 years", mean, na.rm = TRUE)
```

More information and examples are found in Pebesma (2012a), which is obtained by

```
> vignette("sto")
```

and further information about temporal aggregation options are found in the help pages of `aggregate.zoo` of `zoo`.

Just like the spatial matching with `over` explained in Sect. 5.3, spatio-temporal geometry matching by `over` and the related application of `aggregate` only carries out a boolean match: two space-time entities either match or do not match, partial and complete matches are not distinguished.

Matching of time intervals is done by checking whether two time intervals overlap. As time intervals are assumed to be left-closed and right-open, two consecutive days do for instance not overlap when the end time of the first day is identical to the start time of the second day. Interval overlapping uses an on-the-fly indexing provided by package `intervals`.

When one or more of the `endTimes` are later than `time`, time is considered to reflect time *intervals*. If, in that case, some intervals have zero duration (meaning some have `endTime` equal to `time`), these are considered empty intervals and are never matched to another interval, not even to themselves. If time reflects time instances, matches are found by equality.

6.8 Visualisation

Objects deriving from `ST` have a `plot` method that plots the space-time layout, i.e. the space index against the time index. The `stplot` methods provides a number of higher-level plots, including multi-panel plots, similar to how `splot` does this for spatial data. A number of visualisation options will now be discussed.

6.8.1 Multi-panel Plots

In multi-panel (lattice) plots, panels share *x*- and *y*-axis, a common legend or key is used, and the strip above the panel indicates what the panel is about. Three types are implemented for STFDF data, where panels indicate

- Time:** the *x* and *y* axis denote space, an example is shown in Fig. 6.2; this method uses `splot` in package `sp`, and inherits most of its options,
- Feature:** the *x* and *y* axis denote time and value; colours may indicate different variables (`mode="tp"`); see Fig. 6.4 (left)
- Attribute:** *x* and *y* axis denote time and value; colours may denote different features (`mode="ts"`), see Fig. 6.4 (right).

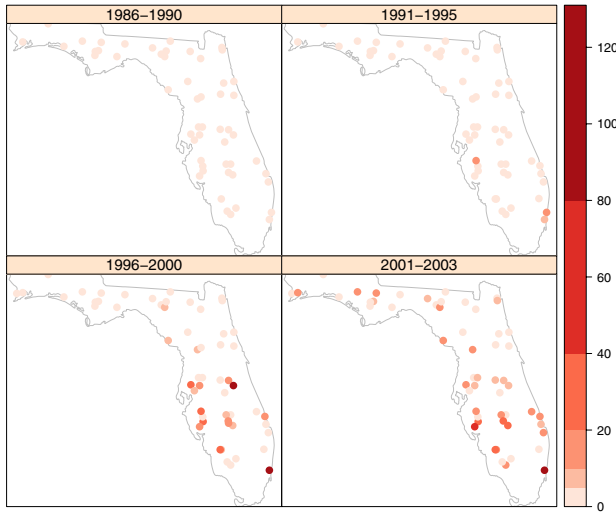


Fig. 6.2 Counts of Eurasian Collared Dove in Florida, 1986–2003, yearly average counts over 5-year periods

For both cases where time is on the y -axis, values over time for different variables or features are connected with lines, as is usual with time series plots. This can be changed to symbols by specifying `type="p"`.

6.8.2 Space-Time Plots

Space-time plots show data in a space-time cross-section, with e.g., space on the x -axis and time on the y -axis, or vice-versa. Hovmöller diagrams (Hovmöller, 1949) are an example of these for full space-time lattices, i.e., objects of class `STFDF`. As space receives one dimension in the plot, these plots usually consider variation along some transect.

To obtain such a plot with `stplot` the arguments `mode` and `scaleX` should be considered. By default, space is taken as the index of the spatial features; some special care is needed when values along a certain transect, or features in a particular order need to be plotted instead. Details are found in the `stplot` documentation.

In the bird count example, we use count-weighted mean time, obtained from n count-time observation pairs (c_i, t_i) by

$$\sum_{i=1}^n c_i t_i / \sum_{i=1}^n c_i$$

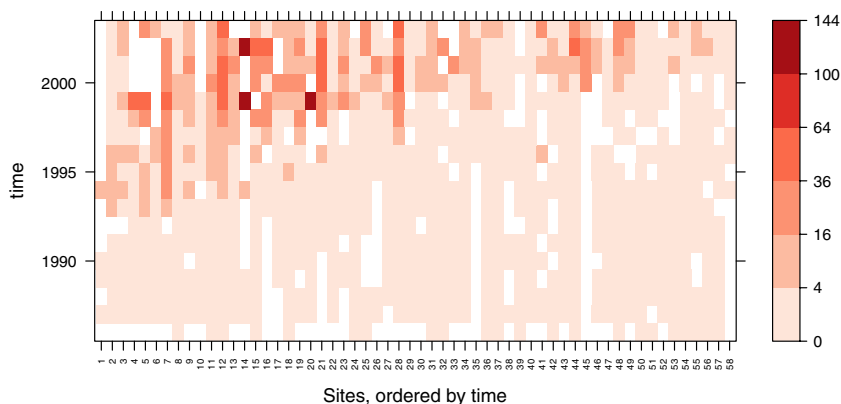


Fig. 6.3 Eurasian Collared Dove counts in Florida, sites ordered by count-weighted mean time; *white cells* indicating missing counts

to obtain an ordering measure (o) that combines abundance and early presence:

```
> ecd.FL <- ecd.st[FL, , "sqrtcounts"]
> x <- as(ecd.FL, "xts")
> x[is.na(x)] <- 0
> o <- order(as.vector(1:18 %*% x)/apply(x, 2, sum))
```

The corresponding Hovmöller diagram

```
> library(RColorBrewer)
> pal <- brewer.pal(6, "Reds")
> cuts <- c(0, 2, 4, 6, 8, 10, 12)
> ck <- list(at = cuts, labels = as.character(cuts^2))
> stplot(ecd.FL[o, ], mode = "xt", col.regions = pal, cuts = 6,
+       asp = 0.5, xlab = "Sites, ordered by time", colorkey = ck)
```

is shown in Fig. 6.3 – note that the replacement of NA values is of influence, and disputable.

6.8.3 Animated Plots

Animation is another way of displaying change over time. Books are notably bad in communicate animating plots.

Animation can be interactive, or non-interactive. Using **stplot**, a sequence of **spplots**, one for each time step, is displayed in a loop when the parameter **animate** is set to a positive value. This value sets the time (seconds) of pause between subsequent plots.

Package **stpp** has, beyond a function **animation** that adds points in a non-interactive progressive loop, a function **stan** that allows interactive animation to view the temporal development of a spatio-temporal point pattern.

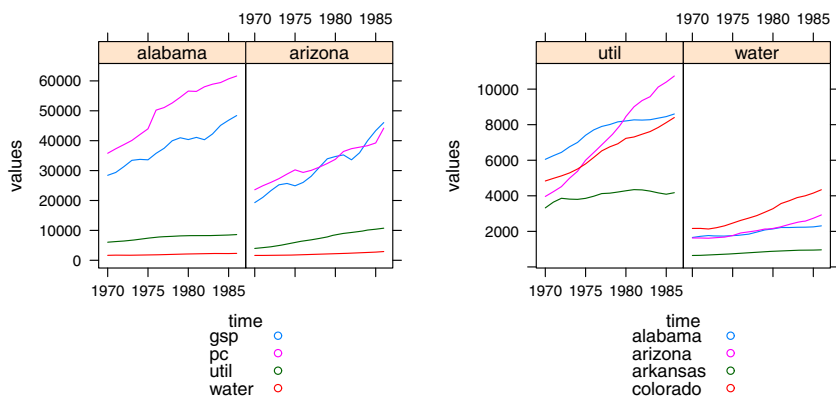


Fig. 6.4 Two `stplot` examples for STDF objects, using feature in panel (*left*: `mode="tp"`) and attribute in panel (*right*: `mode="ts"`) (Data from `Produc` in `plm`, see Pebesma (2012b); Baltagi (2001))

It colours the points in a particular time window, allows this window to be manipulated (increase its width, shift it backwards and forwards in time), and keeps the past points in a background colour.

Another platform that allows specifying a time interval and manipulating it interactively is Google Earth™. Package `plotKML` exports STDF and STIDF objects into KML, which can be shown in Google Earth.

6.8.4 Time Series Plots

Time series plots are a fairly common type of plot in R. Package `xts` has a plot method that allows univariate time series to be plotted. Many (if not most) plot routines in R support time to be along the x- or y-axis.

6.9 Further Packages

An overview of around 50 packages that are relevant for manipulating, analysing and/or viewing spatio-temporal data in R is maintained in the spatio-temporal task view, <http://cran.r-project.org/view=SpatioTemporal>. Some of these will be mentioned below, and/or in later chapters.

6.9.1 Handling Spatio-Temporal Data

Package **raster** has facilities to deal with sequences of raster images, called **RasterStack** or **RasterBrick**. This sequence can correspond to moments (or periods) in time. Coercion between **STFDF** objects and raster stack or bricks is provided by **raster** and **spacetime**, as is an **stplot** method for **RasterStack** or **RasterBrick** objects. Package **rasterVis** has several plot methods for raster objects, building on **lattice**, some of which deal with time.

Several packages interface popular file formats for providing spatio-temporal data, many are dedicated to a single data source or type. Package **sos4R** provides methods to request data from a sensor observation service. Packages **ncdf**, **ncdf4** and **RNetCDF** allow reading and writing netcdf files; **pbdNCDF4** adds collective parallel read and write capability to **ncdf4**.

6.9.2 Analysing Spatio-Temporal Data

Section 8.12 lists options for spatial prediction (geostatistical analysis) of spatio-temporal fields, mentions several relevant packages and points amongst others to a vignette in package **gstat** about this topic. Section 10.7 describes spatio-temporal approaches for disease mapping. Package **surveillance** provides temporal and spatio-temporal modelling and monitoring of epidemic phenomena.

6.10 Outlook

In his outlook over *researchable questions*, Galton (2004) mentions:

First and foremost is the question of what data structures one should use to represent multi-aspect phenomena. Hand in hand with this it is necessary to specify operations on those data structures which can generate the multiple aspects of those structures.

We believe that we made a start with answering these, not by a top-down approach that might start with designing an ontology for the phenomena to be represented and their properties and relations, but by a bottom-up approach, looking at what data analysts do, or, as Jim Gray (SzaLay and Blakeley, 2009) put it, “starting with the 20 most important questions”.

These questions involved which spatial and temporal characteristics our data have, how they are structured in space-time, how construction, selection, combining and aggregation works, and how graphs of spatio-temporal data can be created. We have not addressed how different phenomena should be

distinguished (no distinction is made between objects, events or fields), and many aspects such as (re)sampling data and the analysis of (sets of) trajectories have not yet been addressed. Depending on requirements, resources, and on interactions within or outside the R ecosystem, these may be addressed in the near future.

Part II

Analysing Spatial Data

Analysing Spatial Data

The analysis of spatial data is usually undertaken to make inferences, that is to try to draw conclusions about a hypothesised data generating process or to use an estimated process to predict values at locations for which observations are unavailable. In some cases, the conclusions are sufficient in themselves, and in others, they are carried through to other hierarchical layers in the model under scrutiny. Haining (2003, pp. 184–185) and Bivand (2002, p. 409) suggest (following Tukey, 1977) that our understanding of the data may be partitioned into

$$\text{data} = \text{smooth} + \text{rough}.$$

If the data are spatial, we can see that there is room for another term, irrespective of whether we are more interested in the fit of the model itself or in calibrating the model in order to predict for new data:

$$\text{data} = \text{smooth} + \text{spatial smooth} + \text{rough}.$$

The added term constitutes the ‘added value’ of spatial data analysis, bringing better understanding or predictive accuracy at the price of using specialised methods for fitting the spatial smooth term. We will here be concerned with methods for finding out whether fitting this term is worth the effort, and, if so, how we might choose to go about doing so.

Before rushing off to apply such specialised methods, it is worth thinking through the research problem thoroughly. We have already mentioned the importance of the Distributed Statistical Computing conference in Vienna in 2003 for our work. At that meeting, Bill Venables presented a fascinating study of a real research problem in the management of tiger prawn fisheries. The variable of interest was the proportion by weight of two species of tiger prawn in the logbook on a given night at a given location. In a very careful treatment of the context available, the ‘location’ was not simply taken as a point in space with geographical coordinates:

Rather than use latitude and longitude directly as predictors, we find it more effective to represent station locations using the following two predictors:

- The shortest distance from the station to the coast (variable R_{land}), and
- The distance from an origin in the west to the nearest point to the station along an arbitrary curve running nearly parallel to the coast (variable R_{dist}).

[...] Rather than use R_{dist} itself as a predictor, we use a natural spline basis that allows the fitted linear predictor to depend on the variable in a flexible curvilinear way.

[...] Similarly, we choose a natural spline term with four internal knots at the quantiles of the corresponding variable for the logbook data for the “distance from dry land” variable, R_{land} .

The major reason to use this system, which is adapted to the coastline, is that interactions between R_{land} and R_{dist} are more likely to be negligible than for latitude and longitude, thus simplifying the model. The fact that they do not form a true co-ordinate system equivalent to latitude and longitude is no real disadvantage for the models we propose. Venables and Ripley (2002, pp. 412–413)

The paper deserves to be required reading in its entirety for all spatial data analysts, not least because of its sustained focus on the research problem at hand. It also demonstrates that because applied spatial data analysis builds on and extends applied data analysis, specifically spatial methods should be used when the problem cannot be solved with general methods. Consequently, familiarity with the modelling chapters of textbooks using R for analysis will be of great help in distinguishing between situations calling for spatial solutions, and those that do not, even though the data are spatial. Readers will benefit from having one or more of Fox and Weisberg (2011), Dalgaard (2008), Faraway (2004, 2006), or Venables and Ripley (2002) to refer to in seeking guidance on making often difficult research decisions.

In introducing this part of the book – covering specialised spatial methods but touching in places on non-spatial methods – we use the classification of Cressie (1993) of spatial statistics into three areas, *spatial point patterns*, covered here in Chap. 7, *geostatistical data* in Chap. 8, and *lattice data*, here termed areal data, in Chaps. 9 and 10. In Chap. 1, we mentioned a number of central books on spatial statistics and spatial data analysis; Table II.1 shows very roughly which of our chapters contain material that illustrates some of the methods presented in more recent spatial statistics books, including treatments of all three areas of spatial statistics discussed earlier (see p. 14).

The coverage here is uneven, because only a limited number of the topics covered in these books could be accommodated; the specialised literature within the three areas will be referenced directly in the relevant chapters. On the other hand, the implementations discussed below may be extended to cover alternative methods; for example, the use of WinBUGS and INLA with R is introduced in Chap. 10 in general forms capable of extension. The choice of contributed packages is also uneven; we have used the packages that we maintain, but this does not constitute a recommendation of these rather

Chapter	7	8	9–10
Cressie (1993)	8	2–3	6–7
Schabenberger and Gotway (2005)	3	4–5	1, 6
Waller and Gotway (2004)	5	8	6, 7, 9
Fortin and Dale (2005)	2.1–2.2	3.5	3.1–3.4, 5
O’Sullivan and Unwin (2010)	5–6	9–10	7–8
Gaetan and Guyon (2010)	3, 5.5	1.1–6, 1.9, 5.1	1.7–8, 5.2–4, 5.6
Krivoruchko (2011)	13	7–10	11–12

Table II.1 Thematic cross-tabulation of chapters in this book with chapters and sections of chosen books on spatial statistics and spatial data analysis

than other approaches. Note that complete code examples, data sets, and other support material may be found on the book website.

Chapter 7

Spatial Point Pattern Analysis

7.1 Introduction

The analysis of point patterns appears in many different areas of research. In ecology, for example, the interest may be focused on determining the spatial distribution (and its causes) of a tree species for which the locations have been obtained within a study area. Furthermore, if two or more species have been recorded, it may also be of interest to assess whether these species are equally distributed or competition exists between them. Other factors which force each species to spread in particular areas of the study region may be studied as well. In spatial epidemiology, a common problem is to determine whether the cases of a certain disease are clustered. This can be assessed by comparing the spatial distribution of the cases to the locations of a set of controls taken at random from the population.

In this chapter, we describe how the basic steps in the analysis of point patterns can be carried out using R. When introducing new ideas and concepts we have tried to follow Diggle (2003) as much as possible because this text offers a comprehensive description of point processes and applications in many fields of research. The examples included in this chapter have also been taken from that book and we have tried to reproduce some of the examples and figures included there.

In general, a point process is a stochastic process in which we observe the locations of some *events* of interest within a bounded region A . Diggle (2003) defines a point process as a ‘stochastic mechanism which generates a countable set of events’. Diggle (2003) and Möller and Waagepetersen (2003) give proper definitions of different types of a point process and their main properties. The locations of the events generated by a point process in the area of study A will be called a *point pattern*. Sometimes, additional covariates may have been recorded and they will be attached to the locations of the observed events.

The analysis of spatio-temporal point patterns are described in Gelfand et al. (2010) and Illian et al. (2008). As described in Chap. 6, there are classes for different types of spatio-temporal point patterns.

Other books covering this subject include Schabenberger and Gotway (2005, Chap. 3), Waller and Gotway (2004, Chaps. 5 and 6), O’Sullivan and Unwin (2010, Chaps. 5 and 6), Gaetan and Guyon (2010, Chaps. 3 and 5.5), and Krivoruchko (2011, Chap. 13). More recently, Cressie and Wikle (2011) provide a comprehensive summary of the field, including space-time point patterns.

7.2 Packages for the Analysis of Spatial Point Patterns

There are a number of packages for R which implement different functions for the analysis of spatial point patterns. The **spatial** package provides functions described in Venables and Ripley (2002, pp. 430–434), and **splancs** (Rowlingson and Diggle, 1993) and **spatstat** (Baddeley and Turner, 2005) provide other implementations and additional methods for the analysis of different types of point processes. The Spatial Task View contains a complete list of all the packages available in R for the analysis of point patterns. Other packages worth mentioning include **spatialkernel**, which implements different kernel functions and methods for the analysis of multivariate point processes. Given that most of the examples included in this chapter have been computed using **splancs** and **spatstat**, we focus particularly on these packages.

These packages use different data structures to store the information of a point pattern. Given that it would be tedious to rewrite all the code included in these packages to use **sp** classes, we need a simple mechanism to convert between formats. Package **maptools** offers some functions to convert between **ppp** objects representing two-dimensional point patterns (from **spatstat**, which uses old-style classes, see p. 24) and **sp** classes. Note that, in addition to the point coordinates, **ppp** objects include the boundary of the region where the point data have been observed, whilst **sp** classes do not, and it has to be stored separately. Data types used in **splancs** are based on a two-column matrix for the coordinates of the point pattern plus a similar matrix to store the boundary; the package was written before old-style classes were introduced. Function `as.points` is provided to convert to this type of structure. Hence, it is very simple to convert the coordinates from **sp** classes to use functions included in **splancs**.

Section 2.4 describes different types of **sp** classes to work with point data. They are **SpatialPoints**, for simple point data, and **SpatialPointsDataFrame**, when additional covariates are recorded. More information and examples can be found in the referred section. Hence, it should not be difficult

to have the data available in the format required for the analysis whatever package is used.

To illustrate the use of some of the different techniques available for the analysis of point patterns, we have selected some examples from forest ecology, biology, and spatial epidemiology. The point patterns in Fig. 7.1 show the spatial distribution of cell centres (left), California redwood trees (right), and Japanese black pine (middle). All data sets have been re-scaled to fit into a one-by-one square. These data sets are described in Ripley (1977), Strauss (1975), and Numata (1961) and all of them have been re-analysed in Diggle (2003).

These data sets are available in package **spatstat**. This package uses **ppp** objects to store point patterns, but package **maptools** provides some functions to convert between **ppp** objects and **SpatialPoints**, as shown in the following example. First we take the Japanese black pine saplings example, measured in a square sampling region in a natural forest, reading in the data provided with **spatstat**.

```
> library(spatstat)
> data(japanesepines)

> summary(japanesepines)

Planar point pattern: 65 points
Average intensity 65 points per square unit (one unit = 5.7 metres)

Coordinates are given to 2 decimal places
i.e. rounded to the nearest multiple of 0.01 units (one unit = 5.7 metres)

Window: rectangle = [0, 1] x [0, 1] units
Window area = 1 square unit
Unit of length: 5.7 metres
```

The summary shows the average intensity over the region of interest; this region, known as an observation window, is also reported in the summary; observation windows are stored in objects of class **owin**. In this case, the points have been scaled to the unit square already, but the size of the sampling square can be used to retrieve the actual measurements. Note that **spatstat** windows may be of several forms, here the window is a rectangle. When we coerce a **ppp** object with a rectangular window to a **SpatialPoints** object, the point coordinates will by default be re-scaled to their original values.

```
> library(maptools)

> spjpines <- as(japanesepines, "SpatialPoints")
> summary(spjpines)

Object of class SpatialPoints
Coordinates:
      min max
[1,]   0 5.7
[2,]   0 5.7
```



```
Is projected: NA
proj4string : [NA]
Number of points: 65
```

We can get back to the unit square using the `elide` methods discussed in Chap. 5 as the summary of the output object shows.

```
> spjpines1 <- elide(spjpines, scale = TRUE, unitsq = TRUE)
> summary(spjpines1)
```

```
Object of class SpatialPoints
Coordinates:
      min max
[1,]   0   1
[2,]   0   1
Is projected: NA
proj4string : [NA]
Number of points: 65
```

Getting back to a `ppp` object is also done by coercing, but if we want to preserve the actual dimensions, we have to manipulate the `owin` object belonging to the `ppp` object directly. We return later to see how `SpatialPolygons` objects may be coerced into `owin` objects, and how `spatstat` `im` objects can interface with `SpatialGrid` objects.

```
> pppjap <- as(spjpines1, "ppp")
> summary(pppjap)
```

```
Planar point pattern: 65 points
Average intensity 65 points per square unit
```

```
Coordinates are given to 2 decimal places
i.e. rounded to the nearest multiple of 0.01 units
```

```
Window: rectangle = [0, 1] x [0, 1] units
Window area = 1 square unit
```

These point patterns have been obtained by sampling in different regions, but it is not rare to find examples in which we have different types of events in the same region. In spatial epidemiology, for example, it is common to have two types of points: *cases* of a certain disease and *controls*, which usually reflect the spatial distribution of the population. In general, this kind of point pattern is called a *marked* point pattern because each point is assigned to a group and labelled accordingly.

The Asthma data set records the results of a case-control study carried out in 1992 on the incidence of asthma in children in North Derbyshire (United Kingdom). This data set has been studied by Diggle and Rowlingson (1994), Singleton et al. (1995), and Diggle (2003) to explore the relationship between asthma and the proximity to the main roads and three putative pollution sources (a coking works, chemical plant, and waste treatment centre). In the study, a number of relevant covariates were also collected by means of a questionnaire that was completed by the parents of the children attending

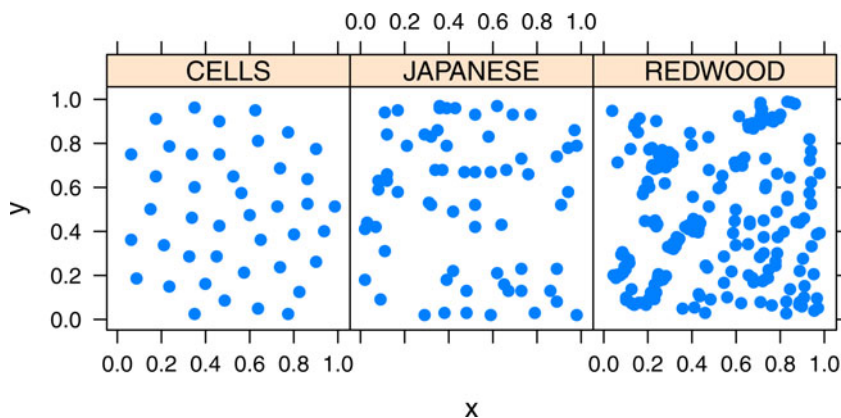


Fig. 7.1 Example of three point patterns re-scaled to fit in the unit square. On the *left*, spatial distribution of the location of cell centres (Ripley, 1977); in the *middle*, Japanese black pine saplings (Numata, 1961); and on the *right*, saplings of California redwood trees (Strauss, 1975)

ten schools in the region. Children having suffered from asthma will act as cases whilst the remainder of the children included in the study will form the set of controls. Although this data set is introduced here, the spatial analysis of case-control data is described in the final part of this chapter.

The data set is available from Prof. Peter J. Diggle's website and comes in anonymised form. Barry Rowlingson provided some of the road lines. The original data were supplied by Dr. Joanna Briggs (University of Leeds, UK). To avoid computational problems in some of the methods described in this section, we have removed a very isolated point, which was one of the cases, and we have selected an appropriate boundary region.

The next example shows how to display the point pattern, including the boundary of the region (that we have created ourselves) and the location of the pollution sources using different **sp** layouts and function **spplot** (see Chap. 3 for more details). Given that the data set is a marked point pattern, we have converted it to a **SpatialPointsDataFrame** to preserve the type (case or control) of the events and all the other relevant information. In addition, we have created a **SpatialPolygons** object to store the boundary of the region and a **SpatialPointsDataFrame** object for the location of the three pollution sources. Given that the main roads are available, we have included them as well using a **SpatialLines** object. The final plot is shown in Fig. 7.2.

```
> library(rgdal)
> spasthma <- readOGR(".", "spasthma")
> spbdry <- readOGR(".", "spbdry")
> spsrc <- readOGR(".", "spsrc")
> sproads <- readOGR(".", "sproads")
```

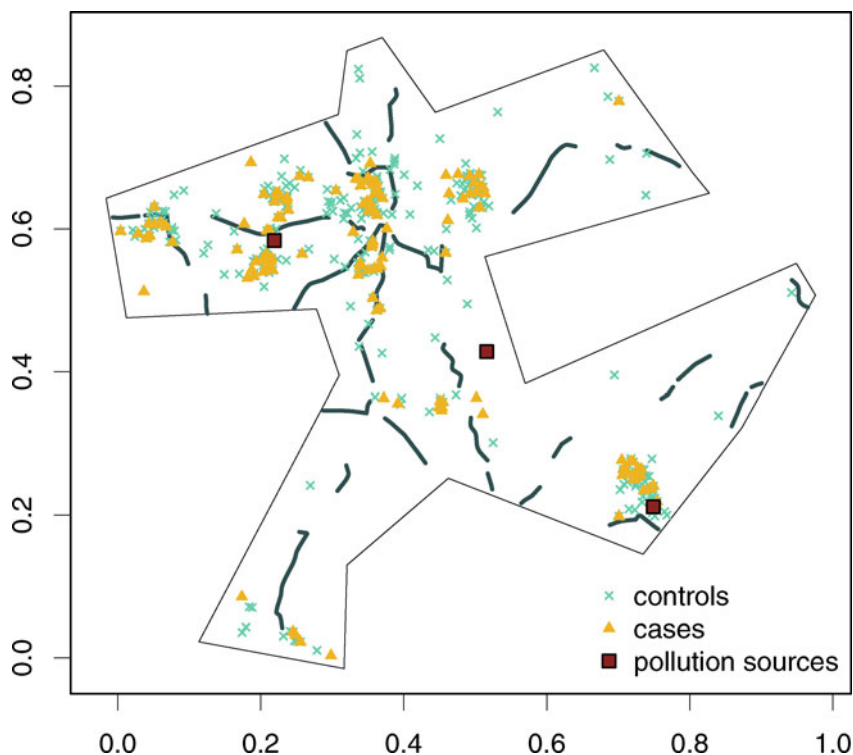


Fig. 7.2 Locations of the residence of asthmatic (cases, *orange filled triangle*) and non-asthmatic (controls, *green cross*) in North Derbyshire, 1992 Diggle and Rowlingson (1994). The boundary has been taken to contain all points in the data set. The map shows the pollution sources (*brown filled square*) and the main roads (*grey lines*)

7.3 Preliminary Analysis of a Point Pattern

The analysis of point patterns is focused on the spatial distribution of the observed events and making inference about the underlying process that generated them. In particular, there are two main issues of interest: the distribution of events in space and the existence of possible interactions between them. For a merely descriptive analysis, we would represent the locations of the point pattern in the study area. This will give us an idea of the distribution of the points, and it can lead to possible hypothesis about the spatial distribution of the events. Further statistical analyses can be done and they are described in this section.

7.3.1 Complete Spatial Randomness

When studying a point process, the most basic test that can be performed is that of *Complete Spatial Randomness* (CSR, henceforth). Intuitively, by CSR we mean that the events are distributed independently at random and uniformly over the study area. This implies that there are no regions where the events are more (or less) likely to occur and that the presence of a given event does not modify the probability of other events appearing nearby.

Informally, this can be tested by plotting the point pattern and observing whether the points tend to appear in clusters or, on the contrary, they follow a regular pattern. In any of these cases, the points are not distributed uniformly because they should be distributed filling all the space in the study area. Usually, clustered patterns occur when there is attraction (i.e. ‘contagion’) between points, whilst regular patterns occur when there is inhibition (i.e. ‘competition’) among points.

Figure 7.1 shows three examples of point patterns that have been generated by different biological mechanisms and seem to have different spatial distributions. In particular, the plot of the Japanese pine trees (middle) seems neither clustered nor regularly distributed, whilst the redwood seeds (right) show a clustered pattern and the cells (left) a regular one. Hence, only the spatial distribution of Japanese pine trees seems to be compatible with CSR.

To measure the degree of accomplishment of the CSR, several functions can be computed on the data. These are described in the following sections, together with methods to measure the uncertainty related to the observed pattern.

Testing for CSR is covered in Waller and Gotway (2004, pp. 118–126), O’Sullivan and Unwin (2010, pp. 99–108, including a discussion on pp. 158–165), and Schabenberger and Gotway (2005, pp. 86–99, including other methods not presented here).

7.3.2 *G Function: Distance to the Nearest Event*

The G function measures the distribution of the distances from an arbitrary event to its nearest event. If these distances are defined as $d_i = \min_j \{d_{ij}, \forall j \neq i\}$, $i = 1, \dots, n$, then the G function can be estimated as

$$\hat{G}(r) = \frac{\#\{d_i : d_i \leq r, \forall i\}}{n},$$

where the numerator is the number of elements in the set of distances that are lower than or equal to d and n is the total number of points. Under CSR, the value of the G function is

$$G(r) = 1 - \exp\{-\lambda\pi r^2\},$$

where λ represents the mean number of events per unit area (or *intensity*).

The compatibility with CSR of the point pattern can be assessed by plotting the empirical function $\hat{G}(d)$ against the theoretical expectation. In addition, point-wise envelopes under CSR can be computed by repeatedly simulating a CSR point process with the same estimated intensity $\hat{\lambda}$ in the study region (Diggle, 2003, p. 13) and check whether the empirical function is contained inside. The next chunk of code shows how to compute this by using **spatstat** functions **Gest** and **envelope**. The results have been merged in a data frame in order to use conditional Lattice graphics.

```
> set.seed(120109)
> r <- seq(0, sqrt(2)/6, by = 0.005)
> envjap <- envelope(as(sppines1, "ppp"), fun = Gest,
+   r = r, nrank = 2, nsim = 99)
> envred <- envelope(as(spred, "ppp"), fun = Gest, r = r,
+   nrank = 2, nsim = 99)
> envcells <- envelope(as(spcells, "ppp"), fun = Gest,
+   r = r, nrank = 2, nsim = 99)
> Gresults <- rbind(envjap, envred, envcells)
> Gresults <- cbind(Gresults, y = rep(c("JAPANESE", "REDWOOD",
+   "CELLS"), each = length(r)))
```

Figure 7.3 shows the empirical function $\hat{G}(r)$ against $G(r)$ together with the 96% pointwise envelopes (because **nrank=2**) of the same point pattern examined using the G function. The plot is produced by taking the pairs $(G(r), \hat{G}(r))$ for a set of reasonable values of the distance r , so that in the x -axis we have the values of the theoretical value of $G(r)$ under CSR and in the y -axis the empirical function $\hat{G}(r)$. The results show that only the Japanese trees seem to be homogeneously distributed, whilst the redwood seeds show a clustered pattern (values of $\hat{G}(r)$ above the envelopes) and the location of the cells shows a more regular pattern (values of $\hat{G}(r)$ below the envelopes).

envelope is a very flexible function that can be used to compute Monte Carlo envelopes of a certain type of functions. Basically, it works by randomly simulating a number of point patterns so that the summary function is computed for all of them. The resulting values are then used to compute point-wise (i.e. at different distances) or global Monte Carlo envelopes. **envelope** can be passed the way the point patterns are generated (by default, CSR). The reader is referred to the manual page for more information.

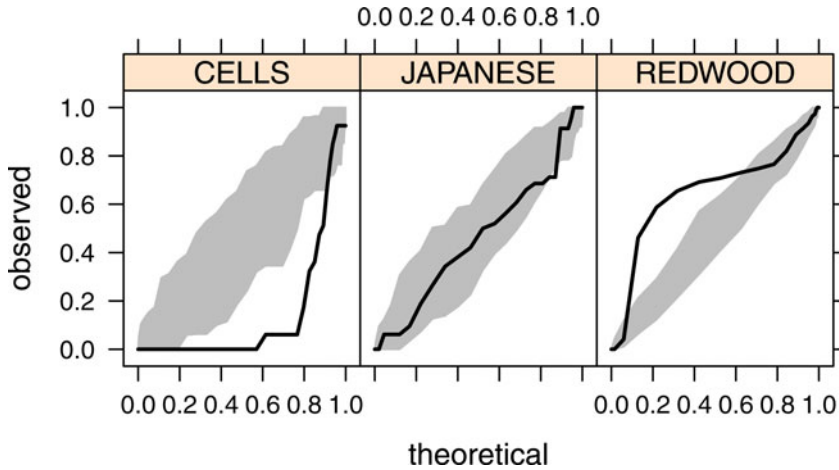


Fig. 7.3 Envelopes and observed values of the G function for three point patterns

7.3.3 F Function: Distance from a Point to the Nearest Event

The F function measures the distribution of all distances from an arbitrary point of the plane to its nearest event. This function is often called the *empty space* function because it is a measure of the average space left between events. Under CSR, the expected value of the F function is

$$F(r) = 1 - \exp\{-\lambda\pi r^2\}.$$

Hence, we can compare the estimated value of the F function to its theoretical value and compute simulation envelopes as before.

```
> set.seed(30)
> Fenvjap <- envelope(as(sppines1, "ppp"), fun = Fest,
+   r = r, nrank = 2, nsim = 99)
> Fenvred <- envelope(as(spred, "ppp"), fun = Fest, r = r,
+   nrank = 2, nsim = 99)
> Fenvcells <- envelope(as(spcells, "ppp"), fun = Fest,
+   r = r, nrank = 2, nsim = 99)
> Fresults <- rbind(Fenvjap, Fenvred, Fenvcells)
> Fresults <- cbind(Fresults, y = rep(c("JAPANESE", "REDWOOD",
+   "CELLS"), each = length(r)))
```

Figure 7.4 shows the empirical F functions and their associated 96% envelopes (because `nrank=2`) for the three data sets presented before. The Japanese data are compatible with the CSR hypothesis, whereas the cells point pattern shows a regular pattern ($\hat{F}(r)$ is above the envelopes) and the redwood points seem to be clustered, given the low values of $\hat{F}(r)$.

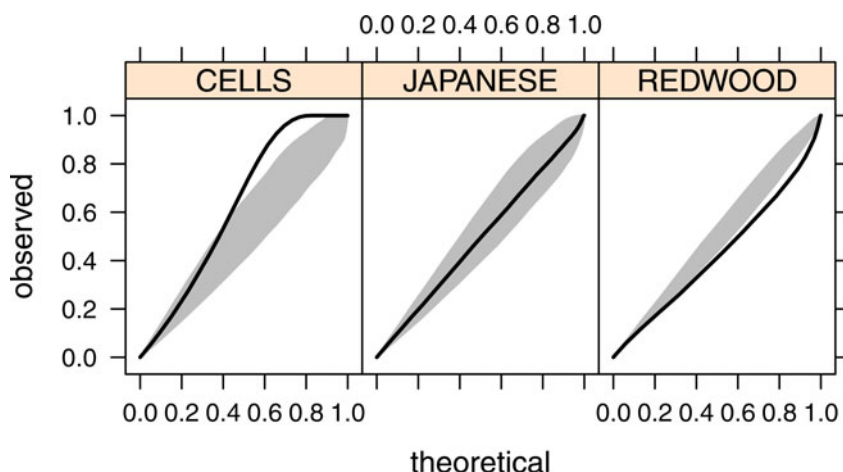


Fig. 7.4 Envelopes and observed values of the F function for three point patterns

7.4 Statistical Analysis of Spatial Point Processes

A first description of the point pattern can be done by estimating the spatial statistical density from the observed data. The spatial density has the same properties as a univariate density, but its domain is the study area where the point process takes place.

As an alternative function to measure the spatial distribution of the events, we can work with the *intensity* $\lambda(x)$ of the point process, which is proportional to its spatial density. The constant of proportionality is the expected number of events of the point process in the area A . That is, for two point processes with the same spatial density but different intensities, the number of events observed will be higher for the process with the highest intensity.

The intensity and spatial density are part of the *first-order* properties because they measure the distribution of events in the study region. Note that neither the intensity nor the spatial density give any information on the interaction between two arbitrary points. This is measured by *second-order properties*, which reflect any tendency of the events to appear clustered, independently, or regularly spaced.

First- and second-order properties are properly defined in, for example, Diggle (2003, p. 43) and Möller and Waagepetersen (2003, Chap. 4). We focus on the estimation of the intensity and the assessment of clustering, as explained in the following sections. Waller and Gotway (2004, pp. 130–146) and Schabenberger and Gotway (2005, pp. 90–103, 110–112) discuss the estimation of the intensity of a point pattern and the assessment of clustering as well.

The separation between first- and second-order properties can be difficult to disentangle without further assumptions. For example, do groups of events

appear at a specific location because the intensity is higher there or because events are clustered? In general, it is assumed that interaction between points occurs at small scale, while large-scale variation is reflected on the intensity (Diggle, 2003, p. 143). Waller and Gotway (2004, 146–147) also discuss the roles of first and second-order properties.

In the remainder of this chapter, we focus on Poisson processes because they offer a simple approach to a wide range of problems. Loosely, we can distinguish between homogeneous and inhomogeneous Poisson point processes (HPP and IPP, respectively). Both HPP and IPP assume that the events occur independently and are distributed according to a given intensity. The main difference between the two point processes is that the HPP assumes that the intensity function is constant, while the intensity of an IPP varies spatially. In a sense, the IPP is a generalisation of the HPP or, inversely, the HPP can be regarded as an IPP with constant intensity. Poisson processes are also described in Schabenberger and Gotway (2005, pp. 81–86, 107–110) and Waller and Gotway (2004, pp. 126–130).

Note that other spatial processes may be required when more complex data sets are to be analysed. For example, when events are clustered, points do not occur independently of each other and a clustered process would be more appropriate. See Diggle (2003, Chap. 5) and Möller and Waagepetersen (2003) for a wider description of other spatial point processes. `spatstat` provides a number of functions to fit some of the models described therein.

7.4.1 Homogeneous Poisson Processes

A homogeneous Poisson process is characterised as representing the kind of point process in which all events are independently and uniformly distributed in the region A where the point process occurs. This means that the location of one point does not affect the probabilities of other points appearing nearby and that there are no regions where events are more likely to appear.

More formally, Diggle (2003) describes an HPP in a region A as fulfilling:

1. The number of events in A , with area $|A|$, is Poisson distributed with mean $\lambda|A|$, where λ is the constant intensity of the point process.
2. Given n observed events in region A , they are uniformly distributed in A .

The HPP is also *stationary* and *isotropic*. It is stationary because the intensity is constant and the second-order intensity depends only on the relative positions of two points (i.e. direction and distance). In addition, it is isotropic because the second-order intensity is invariant to rotation. Hence, the point process has constant intensity and its second-order intensity depends only on the distance between the two points, regardless of the relative positions of the points.

These constraints reflect that the intensity of the point process is constant, that is $\lambda(x) = \lambda > 0, \forall x \in A$, and that events appear independently of each other. Hence, the HPP is the formal definition of a point process which is CSR.

7.4.2 Inhomogeneous Poisson Processes

In most cases assuming that a point process under study is homogeneous is not realistic. Clear examples are the distribution of the population in a city or the location of trees in a forest. In both cases, different factors affect the spatial distribution. In the case of the population, it can be the type of housing, neighbourhood, etc., whilst in the case of the trees, it can be the environmental factors such as humidity, quality of the soil, slope and others.

The IPP is a generalisation of the HPP, which allows for a non-constant intensity. The same principle of independence between events holds, but now the spatial variation can be more diverse, with events appearing more likely in some areas than others. As a result, the intensity will be a generic function $\lambda(x)$ that varies spatially.

7.4.3 Estimation of the Intensity

As stated previously, the intensity of an HPP point process is constant. Hence, the problem of estimating the intensity is the problem of estimating a constant function λ such as the expected number of events in region A ($\int_A \lambda dx$) is equal to the observed number of cases. This is the volume under the surface defined by the intensity in region A . Once we have observed the (homogeneous) point process, we have the locations of a set of n points. So, an unbiased estimator of the intensity is $n/|A|$, where $|A|$ is the area of region A . This ensures that the expected number of points is, in fact, the observed number of points.

For IPP, the estimation of the intensity can be done in different ways. It can be done non-parametrically by means of kernel smoothing or parametrically by proposing a specific function for the intensity whose parameters are estimated by maximising the likelihood of the point process. If we have observed n points $\{x_i\}_{i=1}^n$, the form of a kernel smoothing estimator is the following (Diggle, 1985; Berman and Diggle, 1989):

$$\hat{\lambda}(x) = \frac{1}{h^2} \sum_{i=1}^n \kappa\left(\frac{\|x - x_i\|}{h}\right) / q(\|x\|), \quad (7.1)$$

where $\kappa(u)$ is a bivariate and symmetrical kernel function. $q(\|x\|)$ is a border correction to compensate for the missing observations that occur when x is close to the border of the region A . Bandwidth h measures the level of smoothing. Small values will produce very peaky estimates, whilst large values will produce very smooth functions.

Silverman (1986) gives a detailed description of different kernel functions and their properties. In the examples included in this chapter, we have used a bivariate Gaussian kernel and the *quartic* kernel (also known as *biweight*), whose expression in two dimensions is

$$\kappa(u) = \begin{cases} \frac{3}{\pi}(1 - \|u\|^2)^2 & \text{if } u \in (-1, 1) \\ 0 & \text{Otherwise} \end{cases},$$

where $\|u\|^2$ denotes the squared norm of point $u = (u_1, u_2)$ equal to $u_1^2 + u_2^2$. Figure 7.5 shows an example of estimation of the intensity by kernel smoothing in a one-dimensional setting, but the same ideas are used in a spatial framework.

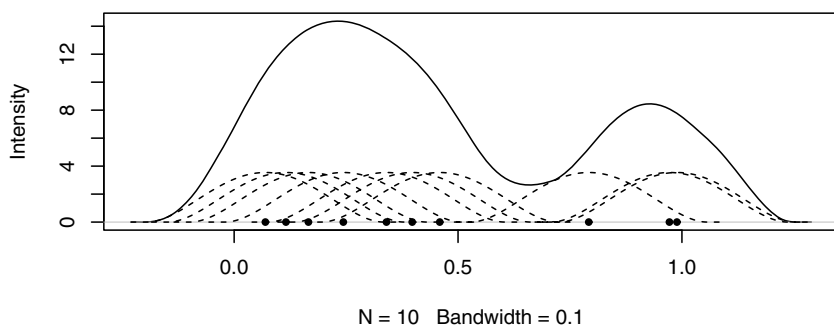


Fig. 7.5 Example of the contribution of the different points to the estimate of the intensity. *Dashed lines* represent the kernel around each observation, whilst the *solid line* is the estimate of the intensity

Methods for the selection of the bandwidth of kernel smoothers in a general setting are given by Silverman (1986). In the context of spatial analysis, a few proposals have been made so far, but it is not clear how to choose an optimal value for the bandwidth in the general case. It seems reasonable to use several values depending on the process under consideration, and choose a value that seems plausible.

Diggle (1985) and Berman and Diggle (1989) propose a criterion based on minimising the Mean Square Error (MSE) of the kernel smoothing estimator when the underlying point process in a stationary Cox process (see Diggle, 2003, p. 68, for details). However, it can still be used as a general exploratory method and a guidance in order to choose the bandwidth. Kelsall and Diggle (1995a,b, 1998) propose and compare different methods for the selection of the bandwidth when a case-control point pattern is used. Clark and Lawson (2004) have compared these and other methods for disease mapping, including some methods for the automatic selection of the bandwidth.

We have applied the approach proposed by Berman and Diggle (1989), which is implemented in functions `mse2d` (`splancs`) and `bw.diggle` (`spatstat`)

to the redwood data set. Note that these two functions can provide different optimal bandwidths because they rely on `kernel2d` (which implements a quartic kernel) and `density` (which implements a Gaussian kernel), respectively. This is shown in the following example:

```
> library(splancs)

> mserwq <- mse2d(as.points(coordinates(spred)), as.points(list(x = c(0,
+   1, 1, 0), y = c(0, 0, 1, 1))), 100, 0.15)
> bwq <- mserwq$h[which.min(mserwq$mse)]
> bwq

[1] 0.039

> mserw <- bw.diggle(as(spred, "ppp"))
> bw <- as.numeric(mserw)
> bw

[1] 0.01977539
```

Figure 7.6 shows different values of the bandwidth and their associated values of the MSE. The value that minimises it for the Gaussian kernel is 0.01978, but it should be noted that the curve is very flat around that point, which means that many other values of the bandwidth are plausible. This is a common problem in the analysis of real data sets.

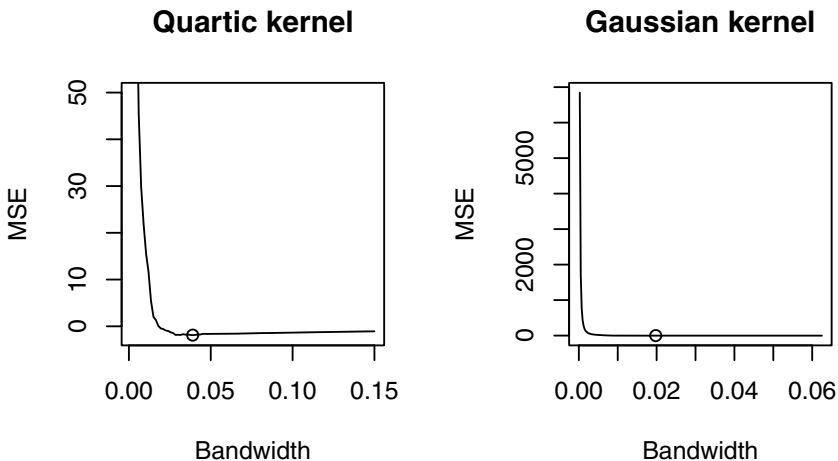


Fig. 7.6 Values of the mean square error for several values of the bandwidth using the redwood data set. These values that minimise the MSE are 0.039 (quartic kernel) and 0.0198 (Gaussian kernel) but many other values seem plausible, given the flatness of the curves

It must be noted that when estimating the intensity by kernel smoothing, the key choice is not that of the kernel function but the bandwidth. Different kernels will produce very similar estimates for equivalent bandwidths, but the

same kernel with different bandwidths will produce dramatically different results. An example of this fact is shown in Fig. 7.7, where four different bandwidths have been used to estimate the intensity of the redwood data.

Kernel smoothing using a quartic kernel can be performed with function `spkernel2d` (in package `splancs`) as follows:

```
> library(splancs)
> poly <- as.points(list(x = c(0, 0, 1, 1), y = c(0, 1,
+      1, 0)))
> sG <- Sobj_SpatialGrid(spred, maxDim = 100)$SG
> grd <- slot(sG, "grid")
> summary(grd)
> k0 <- spkernel2d(spred, poly, h0 = bw, grd)
> k1 <- spkernel2d(spred, poly, h0 = 0.05, grd)
> k2 <- spkernel2d(spred, poly, h0 = 0.1, grd)
> k3 <- spkernel2d(spred, poly, h0 = 0.15, grd)
> df <- data.frame(k0 = k0, k1 = k1, k2 = k2, k3 = k3)
> kernels <- SpatialGridDataFrame(grd, data = df)
> summary(kernels)
```

Package `spatstat` provides similar functions to estimate the intensity by kernel smoothing using an isotropic Gaussian kernel. We have empirically adjusted the value of the bandwidth to make the kernel estimates comparable. See Härdle et al. (2004, Sect. 3.4.2) for a full discussion. When calling `density` on a `ppp` object (which in fact calls `density.ppp`), we have used the additional arguments `dimxy` and `xy` to make sure that the grid used to compute the estimates is compatible with that stored in `kernels`. Finally, the kernel estimate is returned in an `im` class that is converted into a `SpatialGridDataFrame` and the values incorporated into `kernels`.

```
> cc <- coordinates(kernels)
> xy <- list(x = cc[, 1], y = cc[, 2])
> k4 <- density(as(spred, "ppp"), 0.5 * bw, dimyx = c(100,
+      100), xy = xy)
> kernels$k4 <- as(k4, "SpatialGridDataFrame")$v
> k5 <- density(as(spred, "ppp"), 0.5 * 0.05, dimyx = c(100,
+      100), xy = xy)
> kernels$k5 <- as(k5, "SpatialGridDataFrame")$v
> k6 <- density(as(spred, "ppp"), 0.5 * 0.1, dimyx = c(100,
+      100), xy = xy)
> kernels$k6 <- as(k6, "SpatialGridDataFrame")$v
> k7 <- density(as(spred, "ppp"), 0.5 * 0.15, dimyx = c(100,
+      100), xy = xy)
> kernels$k7 <- as(k7, "SpatialGridDataFrame")$v
> summary(kernels)
```

7.4.4 Likelihood of an Inhomogeneous Poisson Process

The previous procedure to estimate the intensity is essentially non-parametric. Alternatively, a specific parametric or semi-parametric form for the intensity

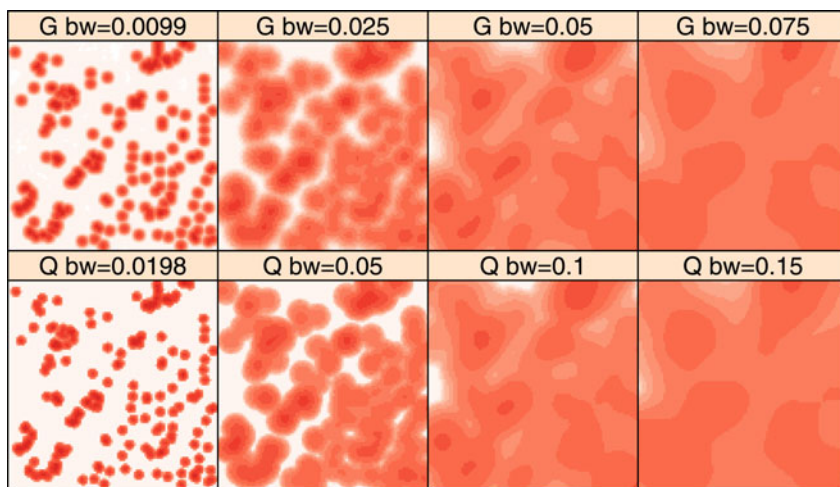


Fig. 7.7 Different estimates of the intensity of the redwood data set using Quartic (Q) and Gaussian (G) kernels and different values of the bandwidth

may be of interest (e.g. to include available covariates). Standard statistical techniques, such as the maximisation of the likelihood, can be used to estimate the parameters that appear in the expression of the intensity.

The expression of the likelihood can be difficult to work out for many point processes. However, in the case of the IPP (and, hence, the HPP) it has a very simple expression. The log-likelihood of a realisation of n independent events of an IPP with intensity $\lambda(x)$ is (Diggle, 2003, p. 104)

$$L(\lambda) = \sum_{i=1}^n \log \lambda(x_i) - \int_A \lambda(x) dx,$$

where $\int_A \lambda(x) dx$ is the expected number of cases of the IPP with intensity $\lambda(x)$ in region A .

When the intensity of the point process is estimated parametrically, the likelihood can be maximised to obtain the estimates of the parameters of the model. Diggle (2003, p. 104) suggests a log-linear model

$$\log \lambda(x) = \sum_{j=1}^p \beta_j z_j(x)$$

using covariates $z_j(x)$, $j = 1, \dots, p$ measured at a location x . These models can be fit using standard numerical integration techniques.

The following example defines the log-intensity (`loglambda`) at a given point $x = (x_1, x_2)$ using the parametric specification given by

$$\log \lambda(x) = \alpha + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1^2 + \beta_4 x_2^2 + \beta_5 x_1 * x_2. \quad (7.2)$$

This expression is in turn used to construct the likelihood of an IPP (L). Function `adaptIntegrate` (from the **cubature** package) is used to compute numerically the integral that appears in the expression of the likelihood.

```
> loglambda <- function(x, alpha, beta) {
+   l <- alpha + sum(beta * c(x, x * x, prod(x)))
+   return(l)
+ }
> L <- function(alphabeta, x) {
+   l <- apply(x, 1, loglambda, alpha = alphabeta[1],
+             beta = alphabeta[-1])
+   l <- sum(l)
+   intL <- adaptIntegrate(lowerLimit = c(0, 0), upperLimit = c(1,
+   1), fDim = 1, tol = 1e-08, f = function(x, alpha = alphabeta[1],
+   beta = alphabeta[-1]) {
+     exp(loglambda(x, alpha, beta))
+   })
+   l <- l - intL$integral
+   return(l)
+ }
```

The following example uses the locations of maple trees from the Lansing Woods data set (Gerard, 1969) in order to show how to fit a parametric intensity using (7.2). The parameters are estimated by maximising the likelihood using function `optim`.

```
> library(cubature)
> data(lansing)
> x <- as.points(lansing[lansing$marks == "maple", ])
> optbeta <- optim(par = c(log(514), 0, 0, 0, 0, 0), fn = L,
+   control = list(maxit = 1000, fnscale = -1), x = x)
```

The values of the coefficients $\alpha, \beta_1, \dots, \beta_5$ are 5.56, 5.66, -0.963 , -5.14 , -1.16 , 0.959 , for a value of the (maximised) likelihood of 2,778.3. Figure 7.8 shows the location of the maple trees and the estimated intensity according to parametric model in (7.2). See Diggle (2003, Chap. 7) for a similar analysis using all the tree species in the Lansing Woods data set.

The same example can be run using function `ppm` from **spatstat** as follows (x and y representing the coordinates of the point pattern):

```
> lmaple <- lansing[lansing$marks == "maple", ]
> ppm(Q = lmaple, trend = ~x + y + I(x^2) + I(y^2) + I(x *
+   y))
```

Nonstationary multitype Poisson process

Possible marks:

blackoak hickory maple misc redoak whiteoak

Trend formula: $\sim x + y + I(x^2) + I(y^2) + I(x * y)$

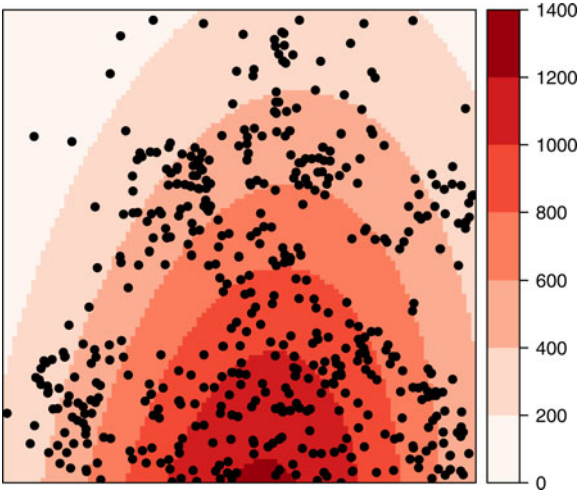


Fig. 7.8 Location of maple trees from the Lansing data set and their estimated parametric intensity using model (7.2)

Fitted coefficients for trend formula:

(Intercept)	x	y	I(x^2)	I(y^2)
3.7310742	5.6400643	-0.7663636	-5.0115142	-1.1983209
I(x * y)				
0.6375824				

	Estimate	S.E.	Ztest	CI95.lo	CI95.hi
(Intercept)	3.7310742	0.2542004	na	3.2328505	4.22929795
x	5.6400643	0.7990009	***	4.0740514	7.20607727
y	-0.7663636	0.6990514		-2.1364792	0.60375200
I(x^2)	-5.0115142	0.7011631	***	-6.3857686	-3.63725974
I(y^2)	-1.1983209	0.6428053		-2.4581962	0.06155433
I(x * y)	0.6375824	0.6989167		-0.7322691	2.00743391

As the authors mention in the manual page, `ppm` can be compared to `glm` because it can be used to fit a specific type of point process model to a particular point pattern. In this case, the `family` argument used in `glm` to define the model is substituted by `interaction`, which defines the point process to be fit. By default, a Poisson point process is used, but many other point processes can be fitted (see manual page for details).

7.4.5 Second-Order Properties

Second-order properties measure the strength and type of the interactions between events of the point process. Hence, they are particularly interesting if we are keen on studying clustering or competition between events.

Informally, the *second-order intensity* of two points x and y reflects the probability of any pair of events occurring in the vicinities of x and y , respectively. Diggle (2003, p. 43) and Möller and Waagepetersen (2003, Chap. 4) give a more formal description of the second-order intensity. Schablenberger and Gotway (2005, pp. 99–103) and Waller and Gotway (2004, pp. 137–147) also discuss second-order properties and the role of the K -function.

An alternative way of measuring second-order properties when the spatial process is HPP is by means of the K -function (Ripley, 1976, 1977). The K -function measures the number of events found up to a given distance of any particular event and it is defined as

$$K(s) = \lambda^{-1} E[N_0(s)],$$

where $E[\cdot]$ denotes the expectation and $N_0(s)$ represents the number of further events up to a distance s around an arbitrary event. To compute this function, Ripley (1976) also proposed an unbiased estimate equal to

$$\hat{K}(s) = (n(n-1))^{-1} |A| \sum_{i=1}^n \sum_{j \neq i} w_{ij}^{-1} |\{x_j : d(x_i, x_j) \leq s\}|, \quad (7.3)$$

where w_{ij} are weights equal to the proportion of the area inside the region A of the circle centred at x_i and radius $d(x_i, x_j)$, the distance between x_i and x_j .

The value of the K -function for an HPP is $K(s) = \pi s^2$. By comparing the estimated value $\hat{K}(s)$ to the theoretical value we can assess what kind of interaction exists. Usually, we assume that these interactions occur at small scales, and so will be interested in relatively small values of s . Values of $\hat{K}(s)$ higher than πs^2 are characteristic of clustered processes, whilst values smaller than that are found when there exists competition between events (regular pattern).

```
> set.seed(30)
> Kenvjap <- envelope(as(spjpines1, "ppp"), fun = Kest,
+   r = r, nrank = 2, nsim = 99)
> Kenvred <- envelope(as(spred, "ppp"), fun = Kest, r = r,
+   nrank = 2, nsim = 99)
> Kenvcells <- envelope(as(spccells, "ppp"), fun = Kest,
+   r = r, nrank = 2, nsim = 99)
> Kresults <- rbind(Kenvjap, Kenvred, Kenvcells)
> Kresults <- cbind(Kresults, y = rep(c("JAPANESE", "REDWOOD",
+   "CELLS"), each = length(r)))
```

Figure 7.9 shows the estimated K -function minus the theoretical value under CSR of the three point patterns that we have considered before. Note that the biological interpretations must be made cautiously because the underlying mechanisms are quite different and the scale of the interactions (if any) will probably be different for each point pattern. This is reflected in two

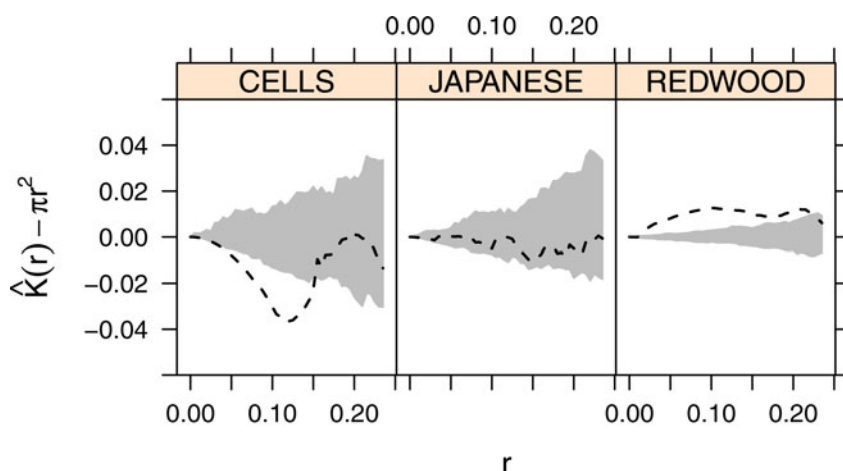


Fig. 7.9 Envelopes and observed values of Ripley's K -function for three point patterns

ways: the width of the envelopes, which reflects the variability of the process under the null hypothesis of CSR, and the scale of the interaction. This seems to exist only for the cells, which follow a regular pattern, and the redwood seeds, which seem to be clustered. The Japanese trees point pattern is compatible with CSR because the estimated K -function is contained within the envelopes.

7.4.5.1 Inhomogeneous K -Function

Baddeley et al. (2000) propose a version of the K -function for non-homogeneous point processes, in particular, for the class of point processes which are second-order reweighted-stationary, which includes IPPs. This means that the second-order intensity of two points, divided by their respective intensities, is stationary. The inhomogeneous K -function is used in Sect. 7.5.5 in the analysis of case-control point patterns.

7.5 Some Applications in Spatial Epidemiology

In this section we focus on different applications of the analysis of point patterns in Spatial Epidemiology. Gatrell et al. (1996) and Diggle (2003) describe most of the methods contained here, but a comprehensive description of spatial methods for the analysis of epidemiological data can be found in Elliott et al. (2000) and Waller and Gotway (2004). Furthermore, Chap. 10 describes the analysis of epidemiological data when they are aggregated.

The distribution of the cases of a certain disease can be regarded as the realisation of a point process, which reflects the underlying distribution of the population (which usually is not homogeneous) plus any other risk factors related to the disease and that are likely to depend on the subjects. Hence, we need to have accurate records of the locations of the disease cases, which can also include additional information on the individuals such as age, gender, and others.

In a spatial setting, the primary interest is on the spatial distribution of the cases, but any underlying risk factor that affects this spatial distribution should be taken into account. It is clear that looking solely at the spatial distribution of the cases in order to detect areas of high incidence is useless because the distribution of the cases will reflect that of the population. To overcome this problem, it would be necessary to have an estimate of the spatial distribution of the population so that it can be compared to that of the cases. For this reason, a set of controls can be randomly selected from the population at risk so that its spatial variation can be estimated Prince et al. (see, e.g. 2001).

Different authors have approached this problem in different ways. Diggle and Chetwynd (1991), for example compute the difference of the homogeneous K -function of cases and controls. Kelsall and Diggle (1995a,b) use non-parametric estimates of the distribution of the ratio between the intensities of cases and controls (i.e. the *relative risk*). Kelsall and Diggle (1998) propose a similar model and the use of binary regression and additive models to account for covariates and a smoothing term to model the residual spatial variation. More recently, Diggle et al. (2007) use the inhomogeneous K -function to compare the spatial distribution of cases and controls after accounting for the effect of relevant covariates.

Many of these methods are also covered, including new examples, and discussed in Schabenberger and Gotway (2005, pp. 103–122), Waller and Gotway (2004, Chap. 6), O’Sullivan and Unwin (2010, see the discussion in Chap. 6), Illian et al. (2008) and Cressie and Wikle (2011, Sect. 4.3). Several chapters in Gelfand et al. (2010) also the analysis of case–control and marked point patterns.

7.5.1 Case–Control Studies

As we need to estimate the spatial distribution of the population, a number of individuals can be taken at random to make a set of controls. Controls are often selected using the population register or, if it is not available, the events of another non-related disease (Diggle, 1990). Furthermore, some strategies, such as stratification and matching (Jarner et al., 2002), can be done in order to account for other sources of confounding, such as age and sex. As discussed by Diggle (2000) when matching is used in the selection of the controls, the

hypothesis of random selection from the population is violated and specific methods to handle this are required (Diggle et al., 2000; Jarner et al., 2002).

In general, we have a set of n_1 cases and n_0 controls. Conditioning on the number of cases and controls, we can assume that they are realisations of two IPP with intensities $\lambda_1(x)$ and $\lambda_0(x)$, respectively. In this setting, assuming that the distribution of cases and controls is the same means that the intensities $\lambda_1(x)$ and $\lambda_0(x)$ are equal up to a proportionality constant, which is equal to the ratio between n_1 and n_0 : $\lambda_1(x) = \frac{n_1}{n_0} \lambda_0(x)$. Note that the ratio between cases and controls is determined only by the study design.

7.5.1.1 Spatial Variation of the Relative Risk

Kelsall and Diggle (1995a,b) consider the estimator of the disease risk given by the ratio between the intensity of the cases and controls $\rho(x) = \lambda_1(x)/\lambda_0(x)$ in order to assess the variation of the risk. Under the null hypothesis of equal spatial distribution, the ratio is a constant $\rho_0 = n_1/n_0$.

Alternatively, a risk estimate $r(x)$ can be estimated by working with the logarithm of the ratio of the densities of cases and controls:

$$r(x) = \log(f(x)/g(x)), \quad (7.4)$$

$f(x) = \lambda_1(x)/\int_A \lambda_1(x) dx$ and $g(x) = \lambda_0(x)/\int_A \lambda_0(x) dx$, respectively. In this case, the null hypothesis of equal spatial distributions becomes $r(x) = 0$. The advantage of this approach is that 0 is the reference value for equal spatial distribution without regarding the number of cases and controls. Unfortunately, this presents several computational problems because the intensity of the controls may be zero at some points, as addressed by, for example, Waller and Gotway (2004, pp. 165–166).

Kelsall and Diggle (1995a) propose the use of a kernel smoothing to estimate each intensity and evaluate different alternatives to estimate the optimum bandwidth for each kernel smoothing. They conclude that the best option is to select the bandwidth by cross-validation and use the same bandwidth in both cases.

They choose the bandwidth that minimises the following criterion:

$$\begin{aligned} CV(h) = & - \int_A \hat{r}_h(x)^2 dx - 2n_1^{-1} \sum_{i=1}^{n_1} \hat{r}_h^{-i}(x_i) / \hat{f}_h^{-i}(x_i) \\ & + 2n_0^{-1} \sum_{i=n_1+1}^{n_1+n_0} \hat{r}_h^{-i}(x_i) / \hat{g}_h^{-i}(x_i), \end{aligned}$$

where the superscript $-i$ means that the function is computed by removing the i th point.

As discussed in Diggle et al. (2007), these automatic methods should be used as a guidance when estimating the bandwidth. For this reason, we have preferred to set the bandwidth manually to a value of 0.06, which provides a reasonable degree of smoothing. In the following example, we have also relabelled the point pattern marks so that the first point type is set to ‘control’. We will need this later when using function `relrisk` to compute the binary regression estimator.

```
> bwasthma <- 0.06
> pppasthma <- as(spasthma, "ppp")
> pppasthma$window <- as(spbdry, "owin")
> marks(pppasthma) <- releval(pppasthma$marks$Asthma, "control")
```

The risk ratio can be computed easily by estimating the intensity of cases and controls first, and then taking the ratio (as shown below) after using the `spkernel2d` function from **spplancs** or `density` from **spatstat**.

First of all, the point locations are divided between cases and controls and the intensities of each subset calculated for grid cells lying within the study area, using the chosen bandwidth. The **spplancs** package uses a simple form of single polygon boundary, while **spatstat** can use multiple separate polygons (**SpatialPolygons** objects can be coerced to suitable **owin** objects). In the following lines we show how to compute the intensity of cases and controls and then the relative risk using `density`.

```
> cases <- unmark(subset(pppasthma, marks(pppasthma) ==
+   "case"))
> ncases <- npoints(cases)
> controls <- unmark(subset(pppasthma, marks(pppasthma) ==
+   "control"))
> ncontrols <- npoints(controls)
> kcases <- density(cases, bwasthma)
> kcontrols <- density(controls, bwasthma)
```

The results are in an **im** object (a Pixel Image Object), which is a square grid with missing values in the points outside the study area. We can first coerce this object to a **SpatialGridDataFrame** object to hold them and coerce to a **SpatialPixelsDataFrame** to drop the missing cells.

```
> spkratio0 <- as(kcases, "SpatialGridDataFrame")
> names(spkratio0) <- "kcases"
> spkratio0$kcontrols <- as(kcontrols, "SpatialGridDataFrame")$v
> spkratio <- as(spkratio0, "SpatialPixelsDataFrame")
> spkratio$kratio <- spkratio$kcases/spkratio$kcontrols
> spkratio$logratio <- log(spkratio$kratio) - log(ncases/ncontrols)
```

To assess departure from the null hypothesis, they propose the following test statistic:

$$T = \int_A (\rho(x) - \rho_0)^2 dx.$$

This integral can be estimated up to a proportionality constant by computing $\rho(x)$ on a regular grid of points $\{s_i, i = 1, \dots, p\}$ and computing the sum of the values $\{(\rho(s_i) - \rho_0)^2, i = 1, \dots, p\}$. Hence, an estimate of T is given by

$$\hat{T} = |c| \sum_{i=1}^p (\hat{\rho}(s_i) - \hat{\rho}_0)^2,$$

where $|c|$ is the area of the cells of the grid, $\hat{\rho}_0$ is n_1/n_0 , and $\hat{\rho}(x)$ the estimate of the risk ratio.

Note that the former test is to assess whether there is constant risk all over the study region. However, risk is likely to vary spatially and another appropriate test can be done by substituting ρ_0 for $\hat{\rho}(x)$ (Kelsall and Diggle, 1995a). Now we are testing for significance of risk given that we assume that its variation is not homogeneous (i.e. equal to $\hat{\rho}(x)$) and the test statistic is

$$T = \int_A (\rho(x) - \hat{\rho}(x))^2 dx.$$

Significance of the observed value of the test statistic can be computed by means of a Monte Carlo test Kelsall and Diggle (1995b). In this test, we compute k values of the test statistic T by re-labelling cases and controls (keeping n_1 and n_0 fixed) and calculating a new risk ratio $\hat{\rho}_i(x)$ $i = 1, \dots, n$ for each new set of cases and controls. This will provide a series of values $T^1, \dots, T^k$ under the null hypothesis. If we call T^0 the value of T for the observed data set, the significance (p -value) can be computed by taking $(t+1)/(k+1)$, where t is the number of values of $T^i, i = 1, \dots, n$ greater than T^0 .

The Monte Carlo test is based on the fact that cases and controls are equally distributed under the null hypothesis. In that case, if we change the label of a case to be a control (or vice versa), the new set of cases (or controls) still have the same spatial distribution and will have the same risk function $\rho(x)$. If that is not the case, then the re-labelling of cases and controls will produce different risk functions.

We will use the previous `SpatialPixelsDataFrame`, which only contain points in the study area, to set up objects to hold the results for the re-labelled cases and controls:

```
> niter <- 99
> ratio <- rep(NA, niter)
> pvaluemap <- rep(0, nrow(spkratio))
> rlabelratio <- matrix(NA, nrow = niter, ncol = nrow(spkratio))
```

The probability map is calculated by repeating the re-labelling process `niter` times, and tallying the number of times that the observed kernel density ratio is less than the re-labelled ratios. In the loop, the first commands carry out the re-labelling from the full set of points, and the remainder calculate the ratio and store the results:

```

> set.seed(1)
> for (i in 1:niter) {
+   pppasthma0 <- rlabel(pppasthma)
+   casesrel <- unmark(subset(pppasthma0, marks(pppasthma0) ==
+     "case"))
+   controlsrel <- unmark(subset(pppasthma0, marks(pppasthma0) ==
+     "control"))
+   kcasesrel <- density(casesrel, bwasthma)
+   kcontrolsrel <- density(controlsrel, bwasthma)
+   kratiorel <- eval.im(kcasesrel/kcontrolsrel)
+   rlabelratio[i, ] <- as(as(kratiorel, "SpatialGridDataFrame"),
+     "SpatialPixelsDataFrame")$v
+   pvaluemap <- pvaluemap + (spkratio$kratio < rlabelratio[i,
+     ])
+ }

```

Figure 7.10 shows the kernel ratio of cases and controls, using a bandwidth of 0.06, as discussed before.

```

> cellsize <- kcontrols$xstep * kcontrols$ystep
> ratorho <- cellsize * sum((spkratio$kratio - ncases/ncontrols)^2)
> ratio <- cellsize * apply(rlabelratio, 1, function(X,
+   rho0) {
+     sum((X - rho0)^2)
+ }, rho0 = ncases/ncontrols)
> pvaluerho <- (sum(ratio > ratorho) + 1)/(niter + 1)

```

The results for the test with null hypothesis $\rho = \hat{\rho}_0$ turned out to be non-significant (p -value of 0.61), which means that the observed risk ratio is consistent with a constant risk ratio. In principle, this agrees with the fact that Diggle and Rowlingson (1994) did not find a significant association with distance from main roads or two of the pollution sources and only a possible association with the remaining site, which should be further investigated. However, they found some relationship with other risk factors, but these were not of a spatial nature and, hence, this particular test is unable to detect it.

Had the p -value of the test been significant, 90% point confidence surfaces could be computed in a similar way to the envelopes shown before, but considering the different values of the estimates of $\rho(x)$ under random labelling and computing the p -value at each point. The procedure computes, for each point x_j in the grid, the proportion of values $\hat{\rho}_i(x_j)$ that are lower than $\hat{\rho}(x_j)$, where the $\hat{\rho}_i(x_j), i = 1, \dots, R$ are the estimated ratios obtained by re-labelling cases and controls. Finally, the 0.05 and 0.95 contours of the p -value surface can be displayed on the plot of $\hat{\rho}(x)$ to highlight areas of significant low and high risk, respectively. This is shown in Fig. 7.10.

The contour lines at a given value can be obtained using function `contourLines`, which takes an `image` object. This will generate contour lines that can be converted to `SpatialLinesDataFrame` objects so that they can be added to a plot as a layout.

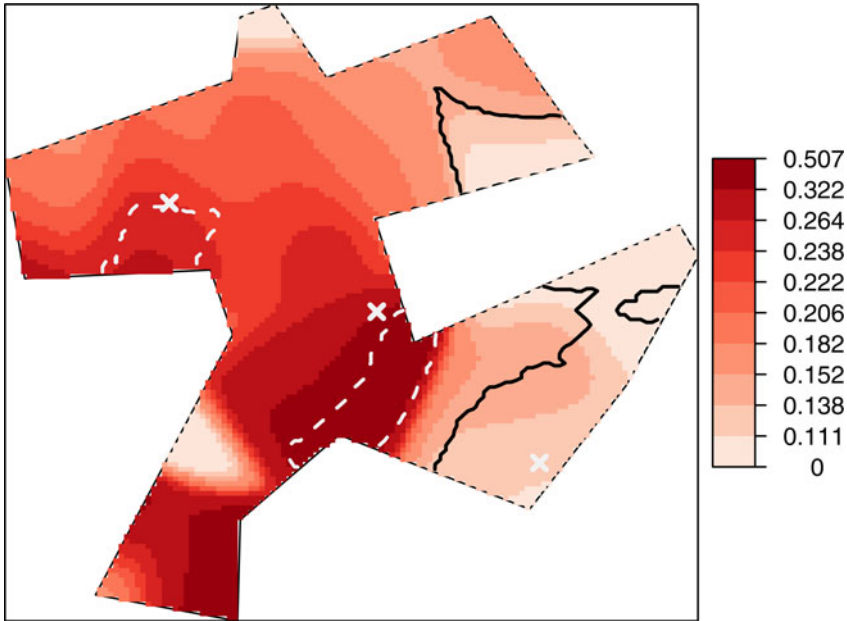


Fig. 7.10 Kernel ratio of the intensity of cases and controls. The continuous and dashed lines show the surfaces associated with 0.95 and 0.05 p -values, respectively, grey crosses mark the pollution sources. The value of $\hat{\rho}_0$ which marks a flat constant risk is 0.2

```
> spkratio$pvaluemap <- (pvaluemap + 1)/(niter + 1)
> imgpvalue <- as.image.SpatialGridDataFrame(spkratio["pvaluemap"])
> clpvalue <- contourLines(imgpvalue, levels = c(0, 0.05,
+ 0.95, 1))
> cl <- ContourLines2SLDF(clpvalue)
```

7.5.2 Binary Regression Estimator

Kelsall and Diggle (1998) propose a binary regression estimator to estimate the probability of being a case at a given location, which can be easily extended to allow for the incorporation of covariates. In principle, the probabilities can be estimated by assuming that we have a variable Y_i , which labels cases ($y_i = 1$) and controls ($y_i = 0$) in a set of $n = n_1 + n_2$ events. Conditioning on the point locations, Y_i is a realisation of a Bernoulli variable Y_i with probability

$$P(Y_i = 1 | X_i = x_i) = p(x_i) = \frac{\lambda_1(x_i)}{\lambda_0(x_i) + \lambda_1(x_i)}.$$

In practise, the following Nadaraya–Watson kernel estimator can be used:

$$\hat{p}_h(x) = \frac{\sum_{i=1}^n h^{-2} \kappa_h((x - x_i)/h) y_i}{\sum_{i=1}^n h^{-2} \kappa_h((x - x_i)/h)}, \quad (7.5)$$

where $\kappa_h(u)$ is a kernel function. Note that $p(x)$ is related to the log-ratio relative risk $r(x)$ as follows:

$$\text{logit}(p(x)) = \log\left(\frac{p(x)}{1 - p(x)}\right) = \log\left(\frac{\lambda_1(x)}{\lambda_0(x)}\right) = r(x) + \log(n_1/n_0).$$

$\hat{p}_h(x)$ can be estimated as

$$\hat{p}_h(x) = \frac{\hat{\lambda}_1(x)}{\hat{\lambda}_1(x) + \hat{\lambda}_0(x)}.$$

To estimate the bandwidth that appears in this new estimator, Kelsall and Diggle (1998) suggest another cross-validation criterion based on the value of h that minimises

$$CV(h) = \left[\prod_{i=1}^n \hat{p}_h^{-i}(x_i)^{y_i} (1 - \hat{p}_h^{-i}(x_i))^{1-y_i} \right]^{-1/n}.$$

This criterion is available in function `bw.relrisk` in the `spatstat` package.

```
> rrbw <- bw.relrisk(pppasmhma, hmax = 0.5)
```

Using the new criterion we obtained a bandwidth of 0.209. However, we believe that this value would over-smooth the data and we have set it to 0.06, as in the estimation of the relative risk ratio. The estimator for $p(x)$ can be computed easily, as is shown below. Figure 7.11 shows the resulting estimate.

```
> bwasthmap <- 0.06
> rr <- relrisk(pppasmhma, bwasthmap)
> spkratio$prob <- as(as(rr, "SpatialGridDataFrame"),
  "SpatialPixelsDataFrame")$v
```

7.5.3 Binary Regression Using Generalised Additive Models

This formulation allows the inclusion of covariates in the model by means of standard logistic regression. In addition, the residual spatial variation can be modelled by including a smooth spatial function. In other words, if u is a

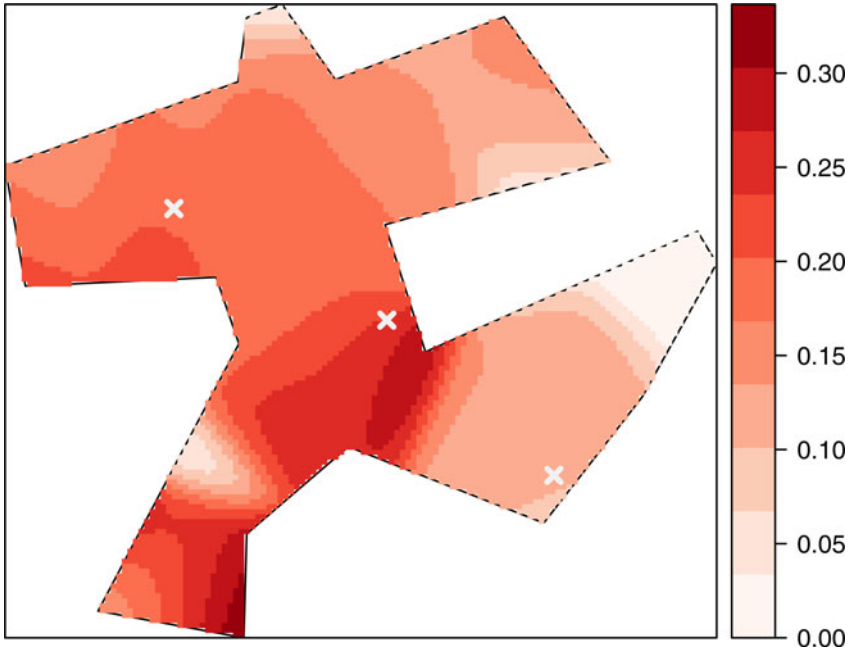


Fig. 7.11 Binary regression estimator using the probability of being a case at every grid cell in the study region

vector of covariates observed at location x and $g(x)$ is a smooth function not dependent on the covariates, the formulation is

$$\text{logit}(p(x)) = u'\beta + g(x).$$

If the covariates are missing, the former expression is just another way of estimating the probability surface. Kelsall and Diggle (1998) estimate $g(x)$ using a kernel weighted regression. We have used package **mgcv** (Wood, 2006) to fit the Generalised Additive Model (GAM) models but, given that this package lacks the same non-parametric estimator used in Kelsall and Diggle (1998), we have preferred the use of a penalised spline instead.

The following example shows how to fit a GAM using the distance of the events to the pollution sources and main roads, and controlling for known and possible risk factors such as gender, age, previous events of hay fever, and having at least one smoker in the house. Rows have been filtered so that only children with a valid value of Gender (1 or 2) are used. We have included the distance as a proxy of the actual exposure to any risk factor caused by the pollution sources or the roads. Other models that consider a special modelling for the distance are considered later.

```
> spasthma$y <- as.integer(!as.integer(spasthma$Asthma) -
+ 1)
```

```

> ccasthma <- coordinates(spasthma)
> spasthma$x1 <- ccasthma[, 1]
> spasthma$x2 <- ccasthma[, 2]
> spasthma$dist1 <- sqrt(spasthma$d2source1)
> spasthma$dist2 <- sqrt(spasthma$d2source2)
> spasthma$dist3 <- sqrt(spasthma$d2source3)
> spasthma$droads <- sqrt(spasthma$roaddist2)
> spasthma$smoking <- as.factor(as.numeric(spasthma$Nsmokers >
+ 0))
> spasthma$Genderf <- as.factor(spasthma$Gender)
> spasthma$HayFeverf <- as.factor(spasthma$HayFever)

> library(mgcv)
> gasthma <- gam(y ~ 1 + dist1 + dist2 + dist3 + droads +
+   Genderf + Age + HayFeverf + smoking + s(x1, x2),
+   data = spasthma[spasthma$Gender == 1 | spasthma$Gender ==
+   2, ], family = binomial)

> summary(gasthma)

```

Family: binomial

Link function: logit

Formula:

```

y ~ 1 + dist1 + dist2 + dist3 + droads + Genderf + Age + HayFeverf +
    smoking + s(x1, x2)

```

Parametric coefficients:

	Estimate	Std. Error	z value	Pr(> z)
(Intercept)	-2.0326790	0.9196841	-2.210	0.0271 *
dist1	0.9822130	6.0721457	0.162	0.8715
dist2	-9.5791583	5.7719999	-1.660	0.0970 .
dist3	11.2248253	7.8743652	1.425	0.1540
droads	0.0001479	0.0001717	0.861	0.3890
Genderf2	-0.3476861	0.1562020	-2.226	0.0260 *
Age	-0.0679031	0.0382349	-1.776	0.0757 .
HayFeverf1	1.1881333	0.1875415	6.335	2.37e-10 ***
smoking1	0.1651213	0.1610364	1.025	0.3052

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Approximate significance of smooth terms:

	edf	Ref.df	Chi.sq	p-value
s(x1,x2)	2.001	2.001	7.002	0.0302 *

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R-sq.(adj) = 0.0403 Deviance explained = 4.94%

UBRE score = -0.12348 Scale est. = 1 n = 1283

The results show that the significant variables are the presence of reported hay fever (p -value $2.4e-10$) and gender (p -value 0.026). The coefficient of the second pollution source is marginally significant (p -value 0.097). The smoothed residual term using splines is significant (p -value 0.0302), which

suggests that there may have been some residual spatial variation unexplained in the generalised linear model.

7.5.4 Point Source Pollution

In the previous model, we have shown how to consider the exposure to a number of pollution sources by including the distance as a covariate in the model. However, this approach does not allow for a more flexible parametric modelling of the exposure according to the distance to a pollution source. Diggle (1990) proposed the use of an IPP for the cases in which their intensity accounts for the distance to the pollution sources. In particular, the intensity is as follows:

$$\lambda_1(x) = \rho \lambda_0(x) f(x - x_0; \theta),$$

ρ measures the overall number of events per unit area, $\lambda_0(x)$ is the spatial variation of the underlying population (independent of the effect of the source), and $f(x - x_0; \theta)$ is a function of the distance from point x to the location of the source x_0 and has parameters θ . Diggle (1990) uses a decaying function with distance

$$f(x - x_0; \alpha, \beta) = 1 + \alpha \exp(-\beta \|x - x_0\|^2).$$

Parameters ρ , α , and β of $\lambda_1(x)$ can be estimated by maximising the likelihood of the IPP, assuming that $\lambda_0(x)$ is estimated by kernel smoothing taking a certain value h_0 of the bandwidth. That is, the value of h_0 is not obtained by the maximisation procedure, but choosing a reasonable value for h_0 can be difficult and it can have an important impact on the results.

A slightly different approach that does not require the choice of a bandwidth is considered in Diggle and Rowlingson (1994). It is based on the previous scenario, but conditioning on the location of cases and controls to model the probability of being a case at location x :

$$p(x) = \frac{\lambda_1(x)}{\lambda_1(x) + \lambda_0(x)} = \frac{\rho f(x - x_0; \alpha, \beta)}{1 + \rho f(x - x_0; \alpha, \beta)}.$$

As in the previous scenario, the remaining parameters of the model can be estimated by maximising the log-likelihood:

$$L(\rho, \theta) = \sum_{i=1}^{n_1} \log(p(x_i)) + \sum_{j=1}^{n_0} \log(1 - p(x_j)).$$

This model can be fitted using function `tribble` from package `splancs`. Given that $\lambda_0(x)$ vanishes we only need to pass the distances to the source and the labels of cases and controls.

To compare models that may include different sets of pollution sources or covariates, Diggle and Rowlingson (1994) compare the difference of the log-likelihoods by means of a chi-square test. The following example shows the results for the exposure model with distance to source two and another model with only the covariate hay fever.

```
> D2_mat <- as.matrix(spsthma$dist2)
> RHO <- ncases/ncontrols
> expsource2 <- tribble(ccflag = spsthma$y, vars = D2_mat,
+   rho = RHO, alphas = 1, betas = 1)
> print(expsource2)

Call:
tribble(ccflag = spsthma$y, vars = D2_mat, alphas = 1, betas = 1,
  rho = RHO)
Kcode = 2

Distance decay parameters:
      Alpha      Beta
[1,] 1.305824 25.14672

rho parameter : 0.163395847627903

log-likelihood : -580.495955916672
null log-likelihood : -581.406203518987

D = 2(L-Lo) : 1.82049520462942

> Hay_mat <- as.matrix(spsthma$HayFever)
> exphay <- tribble(ccflag = spsthma$y, rho = RHO, covars = Hay_mat,
+   thetas = 1)
> print(exphay)

Call:
tribble(ccflag = spsthma$y, rho = RHO, covars = Hay_mat, thetas = 1)
Kcode = 2

Covariate parameters:
[1] 1.103344

rho parameter : 0.163182953009354

log-likelihood : -564.368250327801
null log-likelihood : -581.406203518987

D = 2(L-Lo) : 34.0759063823707
```

As the output shows, the log-likelihood for the model with exposure to source 2 is -580.5 , whilst for the model with the effect of hay fever is only -564.4 . This means that there is a significant difference between the two models and that the model that accounts for the effect of hay fever is preferable. Even though the second source has a significant impact on the increase of the cases of asthma, its effect is not as important as the effect of having suffered from hay fever. However, another model could be proposed to account for both effects at the same time.

```
> expsource2hay <- tribble(ccflag = spasthma$y, vars = D2_mat,
+   rho = RHO, alphas = 1, betas = 1, covars = Hay_mat,
+   thetas = 1)
```

This new model (output not shown) has a log-likelihood of -563 , with two more parameters than the model with hay fever. Hence, the presence of the second source has a small impact on the increase of cases of asthma after adjusting for the effect of hay fever, which can be regarded as the main factor related to asthma, and the model with hay fever only should be preferred. The reader is referred to Diggle and Rowlingson (1994) and Diggle (2003, p. 137) for more details on how the models can be compared and results for other models.

These types of models are extended by Diggle et al. (1997), who consider further options for the choice of the function $f(x - x_0, \alpha, \beta)$ to accommodate different spatial variants of the risk around the source.

In our experience, these models can be very sensitive to the initial values for certain data sets, especially if they are sparse. Hence, it is advised to fit the model using different values for the initial values to ensure that the algorithm is not trapped in a local maximum of the likelihood.

7.5.4.1 Assessment of General Spatial Clustering

As discussed by Diggle (2000), it is important to distinguish between spatial variation of the risk and clustering. Spatial variation occurs when the risk is not homogeneous in the study region (i.e. all individuals do not have the same risk) but cases appear independently of each other according to this risk surface, whilst clustering occurs when the occurrence of cases is not at random and the presence of a case increases the probability of other cases appearing nearby.

The former methods allow us to inspect a raised incidence in the number of cases around certain pre-specified sources. However, no such source is identified a priori, and a different type of test is required to assess clustering in the cases.

Diggle and Chetwynd (1991) propose a test based on the homogeneous K -function to assess clustering of the cases as compared to the controls. The null hypothesis is as before, that is cases and controls are two IPP that have the same intensities up to a proportionality constant. Hence, they will produce

the same K -functions. Note that the inverse is not always true, that is two point processes with the same homogeneous K -function can be completely different (Baddeley and Silverman, 1984). Diggle and Chetwynd (1991) take the difference of the two K -functions to evaluate whether the cases tend to cluster after considering the inhomogeneous distribution of the population: $D(s) = K_1(s) - K_0(s)$, where $K_1(s)$ and $K_0(s)$ are the homogeneous K -functions of cases and controls, respectively.

The test statistic is

$$D = \int_A \frac{D(s)}{\text{var}[D(s)]^{1/2}} \text{d}s,$$

where $\text{var}[D(s)]$ is the variance of $D(s)$ under the null hypothesis. Diggle and Chetwynd (1991) compute the value of this variance under random labelling of cases and controls so that the significance of the test statistic can be assessed. Note that under the null hypothesis the expected value of the test statistic D is zero. Finally, the integral is approximated in practice by a discrete sum at a set of finite distances, as the T statistic was computed before.

Significant departure from 0 means that there is a difference in the distribution of cases and controls, with clustering occurring at the range of those distances for which $D(s) > 0$. Furthermore, pointwise envelopes can be provided for the test statistic by the same Monte Carlo test so that the degree of clustering can be assessed. Function `Kenv.label` (in **splancs**) also provides envelopes for the difference of the K -functions but it does not carry out any test of significance.

A similar test can be implemented using `envelope` in **spatstat**. First of all, we will define function `Kdif` to compute the difference of the K -functions. This function will take the point pattern, a vector of distance $\mathbf{r}$ at which $D(s)$ is computed and the desired edge correction to be used when calling to `Kest`.

```
> Kdif <- function(Xppp, r, cr = "border") {
+   k1 <- Kest(Xppp[marks(Xppp) == "case"], r = r, correction = cr)
+   k2 <- Kest(Xppp[marks(Xppp) == "control"], r = r,
+     correction = cr)
+   res <- data.frame(r = r, D = k1[[cr]] - k2[[cr]])
+   return(fv(res, valu = "D", fname = "D"))
+ }
```

In the call to `envelope` we will also keep the simulated values in order to compute the variance of $D(s)$ used in the test statistic.

```
> r <- seq(0, 0.15, by = 0.01)
> envKdif <- envelope(ppp asthma, Kdif, r = r, nsim = 99,
+   cr = "iso", nrank = 2, savefuns = TRUE, simulate =
+     expression(rlabel(ppp asthma)))
```

```
> khcases <- Kest(cases, r = r, correction = "isotropic")
> khcontrols <- Kest(controls, r = r, correction = "isotropic")
```

Using the saved values `ew` are able to estimate the variance of $D(s)$, the test statistic for the observed data set and the tests statistics for the relabelled data sets to conduct the Monte Carlo test.

```
> simfunns <- as.data.frame(attr(envKdif, "simfunns"))[,
+   -1]
> khcovdiag <- apply(simfunns, 1, var)
> T0 <- sum(((khcases$iso - khcontrols$iso)/sqrt(khcovdiag))[-1])
> T <- apply(simfunns, 2, function(X) {
+   sum((X/sqrt(khcovdiag))[-1])
+ })
> pvalue <- (sum(T > T0) + 1)/(niter + 1)
```

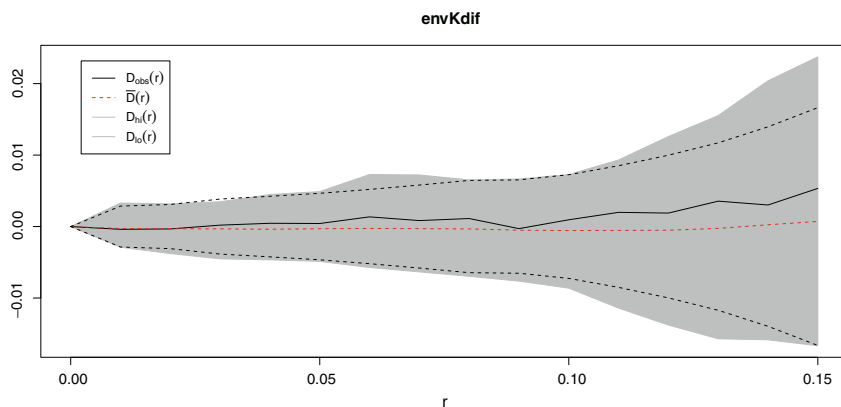


Fig. 7.12 Actual value of $D(s)$ with approximate 95 % confidence intervals (*dashed black lines*) and 95 % envelopes (*gray area*)

The p -value for this data set is 0.31, meaning that there is no significant difference between the distribution of cases and controls. The outcome is consistent with the fact that the observed K -function is contained by the simulation envelopes and approximated 95 % confidence intervals, as shown in Fig. 7.12.

7.5.5 Accounting for Confounding and Covariates

Diggle et al. (2007) propose a similar way of assessing clustering by means of the inhomogeneous K -function $K_{I,\lambda}(s)$ (Baddeley et al., 2000). For an IPP with intensity $\lambda(x)$, it can be estimated as

$$\hat{K}_{I,\lambda}(s) = |A|^{-1} \sum_{i=1}^n \sum_{j \neq i} w_{ij}^{-1} \frac{|\{x_j : d(x_i, x_j) \leq s\}|}{\lambda(x_i)\lambda(x_j)}.$$

Note that this estimator is a generalisation of the estimator of the homogeneous K -function from expression (7.3) and that in fact reduces to it when instead of an IPP we have an HPP (the intensity becomes $\lambda(x) = \lambda$). Similarly, the value of $K_{I,\lambda}(s)$ for an IPP with intensity $\lambda(s)$ is πs^2 .

In practise the intensity $\lambda(x)$ needs to be estimated either parametrically or non-parametrically, so that the estimator that we use is

$$\hat{K}_{I,\hat{\lambda}}(s) = |A|^{-1} \sum_{i=1}^n \sum_{j \neq i} w_{ij}^{-1} \frac{|\{x_j : d(x_i, x_j) \leq s\}|}{\hat{\lambda}(x_i)\hat{\lambda}(x_j)}.$$

Values of $\hat{K}_{I,\hat{\lambda}}(s)$ higher than πs^2 will mean that the point pattern shows more aggregation than that shown by $\lambda(x)$ and values lower than πs^2 reflect more relative homogeneity.

To be able to account for confounding and risk factors, Diggle et al. (2007) propose the use of a semi-parametric estimator of the intensity in a case–control setting. The basic assumption is that controls are drawn from an IPP with spatially varying intensity $\lambda_0(x)$. The cases are assumed to appear as a result of the inhomogeneous distribution of the population, measured by $\lambda_0(x)$, plus other risk factors, measured by a set of spatially referenced covariates $z(x)$. Hence, the intensity of the cases is modelled as

$$\lambda_1(x) = \exp\{\alpha + \beta z(x)\} \lambda_0(x),$$

where α and β are the intercept and covariate coefficients of the model, respectively. When there are no covariates, the intensity of the cases reduces to

$$\lambda_1(x) = \frac{n_1}{n_0} \lambda_0(x).$$

Note that it is possible to use any generic non-negative function $f(z(x); \theta)$ to account for other types of effects

$$\lambda_1(x) = \lambda_0(x) f(z(x); \theta).$$

This way it is possible to model non-linear and additive effects.

To estimate the parameters that appear in the intensity of the cases, we can use the same working variables Y_i that we have used before (see the binary regression estimator in Sect. 7.5.2), with values 1 for cases and 0 for controls. Conditioning on the locations of cases and controls, Y_i is a realisation of a Bernoulli process with probability

$$P(Y_i = 1|x_i, z(x)) = p(x_i) = \frac{\lambda_1(x)}{\lambda_0(x) + \lambda_1(x)} = \frac{\exp\{\alpha + \beta z(x)\}}{1 + \exp\{\alpha + \beta z(x)\}}. \quad (7.6)$$

Hence, conditioning on the locations of cases and controls, the problem is reformulated as a logistic regression and α and β can be estimated using function `glm`.

Baddeley et al. (2000) estimate the intensity non-parametrically and use the same data to estimate both the intensity and the inhomogeneous K -function, but Diggle et al. (2007) show that this can give poor performance in detecting clustering. This problem arises from the difficulty of disentangling inhomogeneous spatial variation of process from clustering of the events (Cox, 1955). Another problem that appears in practise is that the intensities involved must be bounded away from zero. If kernel smoothing is used, a good alternative to the quartic kernel is a Gaussian bivariate kernel.

The following piece of code shows how to estimate the inhomogeneous K -function both without covariates and accounting for hay fever.

```
> glmasthma <- glm(y ~ HayFeverf, data = spasthma, family = "binomial")
> prob <- fitted(glmasthma)
> weights <- exp(glmasthma$linear.predictors)
> lambda0 <- interp.im(kcontrols, coords(cases)[, 1], coords(cases)[,
+ 2])
> lambda1 <- weights[marks(pppasthma) == "case"] * lambda0
> ratiocc <- ncases/ncontrols
> kihnocov <- Kinhom(cases, ratiocc * lambda0, r = r)
> kih <- Kinhom(cases, lambda1, r = r)
```

To assess for any residual clustering left after adjusting for covariates, Diggle et al. (2007) suggest the following test statistic:

$$D = \int_0^{s_0} \frac{\hat{K}_{I, \hat{\lambda}_1}(s) - E[s]}{\text{var}(K_{I, \lambda}(s))^{1/2}} ds,$$

$E[s]$ is the expectation of $\hat{K}_{I, \hat{\lambda}_1}(s)$ under the null hypothesis. In principle, it should be πs^2 , but when kernel estimators are used in the computation of the intensity, the estimate of $K_{I, \lambda}(s)$ may be biased. $E[s]$ can be computed as the average of all the estimates $\hat{K}_{I, \hat{\lambda}_1}(s)$, which have been obtained during the Monte Carlo simulations (as explained below). $\text{var}(K_{I, \lambda}(s))$ can be computed in a similar way.

The Monte Carlo test proposed by Diggle et al. (2007) is similar to the one that we used in the homogeneous case (see Sect. 7.5.4.1), with the difference that the re-labelling must be done taking into account the effects of the covariates. That is, when we relabel cases and controls, the probability of being a case will not be the same for all points but it will depend on the values of $z(x)$. In particular, these probabilities are given by (7.6). The values of the covariates are fixed to the values obtained by fitting the model with the observed data set (i.e. they are not re-estimated when the points are re-

labelled) because we are only interested in testing for the spatial variation and not that related to the estimation of the coefficients of the covariates.

```
> rlabelp <- function(Xppp, ncases, prob) {
+   idxsel <- sample(1:npoints(Xppp), ncases, prob = prob)
+   marks(Xppp) <- "control"
+   marks(Xppp)[idxsel] <- "case"
+   return(Xppp)
+ }

> KIlambda <- function(Xppp, r, cr = "iso", weights, sigma) {
+   idxrel <- marks(Xppp) == "case"
+   casesrel <- unmark(Xppp[idxrel])
+   controlsrel <- unmark(Xppp[!idxrel])
+   lambda0rel <- interp.im(density(controlsrel, sigma),
+     coords(casesrel)[, 1], coords(casesrel)[, 2])
+   lambda1rel <- weights[idxrel] * lambda0rel
+   KI <- Kinhom(casesrel, lambda1rel, r = r, correction = cr)
+   res <- data.frame(r = r, KI = KI[[cr]])
+   return(fv(res, valu = "KI", fname = "K_[I,lambda]"))
+ }

> set.seed(4567)
> envKInocov <- envelope(pppasmhma, KIlambda, r = r, cr = "iso",
+   weights = weights, sigma = bwasmhma, nsim = 99, nrank = 2,
+   savefuns = TRUE, simulate = expression(rlabelp(pppasmhma,
+     ncases = ncases, prob = rep(ratioocc, npoints(pppasmhma))))))
> envKIcov <- envelope(pppasmhma, KIlambda, r = r, cr = "iso",
+   weights = weights, sigma = bwasmhma, nsim = 99, nrank = 2,
+   savefuns = TRUE, simulate = expression(rlabelp(pppasmhma,
+     ncases = ncases, prob = prob)))

> kinhomrelnocov <- as.data.frame(attr(envKInocov, "simfuns")),
+   -1]
> kinhomrel <- as.data.frame(attr(envKIcov, "simfuns")),
+   -1]

> kinhsdnocov <- apply(kinhomrelnocov, 1, sd)
> D0nocov <- sum(((envKInocov$obs - envKInocov$mmean)/kinhsdnocov)[-1])
> Dnocov <- apply(kinhomrelnocov, 2, function(X) {
+   sum(((X - envKInocov$mmean)/kinhsdnocov)[-1])
+ })
> pvaluenocov <- (sum(Dnocov > D0nocov) + 1)/(niter + 1)

> kinhsd <- apply(kinhomrel, 1, sd)
> D0 <- sum(((envKIcov$obs - envKIcov$mmean)/kinhsd)[-1])
> D <- apply(kinhomrel, 2, function(X) {
+   sum(((X - envKIcov$mmean)/kinhsd)[-1])
+ })
> pvalue <- (sum(D > D0) + 1)/(niter + 1)
```

Figure 7.13 shows the estimated values of the inhomogeneous K -function plus 95% envelopes under the null hypothesis. In both cases there are no signs of spatial clustering. The p -values are 0.09 (no covariates) and 0.04 (with hay fever). The differences in the p -values are due to the fact that we are adjusting for hay fever. This is consistent with the plots in Fig. 7.13.

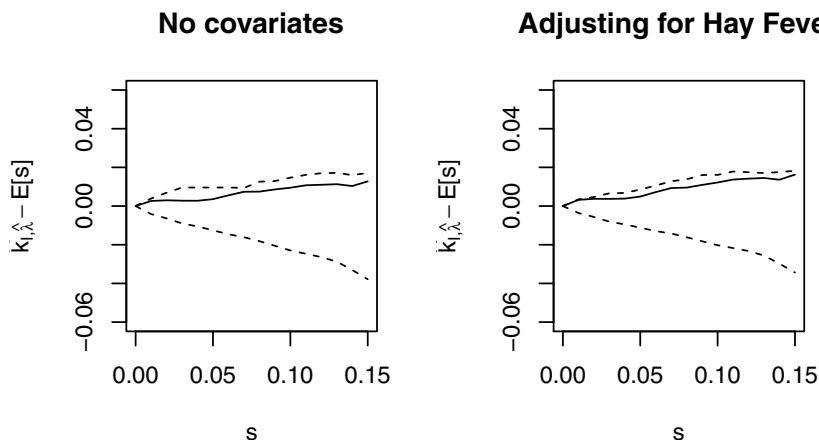


Fig. 7.13 Results of the test based on the inhomogeneous K -function for the asthma data set. The intensity has been modulated to account for the effect of suffering from hay fever

7.6 Further Methods for the Analysis of Point Patterns

In this chapter we have just covered some key examples but the analysis of point patterns with R goes beyond this. Other important problems that we have not discussed here are the analysis of marked point processes (Schabenberger and Gotway 2005, pp. 118–122; Diggle 2003, pp. 82–85), spatio-temporal analysis (see Schabenberger and Gotway 2005, pp. 442–445; Diggle 2006), and complex model fitting and simulation from different point processes (as extensively discussed in Möller and Waagepetersen, 2003). Baddeley et al. (2005) provide a recent compendium of theoretical problems and applications of the analysis of point patterns, including a description of package **spatstat**. Some of the examples described therein should be reproducible using the contents of this chapter. Gelfand et al. (2010) devote several chapters to the analysis of spatial point patterns, including model fitting, marked and spatio-temporal point patterns.

The analysis of spatio-temporal point patterns can be conducted with a number of packages. Package **spacetime** provides some basic classes for spatio-temporal point patterns with some basic subsetting and plotting capabilities **spplans** provides functions for spatio-temporal kernel smoothing and the homogeneous spatio-temporal K -functions. Package **stpp** includes a number of functions to simulate and visualize spatio-temporal point patterns (including the possibility of creating animations) and compute the space-time inhomogeneous K -function, which can be used to assess clustering in space and time (Gabriel and Diggle, 2009). Package **lgcp** focuses on log-Gaussian Cox Processes and it implements functions for model fitting and inference for spatial

and spatio-temporal point processes. Finally, **splancs** includes data types for three-dimensional and spatio-temporal point patterns.

The Spatial and Spatio-Temporal Task Views contain a list of other packages for the analysis and visualisation of point patterns. The reader is referred there for updated information.

Chapter 8

Interpolation and Geostatistics

8.1 Introduction

Geostatistical data are data that could in principle be measured anywhere, but that typically come as measurements at a limited number of observation locations: think of gold grades in an ore body or particulate matter in air samples. The pattern of observation *locations* is usually not of primary interest, as it often results from considerations ranging from economical and physical constraints to being ‘representative’ or random sampling varieties. The interest is usually in inference of aspects of the variable that have not been measured such as maps of the estimated values, exceedance probabilities or estimates of aggregates over given regions, or inference of the process that generated the data. Other problems include monitoring network optimisation: where should new observations be located or which observation locations should be removed such that the operational value of the monitoring network is maximised.

Typical spatial problems where geostatistics are used are the following:

- The estimation of ore grades over mineable units, based on drill hole data
- Interpolation of environmental variables from sample or monitoring network data (e.g. air quality, soil pollution, ground water head, hydraulic conductivity)
- Interpolation of physical or chemical variables from sample data
- Estimation of spatial averages from continuous, spatially correlated data

In this chapter we use the Meuse data set used by Burrough and McDonnell (1998). The notation we use follows mostly that of Christensen (1991), as this text most closely links geostatistics to linear model theory. Good texts on geostatistics are Chilès and Delfiner (2012), Christensen (1991), Cressie (1993), and Journel and Huijbregts (1978). More applied texts are, for example Isaaks and Strivastava (1989), Goovaerts (1997), and Deutsch and Journel (1992).

Geostatistics deals with the analysis of random fields $Z(s)$, with Z random and s the non-random spatial index. Typically, at a limited number of sometimes arbitrarily chosen sample locations, measurements on Z are available, and prediction (interpolation) of Z is required at non-observed locations s_0 , or the mean of Z is required over a specific region B_0 . Geostatistical analysis involves estimation and modelling of spatial correlation (covariance or semivariance), and evaluating whether simplifying assumptions such as stationarity can be justified or need refinement. More advanced topics include the conditional simulation of $Z(s)$, for example over locations on a grid, and model-based inference, which propagates uncertainty of correlation parameters through spatial predictions or simulations.

Much of this chapter will deal with package **gstat**, because it offers the widest functionality in the geostatistics curriculum for R: it covers variogram cloud diagnostics, variogram modelling, everything from global simple kriging to local universal cokriging, multivariate geostatistics, block kriging, indicator and Gaussian conditional simulation, and many combinations. Other R packages that provide additional geostatistical functionality are mentioned where relevant, and discussed at the end of this chapter.

8.2 Exploratory Data Analysis

Spatial exploratory data analysis starts with the plotting of maps with a measured variable. To express the observed value, we can use colour or symbol size:

```
> library(lattice)
> library(sp)
> data(meuse)
> coordinates(meuse) <- c("x", "y")
> spplot(meuse, "zinc", do.log = T, colorkey = TRUE)
> bubble(meuse, "zinc", do.log = T, key.space = "bottom")
```

to produce plots with information similar to that of Fig. 3.8.

The evident structure here is that zinc concentration is larger close to the river Meuse banks. In case of an evident spatial trend, such as the relation between top soil zinc concentration and distance to the river here, we can also plot maps with fitted values and with residuals (Cleveland, 1993), as shown in Fig. 8.1, obtained by

```
> xyplot(log(zinc) ~ sqrt(dist), as.data.frame(meuse))
> zn.lm <- lm(log(zinc) ~ sqrt(dist), meuse)
> meuse$fitted.s <- predict(zn.lm, meuse) - mean(predict(zn.lm,
+   meuse))
> meuse$residuals <- residuals(zn.lm)
> spplot(meuse, c("fitted.s", "residuals"))
```

where the formula $y \sim x$ indicates dependency of y on x . This figure reveals that although the trend removes a large part of the variability, the residuals do

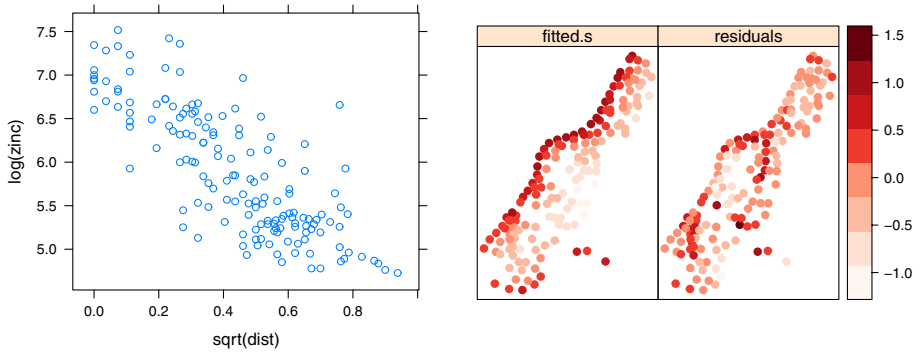


Fig. 8.1 Zinc as a function of distance to river (*left*), and fitted-residual maps (fitted.s: mean subtracted) for the linear regression model of log zinc and square-root transformed distance to the river

not appear to behave as spatially unstructured or white noise: residuals with a similar value occur regularly close to another. More exploratory analysis will take place when we further analyse these data in the context of geostatistical models; first we deal with simple, non-geostatistical interpolation approaches.

8.3 Non-geostatistical Interpolation Methods

Usually, interpolation is done on a regular grid. For the Meuse data set, coordinates of points on a regular grid are already defined in the `meuse.grid` data.frame, and are converted into a `SpatialPixelsDataFrame` by

```
> data(meuse.grid)
> coordinates(meuse.grid) <- c("x", "y")
> meuse.grid <- as(meuse.grid, "SpatialPixelsDataFrame")
```

Alternatively, we could interpolate to individual points, sets of irregularly distributed points, or to averages over square or irregular areas (Sect. 8.5.6).

8.3.1 Inverse Distance Weighted Interpolation

Inverse distance-based weighted interpolation (IDW) computes a weighted average,

$$\hat{Z}(s_0) = \frac{\sum_{i=1}^n w(s_i) Z(s_i)}{\sum_{i=1}^n w(s_i)},$$

where weights for observations are computed according to their distance to the interpolation location,

$$w(s_i) = ||s_i - s_0||^{-p},$$

with $|| \cdot ||$ indicating Euclidean distance and p an inverse distance weighting power, defaulting to 2. If s_0 coincides with an observation location, the observed value is returned to avoid infinite weights.

The inverse distance power determines the degree to which the nearer point(s) are preferred over more distant points; for large values IDW converges to the one-nearest-neighbour interpolation. It can be tuned, for example using cross validation (Sect. 8.7.1). IDW can also be used within local search neighbourhoods (Sect. 8.5.5).

Because the **spatstat** package also offers a function called **idw**, we disambiguate the two, should **spatstat** have been loaded into the session workspace, by calling the **gstat** function using the `::` operator to choose the desired **idw**:

```
> library(gstat)
> idw.out <- gstat::idw(zinc ~ 1, meuse, meuse.grid, idp = 2.5)

[inverse distance weighted interpolation]

> as.data.frame(idw.out)[1:5, ]
      x      y var1.pred var1.var
1 181180 333740  701.9621      NA
2 181140 333700  799.9616      NA
3 181180 333700  723.5780      NA
4 181220 333700  655.3131      NA
5 181100 333660  942.0218      NA
```

The output variable is called **var1.pred**, and the **var1.var** values are NA because inverse distance does not provide prediction error variances.

Inverse distance interpolation results usually in maps that are very similar to kriged maps when a variogram with no or a small nugget is used. In contrast to kriging, by only considering distances to the prediction location it ignores the spatial configuration of observations; this may lead to undesired effects if the observation locations are strongly clustered. Another difference is that weights are guaranteed to be between 0 and 1, resulting in interpolated values never outside the range of observed values.

8.3.2 Linear Regression

For spatial prediction using simple linear models, we can use the R function **lm**:

```
> zn.lm <- lm(log(zinc) ~ sqrt(dist), meuse)
> meuse.grid$pred <- predict(zn.lm, meuse.grid)
> meuse.grid$se.fit <- predict(zn.lm, meuse.grid, se.fit = TRUE)$se.fit
```


Alternatively, the `predict` method used here can provide the prediction or confidence intervals for a given confidence level. Alternatively, we can use the function `krige` in `gstat` for this,

```
> meuse.lm <- krige(log(zinc) ~ sqrt(dist), meuse, meuse.grid)
```

```
[ordinary or weighted least squares prediction]
```

that in this case does not *krige* as no variogram is specified, but uses linear regression.

Used in this form, the result is identical to that of `lm`. However, it can also be used to predict with regression models that are refitted within local neighbourhoods around a prediction location (Sect. 8.5.5) or provide mean predicted values for spatial areas (Sect. 8.5.6). The variance it returns is the prediction error variance when predicting for points or the estimation error variance when used for blocks.

A special form of linear regression is obtained when polynomials of spatial coordinates are used for predictors, for example for a second-order polynomial

```
> meuse.tr2 <- krige(log(zinc) ~ 1, meuse, meuse.grid,
+   degree = 2)
```

```
[ordinary or weighted least squares prediction]
```

This form is called trend surface analysis.

It is possible to use `lm` for trend surface analysis, for example for the second-order trend with a formula using `I` to treat powers and products 'as is':

```
> lm(log(zinc) ~ I(x^2) + I(y^2) + I(x * y) + x + y, meuse)
```

or the short form

```
> lm(log(zinc) ~ poly(x, y, degree = 2), meuse)
```

In the first form `lm` does not standardise coordinates, which often yields huge numbers when powered. The second form does standardise coordinates in such a way that it cannot be used in a subsequent `predict` call with different coordinate ranges. Trend surface fitting is highly sensitive to outlying observations. Another place to look for trend surface analysis is function `surf.ls` in package `spatial`.

8.4 Estimating Spatial Correlation: The Variogram

In geostatistics the spatial correlation is modelled by the variogram instead of a correlogram or covariogram, largely for historical reasons. Here, the word variogram will be used synonymously with semivariogram. The variogram plots semivariance as a function of distance.

In standard statistical problems, correlation can be estimated from a scatterplot, when several data pairs $\{x, y\}$ are available. The spatial correlation between two observations of a variable $z(s)$ at locations s_1 and s_2 cannot be estimated, as only a single pair is available. To estimate spatial correlation from observational data, we therefore need to make stationarity assumptions before we can make any progress. One commonly used form of stationarity is *intrinsic stationarity*, which assumes that the process that generated the samples is a random function $Z(s)$ composed of a mean and residual

$$Z(s) = m + e(s), \quad (8.1)$$

with a constant mean

$$E(Z(s)) = m \quad (8.2)$$

and a variogram defined as

$$\gamma(h) = \frac{1}{2} E(Z(s) - Z(s+h))^2. \quad (8.3)$$

Under this assumption, we basically state that the variance of Z is constant, and that spatial correlation of Z does not depend on location s , but only on separation distance h . Then, we can form *multiple* pairs $\{z(s_i), z(s_j)\}$ that have (nearly) identical separation vectors $h = s_i - s_j$ and estimate correlation from them. If we further assume *isotropy*, which is direction independence of semivariance, we can replace the vector h with its length, $\|h\|$.

Under this assumption, the variogram can be estimated from N_h sample data pairs $z(s_i)$, $z(s_i + h)$ for a number of distances (or distance intervals) $\tilde{h}_j$ by

$$\hat{\gamma}(\tilde{h}_j) = \frac{1}{2N_h} \sum_{i=1}^{N_h} (Z(s_i) - Z(s_i + h))^2, \quad \forall h \in \tilde{h}_j \quad (8.4)$$

and this estimate is called the *sample* variogram.

A wider class of models is obtained when the mean varies spatially, and can, for example be modelled as a linear function of known predictors $X_j(s)$, as in

$$Z(s) = \sum_{j=0}^p X_j(s) \beta_j + e(s) = X\beta + e(s), \quad (8.5)$$

with $X_j(s)$ the known spatial regressors and β_j unknown regression coefficients, usually containing an intercept for which $X_0(s) \equiv 1$. The $X_j(s)$ form the columns of the $n \times (p+1)$ design matrix X , β is the column vector with $p+1$ unknown coefficients.

For varying mean models, stationarity properties refer to the residual $e(s)$, and the sample variogram needs to be computed from estimated residuals.

8.4.1 Exploratory Variogram Analysis

A simple way to acknowledge that spatial correlation is present or not is to make scatter plots of pairs $Z(s_i)$ and $Z(s_j)$, grouped according to their separation distance $h_{ij} = ||s_i - s_j||$. This is done for the `meuse` data set in Fig. 8.2, by

```
> hscat(log(zinc) ~ 1, meuse, (0:9) * 100)
```

where the strip texts indicate the distance classes, and sample correlations are shown in each panel.

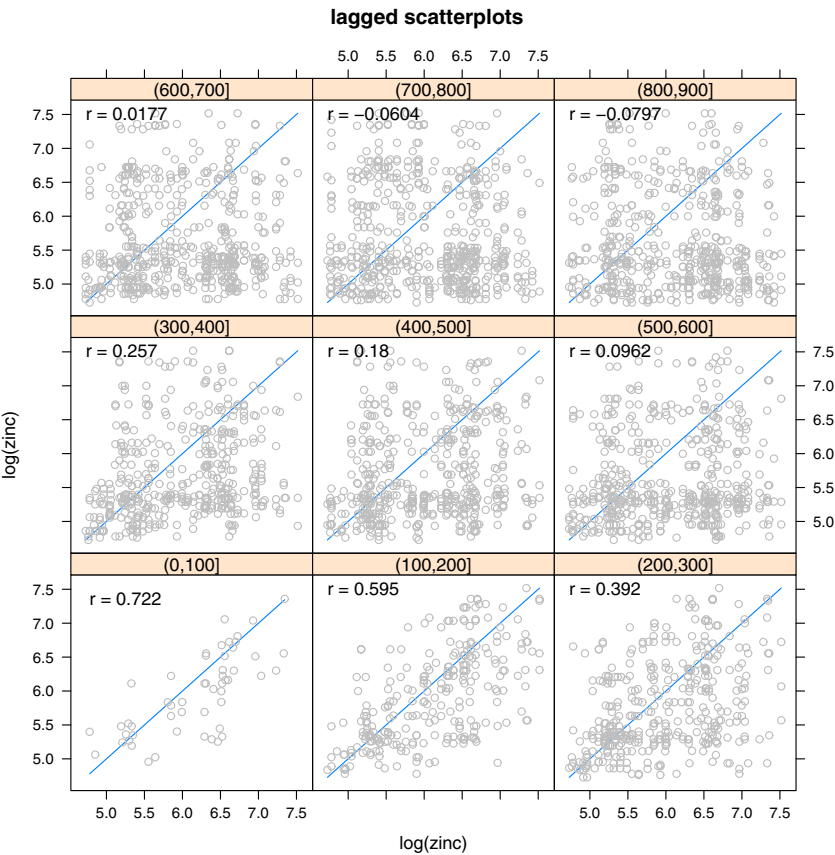


Fig. 8.2 Lagged scatter plot for the log-zinc data in the `meuse` data set

A second way to explore spatial correlation is by plotting the variogram and the variogram cloud. The variogram cloud is obtained by plotting all

possible squared differences of observation pairs $(Z(s_i) - Z(s_j))^2$ against their separation distance h_{ij} . One such variogram cloud, obtained by

```
> hscat(log(zinc) ~ 1, meuse, (0:9) * 100)
```

is plotted in Fig. 8.3 (top). The plot shows a lot of scatter, as could be expected: when $Z(s)$ follows a Gaussian distribution, $(Z(s_i) - Z(s_j))^2$ follows a $\chi^2(1)$ distribution. It does show, however, some increase of maximum values for distances increasing up to 1,000 m.

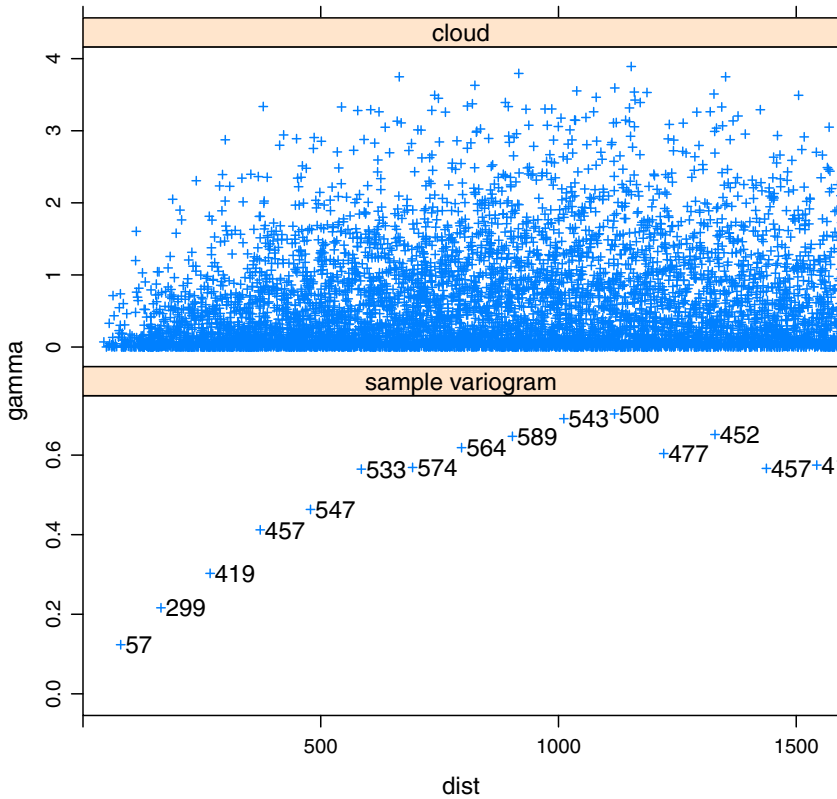


Fig. 8.3 Variogram cloud (*top*) and sample variogram (*bottom*) for log-zinc data; numbers next to symbols refer to the value N_h in (8.4)

Essentially, the sample variogram plot of (8.4) obtained by

```
> library(gstat)
> variogram(log(zinc) ~ 1, meuse, cloud = TRUE)
```

is nothing but a plot of averages of semivariogram cloud values over distance intervals h ; it is shown in Fig. 8.3, bottom. It smooths the variation in the

variogram cloud and provides an estimate of semivariance (8.3), although Stein (1999) discourages this approach.

The `~ 1` defines a single constant predictor, leading to a spatially constant mean coefficient, in accordance with (8.2); see p. 26 for a presentation of formula objects.

As any point in the variogram cloud refers to a pair of points in the data set, the variogram cloud can point us to areas with unusual high or low variability. To do that, we need to select a subset of variogram cloud points. In

```
> sel <- plot(variogram(zinc ~ 1, meuse, cloud = TRUE),
+   digitize = TRUE)
> plot(sel, meuse)
```

the user is asked to digitise an area in the variogram cloud plot after the first command. The second command plots the selected point pairs on the observations map. Figure 8.4 shows the output plots of such a session. The point pairs with largest semivariance at short distances were selected, because they indicate the areas with the strongest gradients. The map shows that these areas are not spread randomly: they connect the maximum values closest to the Meuse river with values usually more inland. This can be an indication of non-stationarity or of anisotropy. Log-transformation or detrending may remove this effect, as we see later.

In case of outlying observations, extreme variogram cloud values are easily identified to find the outliers. These may need removal, or else robust measures for the sample variogram can be computed by passing the logical argument `creessie=TRUE` to the `variogram` function call (Cressie, 1993).

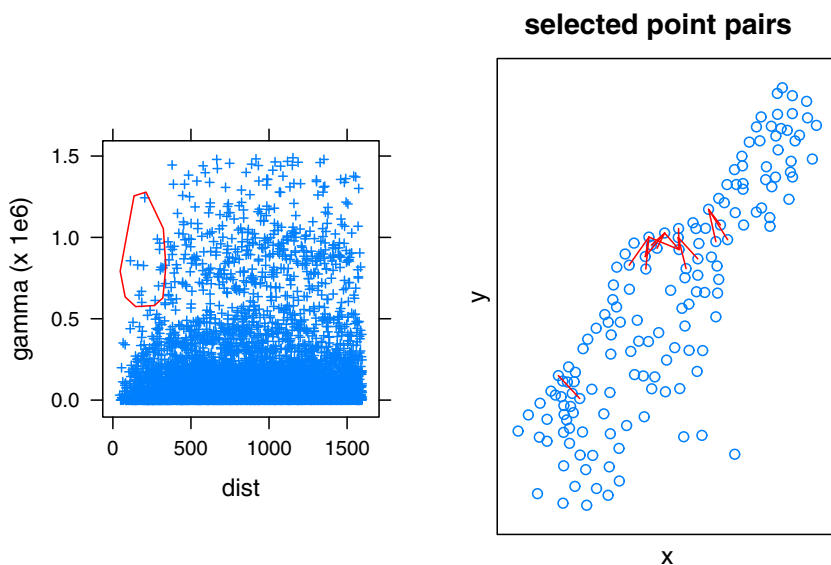


Fig. 8.4 Interactively selected point pairs on the variogram cloud (*left*), and map of selected point pairs (*right*)

A sample variogram $\hat{\gamma}(h)$ always contains a signal that results from the true variogram $\gamma(h)$ and a sampling error, due to the fact that N_h and s are not infinite. To verify whether an increase in semivariance with distance *could* possibly be attributed to chance, we can compute variograms from the same data, after randomly re-assigning measurements to spatial locations. If the sample variogram falls within the (say 95 %) range of these random variograms, complete spatial randomness of the underlying process *may* be a plausible hypothesis. Figure 8.5 shows an example of such a plot for the log zinc data; here the hypothesis of absence of spatial correlation seems unlikely. In general, however, concluding or even assuming that an underlying process is *completely* spatially uncorrelated is quite unrealistic for real, natural processes. A common case is that the spatial correlation is difficult to infer from sample data, because of their distribution, sample size, or spatial configuration. In certain cases spatial correlation is nearly absent.

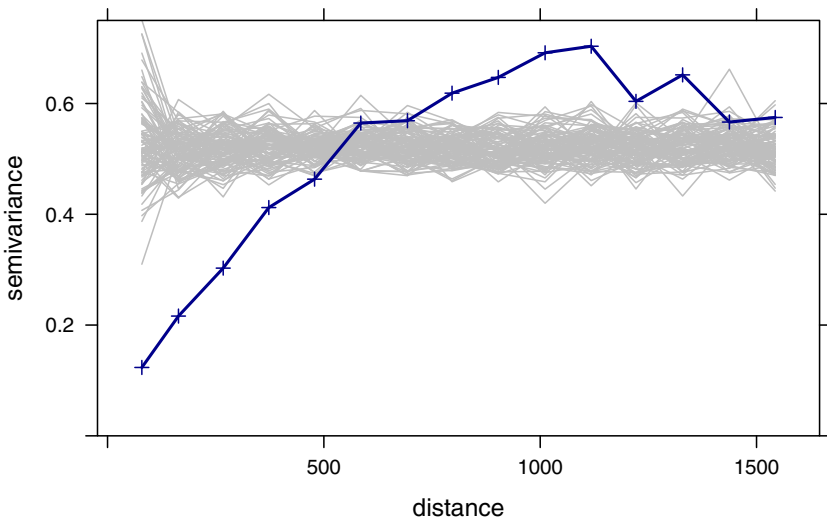


Fig. 8.5 Sample variogram (*bold*) compared to 100 variograms for randomly re-allocated data (*grey lines*)

8.4.2 Cutoff, Lag Width, Direction Dependence

Although the command

```
> plot(variogram(log(zinc) ~ 1, meuse))
```

simply computes and plots the sample variogram, it does make a number of decisions by default. It decides that direction is ignored: point pairs are

merged on the basis of distance, not direction. An alternative is, for example to look in four different angles, as in

```
> plot(variogram(log(zinc) ~ 1, meuse, alpha = c(0, 45,
+      90, 135)))
```

see Fig. 8.7. Directions are now divided over their principal direction, e.g., any point pair between 22.5° and 67.5° is used for the 45° panel. You might want to split into a finer direction subdivision, for example passing `alpha = seq(0, 170, 10)`, but then the noise component of resulting sample variograms will increase, as the number of point pairs for each separate estimate decreases.

A similar issue is the cutoff distance, which is the maximum distance up to which point pairs are considered and the width of distance interval over which point pairs are averaged in bins.

The default value `gstat` uses for the cutoff value is one third of the largest diagonal of the bounding box (or cube) of the data. Just as for time series data autocorrelations are never computed for lags farther than half the series length, there is little point in computing semivariances for long distances other than mere curiosity: wild oscillations usually show up that reveal little about the process under study. Good reasons to decrease the cutoff may be when a local prediction method is foreseen, and only semivariances up to a rather small distance are required. In this case, the modelling effort, and hence the computing of sample variograms should be limited to this distance (e.g. twice the radius of the planned search neighbourhood).

For the interval width, `gstat` uses a default of the cutoff value divided by 15. Usually, these default values will result in some initial overview of the spatial correlation. Choosing a smaller interval width will result in more detail, as more estimates of $\gamma(h)$ appear, but also in estimates with more noise, as N_h inevitably decreases. It should be noted that apparent local fluctuations of consecutive $\hat{\gamma}(h)$ values may still be attributed to sampling error. The errors $\hat{\gamma}(h_i) - \gamma(h_i)$ and $\hat{\gamma}(h_j) - \gamma(h_j)$ will be correlated, because $\gamma(\hat{h}_i)$ and $\gamma(\hat{h}_j)$ usually share a large number of common points used to form pairs.

The default cutoff and interval width values may not be appropriate at all, and can be overridden, for example by

```
> plot(variogram(log(zinc) ~ 1, meuse, cutoff = 1000, width = 50))
```

The distance vector does not have to be cut in regular intervals; one can specify each interval by

```
> variogram(log(zinc) ~ 1, meuse, boundaries = c(0, 50,
+      100, seq(250, 1500, 250)))
```

which is especially useful for data sets that have much information on short distance variability: it allows one to zoom in on the short distance variogram without revealing irrelevant details for the longer distances.

8.4.3 Variogram Modelling

The variogram is often used for spatial prediction (interpolation) or simulation of the observed process based on point observations. To ensure that predictions are associated with non-negative prediction variances, the matrix with semivariance values between all observation points and any possible prediction point needs to be non-negative definite. For this, simply plugging in sample variogram values from (8.4) is not sufficient. One common way is to infer a parametric variogram *model* from the data. A non-parametric way, using smoothing and cutting off negative frequencies in the spectral domain, is given in Yao and Journel (1998); it will not be discussed here.

The traditional way of finding a suitable variogram model is to fit a parametric model to the sample variogram (8.4). An overview of the basic variogram models available in **gstat** is obtained by

```
> show.vgms()
> show.vgms(model = "Mat", kappa.range = c(0.1, 0.2, 0.5,
+      1, 2, 5, 10), max = 10)
```

where the second command gives an overview of various models in the Matérn class.

In **gstat**, valid variogram models are constructed by using one or combinations of two or more basic variogram models. Variogram models are derived from **data.frame** objects, and are built as follows:

```
> vgm(1, "Sph", 300)

  model psill range
1   Sph     1   300

> vgm(1, "Sph", 300, 0.5)

  model psill range
1   Nug    0.5     0
2   Sph    1.0   300

> v1 <- vgm(1, "Sph", 300, 0.5)
> v2 <- vgm(0.8, "Sph", 800, add.to = v1)
> v2

  model psill range
1   Nug    0.5     0
2   Sph    1.0   300
3   Sph    0.8   800

> vgm(0.5, "Nug", 0)

  model psill range
1   Nug    0.5     0
```

and so on. Each component (row) has a model type ('Nug', 'Sph', ...), followed by a partial sill (the vertical extent of the model component) and a

range parameter (the horizontal extent). Nugget variance can be defined in two ways, because it is almost always present. It reflects usually measurement error and/or micro-variability. Note that **gstat** uses range *parameters*, for example for the exponential model with partial sill c and range parameter a

$$\gamma(h) = c(1 - e^{-h/a}).$$

This implies that for this particular model the *practical range*, the value at which this model reaches 95 % of its asymptotic value, is $3a$; for the Gaussian model the practical range is $\sqrt{3}a$. A list of model types is obtained by

```
> vgm()

      short                                long
1   Nug                                Nug (nugget)
2   Exp                                Exp (exponential)
3   Sph                                Sph (spherical)
4   Gau                                Gau (gaussian)
5   Exc                                Exclass (Exponential class)
6   Mat                                Mat (Matern)
7   Ste Mat (Matern, M. Stein's parameterization)
8   Cir                                Cir (circular)
9   Lin                                Lin (linear)
10  Bes                                Bes (bessel)
11  Pen                                Pen (pentaspherical)
12  Per                                Per (periodic)
13  Hol                                Hol (hole)
14  Log                                Log (logarithmic)
15  Pow                                Pow (power)
16  Spl                                Spl (spline)
17  Leg                                Leg (Legendre)
18  Err                                Err (Measurement error)
19  Int                                Int (Intercept)
```

Not all of these models are equally useful, in practice. Most practical studies have so far used exponential, spherical, Gaussian, Matérn, or power models, with or without a nugget, or a combination of those.

For weighted least squares fitting a variogram model to the sample variogram (Cressie, 1985), we need to take several steps:

1. Choose a suitable model (such as exponential, ...), with or without nugget
2. Choose suitable initial values for partial sill(s), range(s), and possibly nugget (Fig. 8.6)
3. Fit this model, using one of the fitting criteria.

For the variogram obtained by

```
> v <- variogram(log(zinc) ~ 1, meuse)
> plot(v)
```

and shown in Fig. 8.6, the spherical model looks like a reasonable choice. Initial values for the variogram fit are needed for **fit.variogram**, because

for the spherical model (and many other models) fitting the range parameter involves non-linear regression. The following fit works:

```
> fit.variogram(v, vgm(1, "Sph", 800, 1))
```

	model	psill	range
1	Nug	0.05065923	0.0000
2	Sph	0.59060463	896.9976

but if we choose initial values too far off from reasonable values, as in

```
> fit.variogram(v, vgm(1, "Sph", 10, 1))
```

Warning: singular model in variogram fit

	model	psill	range
1	Nug	1	0
2	Sph	1	10

the fit will not succeed. To stop execution in an automated fitting script, a construct like

```
> v.fit <- fit.variogram(v, vgm(1, "Sph", 10, 1))
> if (attr(v.fit, "singular")) stop("singular fit")
```

will halt the script on this fitting problem.

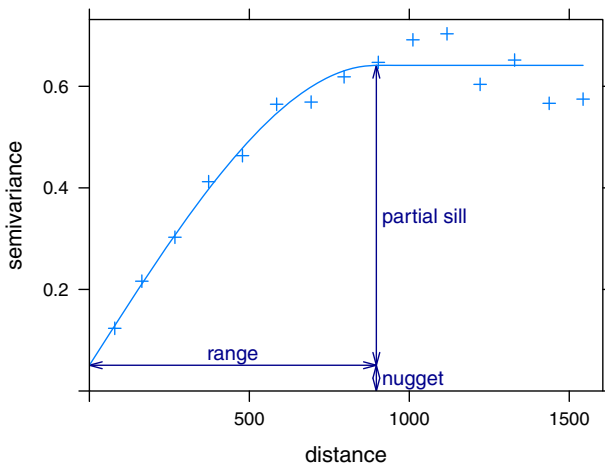


Fig. 8.6 Sample variogram (*plus*) and fitted model (*line*); *blue arrows* indicate the model parameter values

The fitting method uses non-linear regression to fit the coefficients. For this, a weighted sum of square errors $\sum_{j=1}^p w_j (\gamma(h) - \hat{\gamma}(h))^2$, with $\gamma(h)$ the value according to the parametric model, is minimised. The optimisation routine alternates the following two steps until convergence: (i) a direct fit over the partial sills, and (ii) non-linear optimising of the range parameter(s) given the last fit of partial sills. The minimised criterion is available as

Table 8.1 Values for argument `fit.method` in function `fit.variogram`

<code>fit.method</code>	Weight
1	N_j
2	$N_j / \{\gamma(h_j)\}^2$
6	1
7	N_j / h_j^2

```
> attr(v.fit, "SSErr")
[1] 9.011194e-06
```

Different options for the weights w_j are given in Table 8.1, the default value chosen by **gstat** is 7. Two things should be noted: (i) for option 2, the weights change after each iteration, which may confuse the optimisation routine, and (ii) for the linear variogram with no nugget, option 7 is equivalent to option 2. Option 7 is default as it seems to work in many cases; it will, however, give rise to spurious fits when a sample semivariance estimate for distance (very close to) zero gives rise to an almost infinite weight. This may happen when duplicate observations are available.

An alternative approach to fitting variograms is by visual fitting, the so-called eyeball fit. Package **geoR** provides a graphical user interface for interactively adjusting the parameters:

```
> library(geoR)
> v.ey<- eyefit(variog(as.geodata(meuse["zinc"]), max.dist = 1500))
> ve.fit <- as.vgm.variomodel(v.ey[[1]])
```

The last function converts the model saved in `v.ey` to a form readable by **gstat**.

Typically, visual fitting will minimise $|\gamma(h) - \hat{\gamma}(h)|$ with emphasis on short distance/small $\gamma(h)$ values, as opposed to a weighted squared difference, used by most numerical fitting. An argument to prefer visual fitting over numerical fitting may be that the person who fits has knowledge that goes beyond the information in the data. This may for instance be related to information about the nugget effect, which may be hard to infer from data when sample locations are regularly spread. Information may be borrowed from other studies or derived from measurement error characteristics for a specific device. In that case, one could, however, also consider partial fitting, by keeping, for example the nugget to a fixed value.

Partial fitting of variogram coefficients can be done with **gstat**. Suppose we know for some reason that the partial sill for the nugget model (i.e. the nugget variance) is 0.06, and we want to fit the remaining parameters, then this is done by

```
> fit.variogram(v, vgm(1, "Sph", 800, 0.06), fit.sills = c(FALSE,
+ TRUE))
  model    psill    range
1  Nug 0.0600000  0.0000
2  Sph 0.5845836 923.0066
```

Alternatively, the range parameter(s) can be fixed using argument `Data set!Meuse bank fit.ranges`.

Maximum likelihood fitting of variogram models does not need the sample variogram as intermediate form, as it fits a model directly to a quadratic form of the data, that is the variogram cloud. REML (restricted maximum likelihood) fitting of only partial sills, not of ranges, can be done using **gstat** function `fit.variogram.reml`:

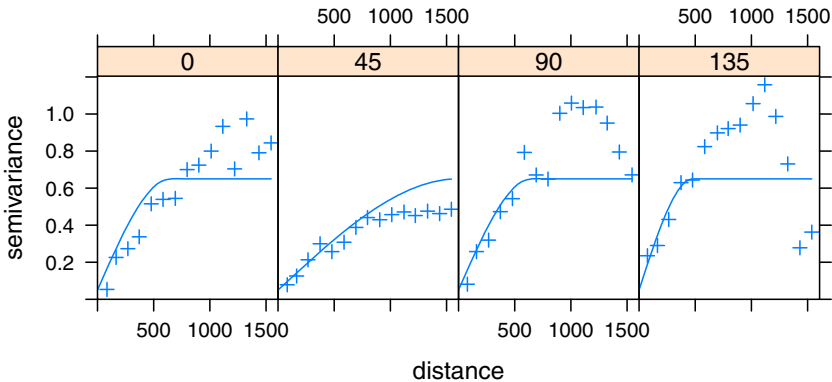


Fig. 8.7 Directional sample variogram (*plus*) and fitted model (*line*), for four directions (0 is North, 90 is East)

```
> fit.variogram.reml(log(zinc) ~ 1, meuse, model = vgm(0.6,
+   "Sph", 800, 0.06))
```

```
model    psill range
1  Nug 0.0201109   0
2  Sph 0.5711620 800
```

Maximum likelihood or restricted maximum likelihood fitting of variogram models, including the range parameters, can be done using function `lik-fit` in package **geoR**, or with function `fitvario` in package **RandomFields**. Maximum likelihood fitting is optimal under the assumption of a Gaussian random field, and can be very time consuming for larger data sets.

8.4.4 Anisotropy

Anisotropy may be modelled by defining a range *ellipse* instead of a circular or spherical range. In the following example

```
> v.dir <- variogram(log(zinc) ~ 1, meuse, alpha = (0:3) *
+   45)
> v.anis <- vgm(0.6, "Sph", 1600, 0.05, anis = c(45, 0.3))
> plot(v.dir, v.anis)
```

the result of which is shown in Fig. 8.7, for four main directions. The fitted model has a range in the principal direction (45° , NE) of 1,600, and of $0.3 \times 1,600 = 480$ in the minor direction (135°).

When more measurement information is available, one may consider plotting a *variogram map*, as in

```
> plot(variogram(log(zinc) ~ 1, meuse, map = TRUE, cutoff = 1000,
+           width = 100))
```

which bins h vectors in square grid cells over x and y , meaning that distance and direction are shown in much more detail. Help is available for the plotting function `plot.variogramMap`.

Package **gstat** does not provide automatic fitting of anisotropy parameters. Function `likfit` in **geoR** does, by using (restricted) maximum likelihood.

8.4.5 Multivariable Variogram Modelling

We use the term multivariable geostatistics here for the case where multiple dependent spatial variables are analysed jointly. The case where the trend of a single dependent variable contains more than a constant only is not called multivariable in this sense, and will be treated in Sect. 8.5.

The main tool for estimating semivariances between different variables is the cross variogram, defined for collocated¹ data as

$$\gamma_{ij}(h) = \frac{1}{2}E[(Z_i(s) - Z_i(s+h))(Z_j(s) - Z_j(s+h))]$$

and for non-collocated data as

$$\gamma_{ij}(h) = \frac{1}{2}E[((Z_i(s) - m_i) - (Z_j(s) - m_j))^2],$$

with m_i and m_j the means of the respective variables. Sample cross variograms are the obvious sums over the available pairs or cross pairs, in the line of (8.4).

As multivariable analysis may involve numerous variables, we need to start organising the available information. For that reason, we collect all the observation data specifications in a **gstat** object, created by the function **gstat**. This function does nothing else than ordering (and actually, copying) information needed later in a single object. Consider the following definitions of four heavy metals:

```
> g <- gstat(NULL, "logCd", log(cadmium) ~ 1, meuse)
> g <- gstat(g, "logCu", log(copper) ~ 1, meuse)
> g <- gstat(g, "logPb", log(lead) ~ 1, meuse)
```

¹ Each observation location has all variables measured.

```

> g <- gstat(g, "logZn", log(zinc) ~ 1, meuse)
> g
data:
logCd : formula = log(cadmium)~~1 ; data dim = 155 x 14
logCu : formula = log(copper)~~1 ; data dim = 155 x 14
logPb : formula = log(lead)~~1 ; data dim = 155 x 14
logZn : formula = log(zinc)~~1 ; data dim = 155 x 14
> vm <- variogram(g)
> vm.fit <- fit.lmc(vm, g, vgm(1, "Sph", 800, 1))
> plot(vm, vm.fit)

```

the plot of which is shown in Fig. 8.8. By default, `variogram` when passing a `gstat` object computes all direct and cross variograms, but this can be turned off. The function `fit.lmc` fits a linear model of coregionalization, which is a particular model that needs to have identical model components (here nugget, and spherical with range 800), and needs to have positive definite partial sill matrices, to ensure non-negative prediction variances when used for spatial prediction (cokriging).

As the variograms in Fig. 8.8 indicate, the variables have a strong cross correlation. Because these variables are collocated, we could compute direct correlations:

```

> cor(as.data.frame(meuse)[c("cadmium", "copper", "lead",
+   "zinc")])

```

	cadmium	copper	lead	zinc
cadmium	1.0000000	0.9254499	0.7989466	0.9162139
copper	0.9254499	1.0000000	0.8183069	0.9082695
lead	0.7989466	0.8183069	1.0000000	0.9546913
zinc	0.9162139	0.9082695	0.9546913	1.0000000

which confirm this, but ignore spatial components. For non-collocated data, the direct correlations may be hard to compute.

The `fit.lmc` function fits positive definite coefficient matrices by first fitting models individually (while fixing the ranges) and then replacing non-positive definite coefficient matrices by their nearest positive definite approximation, taking out components that have a negative eigenvalue. When eigenvalues of exactly zero occur, a small value may have to be added to the direct variogram sill parameters; use the `correct.diagonal` argument for this.

Variables do not need to have a constant mean but can have a trend function specified, as explained in Sect. 8.4.6.

8.4.6 Residual Variogram Modelling

Residual variograms are calculated by default when a more complex model for the trend is used, for example as in

```

> variogram(log(zinc) ~ sqrt(dist), meuse)

```

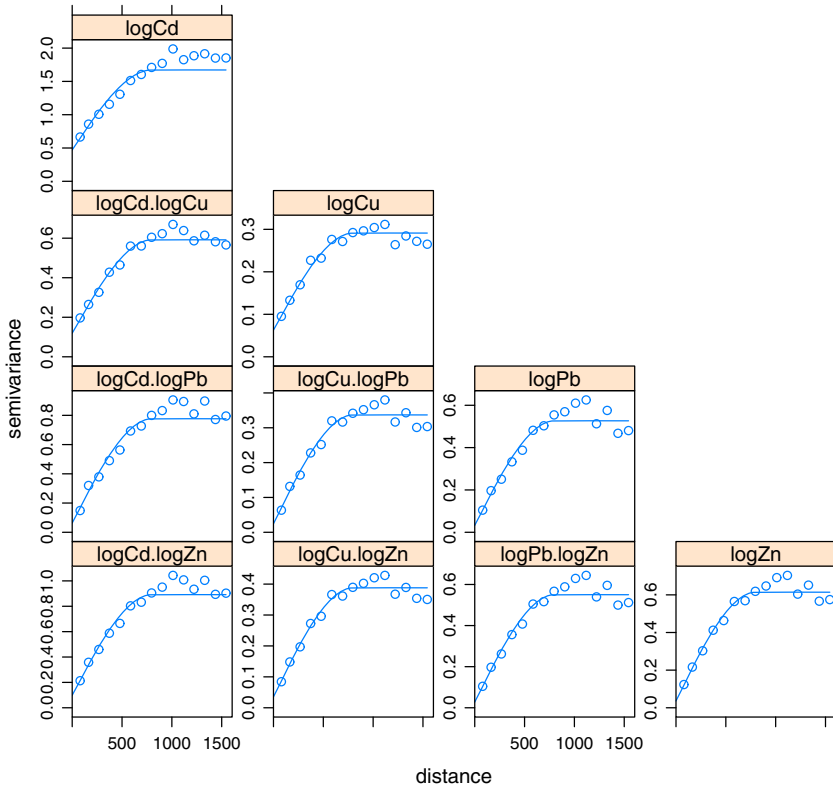


Fig. 8.8 Direct variograms (diagonal) and cross variograms (off-diagonal) along with fitted linear model of coregionalization (—)

where the trend is simple linear (Fig. 8.1), for example reworking (8.5) to

$$\log(Z(s)) = \beta_0 + \sqrt{D(s)}\beta_1 + e(s),$$

with $D(s)$ the distance to the river. For defining trends, the full range of R formulas can be used: the right-hand side may contain factors, in which case trends are calculated with respect to the factor level means, and may contain interactions of all sorts; see p. 26 for explanation on SSformula syntax.

By default, the residuals **gstat** uses are ordinary least squares residuals (i.e. regular regression residuals), meaning that for the sake of estimating the trend, observations are considered independent. To honour a dependence structure present, generalised least squares residuals can be calculated instead. For this, a variogram model to define the covariance structure is needed. In the following example

```
> f <- log(zinc) ~ sqrt(dist)
> vt <- variogram(f, meuse)
```

```

> vt.fit <- fit.variogram(vt, vgm(1, "Exp", 300, 1))
> vt.fit

    model      psill    range
1   Nug 0.05712231    0.0000
2   Exp 0.17641559  340.3201

> g.wls <- gstat(NULL, "log-zinc", f, meuse, model = vt.fit,
+       set = list(gls = 1))
> (variogram(g.wls)$gamma - vt$gamma)/mean(vt$gamma)

[1] 1.133887e-05 -6.800894e-05 -1.588582e-04 -2.520913e-04
[5] -5.461007e-05 -1.257573e-04 2.560629e-04 1.509185e-04
[9] 4.812184e-07 -5.292472e-05 -2.998868e-04 2.169712e-04
[13] -1.771773e-04 1.872195e-04 3.095021e-05

```

it is clear that the difference between the two approaches is marginal, but this does not need to be the case in other examples.

For multivariable analysis, **gstat** objects can be formed where the trend structure can be specified uniquely for each variable. If multivariable residuals are calculated using weighted least squares, this is done on a per-variable basis, ignoring cross correlations for trend estimation.

8.5 Spatial Prediction

Spatial prediction refers to the prediction of unknown quantities $Z(s_0)$, based on sample data $Z(s_i)$ and assumptions regarding the form of the trend of Z and its variance and spatial correlation.

Suppose we can write the trend as a linear regression function, as in (8.5). If the predictor values for s_0 are available in the $1 \times p$ row-vector $x(s_0)$, V is the covariance matrix of $Z(s)$ and v the covariance vector of $Z(s)$ and $Z(s_0)$, then the best linear unbiased predictor of $Z(s_0)$ is

$$\hat{Z}(s_0) = x(s_0)\hat{\beta} + v'V^{-1}(Z(s) - X\hat{\beta}), \quad (8.6)$$

with $\hat{\beta} = (X'V^{-1}X)^{-1}X'V^{-1}Z(s)$ the generalized least squares estimate of the trend coefficients and where X' is the transpose of the design matrix X . The predictor consists of an estimated mean value for location s_0 , plus a weighted mean of the residuals from the mean function, with weights $v'V^{-1}$, known as the *simple kriging weights*.

The predictor (8.6) has prediction error variance

$$\sigma^2(s_0) = \sigma_0^2 - v'V^{-1}v + \delta(X'V^{-1}X)^{-1}\delta', \quad (8.7)$$

where σ_0^2 is $\text{var}(Z(s_0))$, or the variance of the Z process, and where $\delta = x(s_0) - v'V^{-1}X$. The term $v'V^{-1}v$ is zero if v is zero, that is if all observations are uncorrelated with $Z(s_0)$, and equals σ_0^2 when s_0 is identical to

an observation location. The third term of (8.7) is the contribution of the estimation error $\text{var}(\hat{\beta} - \beta) = (X'V^{-1}X)^{-1}$ to the prediction (8.6): it is zero if s_0 is an observation location, and increases, for example when $x(s_0)$ is more distant from X , as when we extrapolate in the space of X .

8.5.1 Universal, Ordinary, and Simple Kriging

The instances of this best linear unbiased prediction method with the number of predictors $p > 0$ are usually called *universal kriging*. Sometimes the term *kriging with external drift* is used for the case where $p = 1$ and X does not include coordinates.

A special case is that of (8.2), for which $p = 0$ and $X_0 \equiv 1$. The corresponding prediction is called *ordinary kriging*.

Simple kriging is obtained when, for whatever reason, β is a priori assumed to be known. The known β can then be substituted for $\hat{\beta}$ in (8.6). The simple kriging variance is obtained by omitting the third term, which is associated with the estimation error of $\hat{\beta}$ in (8.7).

Applying these techniques is much more straightforward than this complicated jargon suggests, as an example will show:

```
> lz.sk <- krige(log(zinc) ~ 1, meuse, meuse.grid, v.fit,
+   beta = 5.9)

[using simple kriging]

> lz.ok <- krige(log(zinc) ~ 1, meuse, meuse.grid, v.fit)

[using ordinary kriging]

> lz.uk <- krige(log(zinc) ~ sqrt(dist), meuse, meuse.grid,
+   vt.fit)

[using universal kriging]
```

Clearly, the `krige` command chooses the kriging method itself, depending on the information it is provided with: are trend coefficients given? is the trend constant or more complex? How this is done is shown in the decision tree of Fig. 8.9.

8.5.2 Multivariable Prediction: Cokriging

The kriging predictions equations can be simply extended to obtain multivariable prediction equations, see, for example Hoef and Cressie (1993) and Pebesma (2004). The general idea is that multiple variables may be cross correlated, meaning that they exhibit not only autocorrelation but that the

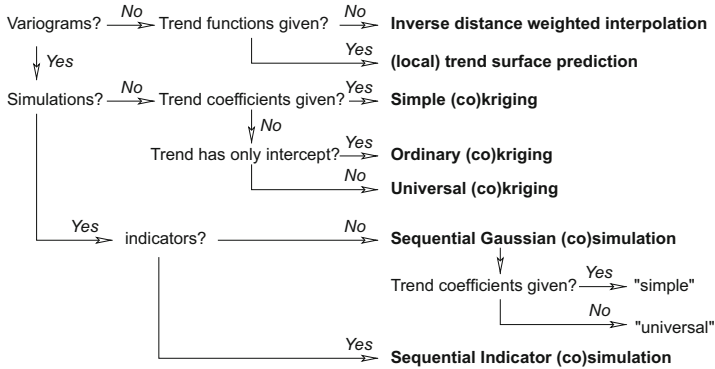


Fig. 8.9 Decision tree for the `gstat predict` method

spatial variability of variable A is correlated with variable B , and can therefore be used for its prediction, and vice versa. Typically, both variables are assumed to be measured on a limited set of locations, and the interpolation addresses unmeasured locations.

The technique is not limited to two variables. For each prediction location, multivariable prediction for q variables yields a $q \times 1$ *vector* with a prediction for each variable, and a $q \times q$ matrix with prediction error variances and covariances from which we can obtain the error correlations:

```
> cok.maps <- predict(vm.fit, meuse.grid)
```

```
Linear Model of Coregionalization found. Good.
[using ordinary cokriging]
```

```
> names(cok.maps)
```

```
[1] "logCd.pred"      "logCd.var"       "logCu.pred"
[4] "logCu.var"       "logPb.pred"      "logPb.var"
[7] "logZn.pred"      "logZn.var"       "cov.logCd.logCu"
[10] "cov.logCd.logPb" "cov.logCu.logPb" "cov.logCd.logZn"
[13] "cov.logCu.logZn" "cov.logPb.logZn"
```

Here, only the unique matrix elements are stored; to get an overview of the prediction error variance and covariances, a utility function wrapping `spplot` is available; the output of

```
> spplot.vcov(cok.maps)
```

is given in Fig. 8.10.

Before the cokriging starts, **gstat** reports success in finding a Linear Model of Coregionalization (LMC). This is good, as it will assure non-negative cokriging variances. If this is not the case, for example because the ranges differ,

```
> vm2.fit <- vm.fit
> vm2.fit$model[[3]]$range = c(0, 900)
> predict(vm2.fit, meuse.grid)
```

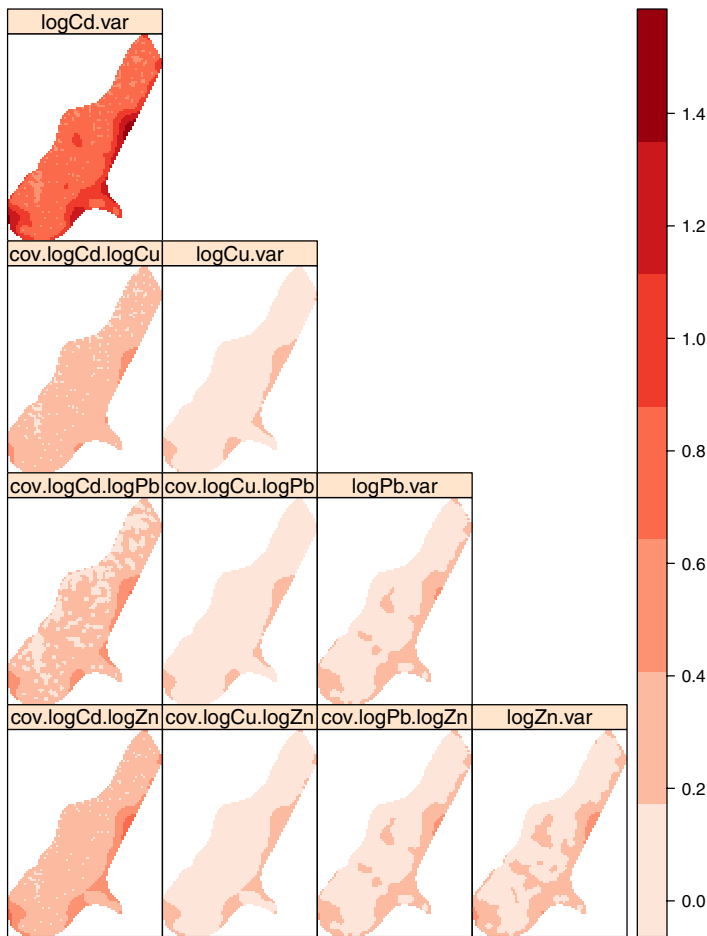


Fig. 8.10 Cokriging variances (diagonal) and covariances (off-diagonal)

will stop with an error message. Stopping on this check can be avoided by

```
> vm2.fit$set <- list(noccheck = 1)
> x <- predict(vm2.fit, meuse.grid)
```

```
Warning: No Intrinsic Correlation or Linear Model of Coregionalization
found
Reason: ranges differ
Now checking for Cauchy-Schwartz inequalities:
variogram(var0,var1) passed Cauchy-Schwartz
variogram(var0,var2) passed Cauchy-Schwartz
variogram(var1,var2) passed Cauchy-Schwartz
variogram(var0,var3) passed Cauchy-Schwartz
variogram(var1,var3) passed Cauchy-Schwartz
variogram(var2,var3) passed Cauchy-Schwartz
[using ordinary cokriging]
```

```
> names(as.data.frame(x))

[1] "x"                "y"                "logCd.pred"
[4] "logCd.var"        "logCu.pred"       "logCu.var"
[7] "logPb.pred"       "logPb.var"        "logZn.pred"
[10] "logZn.var"        "cov.logCd.logCu"  "cov.logCd.logPb"
[13] "cov.logCu.logPb"  "cov.logCd.logZn"  "cov.logCu.logZn"
[16] "cov.logPb.logZn"

> any(as.data.frame(x)[c(2, 4, 6, 8)] < 0)

[1] FALSE
```

which does check for pairwise Cauchy-Schwartz inequalities, that is $|\gamma_{ij}(h)| \leq \sqrt{\gamma_i(h)\gamma_j(h)}$, but will not stop on violations. Note that this latter check is not sufficient to guarantee positive variances. The final check confirms that we actually did not obtain any negative variances, for this particular case.

8.5.3 Collocated Cokriging

Collocated cokriging is a special case of cokriging, where a secondary variable is available at all prediction locations, and instead of choosing all observations of the secondary variable or those in a local neighbourhood, we restrict the secondary variable search neighbourhood to this single value on the prediction location. For instance, consider `log(zinc)` as primary and `dist` as secondary variable:

```
> g.cc <- gstat(NULL, "log.zinc", log(zinc) ~ 1, meuse,
+   model = v.fit)
> meuse.grid$distn <- meuse.grid$dist - mean(meuse.grid$dist) +
+   mean(log(meuse$zinc))
> vd.fit <- v.fit
> vov <- var(meuse.grid$distn)/var(log(meuse$zinc))
> vd.fit$psill <- v.fit$psill * vov
> g.cc <- gstat(g.cc, "distn", distn ~ 1, meuse.grid, nmax = 1,
+   model = vd.fit, merge = c("log.zinc", "distn"))
> vx.fit <- v.fit
> vx.fit$psill <- sqrt(v.fit$psill * vd.fit$psill) * cor(meuse$dist,
+   log(meuse$zinc))
> g.cc <- gstat(g.cc, c("log.zinc", "distn"), model = vx.fit)
> x <- predict(g.cc, meuse.grid)
```

Intrinsic Correlation found. Good.
[using ordinary cokriging]

Figure 8.11 shows the predicted maps using ordinary kriging, collocated cokriging and universal cokriging, using `log(zinc) ~ sqrt(dist)` as trend.

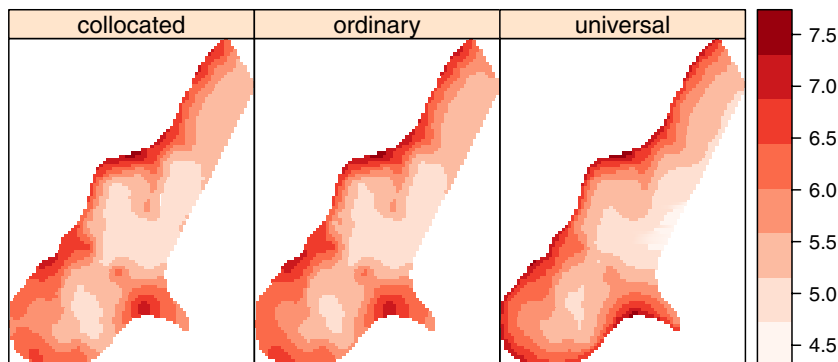


Fig. 8.11 Predictions for collocated cokriging, ordinary kriging and universal kriging

8.5.4 Cokriging Contrasts

Cokriging error covariances can be of value when we want to compute functions of multiple predictions. Suppose Z_1 is measured on time 1, and Z_2 on time 2, and both are non-collocated. When we want to estimate the change $Z_2 - Z_1$, we can use the estimates for both moments. For the prediction error of this difference, in addition to the prediction error variances for $\hat{Z}_1$ and $\hat{Z}_2$ we need the prediction error covariance. The function `get.contr` helps computing the predicted value and prediction error variance for *any* linear combination (contrast) in a set of predictors, obtained by cokriging. A demo in `gstat`,

```
> demo(pcb)
```

gives a full space-time cokriging example that shows how time trends can be estimated for PCB-138 concentration in sea floor sediment, from four consecutive five-yearly rounds of monitoring, using universal cokriging.

8.5.5 Kriging in a Local Neighbourhood

By default, all spatial predictions method provided by `gstat` use all available observations for each prediction. In many cases, it is more convenient to use only the data in the neighbourhood of the prediction location. The reasons for this may be statistical or computational. Statistical reasons include that the hypothesis of constant mean or mean function should apply locally, or that the assumed knowledge of the variogram is only valid up to a small distance. Computational issues may involve both memory and speed: kriging for n data requires solving an $n \times n$ system. For large n (say more than 1,000) this may be too slow, and discarding anything but the closest say 100 observations may not result in notable different predictions.

It should be noted that for global kriging the matrix V needs to be decomposed only once, after which the result is re-used for each prediction location to obtain $V^{-1}v$. Decomposing a linear system of equations is an $O(n^2)$ operation, solving another system $O(n)$. Therefore, if a neighbourhood size is chosen slightly smaller than the global neighbourhood, the computation time may even increase, compared to using a global neighbourhood.

Neighbourhoods in **gstat** are defined by passing the arguments **nmax**, **nmin**, and/or **maxdist** to functions like **predict**, **krige**, or **gstat**. Arguments **nmax** and **nmin** define a neighbourhood size in terms of number of nearest points, **maxdist** specifies a circular search range to select point. They may be combined: when less than **nmin** points are found in a search radius, a missing value is generated.

For finding neighbourhood selections fast, **gstat** first builds a PR bucket quadtree, or for three-dimensional data octree search index (Hjaltason and Samet, 1995). With this index it finds any neighbourhood selection with only a small number of distance evaluations.

8.5.6 Change of Support: Block Kriging

Despite the fact that no measurement can ever be taken on something that has size zero, in geostatistics, by default observations $Z(s_i)$ are treated as being observed on point location. Kriging a value with a physical size equal to that of the observations is called *point kriging*. In contrast, *block kriging* predicts averages of larger areas or volumes. The term block kriging originates from mining, where early geostatistics was developed (Journel and Huijbregts, 1978). In mines, predictions based on bore hole data had to be made for *mineable units*, which tended to have a block shape. *Change of support* occurs when predictions are made for a larger physical support based on small physical support observations. There is no particular reason why the larger support needs to have a rectangular shape, but it is common.

Besides the practical relevance to the mining industry, a consideration in many environmental applications has been that point kriging usually exhibits large prediction errors. This is due to the larger variability in the observations. When predicting averages over larger areas, much of the variability (i.e. that *within* the blocks) averages out and block mean values have lower prediction errors, while still revealing spatial patterns if the blocks are not too large. In environmental problems, legislation may be related to means or medians over larger areas, rather than to point values.

Block kriging (or other forms of prediction for blocks) can be obtained by **gstat** in three ways:

1. For regular blocks, by specifying a block size
2. For irregular but constant 'blocks', by specifying points that discretise the irregular form

3. For blocks or areas of varying size, by passing an object of class `SpatialPolygons` to the `newdata` argument (i.e. replacing `meuse.grid`)

Ordinary block kriging for blocks of size 40×40 is simply obtained by

```
> lz.ok <- krige(log(zinc) ~ 1, meuse, meuse.grid, v.fit,
+   block = c(40, 40))
```

[using ordinary kriging]

For a circular shape with radius 20, centred on the points of `meuse.grid`, one could select points on a regular grid within a circle:

```
> xy <- expand.grid(x = seq(-20, 20, 4), y = seq(-20, 20,
+   4))
> xy <- xy[(xy$x^2 + xy$y^2) <= 20^2, ]
> lz.ok <- krige(log(zinc) ~ 1, meuse, meuse.grid, v.fit,
+   block = xy)
```

[using ordinary kriging]

For block averages over varying regions, the `newdata` argument, usually a grid, can be replaced by a polygons object. Suppose `meuse.polygons` contains polygons for which we want to predict block averages, then this is done by

```
> lz.pols <- krige(log(zinc) ~ 1, meuse, meuse.polygons,
+   v.fit)
```

To discretise each (top level) Polygons object, coordinates that discretize the polygon are obtained by

```
> spsample(polygon, n = 500, type = "regular", offset = c(0.5,
+   0.5))
```

meaning that a regular discretisation is sought with approximately 500 points. These default arguments to `spsample` can be modified by altering the `sps.args` argument to `predict.gstat`; `spsample` is described on p. 146.

A much less efficient way to obtain block kriging predictions and prediction errors is to use Gaussian conditional simulation (Sect. 8.8) over a fine grid, calculate block means from each realisation, and obtain the mean and variance from a large number of realisations. In the limit, this should equal the analytical block kriging prediction.

When instead of a block *average* a non-linear spatial aggregation is required, such as a quantile value of points within a block, or the area fraction of a block where points exceed a threshold (Sect. 8.8.2), the simulation path is the more natural approach.

8.5.7 Stratifying the Domain

When a categorical variable is available that splits the area of interest in a number of disjoint areas, for example based on geology, soil type, land use or some other factor, we might want to apply separate krigings to the different units. This is called stratified kriging. The reason for doing kriging per-class may be that the covariance structure (semivariogram) is different for the different classes. In contrast to universal kriging with a categorical predictor, no correlation is assumed between residuals from different classes.

The example assumes there is a variable `part.a`, which partitions the area in two sub-areas, where `part.a` is 0 and where it is 1. First we can try to find out in which grid cells the observations lie:

```
> meuse$part.a <- gstat::idw(part.a ~ 1, meuse.grid,
+   meuse, nmax = 1)$var1.pred
```

[inverse distance weighted interpolation]

here, any interpolation may do, as we basically use the first nearest neighbour as predictor. A more robust approach may be to use the `over` method,

```
> meuse$part.a <- over(meuse, meuse.grid["part.a"])[[1]]
```

or equivalently,

```
> meuse$part.a <- meuse.grid$part.a[over(meuse, geometry(meuse.grid))]
```

because assign NA to point values not in a grid cell.

Next, we can perform kriging for each of the sub-domains, store them in `x1` and `x2`, and merge the result using `rbind` in their non-spatial representation:

```
> x1 <- krige(log(zinc) ~ 1, meuse[meuse$part.a == 0, ],
+   meuse.grid[meuse.grid$part.a == 0, ], model = vgm(0.548,
+   "Sph", 900, 0.0654), nmin = 20, nmax = 40, maxdist = 1000)
```

[using ordinary kriging]

```
> x2 <- krige(log(zinc) ~ 1, meuse[meuse$part.a == 1, ],
+   meuse.grid[meuse.grid$part.a == 1, ], model = vgm(0.716,
+   "Sph", 900), nmin = 20, nmax = 40, maxdist = 1000)
```

[using ordinary kriging]

```
> lz.stk <- rbind(as.data.frame(x1), as.data.frame(x2))
> coordinates(lz.stk) <- c("x", "y")
> lz.stk <- as(x, "SpatialPixelsDataFrame")
> spplot(lz.stk["var1.pred"], main = "stratified kriging predictions")
```


8.5.8 Trend Functions and Their Coefficients

For cases of exploratory data analysis or analysis of specific regression diagnostics, it may be of interest to limit prediction to the trend component $x(s_0)\hat{\beta}$, ignoring the prediction of the residual, that is ignoring the second term in the right-hand side of (8.6). This can be accomplished by setting argument `BLUE = TRUE` to `predict.gstat`:

```
> g.tr <- gstat(formula = log(zinc) ~ sqrt(dist), data = meuse,
+   model = v.fit)
> predict(g.tr, meuse[1, ])

[using universal kriging]
      coordinates var1.pred   var1.var
1 (181072, 333611)  6.929517 2.409685e-34

> predict(g.tr, meuse[1, ], BLUE = TRUE)

[generalized least squares trend estimation]
      coordinates var1.pred   var1.var
1 (181072, 333611)  6.862085 0.06123864
```

The first output yields the observed value (with zero variance), the second yields the generalised least squares trend component.

If we want to do significance testing of regression coefficients under a full model with spatially correlated residuals, we need to find out what the estimated regression coefficients and their standard errors are. For this, we can use `gstat` in a debug mode, in which case it will print a lot of information about intermediate calculations to the screen; just try

```
> predict(g, meuse[1, ], BLUE = TRUE, debug = 32)
```

but this does not allow saving the actual coefficients as data in R. Another way is to ‘fool’ the prediction mode with a specific contrast on the regression coefficients, for example the vector $x(s_0) = (0, 1)$, such that $x(s_0)\hat{\beta} = 0\hat{\beta}_0 + 1\hat{\beta}_1 = \hat{\beta}_1$. Both regression coefficient estimates are obtained by

```
> meuse$Int <- rep(1, 155)
> g.tr <- gstat(formula = log(zinc) ~ -1 + Int + sqrt(dist),
+   data = meuse, model = v.fit)
> rn <- c("Intercept", "beta1")
> df <- data.frame(Int = c(0, 1), dist = c(1, 0), row.names = rn)
> spdf <- SpatialPointsDataFrame(SpatialPoints(matrix(0,
+   2, 2)), df)
> spdf

      coordinates Int dist
Intercept   (0, 0)   0    1
beta1       (0, 0)   1    0

> predict(g.tr, spdf, BLUE = TRUE)
```

```
[generalized least squares trend estimation]
coordinates var1.pred var1.var
1      (0, 0) -2.471753 0.20018883
2      (0, 0)  6.953173 0.06633691
```

The `Int` variable is a ‘manual’ intercept to replace the automatic intercept, and the `-1` in the formula removes the automatic intercept. This way, we can control it and give it the zero value. The predictions now contain the generalised least squares estimates of the regression model.

8.5.9 Non-linear Transforms of the Response Variable

For predictor variables, a non-linear transform simply yields a new variable and one can proceed as if nothing has happened. Searching for a good transform, such as using `sqrt(dist)` instead of direct `dist` values, may help in approaching the relationship with a straight line. For dependent variables this is not the case: because statistical expectation (‘averaging’) is a linear operation, $E(g(X)) = g(E(X))$ only holds if $g(\cdot)$ is a linear operator. This means that if we compute kriging predictors for zinc on the log scale, we do not obtain the expected zinc concentration by taking the exponent of the kriging predictor.

A large class of monotonous transformations is provided by the Box–Cox family of transformations, which take a value λ :

$$f(y, \lambda) = \begin{cases} (y^\lambda - 1)/\lambda & \text{if } \lambda \neq 0, \\ \ln(y) & \text{if } \lambda = 0. \end{cases}$$

A likelihood profile plot for lambda is obtained by the `boxcox` method in the package bundle **MASS**. For example, the plot resulting from

```
> library(MASS)
> boxcox(zinc ~ sqrt(dist), data = as.data.frame(meuse))
```

suggests that a Box–Cox transform with a slightly negative value for λ , for example $\lambda = -0.2$, might be slightly better in approaching a log-normal distribution.

Yet another transformation is the normal score transform (Goovaerts, 1997) computed by the function `qqnorm`, defined as

```
> meuse$zinc.ns <- qqnorm(meuse$zinc, plot.it = FALSE)$x
```

Indeed, the resulting variable has mean zero, variance close to 1 (exactly one if n is large), and plots a straight line on a normal probability plot. So simple as this transform is, so complex can the back-transform be: it requires linear interpolation between the sample points, and for the extrapolation of values outside the data range the cited reference proposes several different models for tail distributions, all with different coefficients. There seems to be little

guidance as how to choose between them based on sample data. It should be noted that back-transforming normal-score transformed values outside the data range is less of a problem with interpolation, but more so for simulation (Sect. 8.8).

Indicator kriging is obtained by indicator transforming a continuous variable, or reducing a categorical variable to a binary variable. An example for the indicator whether zinc is below 500 ppm is

```
> ind.f <- I(zinc < 500) ~ 1
> ind.fit <- fit.variogram(variogram(ind.f, meuse), vgm(1,
+   "Sph", 800, 1))
> ind.kr <- krige(ind.f, meuse, meuse.grid, ind.fit)

[using ordinary kriging]

> summary(ind.kr$var1.pred)

      Min.   1st Qu.   Median     Mean  3rd Qu.     Max.
-0.03472  0.47490  0.80730  0.70390  0.94540  1.08800
```

Clearly, this demonstrates the difficulty of interpreting the resulting estimates of ones and zeros as probabilities, as one has to deal with negative values and values larger than one.

When it comes to non-linear transformations such as the log transform, the question whether to transform or not to transform is often a hard one. On the one hand, it introduces the difficulties mentioned; on the other hand, transformation solves problems like negative predictions for non-negative variables, and, for example heteroscedasticity: for non-negative variables the variability is larger for areas with larger values, which opposes the stationarity assumption where variability is independent from location.

When a continuous transform is taken, such as the log-transform or the Box–Cox transform, it is possible to back-transform quantiles using the inverse transform. So, under log-normal assumptions the exponent of the kriging mean on the log scale is an estimate of the median on the working scale. From back-transforming a large number of quantiles, the mean value and variance may be worked out.

8.5.10 Singular Matrix Errors

Kriging cannot deal with duplicate observations, or observations that share the same location, because they are perfectly correlated, and lead to singular covariance matrices V , meaning that $V^{-1}v$ has no unique solution. Obtaining errors due to a singular matrix is a common case.

```
> meuse.dup <- rbind(as.data.frame(meuse)[1, ], as.data.frame(meuse))
> coordinates(meuse.dup) = ~x + y
> krige(log(zinc) ~ 1, meuse.dup, meuse[1, ], v.fit)
```

will result in the output:

```
[using ordinary kriging]
```

```
"chfactor.c", line 130: singular matrix in function LDLfactor()
Error in predict.gstat(g, newdata=newdata, block=block, nsim=nsim:
  LDLfactor
```

which points to the C function where the actual error occurred (`LDLfactor`). The most common case where this happens is when duplicate observations are present in the data set. Duplicate observations can be found by

```
> zd <- zerodist(meuse.dup)
> zd

      [,1] [,2]
[1,]     1     2

> meuse0 <- meuse.dup[-zd[, 1], ]
> krige(log(zinc) ~ 1, meuse0, meuse[1, ], v.fit)

[using ordinary kriging]
      coordinates var1.pred var1.var
1 (181072, 333611)  6.929517         0
```

which tells that observations 1 and 2 have identical location; the third command removes the first of the pair. Near-duplicate observations are found by increasing the `zero` argument of function `zerodist` to a very small threshold.

Other common causes for singular matrices are the following:

- The use of variogram models that cause observations to have nearly perfect correlation, despite the fact that they do not share the same location, for example from `vgm(0, "Nug", 0)` or `vgm(1, "Gau", 1e20)`. The Gaussian model is *always* a suspect if errors occur when it is used; adding a small nugget often helps.
- Using a regression model with perfectly correlated variables; note that, for example a global regression model may lead to singularity in a local neighbourhood where a predictor may be constant and correlate perfectly with the intercept, or otherwise perfect correlation occurs.

Stopping execution on singular matrices is usually best: the cause needs to be found and somehow resolved. An alternative is to skip those locations and continue. For instance,

```
> setL <- list(cn_max = 1e+10)
> krige(log(zinc) ~ 1, meuse.dup, meuse[1, ], v.fit, set = setL)

[using ordinary kriging]
Warning:
Covariance matrix (nearly) singular at location [181072,333611,0]:
  skipping...
      coordinates var1.pred var1.var
1 (181072, 333611)      NA      NA
```

checks whether the estimated condition number for V and $X'V^{-1}X$ exceeds 10^{10} , in which case NA values are generated for prediction. Larger condition numbers indicate that a matrix is closer to singular. This is by no means a solution. It will also report whether V or $X'V^{-1}X$ are singular; in the latter case the cause is more likely related to collinear regressors, which reside in X .

Near-singularity may not be picked up, and can potentially lead to dramatically bad results: predictions that are orders of magnitude different from the observation data. The causes should be sought in the same direction as real singularity. Setting the `cn_max` value may help finding where this occurs.

8.6 Kriging, Filtering, Smoothing

Kriging results in spatially smooth interpolators that are, in case of a non-zero nugget effect, discontinuous in the measurement points. At measurement points, kriging prediction yields the measured value with a zero prediction error variance. In the following code, we fit a variogram model, compute the kriging prediction at the first observation location

```
> v <- variogram(log(zinc) ~ 1, meuse)
> v.fit <- fit.variogram(v, vgm(1, "Sph", 800, 1))
> v.fit
```

	model	psill	range
1	Nug	0.05065923	0.0000
2	Sph	0.59060463	896.9976

```
> log(meuse$zinc[1])

[1] 6.929517

> krige(log(zinc) ~ 1, meuse, meuse[1, ], v.fit)

[using ordinary kriging]
      coordinates vari.pred vari.var
1 (181072, 333611) 6.929517      0
```

If we would shift this first point spatially with any small amount, say 1 m, we would see a strongly different prediction:

```
> krige(log(zinc) ~ 1, meuse, meuse_shift[1, ], v.fit)

[using ordinary kriging]
      coordinates vari.pred vari.var
1 (181073, 333612) 6.880461 0.089548
```

As a matter of fact, kriging in presence of a nugget effect, when considered as prediction in a small region around a prediction location, is discontinuous and hence not smooth. This effect goes unnoticed when data locations do not coincide with the (gridded) prediction locations.

An alternative approach could conceive the measured process $Z(s)$ as $Z(s) = U(s) + \epsilon(s)$, the sum of an underlying, smooth process $U(s)$ and

a (rough) measurement error $\epsilon(s)$, and one could focus on predicting $U(s)$ rather than $Z(s)$. The motivation for this is that the reproduction (prediction) of measurement errors, even when known, is not of interest. Spatial prediction, or *filtering*, now follows (8.6) and (8.7) but with v being the covariance vector of $Z(s)$ and $U(s_0)$, which does *not* involve the discontinuity of the nugget effect.

Predicting the $U(s)$ process is obtained by rephrasing the nugget effect as an error:

```
> err.fit <- fit.variogram(v, vgm(1, "Sph", 800, Err = 1))
> err.fit

    model      psill    range
1   Err 0.05065923    0.0000
2   Sph 0.59060463  896.9976

> krige(log(zinc) ~ 1, meuse, meuse[1, ], err.fit)

[using ordinary kriging]
      coordinates var1.pred  var1.var
1 (181072, 333611)  6.884405 0.03648707
```

where one can see that the prediction is no longer equal to the data value but very similar to the prediction at the minimally shifted prediction location. The prediction variance is very similar to the kriging variance at the minimally shifted prediction location minus the nugget variance (the variance of $\epsilon(s)$).

One could also find some compromise between kriging and filtering, or nugget and measurement error. When it is known that the measurement error variance is 0.01, a nugget could still be fitted:

```
> v = fit.variogram(v, vgm(1, "Sph", 800, Err = 0.01, nugget = 1),
+   fit.sill = c(FALSE, TRUE, TRUE))
> v

    model      psill    range
1   Err 0.01000000    0.0000
2   Nug 0.04065923    0.0000
3   Sph 0.59060463  896.9976

> krige(log(zinc) ~ 1, meuse[1, ], meuse[1, ], v)

[using ordinary kriging]
      coordinates var1.pred var1.var
1 (181072, 333611)  6.929517    0.01
```

Usually, nugget and measurement error cannot be fitted separately from typical datasets.

8.7 Model Diagnostics

The model diagnostics we have seen so far are fitted and residual plots (for linear regression models), spatial identification of groups of points in the variogram cloud, visual and numerical inspection of variogram models, and visual and numerical inspection of kriging results. Along the way, we have seen many model decisions that needed to be made; the major ones being the following:

- Possible transformation of the dependent variable
- The form of the trend function
- The cutoff, lag width, and possibly directional dependence for the sample variogram
- The variogram model type
- The variogram model coefficient values, or fitting method
- The size and criterion to define a local neighbourhood

and we have seen fairly little *statistical* guidance as to which choices are better. To some extent we can ‘ask’ the data what a good decision is, and for this we may use cross validation. We see that there are some model choices that do not seem very important, and others that cross validation simply cannot inform us about.

8.7.1 Cross Validation Residuals

Cross validation splits the data set into two sets: a modelling set and a validation set. The modelling set is used for variogram modelling and kriging on the locations of the validation set, and then the validation measurements can be compared to their predictions. If all went well, cross validation residuals are small, have zero mean, and no apparent structure.

How should we choose or isolate a set for validation? A possibility is to randomly partition the data in a model and test set. Let us try this for the `meuse` data set, splitting it in 100 observations for modelling and 55 for testing:

```
> sel100 <- sample(1:155, 100)
> m.model <- meuse[sel100, ]
> m.valid <- meuse[-sel100, ]
> v100.fit <- fit.variogram(variogram(log(zinc) ~ 1, m.model),
+   vgm(1, "Sph", 800, 1))
> m.valid.pr <- krige(log(zinc) ~ 1, m.model, m.valid,
+   v100.fit)

[using ordinary kriging]

> resid.kr <- log(m.valid$zinc) - m.valid.pr$var1.pred
> summary(resid.kr)
```

```

      Min.   1st Qu.   Median     Mean  3rd Qu.    Max.
-0.79990 -0.18240 -0.03922 -0.01881  0.19210  1.06900

> resid.mean <- log(m.valid$zinc) - mean(log(m.valid$zinc))
> R2 <- 1 - sum(resid.kr^2)/sum(resid.mean^2)
> R2

[1] 0.717017

```

which indicates that kriging prediction is a better predictor than the mean, with an indicative R^2 of 0.72. Running this analysis again will result in different values, as another random sample is chosen. Also note that no visual verification that the variogram model fit is sensible has been applied. A map with cross validation residuals can be obtained by

```

> m.valid.pr$res <- resid.kr
> bubble(m.valid.pr, "res")

```

A similar map is shown for 155 residuals in Fig. 8.12. Here, symbol size denotes residual size, with symbol area proportional to absolute value.

To use the data to a fuller extent, we would like to use all observations to create a residual once; this may be used to find influential observations. It can be done by replacing the first few lines in the example above with

```

> nfold <- 3
> part <- sample(1:nfold, 155, replace = TRUE)
> sel <- (part != 1)
> m.model <- meuse[sel, ]
> m.valid <- meuse[-sel, ]

```

and next define `sel = (part != 2)`, etc. Again, the random splitting brings in a random component to the outcomes. This procedure is threefold cross validation, and it can be easily extended to n -fold cross validation. When n equals the number of observations, the procedure is called leave-one-out cross validation.

A more automated way to do this is provided by the **gstat** functions `krige.cv` for univariate cross validation and `gstat.cv` for multivariable cross validation:

```

> v.fit <- vgm(0.59, "Sph", 874, 0.04)
> cv155 <- krige.cv(log(zinc) ~ 1, meuse, v.fit, nfold = 5,
+   verbose = FALSE)

> bubble(cv155, "residual", main = "log(zinc): 5-fold CV residuals")

```

the result of which is shown in Fig. 8.12. It should be noted that these functions do not re-fit variograms for each fold; usually a variogram is fit on the complete data set, and in that case validation residuals are not completely independent from modelling data, as they already did contribute to the variogram model fitting.

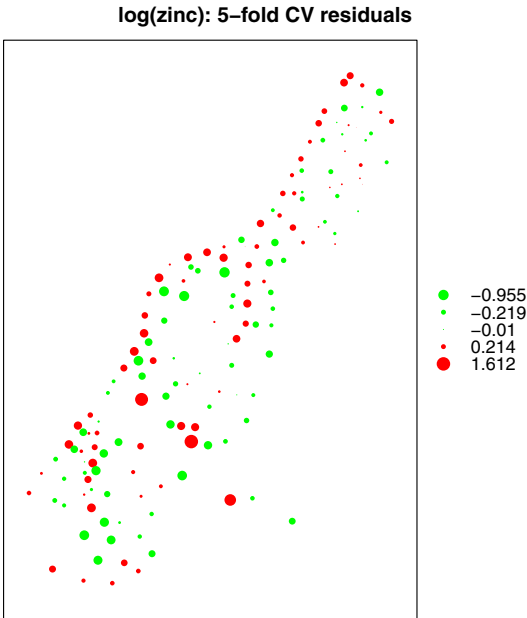


Fig. 8.12 Cross validation residuals for fivefold cross validation; symbol size denotes residual magnitude, positive residuals indicate under-prediction

8.7.2 Cross Validation z-Scores

The `krige.cv` object returns more than residuals alone:

```
> summary(cv155)
```

Object of class SpatialPointsDataFrame
Coordinates:

min	max
x 178605	181390
y 329714	333611

Is projected: NA
proj4string : [NA]
Number of points: 155

Data attributes:

var1.pred		var1.var		observed		residual	
Min.	:4.808	Min.	:0.1102	Min.	:4.727	Min.	:-0.955422
1st Qu.	:5.380	1st Qu.	:0.1519	1st Qu.	:5.288	1st Qu.	:-0.218794
Median	:5.881	Median	:0.1779	Median	:5.787	Median	:-0.010007
Mean	:5.887	Mean	:0.1914	Mean	:5.886	Mean	:-0.001047
3rd Qu.	:6.333	3rd Qu.	:0.2145	3rd Qu.	:6.514	3rd Qu.	: 0.213743
Max.	:7.257	Max.	:0.5370	Max.	:7.517	Max.	: 1.611695

zscore		fold	
Min.	:-2.270139	Min.	:1.000
1st Qu.	:-0.508470	1st Qu.	:2.000
Median	:-0.023781	Median	:3.000

Mean	: 0.001421	Mean	: 2.987
3rd Qu.	: 0.498811	3rd Qu.	: 4.000
Max.	: 3.537729	Max.	: 5.000

the variable `fold` shows to which fold each record belonged, and the variable `zscore` is the z -score, computed as

$$z_i = \frac{Z(s_i) - \hat{Z}_{[i]}(s_i)}{\sigma_{[i]}(s_i)},$$

with $\hat{Z}_{[i]}(s_i)$ the cross validation prediction for s_i , and $\sigma_{[i]}(s_i)$ the corresponding kriging standard error. In contrast to standard residuals the z -score takes the kriging variance into account: it is a standardised residual, and if the variogram model is correct, the z -score should have mean and variance values close to 0 and 1. If, in addition, $Z(s)$ follows a normal distribution, so should the z -score do.

8.7.3 Multivariable Cross Validation

Multivariable cross validation is obtained using the `gstat.cv` function:

```
> g.cv <- gstat.cv(g, nmax = 40)
```

Here, the neighbourhood size is set to the nearest 40 observations for computational reasons. With multivariable cross validation, two additional parameters need be considered:

- `remove.all = FALSE` By default only the first variable is cross-validated, and all other variables are used to their full extent for prediction on the validation locations; if set to `TRUE`, also secondary data at the validation locations are removed.
- `all.residuals = FALSE` By default only residuals are computed and returned for the primary variable; if set to `TRUE`, residuals are computed and returned for all variables.

In a truly multivariable setting, where there is no hierarchy between the different variables to be considered, both should be set to `TRUE`.

8.7.4 Limitations to Cross Validation

Cross validation can be useful to find artefacts in data, but it should be used with caution for confirmatory purposes: one needs to be careful not to conclude that our (variogram and regression) model is correct if cross

validation does not lead to unexpected findings. It is for instance not good at finding what is not in the data.

As an example, the Meuse data set does not contain point pairs with a separation distance closer than 40. Therefore, the two variogram models

```
> v1.fit <- vgm(0.591, "Sph", 897, 0.0507)
> v2.fit <- vgm(0.591, "Sph", 897, add.to = vgm(0.0507,
+   "Sph", 40))
```

that only differ with respect to the spatial correlation at distances smaller than 40 m yield identical cross validation results:

```
> set.seed(13331)
> cv155.1 <- krige.cv(log(zinc) ~ 1, meuse, v1.fit, nfold = 5,
+   verbose = FALSE)
> set.seed(13331)
> cv155.2 <- krige.cv(log(zinc) ~ 1, meuse, v2.fit, nfold = 5,
+   verbose = FALSE)
> summary(cv155.1$residual - cv155.2$residual)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0	0	0	0	0	0

Note that the `set.seed(13331)` was used here to force identical assignments of the otherwise random folding. When used for block kriging, the behaviour of the variogram at the origin is of utmost importance, and the two models yield strongly differing results. As an example, consider block kriging predictions at the `meuse.grid` cells:

```
> b1 <- krige(log(zinc) ~ 1, meuse, meuse.grid, v1.fit,
+   block = c(40, 40))$var1.var
```

[using ordinary kriging]

```
> b2 <- krige(log(zinc) ~ 1, meuse, meuse.grid, v2.fit,
+   block = c(40, 40))$var1.var
```

[using ordinary kriging]

```
> summary((b1 - b2)/b1)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
-0.4313	-0.2195	-0.1684	-0.1584	-0.1071	0.4374

where some kriging variances drop, but most increase, up to 30 % when using the variogram without nugget instead of the one with a nugget. The decision which variogram to choose for distances shorter than those available in the data is up to the analyst, and matters.

8.8 Geostatistical Simulation

Geostatistical simulation refers to the simulation of possible realisations of a random field, given the specifications for that random field (e.g. mean structure, residual variogram, intrinsic stationarity) and possibly observation data. Conditional simulation produces realisations that exactly honour observed data at data locations, unconditional simulations ignore observations and only reproduce means and prescribed variability.

Geostatistical simulation is fun to do, to see how much (or little) realisations can vary given the model description and data, but it is a poor means of quantitatively communicating uncertainty: many realisations are needed and there is no obvious ordering in which they can be viewed. They are, however, often needed when the uncertainty of kriging predictions is, for example input to a next level of analysis, and spatial correlation plays a role. An example could be the use of rainfall fields as input to spatially distributed rainfall-runoff models: interpolated values and their variances are of little value, but running the rainfall-runoff model with a large number of simulated rainfall fields may give a realistic assertion of the uncertainty in runoff, resulting from uncertainty in the rainfall field.

Calculating runoff given rainfall and catchment characteristic can be seen as a non-linear spatial aggregation process. Simpler non-linear aggregations are, for example for a given area or block the fraction of the variable that exceeds a critical limit, the 90th percentile within that area, or the actual area where (or its size distribution for which) a measured concentration exceeds a threshold. Simulation can give answers in terms of predictions as well as predictive distributions for all these cases. Of course, the outcomes can never be better than the degree to which the simulation model reflects reality.

The next subsection introduces sequential simulation, a very generic and flexible method. Package **RandomFields** provides a large number of other simulation algorithms; packages **RandomFields** and **fields** both contain implementations of the circulant embedding method (Dietrich and Newsam, 1993), which is often preferred for its computational speed.

8.8.1 Sequential Simulation

For simulating random fields, package **gstat** only provides the sequential simulation algorithm (see, e.g. Goovaerts, 1997 for an explanation), and provides this for Gaussian simulation and indicator simulation, possibly multivariable, optionally with simulation of trend components, and optionally for block mean values.

Sequential simulation proceeds as follows; following a random path through the simulation locations, it repeats the following steps:

1. Compute the conditional distribution given data and previously simulated values, using simple kriging
2. Draw a value from this conditional distribution
3. Add this value to the data set
4. Go to the next unvisited location, and go back to 1

until all locations have been visited. In step 2, either the Gaussian distribution is used or the indicator kriging estimates are used to approximate a conditional distribution, interpreting kriging estimates as probabilities (after some fudging!).

Step 1 of this algorithm will become the most computationally expensive when all data (observed and previously simulated) are used. Also, as the number of simulation nodes is usually much larger than the number of observations, the simulation process will slow down more and more when global neighbourhoods are used. To obtain simulations with a reasonable speed, we need to set a maximum to the neighbourhood. This is best done with the `nmax` argument, as spatial data density increases when more and more simulated values are added. For simulation we again use the functions `krige` or `predict.gstat`; the argument `nsim` indicates how many realisations are requested:

```
> lzn.sim <- krige(log(zinc) ~ 1, meuse, meuse.grid, v.fit,
+               nsim = 6, nmax = 40)

drawing 6 GLS realisations of beta...
[using conditional Gaussian simulation]

> spplot(lzn.sim)
```

the result of which is shown in Fig. 8.13. It should be noted that these realisations are created following a single random path, in which case the expensive results ($V^{-1}v$ and the neighbourhood selection) can be re-used. Alternatively, one could use six function calls, each with `nsim = 1`.

The simulation procedure above also gave the output line **drawing 6 GLS realisations of beta...**, which confirms that prior to simulation of the field *for each realisation* a trend vector (in this case a mean value only) is drawn from the normal distribution with mean $(X'V^{-1}X)^{-1}X'V^{-1}Z(s)$ and variance $(X'V^{-1}X)^{-1}$, that is the generalised least squares estimate and estimation variance. This procedure leads to simulations that have mean and variance equal to the ordinary or universal kriging mean and variance, and that have residual spatial correlation according to the variogram prescribed (Abrahamsen and Benth, 2001). For simulations that omit the simulation of the trend coefficients, the vector β should be passed, for example as `beta = 5.9` to the `krige` function, as with the simple kriging example. In that case, the simulated fields will follow the simple kriging mean and variance.

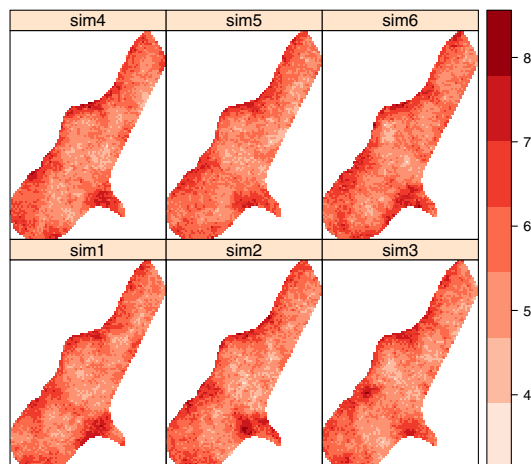


Fig. 8.13 Six realisations of conditional Gaussian simulation for log-zinc

8.8.2 Non-linear Spatial Aggregation and Block Averages

Suppose the area shown in Fig. 8.14 is the target area for which we want to know the fraction above a threshold; the area being available as a `SpatialPolygons` object `area`. We can now compute the distribution of the fraction of the area above a cutoff of 500 ppm by simulation:

```
> nsim <- 1000
> cutoff <- 500
> sel.grid <- meuse.grid[area.sp, ]
> lzn.sim <- krige(log(zinc) ~ 1, meuse, sel.grid, v.fit,
+   nsim = nsim, nmax = 40)

drawing 1000 GLS realisations of beta...
[using conditional Gaussian simulation]

> res <- apply(as.data.frame(lzn.sim)[1:nsim], 2, function(x) mean(x >
+   log(cutoff)))

> hist(res, main = paste("fraction above", cutoff), xlab = NULL,
+   ylab = NULL)

shown in the right-hand side of Fig. 8.14. Note that if we had been interested
in the probability of mean(x) > log(cutoff), which is a rather different
issue, then block kriging would have been sufficient:

> bkr <- krige(log(zinc) ~ 1, meuse, area.sp, v.fit)

[using ordinary kriging]

> 1 - pnorm(log(cutoff), bkr$var1.pred, sqrt(bkr$var1.var))

[1] 0.9998791
```

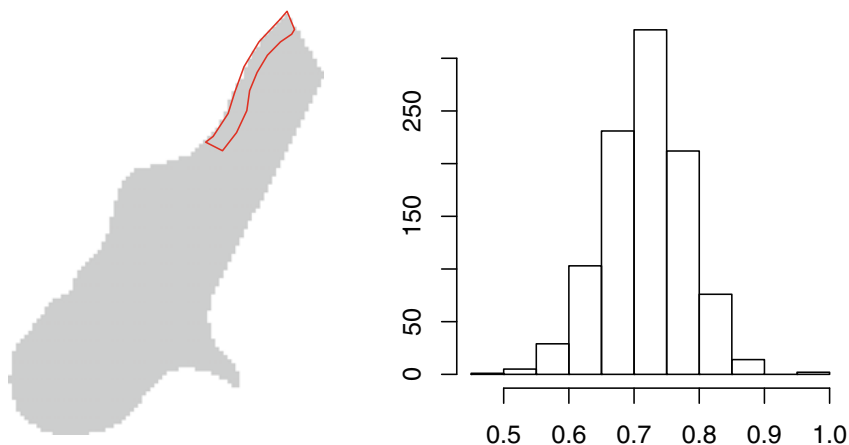


Fig. 8.14 A non-rectangular area for which a non-linear aggregation is required (*left: red*), and distribution of the fraction with zinc concentration above 500 ppm for this area

Block averages can be simulated directly by supplying the `block` argument to `krige`; simulating points and aggregating these to block means *may* be more efficient because simulating blocks calls for the calculation of many block-block covariances, which involves the computation of quadruple integrals.

8.8.3 Multivariable and Indicator Simulation

Multivariable simulation is as easy as cokriging, try

```
> cok.sims <- predict(vm.fit, meuse.grid, nsim = 1000)
```

after passing the `nmax = 40`, or something similar to the `gstat` calls used to build up `vm.fit` (Sect. 8.4.5).

Simulation of indicators is done along the same lines. Suppose we want to simulate soil class 1, available in the Meuse data set:

```
> table(meuse$soil)
 1  2  3
97 46 12

> s1.fit <- fit.variogram(variogram(I(soil == 1) ~ 1, meuse),
+   vgm(1, "Sph", 800, 1))
> s1.sim <- krige(I(soil == 1) ~ 1, meuse, meuse.grid,
+   s1.fit, nsim = 6, indicators = TRUE, nmax = 40)

drawing 6 GLS realisations of beta...
[using conditional indicator simulation]
> spplot(s1.sim)
```

which is shown in Fig. 8.15.

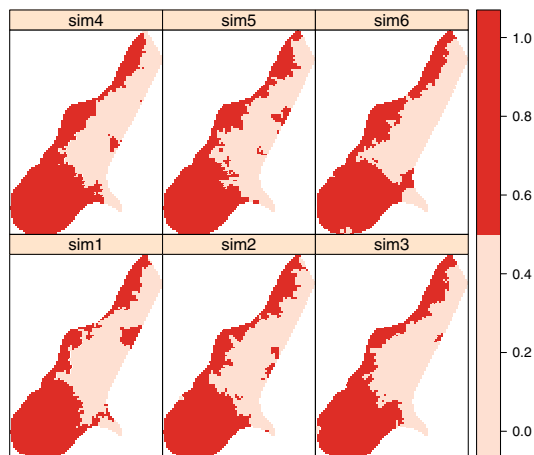


Fig. 8.15 Six realisations of conditional indicator simulation for soil type 1

8.9 Model-Based Geostatistics and Bayesian Approaches

Up to now, we have only seen kriging approaches where it was assumed that the variogram model, fitted from sample data, is assumed to be *known* when we do the kriging or simulation: any uncertainty about it is ignored. Diggle et al. (1998) give an approach, based on linear mixed and generalized linear mixed models, to provide what they call *model-based* geostatistical predictions. It incorporates the estimation error of the variogram coefficients.

When is uncertainty of the variogram an important issue? Obviously, when the sample is small, or, for example when variogram modelling is problematic due to the presence of extreme observations or data that come from a strongly skewed distribution.

8.10 Monitoring Network Optimisation

NA Monitoring costs money. Monitoring programs have to be designed, started, stopped, evaluated, and sometimes need to be enlarged or shrunk. The difficulty of finding optimal network designs (Mateu and Müller, 2013) is that a quantitative criterion is often a priori not present. For example, should one focus on mean kriging variances, or variance of some global mean estimator, or rather on the ability to delineate a particular contour?

A very simple approach towards monitoring network optimisation is to find the point whose removal leads to the smallest increase in mean kriging variance:


```
> m1 <- sapply(1:155, function(x) mean(krige(log(zinc) ~
+ 1, meuse[-x, ], meuse.grid, v.fit)$var1.var))
> which(m1 == min(m1))
```

which will point to observation 72 as the first candidate for removal. Looking at the sorted magnitudes of change in mean kriging variance, by

```
> plot(sort(m1))
```

will reveal that for several other candidate points their removal will have an almost identical effect on the mean variance.

Another approach could be, for example to delineate say the 500ppm contour. We could, for example express the doubt about whether a location is below or above 500 as the closeness of $G((\hat{Z}(s_0) - 500)/\sigma(s_0))$ to 0.5, with $G(\cdot)$ the Gaussian distribution function.

```
> cutoff <- 1000
> f <- function(x) {
+   kr = krige(log(zinc) ~ 1, meuse[-x, ], meuse.grid,
+     v.fit)
+   mean(abs(pnorm((kr$var1.pred - log(cutoff))/sqrt(kr$var1.var)) -
+     0.5))
+ }
> m2 <- sapply(1:155, f)
> which(m2 == max(m2))
```

Figure 8.16 shows that different objectives lead to different candidate points. Also, deciding based on the kriging variance alone results in an outcome that is highly predictable from the points configuration alone: points in the densest areas are candidate for removal.

For *adding* observation points, one could loop over a fixed grid and find the point that increases the objective most; this is more work as the number of options for addition are much larger than that for removal. Evaluating on the kriging variance is not a problem, as the observation value does not affect the kriging variance. For the second criterion, it does.

The problem when two or more points have to be added or removed *jointly* becomes computationally very intensive, as the sequential solution (first the best point, then the second best) is not necessarily equal to the joint solution, for example which configuration of n points is best. Instead of an exhaustive search a more clever optimisation strategy such as simulated annealing or a genetic algorithm should be used.

Package **fields** has a function `cover.design` that finds a set of points on a finite grid that minimises a geometric space-filling criterion



Fig. 8.16 Candidate points for removal. *Left*: for mean kriging variance, *right*: for delineating the 1,000 ppm contour. *Open, red circles*: 10 % most favourite points; *closed, green circles*: 10 % least favourite points

8.11 Other R Packages for Interpolation and Geostatistics

8.11.1 *Non-geostatistical Interpolation*

Besides inverse distance weighted interpolation and kriging, other interpolation methods may be used, for example based on generalised additive models or smoothers, such as given in package **mgcv**. Additive models in coordinates without interaction will not yield rotation-invariant solutions, but two-dimensional smoothing splines will. The interested reader is referred to Wood (2006).

Package **fields** also provides a function **Tps**, for thin plate (smoothing) splines. Package **akima** provides interpolation methods based on bilinear or bicubic splines (Akima, 1978); the package has a restrictive license, and work on an unencumbered replacement is planned.

Package **stinepack** provides a ‘consistently well behaved method of interpolation based on piecewise rational functions using Stineman’s algorithm’ (Stineman, 1980).

An interpolation method that also has this property but that does take observation configuration into account is *natural neighbour* interpolation (Sibson, 1981), but this is not available in R.

None of the packages mentioned in this sub-section accept or return data in one of the **Spatial** classes of package **sp**.

8.11.2 *Spatial*

Package **spatial** is part of the **VR** bundle that comes with Venables and Ripley (2002) and is probably one of the oldest spatial packages available in R. It provides calculation of spatial correlation using functions **correlogram** or **variogram**. It allows ordinary and universal point kriging over a grid for spherical, exponential, and Gaussian covariance models. For universal kriging predictors it only allows polynomials in the spatial coordinates. Function **surf.gls** fits a trend surface (i.e. a polynomial in the coordinates) by generalised least squares, and it has a **predict** method.

8.11.3 *RandomFields*

RandomFields offers sample variogram computation, variogram fitting by least squares or maximum likelihood or restricted maximum likelihood. Simple and ordinary point kriging are provided, and unconditional and conditional simulation using a large variety of modern simulation methods different from sequential simulation, many of them based on spectral methods and Fourier transforms; their abbreviated code is shown as column labels in the table obtained by

```
> PrintModelList()
```

an explanation of its output is available in the help for **PrintMethodList**. The package provides a large set of covariance functions, shown as row labels in the model list.

8.11.4 *geoR and geoRglm*

In addition to variogram estimation, variogram model function fitting using least squares or (restricted) maximum likelihood (**likfit**), and ordinary and universal point kriging, package **geoR** allows for Bayesian kriging (function **krige.bayes** of (transformed) Gaussian variables). This requires the user to specify priors for each of the variogram model parameters (but not for the trend coefficients); **krige.bayes** will then compute the posterior kriging distribution. The function documentation points to a long list of documents that describe the implementation details. The book by Diggle and Ribeiro Jr. (2007) describes more details and gives worked examples.

Package **geoR** uses its own class for spatial data, called **geodata**. It contains coercion method for point data in **sp** format, try, for example

```
> library(geoR)
> plot(variog(as.geodata(meuse["zinc"]), max.dist = 1500))
```

Package **geoR** also has an `xvalid` function for leave-one-out cross validation that (optionally) allows for re-estimating model parameters (trend and variogram coefficients) when leaving out each observation. It also provides the **eyefit**, for interactive visual fitting of functions to sample variograms (see Sect. 8.4.3).

Package **geoRglm** extends package **geoR** for binomial and Poisson processes, and includes Bayesian and conventional kriging methods for trans-Gaussian processes. It mostly uses MCMC approaches, and may be slow for larger data sets.

8.11.5 *Fields*

Package **fields** is an R package for curve and function fitting with an emphasis on spatial data. The main spatial prediction methods are thin plate splines (function `Tps`) and kriging (function `Krig` and `krig.image`). The kriging functions allow you to supply a covariance function that is written in native code. Functions that are positive definite on a sphere (i.e. for unprojected data) are available. Function `cover.design` is written for monitoring network optimisation.

8.11.6 *spBayes*

Package **spBayes** fits univariate and multivariate models with Markov chain Monte Carlo (MCMC). Core functions include univariate and multivariate Gaussian regression with spatial random effects (functions `spLM`, `spMvLM`) and univariate and multivariate logistic and Poisson regression with spatial random effects (functions `spGLM` and `spMvGLM`).

8.12 Spatio-Temporal Prediction

Spatio-temporal prediction methods are provided by several R packages. Package **gstat** contains a vignette that explains how to compute and fit spatio-temporal variogram models, and use them in spatio-temporal kriging. It uses the classes of **spacetime** for this. Variogram models include the metric, product-sum, sum-metric, and separable model. The vignette is accessed by

```
> vignette("st", package = "gstat")
```

Other packages providing particular space-time variogram model fitting and prediction options include **RandomFields**, **SpatioTemporal**, and **spTimer**. The SpatioTemporal task view on CRAN gives an overview of packages for analyzing spatio-temporal data, which includes spatio-temporal geostatistics. It is found at <http://cran.r-project.org/view=SpatioTemporal>.

Chapter 9

Modelling Areal Data

9.1 Introduction

Spatial data are often observed on polygon entities with defined boundaries. The polygon boundaries are defined by the researcher in some fields of study, may be arbitrary in others and may be administrative boundaries created for very different purposes in others again. The observed data are frequently aggregations within the boundaries, such as population counts. The areal entities may themselves constitute the units of observation, for example when studying local government behaviour where decisions are taken at the level of the entity, for example setting local tax rates. By and large, though, areal entities are aggregates, bins, used to tally measurements, like voting results at polling stations. Very often, the areal entities are an exhaustive tessellation of the study area, leaving no part of the total area unassigned to an entity. Of course, areal entities may be made up of multiple geometrical entities, such as islands belonging to the same county; they may also surround other areal entities completely, and may contain holes, like lakes.

The boundaries of areal entities may be defined for some other purpose than their use in data analysis. Postal code areas can be useful for analysis, but were created to facilitate postal deliveries. It is only recently that national census organisations have accepted that frequent, apparently justified, changes to boundaries are a major problem for longitudinal analyses. In Sect. 5.1, we discussed the concept of spatial support, which here takes the particular form of the *modifiable areal unit problem* (Waller and Gotway, 2004, pp. 104–108). Arbitrary areal unit boundaries are a problem if their modification could lead to different results, with the case of political gerrymandering being a sobering reminder of how changes in aggregation may change outcomes.¹ They may also get in the way of the analysis if the spatial scale

¹ The **BARD** package for automated redistricting and heuristic exploration of redistricter revealed preference is an example of the use of R for studying this problem.

or footprint of an underlying data generating process is not matched by the chosen boundaries.

If data collection can be designed to match the areal entities to the data, the influence of the choice of aggregates will be reduced. An example could be the matching of labour market data to local labour markets, perhaps defined by journeys to work. On the other hand, if we are obliged to use arbitrary boundaries, often because there are no other feasible sources of secondary data, we should be aware of potential difficulties. Such mismatches are among the reasons for finding spatial autocorrelation in analysing areal aggregates; other reasons include substantive spatial processes in which entities influence each other by contagion, such as the adoption of similar policies by neighbours, and model misspecification leaving spatially patterned information in the model residuals. These causes of observed spatial autocorrelation can occur in combination, making the correct identification of the actual spatial processes an interesting undertaking; Dray et al. (2012) discuss many of the issues involved in undertaking such identification.

A wide range of scientific disciplines have encountered spatial autocorrelation among areal entities, with the term ‘Galton’s problem’ used in several. The problem is to establish how many effectively independent observations are present, when arbitrary boundaries have been used to divide up a study area. In his exchange with Tyler in 1889, Galton questioned whether observations of marriage laws across areal entities constituted independent observations, since they could just reflect a general pattern from which they had all descended. So positive spatial dependence tends to reduce the amount of information contained in the observations, because proximate observations can in part be used to predict each other.

In Chap. 8, we have seen how distances on a continuous surface can be used to structure spatial autocorrelation, for example with the variogram. Here we will be concerned with areal entities that are defined as neighbours, for chosen definitions of neighbours. On a continuous surface, all points are neighbours of each other, though some may carry very little weight, because they are very distant. On a tessellated surface, we can choose neighbour definitions that partition the set of all entities (excluding observation i) into members or non-members of the neighbour set of observation i . We can also decide to give each neighbour relationship an equal weight, or vary the weights on the arcs of the directed graph describing the spatial dependence.

The next section will cover the construction of neighbours and weights that can be applied to neighbourhoods. Once this important and often demanding prerequisite is in place, we go on to look at ways of measuring spatial autocorrelation. While the tests build on models of spatial processes, we look at tests first, and only subsequently move on to modelling. We will also be interested to show how spatial autocorrelation can be introduced into independent data, so that simulations can be undertaken. It is worth bearing in mind that the spatial patterning we find may only indicate that our current model of

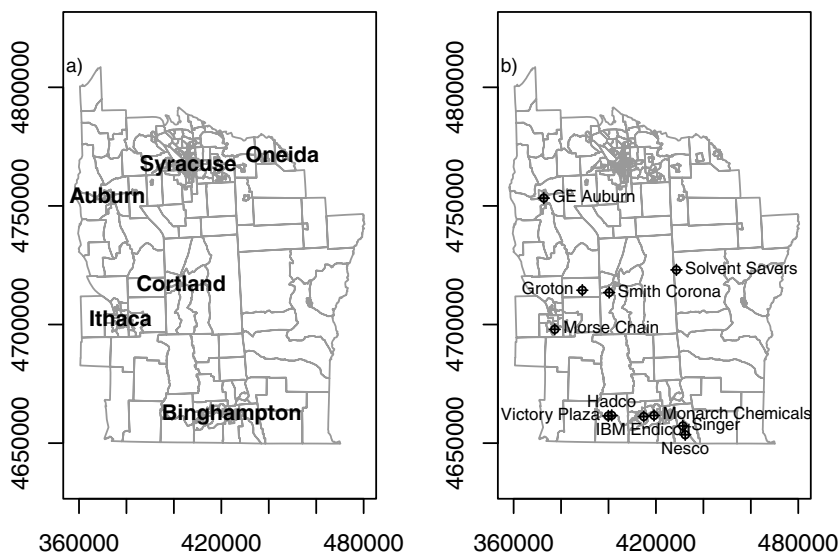


Fig. 9.1 (a) Major cities in the eight-county upper New York State study area; (b) locations of 11 inactive hazardous waste sites in the study area

the data is not appropriate. This applies to areal units not fitting the data generation process, to missing variables including variables with the wrong functional form, and differences between our assumptions about the data and their actual distributions, often shown as over-dispersion in count data. The modelling of areal data will be dealt with in the subsequent sections, with extensions in Chap. 10.

The 281 census tract data set for eight central New York State counties featured prominently in Waller and Gotway (2004) will be used in many of the examples,² supplemented with tract boundaries derived from TIGER 1992 and distributed by SEDAC/CIESIN. This file is not identical with the boundaries used in the original source, but is very close and may be re-distributed, unlike the version used in the book. The area has an extent of about 160 km north–south and 120 km east–west; Fig. 9.1 shows the major cities in the study area and the location of 11 hazardous waste sites. The figures in Waller and Gotway (2004) include water bodies, which are not present in this version of the tract boundaries, in which tract boundaries follow the centre lines of lakes, rather than their shores.

² The boundaries have been projected from geographical coordinates to UTM zone 18.

9.2 Spatial Neighbours and Spatial Weights

Creating spatial weights is a necessary step in using areal data, perhaps just to check that there is no remaining spatial patterning in residuals. The first step is to define which relationships between observations are to be given a non-zero weight, that is to choose the neighbour criterion to be used; the second is to assign weights to the identified neighbour links. Trying to detect pattern in maps of residuals visually is not an acceptable choice, although one sometimes hears comments explaining the lack of formal analysis such as ‘they looked random’, or alternatively ‘I can see the clusters’. Making the neighbours and weights is, however, not easy to do, and so a number of functions are included in the **spdep** package to help. Further functions are found in some ecology packages, such as the **ade4** package – this package also provides **nb2neig** and **neig2nb** converters for interoperability. The construction of spatial weights is touched on by Cressie (1993, pp. 384–385), Schabenberger and Gotway (2005, p. 18), Waller and Gotway (2004, pp. 223–225), Fortin and Dale (2005, pp. 113–118), O’Sullivan and Unwin (2010, pp. 200–204), Ward and Gleditsch (2008, pp. 14–22), Chun and Griffith (2013, pp. 9, 57–59), and Banerjee et al. (2004, pp. 70–71). The paucity of treatments in the literature contrasts with the strength of the prior information being introduced by the analyst at this stage. Since analysing areal data is dependent on the choices made in constructing the spatial weights, we have chosen to move the detailed discussion of the creation of neighbour objects included in the first edition of this book to a vignette included in the **spdep** package. The vignettes: “nb”, “C069” and “sids” in **spdep** all include discussions of the creation and use of spatial weights, and may be accessed by:

```
> vignette("nb", package = "spdep")
```

9.2.1 Neighbour Objects

In the **spdep** package, neighbour relationships between n observations are represented by an object of class **nb**; the class is an old-style class as presented on p. 24. It is a list of length n with the index numbers of neighbours of each component recorded as an integer vector. If any observation has no neighbours, the component contains an integer zero. It also contains attributes, typically a vector of character region identifiers, and a logical value indicating whether the relationships are symmetric. The region identifiers can be used to check for integrity between the data themselves and the neighbour object. The helper function **card** returns the cardinality of the neighbour set for each object, that is, the number of neighbours; it differs from the application of **length** to the list components because no-neighbour entries are coded as a single element integer vector with the value of zero.

```

> library(spdep)

> library(rgdal)
> NY8 <- readOGR(".", "NY8_utm18")
> NY_nb <- read.gal("NY_nb.gal", region.id = row.names(NY8))

> summary(NY_nb)

Neighbour list object:
Number of regions: 281
Number of nonzero links: 1522
Percentage nonzero weights: 1.927534
Average number of links: 5.41637
Link number distribution:

 1  2  3  4  5  6  7  8  9 10 11
6 11 28 45 59 49 45 23 10  3  2
6 least connected regions:
55 97 100 101 244 245 with 1 link
2 most connected regions:
34 82 with 11 links

> plot(NY8, border = "grey60")
> plot(NY_nb, coordinates(NY8), pch = 19, cex = 0.6, add = TRUE)

```

Starting from the census tract queen contiguities, where all touching polygons are neighbours, used in Waller and Gotway (2004) and provided as a DBF file on their website, a GAL format file has been created and read into R— we return to the import and export of neighbours and weights on p. 273.

Since we now have an `nb` object to examine, we can present the standard methods for these objects. There are `print`, `summary`, `plot`, and other methods; the `summary` method presents a table of the link number distribution, and both `print` and `summary` methods report asymmetry and the presence of no-neighbour observations; asymmetry is present when i is a neighbour of j but j is not a neighbour of i . Figure 9.2 shows the complete neighbour graph for the eight-county study area. For the sake of simplicity in showing how to create neighbour objects, we work on a subset of the map consisting of the census tracts within Syracuse, although the same principles apply to the full data set. We retrieve the part of the neighbour list in Syracuse using the `subset` method.

```

> Syracuse <- NY8[NY8$AREANAME == "Syracuse city", ]
> Sy0_nb <- subset(NY_nb, NY8$AREANAME == "Syracuse city")
> summary(Sy0_nb)

Neighbour list object:
Number of regions: 63
Number of nonzero links: 346
Percentage nonzero weights: 8.717561
Average number of links: 5.492063
Link number distribution:

```

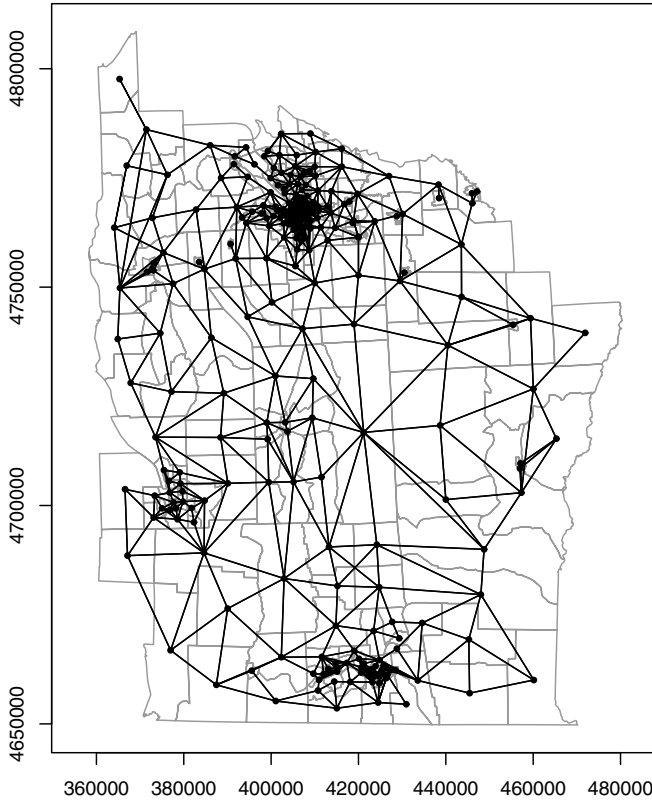


Fig. 9.2 Census tract contiguities, New York eight-county census tracts

```

1 2 3 4 5 6 7 8 9
1 1 5 9 14 17 9 6 1
1 least connected region:
164 with 1 link
1 most connected region:
136 with 9 links

```

For use later in this discussion, we create a number of neighbour objects in other ways. Three are k nearest neighbour objects, taking k nearest points as neighbours, for $k = 1, 2, 4$ – this adapts across the study area, taking account of differences in the densities of areal entities. Naturally, in the overwhelming majority of cases, it leads to asymmetric neighbours, but will ensure that all areas have k neighbours.

```

> coords <- coordinates(Syracuse)
> IDs <- row.names(Syracuse)
> Sy8_nb <- knn2nb(knearneigh(coords, k = 1), row.names = IDs)
> Sy9_nb <- knn2nb(knearneigh(coords, k = 2), row.names = IDs)

```

```
> Sy10_nb <- knn2nb(knearneigh(coords, k = 4), row.names = IDs)
> dsts <- unlist(nbdists(Sy8_nb, coords))
> Sy11_nb <- dnearneigh(coords, d1 = 0, d2 = 0.75 * max(dsts),
+   row.names = IDs)
```

The $k = 1$ object is also useful in finding the minimum distance at which all areas have a distance-based neighbour. Using the `nbdists` function, we can calculate a list of vectors of distances corresponding to the neighbour object, here for first nearest neighbours. The greatest value will be the minimum distance needed to make sure that all the areas are linked to at least one neighbour, so we can easily create an object with no-neighbour observations by setting the upper threshold under this value.

9.2.2 Spatial Weights Objects

The literature on spatial weights is surprisingly small, given their importance in measuring and modelling spatial dependence in areal data. Griffith (1995) provides sound practical advice, while Bavaud (1998) seeks to insert conceptual foundations under ad hoc spatial weights. Spatial weights can be seen as a list of weights indexed by a list of neighbours, where the weight of the link between i and j is the k th element of the i th weights list component, and k tells us which of the i th neighbour list component values is equal to j . If j is not present in the i th neighbour list component, j is not a neighbour of i . Consequently, some weights w_{ij} in the $\mathbf{W}$ weights matrix representation will set to zero, where j is not a neighbour of i . Here, we follow Tiefelsdorf et al. (1999) in our treatment, using their abstraction of spatial weights styles.

Once the list of sets of neighbours for our study area is established, we proceed to assign spatial weights to each relationship. If we know little about the assumed spatial process, we try to avoid moving far from the binary representation of a weight of unity for neighbours (Bavaud, 1998), and zero otherwise. In this section, we review the ways that weights objects – `listw` objects – are constructed; the class is an old-style class as described on p. 24. Next, the conversion of these objects into dense and sparse matrix representations will be shown, concluding with functions for importing and exporting neighbour and weights objects.

The `nb2listw` function takes a neighbours list object and converts it into a weights object. The default conversion style is `W`, where the weights for each areal entity are standardised to sum to unity; this is also often called row standardisation. The `print` method for `listw` objects shows the characteristics of the underlying neighbours, the style of the spatial weights, and the spatial weights constants used in calculating tests of spatial autocorrelation. The `neighbours` component of the object is the underlying `nb` object, which gives the indexing of the `weights` component.

```

> Sy0_lw_W <- nb2listw(Sy0_nb)
> Sy0_lw_W

Characteristics of weights list object:
Neighbour list object:
Number of regions: 63
Number of nonzero links: 346
Percentage nonzero weights: 8.717561
Average number of links: 5.492063

Weights style: W
Weights constants summary:
      n  nn S0      S1      S2
W 63 3969 63 24.78291 258.564

> names(Sy0_lw_W)

[1] "style"      "neighbours" "weights"

> names(attributes(Sy0_lw_W))

[1] "names"      "class"      "region.id"  "call"      "GeoDa"

```

For `style="W"`, the weights vary between unity divided by the largest and smallest numbers of neighbours, and the sums of weights for each areal entity are unity. This spatial weights style can be interpreted as allowing the calculation of average values across neighbours. The weights for links originating at areas with few neighbours are larger than those originating at areas with many neighbours, perhaps boosting areal entities on the edge of the study area unintentionally. This representation is no longer symmetric, but is similar to symmetric – this matters as we see below in Sect. 9.4.1.1.

```

> 1/rev(range(card(Sy0_lw_W$neighbours)))

[1] 0.1111111 1.0000000

> summary(unlist(Sy0_lw_W$weights))

  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.1111 0.1429 0.1667 0.1821 0.2000 1.0000

> summary(sapply(Sy0_lw_W$weights, sum))

  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    1      1      1      1      1      1

```

Setting `style="B"` – ‘binary’ – retains a weight of unity for each neighbour relationship, but in this case, the sums of weights for areas differ according to the numbers of neighbour areas have.

```

> Sy0_lw_B <- nb2listw(Sy0_nb, style = "B")
> summary(unlist(Sy0_lw_B$weights))

  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    1      1      1      1      1      1

```

```
> summary(sapply(Sy0_lw_B$weights, sum))
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
1.000	4.500	6.000	5.492	6.500	9.000

The `glist` argument can be used to pass a list of vectors of general weights corresponding to the neighbour relationships to `nb2listw`. Say that we believe that the strength of neighbour relationships attenuates with distance, one of the cases considered by Cliff and Ord (1981, pp.17–18) and O’Sullivan and Unwin (2010, pp.200–204) provide a similar discussion. We could set the weights to be proportional to the inverse distance between points representing the areas, using `nbdists` to calculate the distances for the given `nb` object. Using `lapply` to invert the distances, we can obtain a different structure of spatial weights from those above. If we have no reason to assume any more knowledge about neighbour relations than their existence or absence, this step is potentially misleading. If we do know, on the other hand, that migration or commuting flows describe the spatial weights’ structure better than the binary alternative, it may be worth using them as general weights; there may, however, be symmetry problems, because such flows – unlike inverse distances – are only rarely symmetric.

```
> dsts <- nbdists(Sy0_nb, coordinates(Syracuse))
> idw <- lapply(dsts, function(x) 1/(x/1000))
> Sy0_lw_idwB <- nb2listw(Sy0_nb, glist = idw, style = "B")
> summary(unlist(Sy0_lw_idwB$weights))
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0.3886	0.7374	0.9259	0.9963	1.1910	2.5270

```
> summary(sapply(Sy0_lw_idwB$weights, sum))
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
1.304	3.986	5.869	5.471	6.737	9.435

Figure 9.3 shows three representations of spatial weights for Syracuse displayed as matrices. The `style="W"` image on the left is evidently asymmetric, with darker colours showing larger weights for areas with few neighbours. The other two panels are symmetric, but express different assumptions about the strengths of neighbour relationships.

The final argument to `nb2listw` allows us to handle neighbour lists with no-neighbour areas. It is not obvious that the weight representation of the empty set is zero – perhaps it should be `NA`, which would lead to problems later.

For this reason, the default value of the argument is `zero.policy=FALSE`, leading to an error when given an `nb` argument with areas with no neighbours. Setting the argument to `TRUE` permits the creation of the spatial weights object, with zero weights. The `zero.policy` argument will subsequently need to be used in each function called, unless set `TRUE` for the current R session with `set.ZeroPolicyOption(TRUE)`. The contrast between the set-based understanding of neighbours and conversion to a matrix representation

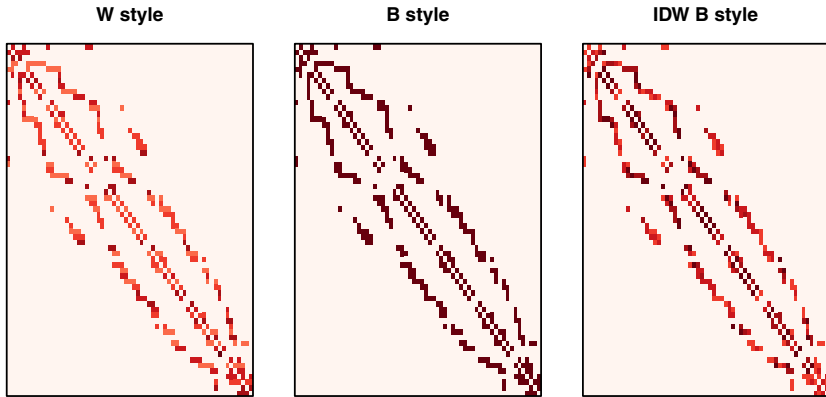


Fig. 9.3 Three spatial weights representations for Syracuse

is discussed by Bivand and Portnov (2004), and boils down to whether the product of a no-neighbour area's weights and an arbitrary n -vector should be a missing value or numeric zero. As we see later (p. 279), keeping the no-neighbour areal entities raises questions about the relevant size of n when testing for autocorrelation, among other issues.

```
> Sy0_lw_D1 <- nb2listw(Sy11_nb, style = "B")

Error in nb2listw(Sy11_nb, style = "B") : Empty neighbour sets found

> Sy0_lw_D1 <- nb2listw(Sy11_nb, style = "B", zero.policy = TRUE)
> print(Sy0_lw_D1, zero.policy = TRUE)

Characteristics of weights list object:
Neighbour list object:
Number of regions: 63
Number of nonzero links: 230
Percentage nonzero weights: 5.794911
Average number of links: 3.650794
2 regions with no links:
154 168

Weights style: B
Weights constants summary:
  n  nn  S0  S1  S2
B 61 3721 230 460 4496
```

The parallel problem of data sets with missing values in variables but with fully specified spatial weights is approached through the `subset.listw` method, which re-generates the weights for the given subset of areas, for example given by `complete.cases`. Knowing which observations are incomplete, the underlying neighbours and weights can be subsetting in some cases, with the aim of avoiding the propagation of NA values when calculating spatially lagged values. Many tests and model fitting functions can carry this

out internally if the appropriate argument flag is set, although the careful analyst will prefer to subset the input data and the weights before testing or modelling.

9.2.3 Handling Spatial Weights Objects

There are several contributed packages providing support for sparse matrices, among which **Matrix**, is a recommended package. The `as_dgRMatrix_listw` wrapper converts a `listw` object to the **Matrix** sorted compressed row-oriented form sparse matrix, as a `dgRMatrix` object, a subclass of the virtual `RsparseMatrix` class. It is easier to make a row-oriented sparse matrix from a spatial weights object as the weights are themselves row-oriented. A function that is used a good deal within testing and model fitting functions is `listw2U`, which returns a symmetric `listw` object representing the $\frac{1}{2}(W + W^T)$ spatial weights matrix.

Neighbour and weights objects produced in other software can be imported into R without difficulty, and such objects can be exported to other software too. As examples, some files have been generated in GeoDa³ from the Syracuse census tracts written out as a shapefile, with the centroid used here stored in the data frame. The first two are for contiguity neighbours, using the queen and rook criteria, respectively. These so-called GAL-format files contain only neighbour information, and are described in detail in the help file accompanying the function `read.gal`.

```
> Sy14_nb <- read.gal("Sy_GeoDa1.GAL")
> isTRUE(all.equal(Sy0_nb, Sy14_nb, check.attributes = FALSE))

[1] TRUE
```

The `write.nb.gal` function is used to write GAL-format files from `nb` objects. GeoDa also makes GWT-format files, described in the GeoDa documentation and the help file, which also contain distance information for the link between the areas, and are stored in a three-column sparse representation. They can be read using `read.gwt2nb`, here for a four-nearest-neighbour scheme, and only using the neighbour links.

```
> Sy16_nb <- read.gwt2nb("Sy_GeoDa4.GWT")
> isTRUE(all.equal(Sy10_nb, Sy16_nb, check.attributes = FALSE))

[1] TRUE
```

A similar set of functions is available for exchanging spatial weights with the Spatial Econometrics Library.⁴ The sparse representation of weights is similar to the GWT-format and can be imported using `read.dat2listw`.

³ <http://geodacenter.asu.edu/>, Anselin et al. (2006).

⁴ <http://www.spatial-econometrics.com>

Export to three different formats goes through the `listw2sn` function, which converts a spatial weights object to a three-column sparse representation. The output data frame can be written as a GWT-format file with `write.sn2gwt` or as a text representation of a sparse matrix for Matlab™ with `write.sn2dat`. Files written using `write.sn2gwt` may be read into Stata™ with `spmat import W using`. There is a function called `listw2WB` for creating a list of spatial weights for WinBUGS, to be written to file using `dput`. For neighbour objects, `nb2WB` may be used, setting all weights to unity. In a similar way, a neighbour object may be exported to file with `nb2INLA`, in order to pass data to the `graph` argument for the "bym" model in fitting using `inla`. It is also possible to use the `nb2gra` in the **BayesX** package to convert `nb` objects to the `gra` graph format; the model estimation methods using these objects are discussed in Chap. 10.

The `mat2listw` can be used to reverse the process, when a weights matrix has been read into R, and needs to be made into a neighbour and weights list object. Unfortunately, this function does not set the `style` of the `listw` object to a known value, using `M` to signal this lack of knowledge. It is then usual to rebuild the `listw` object, treating the `neighbours` component as an `nb` object, the `weights` component as a list of general weights and setting the style in the `nb2listw` function directly. It was used for the initial import of the eight-county contiguities, as shown in detail on the `NY_data` help page provided with `spdep`.

Finally, there is a function `nb2lines` to convert neighbour lists into `SpatialLinesDataFrame` objects, given point coordinates representing the areas. This allows neighbour objects to be plotted in an alternative way, and if need be, to be exported as shapefiles.

9.2.4 Using Weights to Simulate Spatial Autocorrelation

In Fig. 9.3, use was made of `listw2mat` to turn a spatial weights object into a dense matrix for display. The same function is used for constructing a dense representation of the $(\mathbf{I} - \rho\mathbf{W})$ matrix to simulate spatial autocorrelation within the `invIrW` function, where $\mathbf{W}$ is a weights matrix, ρ is a spatial autocorrelation coefficient, and $\mathbf{I}$ is the identity matrix. This approach was introduced by Cliff and Ord (1973, pp. 146–147), and does not impose strict conditions on the matrix to be inverted (only that it be non-singular), and only applies to simulations from a simultaneous autoregressive process. The underlying framework for the covariance representation used here – simultaneous autoregression – will be presented in Sect. 9.4.1.1.

Starting with a vector of random numbers corresponding to the number of census tracts in Syracuse, we use the row-standardised contiguity weights to introduce autocorrelation.

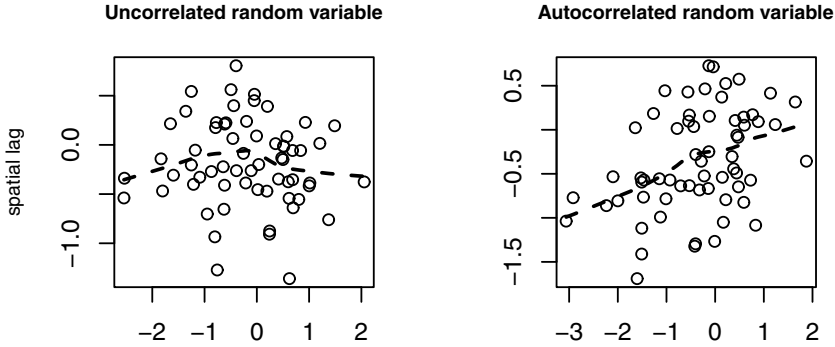


Fig. 9.4 Simulating spatial autocorrelation: spatial lag plots, showing a locally weighted smoother line

```
> set.seed(987654)
> n <- length(Sy0_nb)
> uncorr_x <- rnorm(n)
> rho <- 0.5
> autocorr_x <- invIrW(Sy0_lw_W, rho) %*% uncorr_x
```

The outcome is shown in Fig. 9.4, where the spatial lag plot of the original, uncorrelated variable contrasts with that of the autocorrelated variable, which now has a strong positive relationship between tract values and the spatial lag – here the average of values of neighbouring tracts.

The `lag` method for `listw` objects creates ‘spatial lag’ values: $\text{lag}(y_i) = \sum_{j \in N_i} w_{ij} y_j$ for observed values y_i ; N_i is the set of neighbours of i . If the weights object style is row-standardisation, the $\text{lag}(y_i)$ values will be averages over the sets of neighbours for each i , rather like a moving window defined by N_i and including values weighted by w_{ij} .

Analysing areal data is crucially dependent on the construction of the spatial weights, which is why it has taken some time to describe the breadth of choices facing the researcher. We can now go on to test for spatial autocorrelation, and to model using assumptions about underlying spatial processes.

9.3 Testing for Spatial Autocorrelation

Now that we have a range of ways of constructing spatial weights, we can start using them to test for the presence of spatial autocorrelation. Before doing anything serious, it would be very helpful to review the assumptions being made in the tests; we will be using Moran’s I as an example, but the consequences apply to other tests too. As Schabenberger and Gotway (2005, pp. 19–23) explain clearly, tests assume that the mean model of the data removes systematic spatial patterning from the data. If we are examin-

ing ecological data, but neglect environmental drivers such as temperature, precipitation, or elevation, we should not be surprised if the data seem to display spatial autocorrelation (for a discussion, see Bivand, 2008, pp. 9–15). Such misspecification of the mean model is not at all uncommon, and may be unavoidable where observations on variables needed to specify it correctly are not available. In fact, Cressie (1993, p. 442) only discusses the testing of residual autocorrelation, and then very briefly, preferring to approach autocorrelation through modelling.

Another issue that can arise is that the spatial weights we use for testing are not those that generated the autocorrelation – our chosen weights may, for example not suit the actual scales of interaction between areal entities. This is a reflection of misspecification of the model of the variance of the residuals from the mean model, which can also include making distributional assumptions that are not appropriate for the data, for example assuming homoskedasticity or regular shape parameters (for example, skewness and kurtosis). Some of these can be addressed by transforming the data and by using weighted estimation, but in any case, care is needed in interpreting apparent spatial autocorrelation that may actually stem from misspecification.

The use of global tests for spatial autocorrelation is covered in much more detail than the construction of spatial weights in the spatial data analysis texts that we are tracking. Waller and Gotway (2004, pp. 223–236) follow up the problem of mistaking the misspecification of the mean model for spatial autocorrelation. This is less evident in Fortin and Dale (2005, pp. 122–132), Ward and Gleditsch (2008, pp. 23–24), Chun and Griffith (2013, pp. 9–14), and O’Sullivan and Unwin (2010, pp. 199–211). Banerjee et al. (2004, pp. 71–73) are, like Cressie (1993), more concerned with modelling than testing.

We begin with the simulated variable for the Syracuse census tracts (see Sect. 9.2.4). Since the input variable is known to be drawn at random from the Normal distribution, we can manipulate it to see what happens to test results under different conditions. The test to be used in this introductory discussion is Moran’s I , which is calculated as a ratio of the product of the variable of interest and its spatial lag, with the cross-product of the variable of interest, and adjusted for the spatial weights used:

$$I = \frac{n}{\sum_{i=1}^n \sum_{j=1}^n w_{ij}} \frac{\sum_{i=1}^n \sum_{j=1}^n w_{ij} (y_i - \bar{y})(y_j - \bar{y})}{\sum_{i=1}^n (y_i - \bar{y})^2},$$

where y_i is the i th observation, $\bar{y}$ is the mean of the variable of interest, and w_{ij} is the spatial weight of the link between i and j . Centring on the mean is equivalent to asserting that the correct model has a constant mean, and that any remaining patterning after centring is caused by the spatial relationships encoded in the spatial weights.

The results for Moran’s I are collated in Table 9.1 for five settings. The first column contains the observed value of I , the second is the expectation,

	<i>I</i>	<i>E(I)</i>	<i>var(I)</i>	St. deviate	<i>p</i> -value
uncorr_x	-0.03329	-0.01613	0.00571	-0.227	0.59
autocorr_x	0.2182	-0.0161	0.0057	3.1	0.00096
autocorr_x k=1	0.1921	-0.0161	0.0125	1.86	0.031
trend_x	0.23747	-0.01613	0.00575	3.34	0.00041
lm(trend_x ~ et)	-0.0538	-0.0309	0.0054	-0.312	0.62

Table 9.1 Moran’s *I* test results for five different data generating processes

which is $-1/(n-1)$ for the mean-centred cases, the third the variance of the statistic under randomisation, next the standard deviate $(I - E(I))/\sqrt{\text{var}(I)}$, and finally the *p*-value of the test for the alternative that $I > E(I)$. The test results are for the uncorrelated case first (`uncorr_x`) – there is no trace of spatial dependence with these weights. Even though a random drawing could show spatial autocorrelation, we would be unfortunate to find a pattern corresponding to our spatial weights by chance for just one draw. When the spatially autocorrelated variable is tested (`autocorr_x`), it shows, as one would expect, a significant result for these spatial weights. If we use spatial weights that differ from those used to generate the spatial autocorrelation (`autocorr_x k=1`), the value of *I* falls, and although it is marginally significant, it is worth remembering that, had the generating process been less strong, we might have come to the wrong conclusion based on the choice of spatial weights not matching the actual generating process.

```
> moran_u <- moran.test(uncorr_x, listw = Sy0_lw_W)
> moran_a <- moran.test(autocorr_x, listw = Sy0_lw_W)
> moran_a1 <- moran.test(autocorr_x, listw = nb2listw(Sy9_nb,
+ style = "W"))
```

The final two rows of Table 9.1 show what can happen when our assumption of a constant mean is erroneous (Schabenberger and Gotway, 2005, pp. 22–23). Introducing a gentle trend rising from west to east into the uncorrelated random variable, we have a situation in which there is no underlying spatial autocorrelation, just a simple linear trend. If we assume a constant mean, we reach the wrong conclusion shown in the fourth row of the table (`trend_x`). The final row shows how we get back to the uncorrelated residuals by including the trend in the mean, and again have uncorrelated residuals (`lm(trend_x ~ et)`).

```
> et <- coords[, 1] - min(coords[, 1])
> trend_x <- uncorr_x + 0.00025 * et
> moran_t <- moran.test(trend_x, listw = Sy0_lw_W)
> moran_t1 <- lm.morantest(lm(trend_x ~ et), listw = Sy0_lw_W)
```

This shows how important it can be to understand that tests for spatial autocorrelation can also react to a misspecified model of the mean, and that the omission of a spatially patterned variable from the mean function will ‘look like’ spatial autocorrelation to the tests.

9.3.1 Global Tests

Moran's I – `moran.test` – is perhaps the most common global test, and for this reason we continue to use it here. Other global tests implemented in the **spdep** package include Geary's C (`geary.test()`), the global Getis-Ord G (`globalG.test()`), and the spatial general cross product Mantel test, which includes Moran's I , Geary's C , and the Sokal variant of Geary's C as alternative forms (`sp.mantel.mc()`). All these are for continuous variables, with `moran.test()` having an argument to use an adjustment for a ranked continuous variable, that is where the metric of the variable is by the ranks of its values rather than the values themselves. There are also join count tests for categorical variables, with the variable of interest represented as a factor (`joincount.test()` for same-colour joins, `joincount.multi()` for same-colour and different colour joins).

The values of these statistics may be of some interest in themselves, but are not directly interpretable. The approach taken most generally is to standardise the observed value by subtracting the analytical expected value, and dividing the difference by the square root of the analytical variance for the spatial weights used, for a set of assumptions. The result is a standard deviate, and is compared with the Normal distribution to find the probability value of the observed statistic under the null hypothesis of no spatial dependence for the chosen spatial weights – most often the test is one-sided, with an alternative hypothesis of the observed statistic being significantly greater than its expected value.

As we see, outcomes can depend on the choices made, for example the style of the weights and to what extent the assumptions made are satisfied. It might seem that Monte Carlo or equivalently bootstrap permutation-based tests, in which the values of the variable of interest are randomly assigned to spatial entities, would provide protection against errors of inference. In fact, because tests for spatial autocorrelation are sensitive to spatial patterning in the variable of interest from any source, they are not necessarily – as we saw above – good guides to decide what is going on in the data generation process. Parametric bootstrapping or tests specifically tuned to the setting – or better specification of the variable of interest – are sometimes needed.

A further problem for which there is no current best advice is how to proceed if some areal entities have no neighbours. By default, test functions in **spdep** do not accept spatial weights with no-neighbour entities unless the `zero.policy` argument is set to `TRUE`. But even if the analyst accepts the presence of rows and columns with only zero entries in the spatial weights matrix, the correct size of n can be taken as the number of observations, or may be reduced to reflect the fact that some of the observations are effectively being ignored. By default, n is adjusted, but the `adjust.n` argument may be set to `FALSE`. If n is not adjusted, for example for Moran's I , the absolute value of the statistic will increase, and the absolute value of its ex-

pectation and variance will decrease. When measures of autocorrelation were developed, it was generally assumed that all entities would have neighbours, so what one should do when some do not, is not obvious. The problem is not dissimilar to the choice of variogram bin widths and weights in geostatistics (Sect. 8.4.3).

We have already used the New York state eight-county census tract data set for examining the construction of neighbour lists and spatial weights. Now we introduce the data themselves, based on Waller and Gotway (2004, pp. 98, 345–353). There are 281 census tract observations, including as we have seen sparsely populated rural areas contrasting with dense, small, urban tracts. The numbers of incident leukaemia cases are recorded by tract, aggregated from census block groups, but because some cases could not be placed, they were added proportionally to other block groups, leading to non-integer counts. The counts are for the 5 years 1978–1982, while census variables, such as the tract population, are just for 1980. Other census variables are the percentage aged over 65, and the percentage of the population owning their own home. Exposure to TCE waste sites is represented as the logarithm of 100 times the inverse of the distance from the tract centroid to the nearest site. We return to these covariates in following sections.

The first example is of testing the number of cases by census tract (following Waller and Gotway, 2004, p. 231) for autocorrelation using the default spatial weights style of row standardisation, and using the analytical randomisation assumption in computing the variance of the statistic. The outcome, as we see, is that the spatial patterning of the variable of interest is significant, with neighbouring tracts very likely to have similar values for whatever reason.

```
> moran.test(NY8$Cases, listw = nb2listw(NY_nb))

Moran's I test under randomisation

data: NY8$Cases
weights: nb2listw(NY_nb)

Moran I statistic standard deviate = 3.978, p-value = 3.477e-05
alternative hypothesis: greater
sample estimates:
Moran I statistic      Expectation      Variance
      0.146883         -0.003571         0.001431
```

Changing the style of the spatial weights to make all weights equal and summing to the number of observations, we see that the resulting probability value is reduced about 20 times – we recall that row-standardisation favours observations with few neighbours, and that styles ‘B’, ‘C’, and ‘U’ ‘weight up’ observations with many neighbours. In this case, style ‘S’ comes down between ‘C’ and ‘W’.

```
> lw_B <- nb2listw(NY_nb, style = "B")
> moran.test(NY8$Cases, listw = lw_B)

Moran's I test under randomisation
```

```
data: NY8$Cases
weights: lw_B
```

```
Moran I statistic standard deviate = 3.186, p-value = 0.0007207
alternative hypothesis: greater
sample estimates:
Moran I statistic      Expectation      Variance
      0.110387         -0.003571         0.001279
```

By default, `moran.test` uses the randomisation assumption, which differs from the simpler normality assumption by introducing a correction term based on the kurtosis of the variable of interest (here 3.63). When the kurtosis value corresponds to that of a normally distributed variable, the two assumptions yield the same variance, but as the variable departs from normality, the randomisation assumption compensates by increasing the variance and decreasing the standard deviate. In this case, there is little difference and the two return similar outcomes.

```
> moran.test(NY8$Cases, listw = lw_B, randomisation = FALSE)

Moran's I test under normality
```

```
data: NY8$Cases
weights: lw_B
```

```
Moran I statistic standard deviate = 3.183, p-value = 0.0007301
alternative hypothesis: greater
sample estimates:
Moran I statistic      Expectation      Variance
      0.110387         -0.003571         0.001282
```

It is useful to show here that the standard test under normality is in fact the same test as the Moran test for regression residuals for the model, including only the intercept. Making this connection here shows that we could introduce additional variables on the right-hand side of our model, over and above the intercept, and potentially other ways of handling misspecification.

```
> lm.morantest(lm(Cases ~ 1, NY8), listw = lw_B)

Global Moran's I for regression residuals
```

```
data:
model: lm(formula = Cases ~ 1, data = NY8)
weights: lw_B
```

```
Moran I statistic standard deviate = 3.183, p-value = 0.0007301
alternative hypothesis: greater
sample estimates:
Observed Moran's I      Expectation      Variance
      0.110387         -0.003571         0.001282
```

Using the same construction, we can also use a Saddlepoint approximation rather than the analytical normal assumption (Tiefelsdorf, 2002), and an exact test (Tiefelsdorf, 1998, 2000; Hepple, 1998; Bivand et al., 2009). These methods are substantially more demanding computationally, and were originally regarded as impractical. For moderately sized data sets such as the one we are using, however, need less than double the time required for reaching a result. In general, exact and Saddlepoint methods make little difference to outcomes for global tests when the number of spatial entities is not small, as here, with the probability value only changing by a factor of 2. We see later that the impact of differences between the normality assumption and the Saddlepoint approximation and exact test is stronger for local indicators of spatial association.

```
> lm.morantest.sad(lm(Cases ~ 1, NY8), listw = lw_B)

      Saddlepoint approximation for global Moran's I
      (Barndorff-Nielsen formula)

data:
model:lm(formula = Cases ~ 1, data = NY8)
weights: lw_B

Saddlepoint approximation = 2.993, p-value = 0.001382
alternative hypothesis: greater
sample estimates:
Observed Moran's I
      0.1104

> lm.morantest.exact(lm(Cases ~ 1, NY8), listw = lw_B)

      Global Moran's I statistic with exact p-value

data:
model:lm(formula = Cases ~ 1, data = NY8)
weights: lw_B

Exact standard deviate = 2.992, p-value = 0.001384
alternative hypothesis: greater
sample estimates:
[1] 0.1104
```

We can also use a Monte Carlo test, a permutation bootstrap test, in which the observed values are randomly assigned to tracts, and the statistic of interest computed `nsim` times. Since, in the global case in contrast to the local, we have enough observations and can repeat this permutation potentially very many times without repetition.

```
> set.seed(1234)
> bperm <- moran.mc(NY8$Cases, listw = lw_B, nsim = 999)
> bperm

      Monte-Carlo simulation of Moran's I
```



```
data: NY8$Cases
weights: lw_B
number of simulations + 1: 1000

statistic = 0.1104, observed rank = 998, p-value = 0.002
alternative hypothesis: greater
```

Waller and Gotway (2004, p.231) also include a Poisson constant risk parametric bootstrap assessment of the significance of autocorrelation in the case counts. The constant global rate r is calculated first, and used to create expected counts for each census tract by multiplying by the population.

```
> r <- sum(NY8$Cases)/sum(NY8$POP8)
> rni <- r * NY8$POP8
> CR <- function(var, mle) rpois(length(var), lambda = mle)
> MoranI.pboot <- function(var, i, listw, n, SO, ...) {
+   return(moran(x = var, listw = listw, n = n, SO = SO)$I)
+ }
> set.seed(1234)

> boot2 <- boot(NY8$Cases, statistic = MoranI.pboot,
+   R = 999, sim = "parametric", ran.gen = CR,
+   listw = lw_B, n = length(NY8$Cases), SO = Szero(lw_B),
+   mle = rni)

> pnorm((boot2$t0 - mean(boot2$t))/sd(boot2$t[, 1]), lower.tail = FALSE)

[1] 0.1472
```

The expected counts can also be expressed as the fitted values of a null Poisson regression with an offset set to the logarithm of tract population – with a log-link, this shows the relationship to generalised linear models (because `Cases` are not all integer, warnings are generated):

```
> rni <- fitted(glm(Cases ~ 1 + offset(log(POP8)), data = NY8,
+   family = "poisson"))
```

These expected counts `rni` are fed through to the `lambda` argument to `rpois` to generate the synthetic data sets by sampling from the Poisson distribution. The output probability value is calculated from the same observed Moran's I minus the mean of the simulated I values, and divided by their standard deviation. Figure 9.5 corresponds to Waller and Gotway (2004, p. 232, Fig. 7.8), with the parametric simulations shifting the distribution of Moran's I rightwards, because it is taking the impact of the heterogeneous tract populations into account.

There is a version of Moran's I adapted to use an Empirical Bayes rate by Assunção and Reis (1999) that, unlike the rate results above, shrinks extreme rates for tracts with small populations at risk towards the rate for the area as a whole – it also uses Monte Carlo methods for inference:

```
> set.seed(1234)
> EBImoran.mc(n = NY8$Cases, x = NY8$POP8, listw = nb2listw(NY_nb,
+   style = "B"), nsim = 999)
```

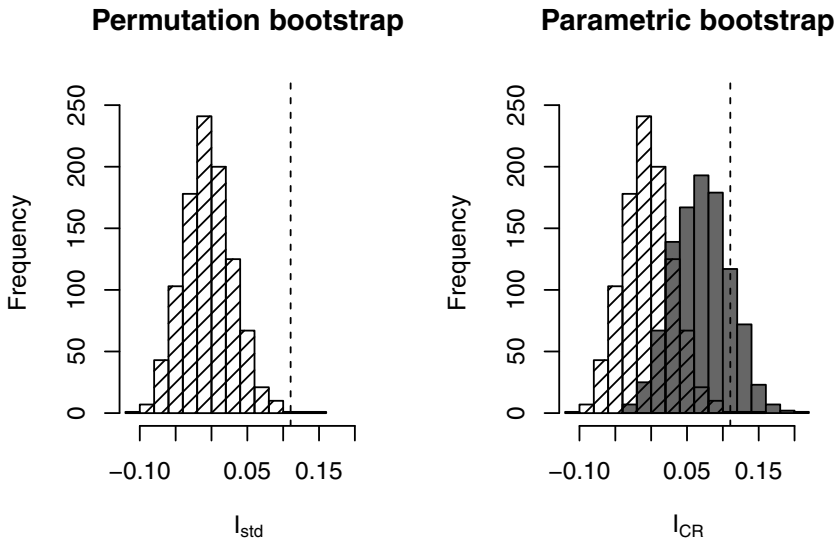


Fig. 9.5 Histograms of simulated values of Moran's I under random permutations of the data and parametric samples from constant risk expected values; the observed value of Moran's I is marked by a vertical line

Monte-Carlo simulation of Empirical Bayes Index

```
data: cases: NY8$Cases, risk population: NY8$POP8
weights: nb2listw(NY_nb, style = "B")
number of simulations + 1: 1000
```

```
statistic = 0.0735, observed rank = 980, p-value = 0.02
alternative hypothesis: greater
```

The results for the Empirical Bayes rates suggest that one reason for the lack of significance of the parametric bootstrapping of the constant risk observed and expected values could be that unusual and extreme values were observed in tracts with small populations. Once the rates have been smoothed, some global autocorrelation is found.

Another approach is to plot and tabulate values of a measure of spatial autocorrelation for higher orders of neighbours or bands of more distant neighbours where the spatial entities are points, as discussed by Borcard et al. (2011, pp. 232–236). The **spdep** package provides the first type as a wrapper to **nblag** and **moran.test**, so that here the first-order contiguous neighbours we have used until now are ‘stepped out’ to the required number of orders. Figure 9.6 shows the output plot in the left panel, and suggests that second-order neighbours are also positively autocorrelated (although the probability values should be adjusted for multiple comparisons).

```
> cor8 <- sp.correlogram(neighbours = NY_nb, var = NY8$Cases,
+   order = 8, method = "I", style = "C")
```

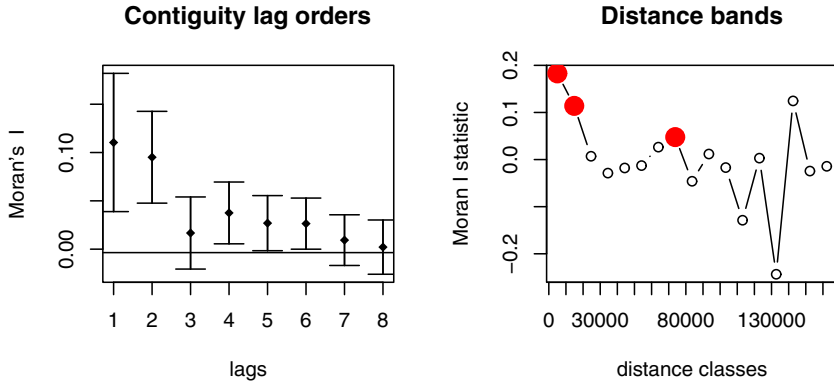


Fig. 9.6 Correlograms: (*left*) values of Moran's I for eight successive lag orders of contiguous neighbours; (*right*) values of Moran's I for a sequence of distance band neighbour pairs

The right panel in Fig. 9.6 presents the output of the `correlog` function in the `pgirmess` package by Patrick Giraudoux; the function is a wrapper for `dneareigh` and `moran.test`. The function automatically selects distance bands of almost 10 km, spanning the whole study area. In this case, the first two bands of 0–10 and 10–20 km have significant values.

```
> library(pgirmess)
> corD <- correlog(coordinates(NY8), NY8$Cases, method = "Moran")
```

9.3.2 Local Tests

Global tests for spatial autocorrelation are calculated from the local relationships between the values observed at a spatial entity and its neighbours, for the neighbour definition chosen. Because of this, we can break global measures down into their components, and by extension, construct localised tests intended to detect 'clusters' – observations with very similar neighbours – and 'hotspots' – observations with very different neighbours. These are discussed briefly by Schabenberger and Gotway (2005, pp. 23–25), and at greater length by O'Sullivan and Unwin (2010, pp. 216–226), Waller and Gotway (2004, pp. 236–242), Chun and Griffith (2013, pp. 94–98) and Fortin and Dale (2005, pp. 153–159). They are covered in some detail by Lloyd (2007, pp. 65–70) in a book concentrating on local models; further examples are given by Bivand (2010, pp. 236–244).

First, let us examine a Moran scatterplot of the leukaemia case count variable; for a discussion of Moran scatterplots, see Ward and Gleditsch (2008, pp. 24–27). The plot (shown in Fig. 9.7) by convention places the variable of

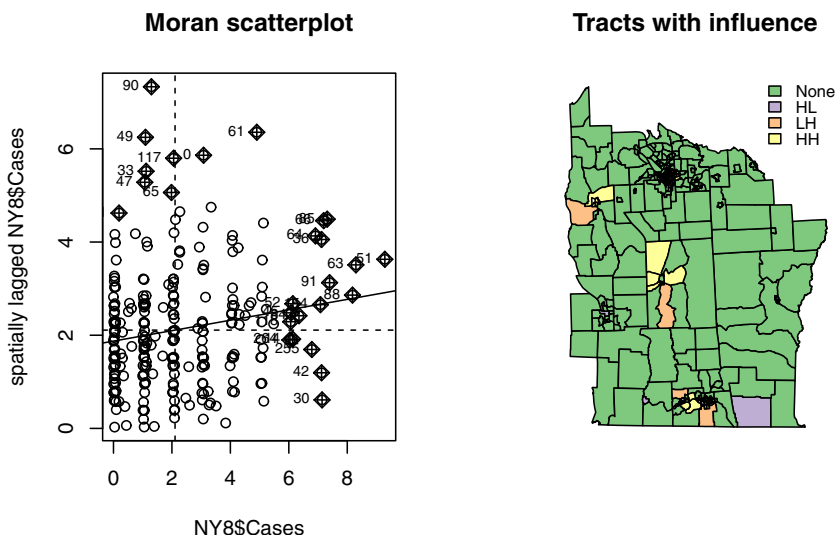


Fig. 9.7 (Left) Moran scatterplot of leukaemia incidence; (right) tracts with influence by Moran scatterplot quadrant

interest on the x -axis, and the spatially weighted sum of values of neighbours – the spatially lagged values – on the y -axis. Global Moran's I is a linear relationship between these and is drawn as a slope. The plot is further partitioned into quadrants at the mean values of the variable and its lagged values: low–low, low–high, high–low, and high–high.

```
> moran.plot(NY8$Cases, listw = nb2listw(NY_nb, style = "C"))
```

Since global Moran's I is, like similar correlation coefficients, a linear relationship, we can also apply standard techniques for detecting observations with unusually strong influence on the slope. Specifically, `moran.plot` calls `influence.measures` on the linear model of `lm(wx ~ x)` providing the slope coefficient, where `wx` is the spatially lagged value of `x`. This means that we can see whether particular local relationships are able to influence the slope more than proportionally. The map in the right panel of Fig. 9.7 shows tracts with significant influence (using standard criteria) coded by their quadrant in the Moran scatterplot.

Local Moran's I_i values are constructed as the n components summed to reach global Moran's I :

$$I_i = \frac{(y_i - \bar{y}) \sum_{j=1}^n w_{ij} (y_j - \bar{y})}{\frac{\sum_{i=1}^n (y_i - \bar{y})^2}{n}},$$

where once again we assume that the global mean $\bar{y}$ is an adequate representation of the variable of interest y . The two components in the numerator, $(y_i - \bar{y})$ and $\sum_{j=1}^n w_{ij}(y_j - \bar{y})$, appear without centring in the Moran scatterplot.

As with the global statistic, the local statistics can be tested for divergence from expected values, under assumptions of normality, and randomisation analytically, and using Saddlepoint approximations and exact methods. The two latter methods can be of importance because the number of neighbours of each observation is very small, and this in turn may make the adoption of the normality assumption problematic. Using numerical methods, which would previously have been considered demanding, the Saddlepoint approximation or exact local probability values can be found in well under 10 s, about 20 times slower than probability values based on normality or randomisation assumptions, for this moderately sized data set.

Trying to detect residual local patterning in the presence of global spatial autocorrelation is difficult. For this reason, results for local dependence are not to be seen as ‘absolute’, but are conditioned at least by global spatial autocorrelation, and more generally by the possible influence of spatial data generating processes at a range of scales from global through local to dependence not detected at the scale of the observations.

```
> lm1 <- localmoran(NY8$Cases, listw = nb2listw(NY_nb,
+ style = "C"))
> lm2 <- as.data.frame(localmoran.sad(lm(Cases ~ 1, NY8),
+ nb = NY_nb, style = "C"))
> lm3 <- as.data.frame(localmoran.exact(lm(Cases ~ 1, NY8),
+ nb = NY_nb, style = "C"))
```

Waller and Gotway (2004, p.239) extend their constant risk hypothesis treatment to local Moran’s I_i , and we can follow their lead:

```
> r <- sum(NY8$Cases)/sum(NY8$POP8)
> rni <- r * NY8$POP8
> lw <- nb2listw(NY_nb, style = "C")
> sdCR <- (NY8$Cases - rni)/sqrt(rni)
> wsdCR <- lag(lw, sdCR)
> I_CR <- sdCR * wsdCR
```

Figure 9.8 shows the two sets of values of local Moran’s I_i , calculated in the standard way and using the Poisson assumption for the constant risk hypothesis. We already know that global Moran’s I can vary in value and in inference depending on our assumptions – for example that inference should take deviations from our distributional assumptions into account. The same applies here to the assumption for the Poisson distribution that its mean and standard deviation are equal, whereas over-dispersion seems to be a problem in data also displaying autocorrelation. There are some sign changes between the maps, with the constant risk hypothesis values somewhat farther from zero.

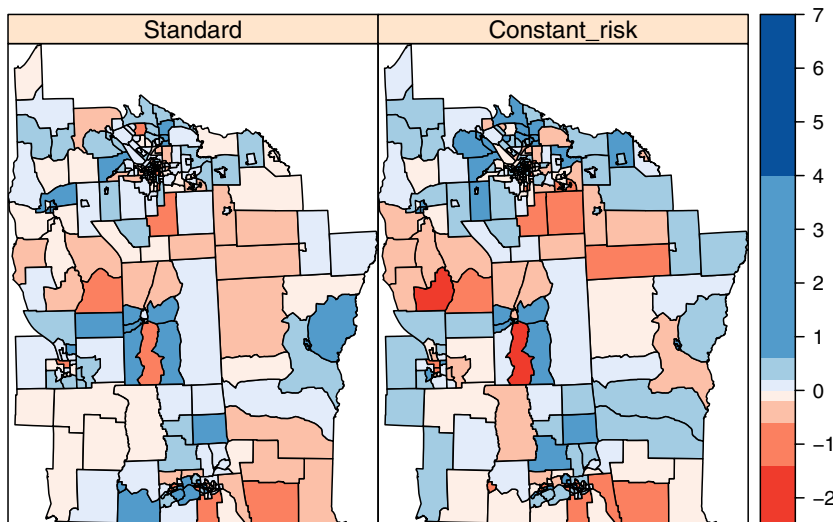


Fig. 9.8 Local Moran's I_i values calculated directly and using the constant risk hypothesis

We can also construct a simple Monte Carlo test of the constant risk hypothesis local Moran's I_i values, simulating very much as in the global case, but now retaining all of the local results. Once the simulation is completed, we extract the rank of the observed constant risk local Moran's I_i value for each tract, and calculate its probability value for the number of simulations made. We use a parametric approach to simulating the local counts using the local expected count as the parameter to `rpois`, because the neighbour counts are very low and make permutation unwise. Carrying out permutation testing using the whole data set also seems unwise, because we would then be comparing like with unlike (Fig. 9.9).

```
> set.seed(1234)
> nsim <- 999
> N <- length(rni)
> sims <- matrix(0, ncol = nsim, nrow = N)
> for (i in 1:nsim) {
+   y <- rpois(N, lambda = rni)
+   sdCRi <- (y - rni)/sqrt(rni)
+   wsdCRi <- lag(lw, sdCRi)
+   sims[, i] <- sdCRi * wsdCRi
+ }
> xrank <- apply(cbind(I_CR, sims), 1, function(x) rank(x)[1])
> diff <- nsim - xrank
> diff <- ifelse(diff > 0, diff, 0)
> pval <- punif((diff + 1)/(nsim + 1))
```

Finally, we zoom in to examine the local Moran's I_i probability values for three calculation methods for the tracts in and near the city of Binghampton

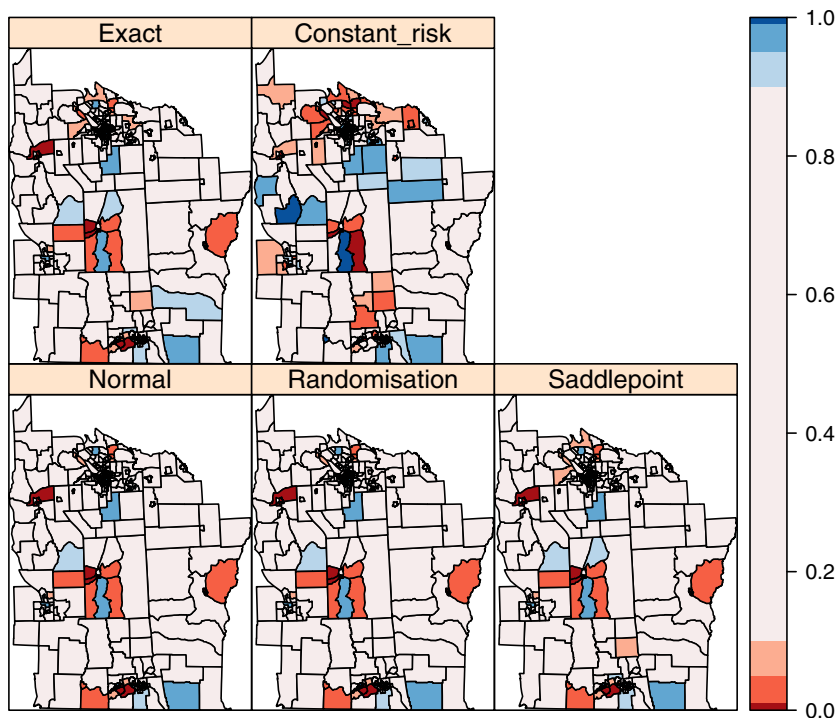


Fig. 9.9 Probability values for all census tracts, local Moran's I_i : normality and randomisation assumptions, Saddlepoint approximation, exact values, and constant risk hypothesis

(Fig. 9.10). It appears that the use of the constant risk approach handles the heterogeneity in the counts better than the alternatives. These results broadly agree with those reached by Waller and Gotway (2004, p. 241), but we note that our underlying model is very simplistic. Finding spatial autocorrelation is not a goal in itself, be it local or global, but rather just one step in a process leading to a proper model. It is to this task that we now turn.

9.4 Fitting Models of Areal Data

We have seen above that the lack of independence between observations in spatial data – spatial autocorrelation – is commonplace, and that tests are available. In an ideal world, one would prefer to gather data in which the observations were mutually independent, and so avoid problems in inference from analytical results. Most applied data analysts, however, do not have

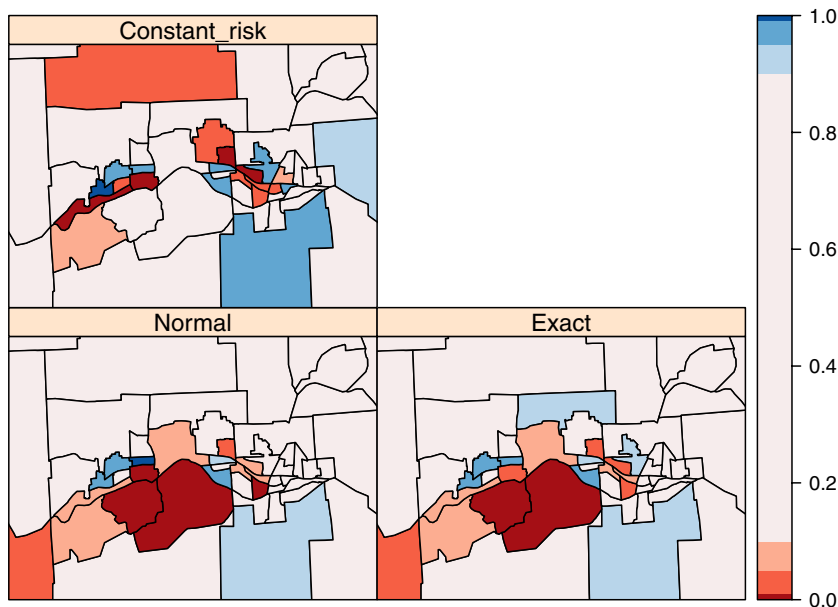


Fig. 9.10 Probability values for census tracts in and near the city of Binghamton, local Moran's I_i : normality assumption, exact values, and constant risk hypothesis

this option, and must work with the data that are available, or that can be collected with available technologies. It is quite often the case that observations on relevant covariates are not available at all, and that the detection of spatial autocorrelation in data or model residuals in fact constitutes the only way left to model the remaining variation.

In this section, we show how spatial structure in dependence between observations may be modelled, in particular for areal data, but where necessary also using alternative representations. We look at spatial econometrics approaches separately, because the terminology used in that domain differs somewhat from other areas of spatial statistics. In conclusion, we mention some alternative methods, but leave Bayesian hierarchical models until Chap. 10.

The problems we face when trying to fit models in the presence of spatial autocorrelation are challenging, not least because the spatial autocorrelation that we seem to have found may actually come from model misspecification (see Sect. 9.3). If this is the case, effort spent on modelling the spatial structure would be better used on improving the model itself, perhaps by handling heteroskedasticity, by adding a missing covariate, by revisiting the functional form of included covariates, or by reconsidering the distributional representation of the response variable.

9.4.1 Spatial Statistics Approaches

Spatial dependence can be modelled in different ways using statistical models. In many cases, it is common to assume that observations are independent and identically distributed, but this may not be the case when working with spatial data. Observations are not independent because there may exist some correlation between neighbouring areas. It may also be difficult to pick apart the impact of spatial autocorrelation and spatial differences in the distribution of the observation. Cressie (1993, pp. 402–448, 458–477, 548–568) provides a very wide discussion of these approaches, including reviews of the background for their development and comprehensive worked examples. Schabenberger and Gotway (2005, pp. 335–348) and Waller and Gotway (2004, pp. 362–380) concentrate on the spatial autoregressive models to be used in this section. Wall (2004) provides a useful comparative review of the ways in which spatial processes for areal data are modelled. Banerjee et al. (2004, pp. 79–87) also focus on these models, because the key features carry through to hierarchical models. Fortin and Dale (2005, pp. 229–233) indicate that spatial autoregressive models may play a different role in ecology, although reviews like Dormann et al. (2007) suggest that they may be of use.

In this section, we have followed Waller and Gotway (2004, Chap. 9) quite closely, as their examples highlight issues such as transforming the response variable and using weights to try to handle heteroskedasticity.

From a statistical point of view, it is possible to account for correlated observations by considering a structure of the following kind in the model. If the vector of response variables is multivariate normal, we can express the model as follows:

$$Y = \mu + e,$$

where μ is the vector of area means, which can be modelled in different ways and e is the vector of random errors, which we assume is normally distributed with zero mean and generic variance V . The mean is often supposed to depend on a linear term on some covariates X , so that we will substitute the mean by $X^T\beta$ in the model. On the other hand, correlation between areas is taken into account by considering a specific form of the variance matrix V .

For the case of non-Normal variables, we could transform the original data to achieve the desired Normality. Hence, the techniques described below can still be applied on the transformed data. In principle, many correlation structures could be feasible in order to account for spatial correlation. However, we focus on two approaches that are commonly used in practise, such as SAR (Simultaneous Autoregressive)⁵ and CAR (Conditionally Autoregressive) models.

⁵ In spatial econometrics, the abbreviation SAR means Spatial Autoregressive, and refers to the spatial lag model defined later in this chapter on p. 305.

Earlier in this chapter, we took the mean of the counts of leukaemia cases by tract as our best understanding of the data generation process, supplementing this with the constant risk approach to try to handle heterogeneity coming from variations in tract populations. One of the alternatives examined by Waller and Gotway (2004, p. 348) is to take a log transformation of the rate:

$$Z_i = \log \frac{1000(Y_i + 1)}{n_i}.$$

The transformed incidence proportions are not yet normal, with three outliers, tracts with small populations but unexpectedly large case counts. They could be smoothed away, but may in fact be interesting, as the patterns they display may be related to substantive covariates, such as closeness to TCE locations. As covariates, we have used the inverse distance to the closest TCE (PEXPOSURE), the proportion of people aged 65 or higher (PCTAGE65P) and the proportion of people who own their own home (PCTOWNHOME).

To set the scene, let us start with a linear model of the relationship between the transformed incidence proportions and the covariates. Note that most model fitting functions accept `Spatial*DataFrame` objects as their `data` argument values, and simply treat them as regular `data.frame` objects. This is not by inheritance, but because the same access methods are provided (see p. 35).

```
> nylm <- lm(Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME, data = NY8)
> summary(nylm)
```

Call:

```
lm(formula = Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME, data = NY8)
```

Residuals:

Min	1Q	Median	3Q	Max
-1.742	-0.396	-0.033	0.335	4.140

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-0.5173	0.1586	-3.26	0.0012 **
PEXPOSURE	0.0488	0.0351	1.39	0.1648
PCTAGE65P	3.9509	0.6055	6.53	3.2e-10 ***
PCTOWNHOME	-0.5600	0.1703	-3.29	0.0011 **

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.657 on 277 degrees of freedom

Multiple R-squared: 0.193, Adjusted R-squared: 0.184

F-statistic: 22.1 on 3 and 277 DF, p-value: 7.31e-13

```
> NY8$lmresid <- residuals(nylm)
```

Figure 9.12 shows the spatial distribution of residual values for the study area census tracts. The two census variables appear to contribute for explaining the variance in the response variable, but exposure to TCE does

not. Moreover, although there is less spatial autocorrelation in the residuals from the model with covariates than in the null model, it is clear that there is information in the residuals that we should try to use. An exact test for spatial autocorrelation in the residuals leads to similar conclusions.

Since the Moran test is intended to detect spatial autocorrelation, we can try to fit a model taking this into account. We should not, however, forget that the misspecifications detected by Moran's I can have a range of causes (see Sect. 9.3). It is also the case that if the fitted model exhibits multi-collinearity, the results of the test may be affected because of the numerical consequences of the model matrix not being of full rank for the expectation and variance of the statistic.

```
> library(spdep)

> NYlistw <- nb2listw(NY_nb, style = "B")
> lm.morantest(nylm, NYlistw)

Global Moran's I for regression residuals

data:
model: lm(formula = Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME,
data = NY8)
weights: NYlistw

Moran I statistic standard deviate = 2.638, p-value = 0.004169
alternative hypothesis: greater
sample estimates:
Observed Moran's I      Expectation      Variance
      0.083090         -0.009891         0.001242
```

Before looking at model estimation in more detail, it is worth mentioning the approximate profile-likelihood estimator introduced by Li et al. (2007) and refined in further work (Li et al., 2012). Assuming that the variable of interest has been detrended, and that the spatial weights are row-standardised, the one-step estimator of the spatial regression parameter may yield more comparative information than Moran's I . For comparison, we show the maximum likelihood parameter estimate fitted using the same data:

```
> NYlistwW <- nb2listw(NY_nb, style = "W")
> aple(residuals(nylm), listw = NYlistwW)

[1] 0.2052

> spautolm(Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME, data = NY8,
+         listw = NYlistwW)$lambda

lambda
0.2169
```

9.4.1.1 Simultaneous Autoregressive Models

The SAR specification uses a regression on the values from the other areas to account for the spatial dependence. This means that the error terms e_i are modelled so that they depend on each other in the following way:

$$e_i = \sum_{j=1}^m b_{ij} e_j + \varepsilon_i.$$

Here, ε_i are used to represent residual errors, which are assumed to be independently distributed according to a Normal distribution with zero mean and diagonal covariance matrix Σ_ε with elements $\sigma_{\varepsilon_i}^2, i = 1, \dots, m$ (the same variance σ_ε^2 is often considered though). The b_{ij} values are used to represent spatial dependence between areas. b_{ii} must be set to zero so that each area is not regressed on itself.

Note that if we express the error terms as $e = B(Y - X^T\beta) + \varepsilon$, the model can also be expressed as

$$Y = X^T\beta + B(Y - X^T\beta) + \varepsilon.$$

Hence, this model can be formulated in a matrix form as follows:

$$(I - B)(Y - X^T\beta) = \varepsilon,$$

where B is a matrix that contains the dependence parameters b_{ij} and I is the identity matrix of the required dimension. It is important to point out that in order for this SAR model to be well defined, the matrix $I - B$ must be non-singular.

Under this model, Y is distributed according to a multivariate normal with mean

$$E[Y] = X^T\beta$$

and covariance matrix

$$\text{Var}[Y] = (I - B)^{-1} \Sigma_\varepsilon (I - B^T)^{-1}.$$

Often Σ_ε is taken to depend on a single parameter σ^2 , so that $\Sigma_\varepsilon = \sigma^2 I$ and then $\text{Var}[Y]$ simplifies to

$$\text{Var}[Y] = \sigma^2 (I - B)^{-1} (I - B^T)^{-1}.$$

It is also possible to specify Σ_ε as a diagonal matrix of weights associated with heterogeneity among the observations.

A useful re-parametrisation of this model can be obtained by writing $B = \lambda W$, where λ is a spatial autocorrelation parameter and W is a matrix

that represents spatial dependence – it is often assumed to be symmetric. These structures can be chosen among those described above. With this specification, the variance of Y becomes

$$\text{Var}[Y] = \sigma^2(I - \lambda W)^{-1}(I - \lambda W^T)^{-1}.$$

These models can be estimated efficiently by maximum likelihood. In R this can be done by using function `spautolm` in package `spdep`. The model can be specified using a formula for the linear predictor, whilst matrix W must be passed as a `listw` object. To create this object from the list of neighbours we can use function `nb2listw`, which will take an object of class `nb`, as explained above.

The following code shows how to fit a simultaneous autoregression to the chosen model. We have fitted the standard model and the weighted model using the population size in 1980 (according to the US Census) in the areas as weights. This reproduces the example developed in Waller and Gotway (2004, Chap. 9, pp. 375–379), and the reader is referred to their discussion for more information. In the call to `nb2listw`, we specified `style = "B"` to construct W using a binary indicator of neighbourhood.

```
> nysar <- spautolm(Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME,
+   data = NY8, listw = NYlistw)
> summary(nysar)
```

Call:

```
spautolm(formula = Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME, data = NY8,
  listw = NYlistw)
```

Residuals:

	Min	1Q	Median	3Q	Max
	-1.56754	-0.38239	-0.02643	0.33109	4.01219

Coefficients:

	Estimate	Std. Error	z value	Pr(> z)
(Intercept)	-0.618193	0.176784	-3.4969	0.0004707
PEXPOSURE	0.071014	0.042051	1.6888	0.0912635
PCTAGE65P	3.754200	0.624722	6.0094	1.862e-09
PCTOWNHOME	-0.419890	0.191329	-2.1946	0.0281930

Lambda: 0.04049 LR test value: 5.244 p-value: 0.022026

Numerical Hessian standard error of lambda: 0.01718

Log likelihood: -276.1

ML residual variance (sigma squared): 0.4139, (sigma: 0.6433)

Number of observations: 281

Number of parameters estimated: 6

AIC: 564.2

According to the results obtained it seems that there is significant spatial correlation in the residuals because the estimated value of λ is 0.0405 and the p -value of the likelihood ratio test is 0.022. In the likelihood ratio test

we compare the model with no spatial autocorrelation (i.e. $\lambda = 0$) to the one which allows for it (i.e. the fitted model with non-zero autocorrelation parameter).

The proximity to a TCE seems not to be significant, although its p -value is close to being significant at the 95 % level and it would be advisable not to discard a possible association and to conduct further research on this. The other two covariates are significant, suggesting that census tracts with larger percentages of older people and with lower percentages of house owners have higher transformed incidence rates.

Figure 9.11 shows the two components of the fitted values of the SAR model following Cressie (1993, p. 564) and Haining (2003, p. 333). Recalling our definition of the model above as:

$$Y = X^T\beta + \lambda W(Y - X^T\beta) + \varepsilon.$$

the first term $X^T\beta$ is the aspatial trend component, while $\lambda W(Y - X^T\beta)$ is the spatial stochastic component.

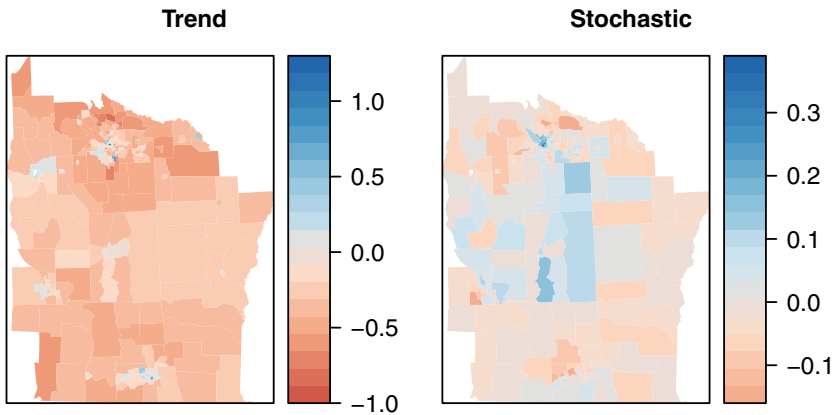


Fig. 9.11 (*Left*): Trend component of SAR model fitted values; (*right*): Spatial stochastic component of SAR model fitted values

However, this model does not account for the heterogeneous distribution of the population by tracts beyond the correction introduced in transforming incidence proportions. Weighted version of these models can be fitted so that tracts are weighted proportionally to the inverse of their population size. For this purpose, we include the parameter `weights=POP8` in the call to the function `lm`.

```
> nylmw <- lm(Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME, data = NY8,
+             weights = POP8)
> summary(nylmw)
```

Call:

```
lm(formula = Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME, data = NY8,
   weights = POP8)
```

Weighted Residuals:

Min	1Q	Median	3Q	Max
-129.07	-14.71	5.82	25.62	70.72

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-0.7784	0.1412	-5.51	8.0e-08 ***
PEXPOSURE	0.0763	0.0273	2.79	0.0056 **
PCTAGE65P	3.8566	0.5713	6.75	8.6e-11 ***
PCTOWNHOME	-0.3987	0.1531	-2.60	0.0097 **

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 33.5 on 277 degrees of freedom

Multiple R-squared: 0.198, Adjusted R-squared: 0.189

F-statistic: 22.8 on 3 and 277 DF, p-value: 3.38e-13

```
> NY8$lmwresid <- residuals(nylmw)
```

Starting with the weighted linear model, we can see that the TCE exposure variable has become significant with the expected sign, indicating that tracts closer to the TCE sites have slightly higher transformed incidence proportions. The other two covariates now also have more significant coefficients. The right panel of Fig. 9.12 shows that information has been shifted from the model residuals to the model itself, with little remaining spatial structure visible on the map.

```
> lm.morantest(nylmw, NYlistw)
```

Global Moran's I for regression residuals

data:

```
model: lm(formula = Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME,
data = NY8, weights = POP8)
weights: NYlistw
```

Moran I statistic standard deviate = 0.4773, p-value = 0.3166

alternative hypothesis: greater

sample estimates:

Observed Moran's I	Expectation	Variance
0.007533	-0.009310	0.001245

The Moran tests for regression residuals can also be used with a weighted linear model object. The results are interesting, suggesting that the misspecification detected by Moran's *I* is in fact related to heteroskedasticity more than to spatial autocorrelation. We can check this for the SAR model too, since `spautolm` also takes a `weights` argument:

```
> nysarw <- spautolm(Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME,
+ data = NY8, listw = NYlistw, weights = POP8)
> summary(nysarw)
```

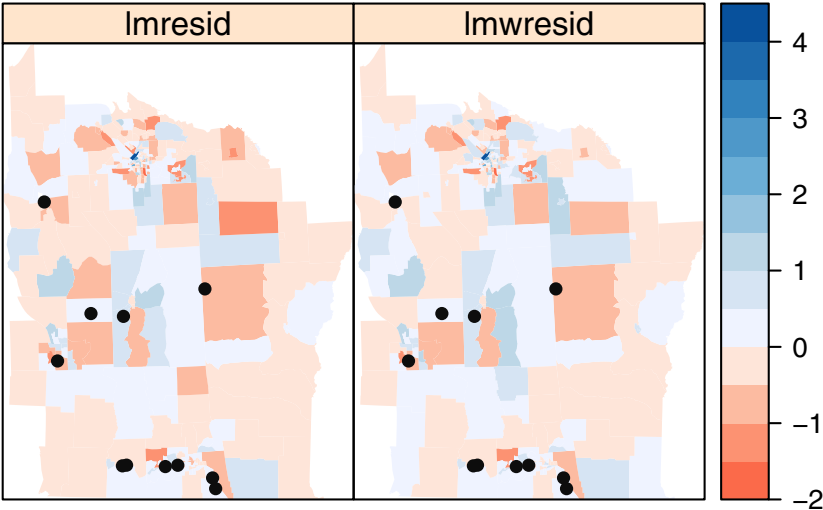


Fig. 9.12 (Left): Residuals from the linear model of transformed incidence proportions; (right): Residuals from the weighted linear model of transformed incidence proportions; TCE site locations shown for comparative purposes

```
Call:
spautolm(formula = Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME, data = NY8,
  listw = NYlistw, weights = POP8)

Residuals:
    Min       1Q   Median       3Q      Max
-1.48488 -0.26823  0.09489  0.46552  4.28343

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.797063   0.144054  -5.5331 3.146e-08
PEXPOSURE    0.080545   0.028334   2.8428 0.004473
PCTAGE65P    3.816731   0.576037   6.6258 3.453e-11
PCTOWNHOME  -0.380778   0.156507  -2.4330 0.014975

Lambda: 0.009564 LR test value: 0.3266 p-value: 0.56764
Numerical Hessian standard error of lambda: 0.01625

Log likelihood: -251.6
ML residual variance (sigma squared): 1104, (sigma: 33.23)
Number of observations: 281
Number of parameters estimated: 6
AIC: 515.2
```

The coefficients of the covariates change slightly in the new model, and all the coefficient *p*-values drop substantially. In this weighted SAR fit, proximity to a TCE site becomes significant. However, there are no traces of spatial autocorrelation left after adjusting for the heterogeneous size of the popula-

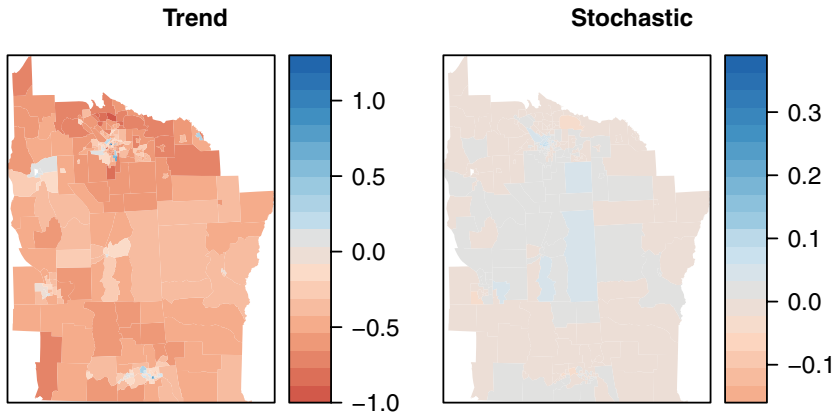


Fig. 9.13 (Left): Trend component of weighted SAR model fitted values; (right): Spatial stochastic component of weighted SAR model fitted values

tion, as Fig. 9.13 shows clearly when compared with Fig. 9.11, using the same break points and colours. This suggests that the spatial variation in population between tracts is responsible for the observed residual spatial correlation after adjusting for covariates.

To compare both models and choose the best one, we use Akaike's Information Criterion (AIC) reported in the model summaries. The AIC is a weighted sum of the log-likelihood of the model and the number of fitted coefficients; according to the criterion, better models are those with the lower values of the AIC. Hence, the weighted model provides a better fitting since its AIC is considerably lower. This indicates the importance of accounting for heterogeneous populations in the analysis of this type of lattice data.

9.4.1.2 Conditional Autoregressive Models

The CAR specification relies on the conditional distribution of the spatial error terms. In this case, the distribution of e_i conditioning on e_{-i} (the vector of all random error terms minus e_i itself) is given. Instead of the whole e_{-i} vector, only the neighbours of area i , defined in a chosen way, are used. We represent them by $e_{j \sim i}$. Then, a simple way of putting the conditional distribution of e_i is

$$e_i | e_{j \sim i} \sim N \left(\frac{\sum_{j \sim i} c_{ij} e_j}{\sum_{j \sim i} c_{ij}}, \frac{\sigma_{e_i}^2}{\sum_{j \sim i} c_{ij}} \right),$$

where c_{ij} are dependence parameters similar to b_{ij} . However, specifying the conditional distributions of the error terms does not imply that the joint distribution exists. To have a proper distribution some constraints must be set

on the parameters of the model. The reader is referred to Schabenberger and Gotway (2005, pp. 338–339) for a detailed description of CAR specifications. For our modelling purposes, the previous guidelines will be enough to obtain a proper CAR specification in most cases.

To fit a CAR model, we can use function `spautolm` again. This time we set the argument `family="CAR"` to specify that we are fitting this type of models.

```
> nycar <- spautolm(Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME,
+   data = NY8, family = "CAR", listw = NYlistw)
> summary(nycar)
```

Call:

```
spautolm(formula = Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME, data = NY8,
  listw = NYlistw, family = "CAR")
```

Residuals:

	Min	1Q	Median	3Q	Max
	-1.539732	-0.384311	-0.030646	0.335126	3.808848

Coefficients:

	Estimate	Std. Error	z value	Pr(> z)
(Intercept)	-0.648362	0.181129	-3.5796	0.0003442
PEXPOSURE	0.077899	0.043692	1.7829	0.0745986
PCTAGE65P	3.703830	0.627185	5.9055	3.516e-09
PCTOWNHOME	-0.382789	0.195564	-1.9574	0.0503053

Lambda: 0.08412 LR test value: 5.801 p-value: 0.016018

Numerical Hessian standard error of lambda: 0.03083

Log likelihood: -275.8

ML residual variance (sigma squared): 0.4076, (sigma: 0.6384)

Number of observations: 281

Number of parameters estimated: 6

AIC: 563.7

The estimated coefficients of the covariates in the model are very similar to those obtained with the SAR models. Nevertheless, the p -values of two covariates, the distance to the nearest TCE and the percentage of people owning a home, are slightly above the 0.05 threshold. The likelihood ratio test indicates that there is significant spatial autocorrelation and the estimated value of λ is 0.0841.

Considering a weighted regression, using the population size as weights, for the same model to account for the heterogeneous distribution of the population completely removes the spatial autocorrelation in the data. The coefficients of the covariates do not change much and all of them become significant. Hence, modelling spatial autocorrelation by means of SAR or

CAR specifications does not change the results obtained; Waller and Gotway (2004, pp. 375–379) give a complete discussion of these results.⁶

```
> nycarw <- spautolm(Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME,
+   data = NY8, family = "CAR", listw = NYlistw, weights = POP8)
> summary(nycarw)
```

Call:

```
spautolm(formula = Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME, data = NY8,
  listw = NYlistw, weights = POP8, family = "CAR")
```

Residuals:

	Min	1Q	Median	3Q	Max
	-1.491042	-0.270906	0.081435	0.451556	4.198134

Coefficients:

	Estimate	Std. Error	z value	Pr(> z)
(Intercept)	-0.790154	0.144862	-5.4545	4.910e-08
PEXPOSURE	0.081922	0.028593	2.8651	0.004169
PCTAGE65P	3.825858	0.577720	6.6223	3.536e-11
PCTOWNHOME	-0.386820	0.157436	-2.4570	0.014010

Lambda: 0.02242 LR test value: 0.3878 p-value: 0.53343

Numerical Hessian standard error of lambda: 0.03819

Log likelihood: -251.6

ML residual variance (sigma squared): 1103, (sigma: 33.21)

Number of observations: 281

Number of parameters estimated: 6

AIC: 515.1

9.4.1.3 Fitting Spatial Regression Models

The `spautolm` function fits spatial regression models by maximum likelihood, by first finding the value of the spatial autoregressive coefficient, which maximises the log likelihood function for the model family chosen, and then fitting the other coefficients by generalised least squares at that point. This means that the spatial autoregressive coefficient can be found by line search using `optimize`, rather than by optimising over all the model parameters at the same time.

The most demanding part of the functions called to optimise the spatial autoregressive coefficient is the computation of the Jacobian, the log determinant of the $n \times n$ matrix $|I - B|$, or $|I - \lambda W|$ in our parametrisation (for an extended discussion, see Bivand et al., 2013). As n increases, the use of the short-cut of

⁶ The fitted coefficient values of the weighted CAR model do not exactly reproduce those of Waller and Gotway (2004, p. 379), although the spatial coefficient is reproduced.

$$\log(|I - \lambda W|) = \log\left(\prod_{i=1}^n (1 - \lambda \zeta_i)\right),$$

where ζ_i are the eigenvalues of W , becomes more difficult. The default method of `method="eigen"` uses eigenvalues, and can thus also set the lower and upper bounds for the line search for λ accurately (as $[1/\min_i(\zeta_i), 1/\max_i(\zeta_i)]$), but is not feasible for large n . It should also be noted that complex eigenvalues are computed for intrinsically asymmetric spatial weights matrices, and their imaginary parts are included, so that the values of the log determinant are correct (see Bivand et al., 2013).

Alternative approaches involve finding the log determinant of a Cholesky decomposition of the sparse matrix $(I - \lambda W)$ directly. Here it is not possible to pre-compute eigenvalues, so one log determinant is computed for each value of λ used, but the number needed is in general not excessive, and much larger n become feasible on ordinary computers. A number of different sparse matrix approaches have been tried, with the use of `Matrix` and `method="Matrix"`, the one suggested currently. All of the Cholesky decomposition approaches to computing the Jacobian require that matrix W be symmetric or at least similar to symmetric, thus providing for weights with "W" and "S" styles based on symmetric neighbour lists and symmetric general spatial weights, such as inverse distance. Matrices that are similar to symmetric have the same eigenvalues, so that the eigenvalues of symmetric $W^* = D^{1/2}WD^{1/2}$ and row-standardised $W = DB$ are the same, for symmetric binary or general weights matrix B , and D a diagonal matrix of inverse row sums of B , $d_{ii} = 1/\sum_{j=1}^n b_{ij}$ (Ord, 1975, p. 125).

```
> nysarwM <- spatoolm(Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME,
+   data = NY8, family = "SAR", listw = NYlistw, weights = POP8,
+   method = "Matrix")

> summary(nysarwM)

Call:
spatoolm(formula = Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME, data = NY8,
  listw = NYlistw, weights = POP8, family = "SAR", method = "Matrix")

Residuals:
    Min       1Q   Median       3Q      Max
-1.48488 -0.26823  0.09489  0.46552  4.28343

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.797063   0.144054 -5.5331 3.146e-08
PEXPOSURE    0.080545   0.028334  2.8428 0.004473
PCTAGE65P    3.816730   0.576037  6.6258 3.453e-11
PCTOWNHOME  -0.380777   0.156507 -2.4330 0.014975

Lambda: 0.009564 LR test value: 0.3266 p-value: 0.56764
Numerical Hessian standard error of lambda: 0.01384
```

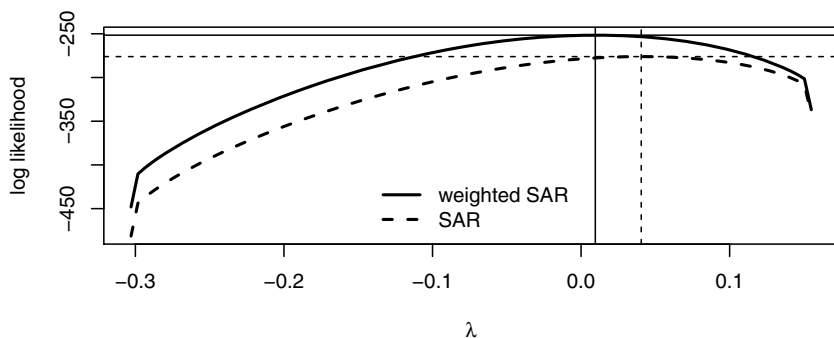


Fig. 9.14 Log likelihood values for a range of values of λ , weighted and unweighted SAR models; fitted spatial coefficient values and maxima shown

```
Log likelihood: -251.6
ML residual variance (sigma squared): 1104, (sigma: 33.23)
Number of observations: 281
Number of parameters estimated: 6
AIC: 515.2
```

The output from fitting the weighted SAR model using functions from the **Matrix** package is identical with that from using the eigenvalues of W .

If it is of interest to examine values of the log likelihood function for a range of values of λ , the `llprof` argument may be used to give the number of equally spaced λ values to be chosen between the inverse of the smallest and largest eigenvalues for `method="eigen"`, or a sequence of such values more generally.

```
> 1/range(eigenw(NYlistw))
[1] -0.3029  0.1550

> nysar_ll <- spautolm(Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME,
+   data = NY8, family = "SAR", listw = NYlistw, llprof = 100)
> nysarw_ll <- spautolm(Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME,
+   data = NY8, family = "SAR", listw = NYlistw, weights = POP8,
+   llprof = 100)
```

Figure 9.14 shows the shape of the values of the log likelihood function along the feasible range of λ for the weighted and unweighted SAR models. We can see easily that the curves are very flat at the maxima, meaning that we could shift λ a good deal without impacting the function value much. The figure also shows the sharp fall-off in function values as the large negative values of the Jacobian kick in close to the ends of the feasible range.

Finally, `family="SMA"` for simultaneous moving average models is also available within the same general framework, but always involves handling dense matrices for fitting.

```

> nysmaw <- spautolm(Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME,
+   data = NY8, family = "SMA", listw = NYlistw, weights = POP8)
> summary(nysmaw)

Call:
spautolm(formula = Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME, data = NY8,
  listw = NYlistw, weights = POP8, family = "SMA")
Residuals:
    Min       1Q   Median       3Q      Max
-1.487080 -0.268990  0.093956  0.466055  4.284087

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.795243    0.143749 -5.5321 3.163e-08
PEXPOSURE    0.080153    0.028237  2.8386 0.004531
PCTAGE65P    3.820316    0.575463  6.6387 3.165e-11
PCTOWNHOME  -0.382529    0.156160 -2.4496 0.014302

Lambda: 0.009184 LR test value: 0.3077 p-value: 0.57909
Numerical Hessian standard error of lambda: 0.01652

Log likelihood: -251.6
ML residual variance (sigma squared): 1105, (sigma: 33.24)
Number of observations: 281
Number of parameters estimated: 6
AIC: 515.2

```

9.4.2 Spatial Econometrics Approaches

One of the attractions of spatial data analysis is the wide range of scientific disciplines involved. Naturally, this leads to multiple approaches to many kinds of analysis, including accepted ways of applying tests and model fitting methods. It also leads to some sub-communities choosing their own sets of tools, not infrequently diverging from other sub-communities. During the 2003 Distributed Computational Statistics meeting, surprise and amusement was caused by the remark that the Internet domain www.spatial-statistics.com contains material chiefly relating to real estate research. But this connection is in fact quite reasonable, as real estate generates a lot of spatial data, and requires suitable methods. Indeed, good understanding of real estate markets and financing is arguably as important to society as a good understanding of the spatial dimensions of disease incidence.

Spatial econometrics is described by Anselin (1988, 2002), and authoritatively advanced by LeSage and Pace (2009), with additional comments by Bivand (2002, 2006) with regard to doing spatial econometrics in R. While the use of case weights, as we have seen above, has resolved a serious model mis-specification in this public health data case, it might be more typical in a legacy econometrics framework to test first for heteroskedasticity, and to try to relieve it by adjusting coefficient standard errors:

```
> library(lmtest)
> bptest(nylm)

studentized Breusch-Pagan test

data:  nylm
BP = 9.214, df = 3, p-value = 0.02658
```

The Breusch–Pagan test (Johnston and DiNardo, 1997, pp. 198–200) results indicate the presence of heteroskedasticity when the residuals from the original linear model are regressed on the right-hand-side variables – the default test set. This might suggest the need to adjust the estimated coefficient standard errors using a variance–covariance matrix (Zeileis, 2004) taking heteroskedasticity into account:

```
> library(sandwich)
> coeftest(nylm)

t test of coefficients:

              Estimate Std. Error t value Pr(>|t|)
(Intercept)  -0.5173    0.1586   -3.26  0.0012 **
PEXPOSURE      0.0488    0.0351    1.39  0.1648
PCTAGE65P      3.9509    0.6055    6.53  3.2e-10 ***
PCTOWNHOME    -0.5600    0.1703   -3.29  0.0011 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

> coeftest(nylm, vcov = vcovHC(nylm, type = "HC4"))

t test of coefficients:

              Estimate Std. Error t value Pr(>|t|)
(Intercept)  -0.5173    0.1617   -3.20  0.00154 **
PEXPOSURE      0.0488    0.0343    1.42  0.15622
PCTAGE65P      3.9509    0.9992    3.95  9.8e-05 ***
PCTOWNHOME    -0.5600    0.1672   -3.35  0.00092 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

There are only minor changes in the standard errors, and they do not affect our inferences.⁷

In spatial econometrics, Moran's I is supplemented by Lagrange Multiplier tests fully described in Anselin (1988, 2002) and Anselin et al. (1996). The development of these tests, as more generally in spatial econometrics, seems to assume the use of row-standardised spatial weights, so we move from symmetric binary weights used above to row-standardised similar to symmetric weights. A key concern is to try to see whether the data generating process is a spatial error SAR or a spatial lag SAR. The former is the SAR that we

⁷ Full details of the test procedures can be found in the references to the function documentation in `lmtest` and `sandwich`.

have already met, while the spatial lag model includes only the endogenous spatially lagged dependent variable in the model.

```
> NYlistwW <- nb2listw(NY_nb, style = "W")
> res <- lm.LMtests(nylm, listw = NYlistwW, test = "all")
> tres <- t(sapply(res, function(x) c(x$statistic, x$parameter,
+   x$p.value)))
> colnames(tres) <- c("Statistic", "df", "p-value")
> printCoefmat(tres)
```

	Statistic	df	p-value
LMerr	5.17	1.00	0.02
LMlag	8.54	1.00	0.00
RLMerr	1.68	1.00	0.20
RLMlag	5.05	1.00	0.02
SARMA	10.22	2.00	0.01

The robust LM tests take into account the alternative possibility, that is the **LMerr** test will respond to both an omitted spatially lagged dependent variable and spatially autocorrelated residuals, while the robust **RLMerr** is designed to test for spatially autocorrelated residuals in the possible presence of an omitted spatially lagged dependent variable. The `lm.LMtests` function here returns a list of five LM tests, which seem to point to a spatial lag specification. Further variants have been developed to take into account both spatial autocorrelation and heteroskedasticity, but are not yet available in R. Again, it is the case that if the fitted model exhibits multicollinearity, the results of the tests will be affected.

The spatial lag model takes the following form:

$$\mathbf{y} = \rho \mathbf{W}\mathbf{y} + \mathbf{X}\beta + \mathbf{e},$$

where $\mathbf{y}$ is the endogenous variable, $\mathbf{X}$ is a matrix of exogenous variables, and $\mathbf{W}$ is the spatial weights matrix. This contrasts with the spatial Durbin model, including the spatial lags of the covariates (independent variables) with coefficients γ :

$$\mathbf{y} = \rho \mathbf{W}\mathbf{y} + \mathbf{X}\beta + \mathbf{W}\mathbf{X}\gamma + \mathbf{e},$$

and the spatial error model:

$$\mathbf{y} - \lambda \mathbf{W}\mathbf{y} = \mathbf{X}\beta - \lambda \mathbf{W}\mathbf{X}\beta + \mathbf{e},$$

$$(\mathbf{I} - \lambda \mathbf{W})\mathbf{y} = (\mathbf{I} - \lambda \mathbf{W})\mathbf{X}\beta + \mathbf{e},$$

which can also be written as

$$\mathbf{y} = \mathbf{X}\beta + \mathbf{u},$$

$$\mathbf{u} = \lambda \mathbf{W}\mathbf{u} + \mathbf{e}.$$

All these models are simultaneous autoregressive models in the sense used in the previous section. Let us now fit a spatial lag model by maximum likelihood, once again finding the spatial lag coefficient by line search, then the remaining coefficients by generalised least squares. In this case, the implication of the spatial lag model is that the incidence rates by census tract depend on one-another directly, not a realistic hypothesis⁸ for this data set:

```
> nylag <- lagsarlm(Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME,
+   data = NY8, listw = NYlistwW)
> summary(nylag)

Call:
lagsarlm(formula = Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME, data = NY8,
  listw = NYlistwW)

Residuals:
    Min       1Q   Median       3Q      Max
-1.626029 -0.393321 -0.018767  0.326616  4.058315

Type: lag
Coefficients: (asymptotic standard errors)
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.505343   0.155850 -3.2425 0.001185
PEXPOSURE    0.045543   0.034433  1.3227 0.185943
PCTAGE65P    3.650055   0.599219  6.0914 1.12e-09
PCTOWNHOME  -0.411829   0.169095 -2.4355 0.014872

Rho: 0.2252, LR test value: 7.75, p-value: 0.0053703
Asymptotic standard error: 0.07954
      z-value: 2.831, p-value: 0.0046378
Wald statistic: 8.015, p-value: 0.0046378

Log likelihood: -274.9 for lag model
ML residual variance (sigma squared): 0.41, (sigma: 0.6403)
Number of observations: 281
Number of parameters estimated: 6
AIC: 561.7, (AIC for lm: 567.5)
LM test for residual autocorrelation
test value: 0.6627, p-value: 0.41561

> bptest.sarlm(nylag)

      studentized Breusch-Pagan test

data:
BP = 7.701, df = 3, p-value = 0.05261
```

⁸ One expects the spatial lag model and its extensions to propose a hypothesis of spillover between observations of the response variable, such as contagion or competition, which is not the case for these leukemia incidence rates; the hypothesised spatial process is modelled by distances from TCE sites (the PEXPOSURE variable).

The spatial econometrics model fitting functions can also use sparse matrix techniques, but when the eigenvalue technique is used, asymptotic standard errors are calculated for the spatial coefficient. There is a numerical snag here, that if the variables in the model are scaled such that the other coefficients are scaled differently from the spatial autocorrelation coefficient, the inversion of the coefficient variance–covariance matrix may fail. The correct resolution is to re-scale the variables, but the tolerance of the inversion function called internally may be relaxed. In addition, an LM test on the residuals is carried out, suggesting that no spatial autocorrelation remains, and a spatial Breusch–Pagan test shows a lessening of heteroskedasticity.

The **McSpatial** package is a welcome addition to the selection of packages for spatial econometrics released on CRAN; we will return to some of its notable features later in this section. The package provides a maximum likelihood function using eigenvalues to estimate the spatial lag model, yielding the same results for the coefficients as **lagsarlm**. Coefficient standard errors differ slightly when asymptotic variances are used in **lagsarlm** because **sarml** always uses variances taken from the numerical Hessian of the optimisation.

```
> library(McSpatial)

> McRes <- sarml(Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME,
+   wmat = listw2mat(NYlistwW), eigvar = eigenw(NYlistwW),
+   print = FALSE, data = NY8)
> c(McRes$beta, rho = McRes$rho, sig2 = McRes$sig2)

(Intercept)    PEXPOSURE    PCTAGE65P    PCTOWNHOME         rho
   -0.50534      0.04554      3.65007     -0.41184      0.22518
      sig2
   0.40998
```

Fitting a spatial Durbin model, a spatial lag model including the spatially lagged explanatory variables (but not the lagged intercept when the spatial weights are row standardised), we see that the fit is not improved significantly.

```
> nymix <- lagsarlm(Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME,
+   data = NY8, listw = NYlistwW, type = "mixed")
> nymix

Call:
lagsarlm(formula = Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME, data = NY8,
  listw = NYlistwW, type = "mixed")
Type: mixed

Coefficients:
      rho      (Intercept)    PEXPOSURE    PCTAGE65P
  0.17578      -0.32260      0.09039      3.61356
PCTOWNHOME lag.PEXPOSURE lag.PCTAGE65P lag.PCTOWNHOME
 -0.02687      -0.05188      0.13123     -0.69950

Log likelihood: -272.7
```

```
> anova(nymix, nylag)
```

	Model	df	AIC	logLik	Test	L.Ratio	p-value
nymix	1	9	563	-273	1		
nylag	2	6	562	-275	2	4.37	0.224

In fitting spatial lag and spatial Durbin models, it has emerged over time that, unlike the spatial error model, the spatial dependence in the parameter ρ feeds back, obliging analysts to base interpretation not on the fitted parameters β , and γ where appropriate, but rather on correctly formulated impact measures (LeSage and Pace, 2009). Discussions of impacts are given by LeSage and Fischer (2008), LeSage and Pace (2009), Anselin and Lozano-Gracia (2008) and Elhorst (2010), as equilibrium effects by Ward and Gleditsch (2008, pp. 44–50), and as emanating effects by Kelejian et al. (2006), extended in Kelejian et al. (2013); an innovative use of impacts in ecology is given by Folmer et al. (2012).

This feedback comes from the elements of the variance–covariance matrix of the coefficients for the maximum likelihood spatial error model linking λ and β being zero, $\partial^2 \ell / (\partial \beta \partial \lambda) = \mathbf{0}$, while in the spatial lag model (and by extension, in the spatial Durbin model), $\partial^2 \ell / (\partial \beta \partial \rho) \neq \mathbf{0}$. In the spatial error model, for right hand side variable r , $\partial y_i / \partial x_{ir} = \beta_r$ and $\partial y_i / \partial x_{jr} = 0$ for $i \neq j$; in the spatial lag model, $\partial y_i / \partial x_{jr} = ((\mathbf{I} - \rho \mathbf{W})^{-1} \mathbf{I} \beta_r)_{ij}$, where $(\mathbf{I} - \rho \mathbf{W})^{-1}$ is known to be dense (LeSage and Pace, 2009, pp. 33–42).

The variance–covariance matrix of the coefficients and the series of traces of the powered weights matrix are the key ingredients needed to compute impact measures for spatial lag and spatial Durbin models. An estimate of the coefficient variance–covariance matrix is needed for Monte Carlo simulation of the impact measures, although the measures themselves may be computed without an estimate of this matrix. LeSage and Pace (2009, pp. 33–42, 114–115) and LeSage and Fischer (2008) provide the background and implementation details for impact measures.

The awkward $S_r(\mathbf{W}) = ((\mathbf{I} - \rho \mathbf{W})^{-1} \mathbf{I} \beta_r)$ matrix term needed to calculate impact measures for the lag model, and $S_r(\mathbf{W}) = ((\mathbf{I} - \rho \mathbf{W})^{-1} (\mathbf{I} \beta_r - \mathbf{W} \gamma_r))$ for the spatial Durbin model, may be approximated using traces of powers of the spatial weights matrix as well as analytically. The average direct impacts are represented by the sum of the diagonal elements of the matrix divided by N for each exogenous variable, the average total impacts are the sum of all matrix elements divided by N for each exogenous variable, while the average indirect impacts are the differences between these two impact vectors.

In **spdep**, **impacts** methods are available for ML spatial lag, spatial Durbin, and other fitted model objects including the spatially lagged response variable. The methods use truncated series of traces using different ways of computing the traces, here powering a sparse matrix, which goes dense, to get exact traces. After coercing the spatial weights to a sorted compressed column-oriented sparse matrix, a form for which many functions and methods

are available, the `trW` function may be used to calculate traces, for the default $m = 30$ traces:

```
> W <- as(as_dgRMatrix_listw(NYlistwW), "CsparseMatrix")
> trMat <- trW(W, type = "mult")
> head(trMat)

[1] 0.00 52.45 16.46 23.58 14.94 15.38
```

Methods for calculating impacts have been written for all relevant spatial econometrics estimators in **spdep**, all supporting the same Monte Carlo testing mechanism as described by LeSage and Pace (2009). This involves sampling from the multivariate Normal distribution given by the fitted coefficients and their covariance matrix, to generate distributions of the impacts. The simulated impacts are returned as `mcmc` objects as defined in the **coda** package (Best et al., 1995), and may be shown for example using `HPDinterval` for Highest Posterior Density intervals:

```
> set.seed(987654)
> imps <- impacts(nymix, tr = trMat, R = 1999)
> imps
```

Impact measures (mixed, trace):

	Direct	Indirect	Total
PEXPOSURE	0.08913	-0.0424	0.04673
PCTAGE65P	3.64049	0.9030	4.54346
PCTOWNHOME	-0.05161	-0.8297	-0.88128

```
> HPDinterval(imps, choice = "direct")
```

	lower	upper
PEXPOSURE	-0.1206	0.300
PCTAGE65P	2.2710	4.748
PCTOWNHOME	-0.5329	0.399

attr(,"Probability")
[1] 0.95

```
> HPDinterval(imps, choice = "indirect")
```

	lower	upper
PEXPOSURE	-0.2832	0.1923
PCTAGE65P	-1.7058	3.3885
PCTOWNHOME	-1.4729	-0.1646

attr(,"Probability")
[1] 0.95

```
> HPDinterval(imps, choice = "total")
```

	lower	upper
PEXPOSURE	-0.04696	0.1346
PCTAGE65P	1.98627	6.9723
PCTOWNHOME	-1.38824	-0.3637

attr(,"Probability")
[1] 0.95

Examining the distributions of the direct impacts, the means of the diagonals of the $S_r(\mathbf{W})$ matrices, and of the indirect impacts, the sums of off-diagonal elements divided by the number of observations, we often find sign differences. Use of impact measures in spatial econometrics is now effectively mandatory for applied analyses using the lagged dependent variable, as the interpretation of the fitted coefficients on the independent variables is misleading if $\rho \neq 0$.

If we impose the Common Factor constraint on the spatial Durbin model, that $\gamma = -\rho\beta$, we fit the spatial error model:

```
> nyerr <- errorsarlm(Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME,
+   data = NY8, listw = NYlistwW)
> summary(nyerr)

Call:errorsarlm(formula = Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME,
  data = NY8, listw = NYlistwW)

Residuals:
      Min       1Q   Median       3Q      Max
-1.628589 -0.384745 -0.030234  0.324747  4.047906

Type: error
Coefficients: (asymptotic standard errors)
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.58662     0.17471  -3.3577 0.000786
PEXPOSURE    0.05933     0.04226   1.4039 0.160335
PCTAGE65P    3.83746     0.62345   6.1552 7.496e-10
PCTOWNHOME  -0.44428     0.18897  -2.3510 0.018721

Lambda: 0.2169, LR test value: 5.425, p-value: 0.019853
Asymptotic standard error: 0.08504
      z-value: 2.551, p-value: 0.010749
Wald statistic: 6.506, p-value: 0.010749

Log likelihood: -276 for error model
ML residual variance (sigma squared): 0.4137, (sigma: 0.6432)
Number of observations: 281
Number of parameters estimated: 6
AIC: 564, (AIC for lm: 567.5)

> LR.sarlm(nyerr, nymix)

Likelihood ratio for spatial linear models

data:
Likelihood ratio = -6.693, df = 3, p-value = 0.08234
sample estimates:
Log likelihood of nyerr Log likelihood of nymix
      -276.0              -272.7
```

Both the spatial lag and Durbin models appear to fit the data somewhat better than the spatial error model. However, in relation to our initial interest in the relationship between transformed incidence proportions and exposure

to TCE sites, we are no further forward than we were with the linear model, and although we seem to have reduced the mis-specification found in the linear model by choosing the spatial lag model, the reduction in error variance is only moderate.

In considering impacts, LeSage and Pace (2009, pp. 41–42) also suggest interpreting the regression coefficients of the spatial error model with independent and spatially lagged independent variables, a spatial Durbin error model, as direct and indirect impacts respectively. The model also accommodates modelling error autocorrelation:

```
> nyerr1 <- errorsarlm(Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME,
+   data = NY8, listw = NYlistwW, etype = "emixed")
> coef(nyerr1)
```

	lambda	(Intercept)	PEXPOSURE	PCTAGE65P
	0.16152	-0.40046	0.09337	3.65813
PCTOWNHOME	lag.PEXPOSURE	lag.PCTAGE65P	lag.PCTOWNHOME	
	-0.06921	-0.04791	0.79233	-0.77312

Once again we see negative indirect impact measures for the exposure variable. We can draw MCMC samples with the `MCMCsamp` method for fitted spatial regression models to examine the shape of the distributions of direct and indirect impacts from the spatial Durbin error model. The `MCMCsamp` method uses the `rwmetrop` random walk Metropolis algorithm from the **LearnBayes** package to sample the fitted model using the same function as that used to compute its numerical Hessian. Figure 9.15 shows the similarity of the distributions of direct and indirect impacts for the exposure variable of the spatial Durbin error model and those of the spatial Durbin derived from the MC simulation used to estimate impacts on p. 310.

```
> set.seed(987654)
> resMCMC <- MCMCsamp(nyerr1, mcmc = 5000, burnin = 500,
+   listw = NYlistwW)
```

Spatial econometrics has also seen the development of alternatives to maximum likelihood methods for fitting models. For example, the spatial lag model may be fitted by analogy with two-stage least squares in a simultaneous system of equations, by using the spatial lags of the explanatory variables as instruments for the spatially lagged dependent variable; typically, $[X, WX, WWX]$ are used as instruments. The **sphet** package described in detail by Piras (2010) and under active development, provides implementations of many of the estimators described by Kelejian and Prucha (2007), Kelejian and Prucha (2010) and Arraiz et al. (2010); it is the convention in this branch of the literature to reverse the names of the spatial coefficients – ρ is here the spatial error coefficient, λ the spatial lag coefficient. The `spreg` estimator functions takes a number of arguments controlling the kind of model to be fitted, also accommodating models handling heteroskedastic innovations:

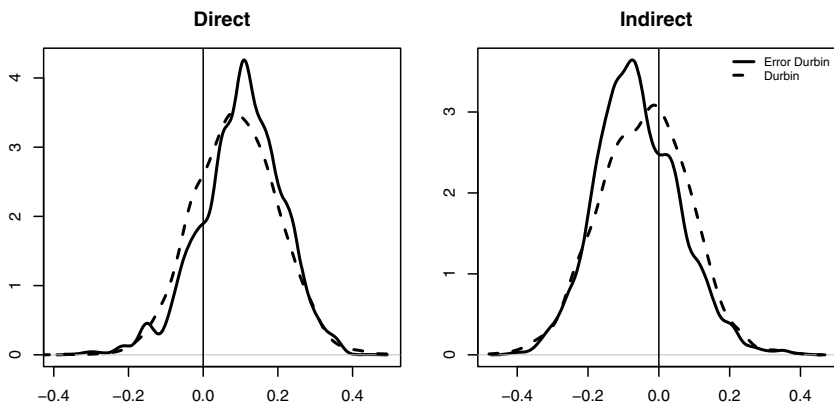


Fig. 9.15 Impacts of the exposure (inverse distance to the closest TCE location) variable for spatial Durbin error and spatial Durbin models: (*Left*) Direct; (*right*) Indirect

```
> library(sphet)
> nyGMLag <- spreg(Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME,
+   data = NY8, listw = NYlistwW, model = "lag", het = FALSE)
> summary(nyGMLag)
```

Stsls

Call:

```
spreg(formula = Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME, data = NY8,
      listw = NYlistwW, model = "lag", het = FALSE)
```

Residuals:

Min.	1st Qu.	Median	3rd Qu.	Max.
-1.7000	-0.3640	-0.0189	0.3370	3.9600

Coefficients:

	Estimate	Std. Error	t-value	Pr(> t)
(Intercept)	-0.4912	0.1561	-3.15	0.0017 **
PEXPOSURE	0.0416	0.0346	1.20	0.2284
PCTAGE65P	3.2926	0.6360	5.18	2.3e-07 ***
PCTOWNHOME	-0.2357	0.2005	-1.18	0.2398
lambda	0.4928	0.1676	2.94	0.0033 **

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

The `spreg` function can also be used to fit a spatial error model using a Generalised Moments (GM) estimator for the autoregressive parameter. It uses a GM approach to optimise λ and σ^2 jointly, and where the numerical search surface is not too flat, can be an alternative to maximum likelihood methods when n is large; heteroskedastic innovations can also be handled.

```
> nyGMerr <- spreg(Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME,
+   data = NY8, listw = NYlistwW, model = "error", het = FALSE)
> summary(nyGMerr)
```

Generalized stsls

Call:

```
spregr(formula = Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME, data = NY8,
        listw = NYlistwW, model = "error", het = FALSE)
```

Residuals:

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
-1.68	-0.40	-0.04	0.00	0.34	4.17

Coefficients:

	Estimate	Std. Error	t-value	Pr(> t)
(Intercept)	-0.5781	0.1752	-3.30	0.00097 ***
PEXPOSURE	0.0580	0.0425	1.37	0.17209
PCTAGE65P	3.8485	0.6240	6.17	6.9e-10 ***
PCTOWNHOME	-0.4578	0.1895	-2.42	0.01571 *
rho	0.2223	0.0942	2.36	0.01836 *

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Another recently published package, **splm**, provides spatial panel estimators described by Millo and Piras (2012), extending the aspatial panel estimator package **plm** (Croissant and Millo, 2008). Finally, the **McSpatial** package provides functions for spatial quantile regression, described by McMillen (2012, 2013), and several spatial probit estimators, which will not be covered here.

```
> fit <- qregspiv(Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME,
+               wmat = listw2mat(NYlistwW), data = NY8, tau = 0.5,
+               nboot = 200)
```

Kim and Muller Two-Stage Quantile Regression Results

	Coef.	Bootstrap SE	Bootstrap Z-values	Pr(> z)
(Intercept)	-0.63668	0.18347	-3.470	0.0005201
PEXPOSURE	0.08967	0.02819	3.181	0.0014683
PCTAGE65P	3.14625	0.81257	3.872	0.0001080
PCTOWNHOME	-0.18028	0.17070	-1.056	0.2909145
WY	0.43563	0.21768	2.001	0.0453660

	Percentile-Lo	Percentile-Hi
(Intercept)	-0.95445	-0.2222
PEXPOSURE	0.03642	0.1560
PCTAGE65P	1.32939	4.6766
PCTOWNHOME	-0.54874	0.1399
WY	0.01448	0.8997

The **qregspiv** function estimates a spatial quantile regression including the lagged dependent variable using instruments $[X, WX]$. It does appear that quantile regression may permit more insight into the relationship between the exposure variable and the dependent variable, but as we can see from the presentation in McMillen (2013), the possibilities of these methods will become clearer when they have been used in more applied work.

9.4.3 Other Methods

Other methods can be used to model dependency between areas. In this section we introduce some of them, based in part on the applied survey reported by Dormann et al. (2007). A specific difficulty that we met above when considering mixed-effects models is that available functions for model fitting use point support rather than polygon support. This means that our prior description of the relationships between observations are distance-based, and so very similar to those described in detail in Chap. 8, where the focus was more on interpolation than modelling. These methods are discussed in the spatial context by Schabenberger and Gotway (2005, pp. 352–382) and Waller and Gotway (2004, pp. 380–409), and hierarchical methods are being employed with increasing frequency (Banerjee et al., 2004). The term *geoadditive* model has begun to be used in recent years, to express the addition of a spatially smoothed component to models of various kinds, see Kneib et al. (2009) and Hothorn et al. (2011) for examples.

Generalised Additive Models (GAM) are very similar to generalised linear models, but they also allow for including non-linear terms in the linear predictor term (Hastie and Tibshirani, 1990; Wood, 2006). It is worth noting that the `formula` argument to linear, generalised linear, spatial, and many other models may contain polynomial and spline terms if desired, but these need to be configured manually. Different types of non-linear functions are available, and may be chosen in the `s()` function in the formula. Here, an isotropic thin plate regression spline is used effectively as a semi-parametric trend surface to add smooth spatial structure from the residuals to the fit, as in Chap. 7 (p. 200).

```
> library(mgcv)
> NY8$x <- coordinates(NY8)[, 1]/1000
> NY8$y <- coordinates(NY8)[, 2]/1000
> nyGAM1 <- gam(Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME +
+ s(x, y), weights = POP8, data = NY8)
> anova(nylmw, nyGAM1, test = "Chisq")
```

Analysis of Variance Table

```
Model 1: Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME
Model 2: Z ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME + s(x, y)
  Res.Df    RSS    Df Sum of Sq Pr(>Chi)
1      277 310778
2      273 305229  3.81      5550    0.27
```

This does not add much to what we already knew from the weighted linear model, with the differences in the residual degrees of freedom showing that the thin plate regression spline term only takes 3.81 estimated degrees of freedom. This does not, however, exploit the real strengths of the technique. Because it can fit generalised models, we can step back from using the transformed incidence proportions to use the case counts (admittedly not integer because of the sharing-out of cases with unknown tract within blocks),

offset by the logarithm of tract populations. Recall that we have said that distributional assumptions about the response variable matter – our response variable perhaps ought to be treated as discrete, so methods respecting this may be more appropriate.

Using the Poisson Generalised Linear Model (GLM) fitting approach, we fit first with `glm`; the Poisson model is introduced in Chap. 10. We can already see that this GLM approach yields interesting insights and that the effects of TCE exposure on the numbers of cases are significant (Fig. 9.16).

```
> nyGLMp <- glm(Cases ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME +
+   offset(log(POP8)), data = NY8, family = "poisson")
```

```
> summary(nyGLMp)
```

Call:

```
glm(formula = Cases ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME +
    offset(log(POP8)), family = "poisson", data = NY8)
```

Deviance Residuals:

Min	1Q	Median	3Q	Max
-2.678	-1.057	-0.198	0.633	3.266

Coefficients:

	Estimate	Std. Error	z value	Pr(> z)
(Intercept)	-8.1344	0.1826	-44.54	< 2e-16 ***
PEXPOSURE	0.1489	0.0312	4.77	1.8e-06 ***
PCTAGE65P	3.9982	0.5978	6.69	2.3e-11 ***
PCTOWNHOME	-0.3571	0.1903	-1.88	0.061 .

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

Null deviance: 428.25 on 280 degrees of freedom
 Residual deviance: 353.35 on 277 degrees of freedom
 AIC: Inf

Number of Fisher Scoring iterations: 5

With the GLM to start from, we again add an isotropic thin plate regression spline in `gam`. There is little over-dispersion present – fitting with `family=quasipoisson`, in which the dispersion parameter is not fixed at unity, so they can model over-dispersion that does not result in large changes. Model comparison shows that the presence of the spline term is now significant. While the coefficient values of the Poisson family fits are not directly comparable with the linear fits on the transformed incidence proportions, we can see that exposure to TCE sites is clearly more significant.

```
> nyGAMp <- gam(Cases ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME +
+   offset(log(POP8)) + s(x, y), data = NY8, family = "poisson")
> summary(nyGAMp)
```

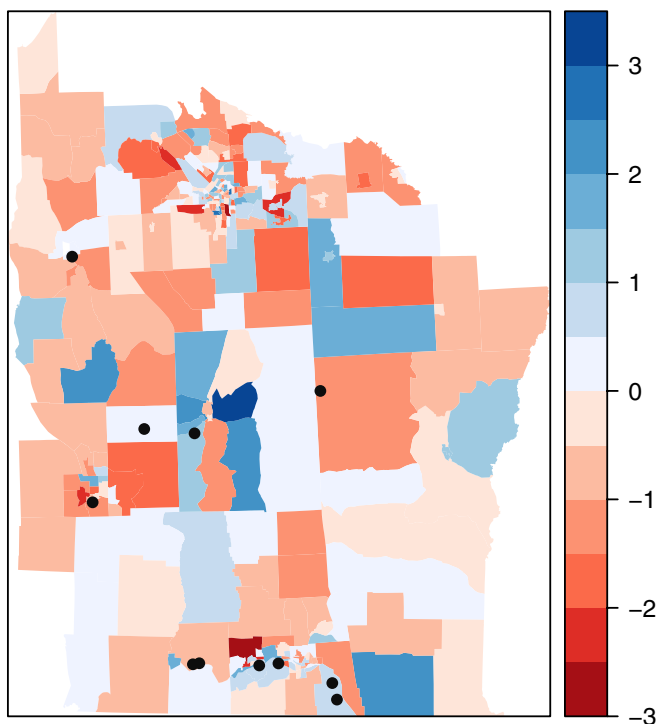


Fig. 9.16 Residuals from the Poisson regression model; TCE site locations shown for comparative purposes

Family: poisson

Link function: log

Formula:

Cases ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME + offset(log(POP8)) +
s(x, y)

Parametric coefficients:

	Estimate	Std. Error	z value	Pr(> z)
(Intercept)	-8.1366	0.2147	-37.89	< 2e-16 ***
PEXPOSURE	0.1681	0.0598	2.81	0.005 **
PCTAGE65P	3.7199	0.6431	5.78	7.3e-09 ***
PCTOWNHOME	-0.3602	0.1994	-1.81	0.071 .

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Approximate significance of smooth terms:

	edf	Ref.df	Chi.sq	p-value
s(x,y)	7.71	10.7	8.64	0.63

R-sq.(adj) = 0.394 Deviance explained = 21.4%
UBRE score = 0.2815 Scale est. = 1 n = 281

```
> anova(nyGLMp, nyGAMP, test = "Chisq")
```

Analysis of Deviance Table

```
Model 1: Cases ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME +  
offset(log(POP8))
```

```
Model 2: Cases ~ PEXPOSURE + PCTAGE65P + PCTOWNHOME +  
offset(log(POP8)) + s(x, y)
```

	Resid. Df	Resid. Dev	Df	Deviance	Pr(>Chi)
1	277	353			
2	269	337	7.71	16.7	0.029 *

Generalised Estimating Equations (GEE) are an alternative to the estimation of GLMs when we have correlated data. They are often used in the analysis of longitudinal data, when we have several observations for the same subject. In a spatial setting, the correlation arises between neighbouring areas. The treatment in Dormann et al. (2007) is promising for the restricted case of clusters of grid cells, but has not yet been extended to irregular point or polygon support.

Generalised linear mixed-effect models (GLMM) extend GLMs by allowing the incorporation of mixed effects into the linear predictor; see Waller and Gotway (2004, pp.387–392) and Schabenberger and Gotway (2005, pp. 359–369). These random effects can account for correlation between observations. GLMM have developed from mixed effects models described by Pinheiro and Bates (2000) and Zuur et al. (2009) to GLMM, often estimated using Bayesian methods (Zuur et al., 2012).

The Moran eigenvector approach (Dray et al., 2006; Griffith and Peres-Neto, 2006) involves the spatial patterns represented by maps of eigenvectors of the doubly-centred spatial weights matrix; by choosing suitable orthogonal patterns and adding them to a linear or generalised linear model, the spatial dependence present in the residuals can be moved into the model. Two estimating functions are provided in **spdep**. In its general form, **SpatialFiltering** chooses the subset of the n eigenvectors that reduce the residual spatial autocorrelation in the error of the model with covariates. The lag form adds the covariates in assessment of which eigenvectors to choose, but does not use them in constructing the eigenvectors. **SpatialFiltering** was implemented and contributed by Yongwan Chun and Michael Tiefelsdorf, and is presented in Tiefelsdorf and Griffith (2007); **ME** is based on Matlab code by Pedro Peres-Neto and is discussed in Dray et al. (2006), Dormann et al. (2007) and Griffith and Peres-Neto (2006). The use of these methods is covered in detail in Borcard et al. (2011, pp.243–284) and reviewed in broader context by Dray et al. (2012).

Geographically weighted regression is described by Fotheringham et al. (2002) and involves first selecting a bandwidth for an isotropic spatial weights kernel, typically a Gaussian kernel with a fixed bandwidth chosen by leave-one-out cross-validation. Choice of the bandwidth can be very demanding,

as n regressions must be fitted at each step. Alternative techniques are available, for example for adaptive bandwidths, but they may often be even more compute-intensive. Estimating functions are provided in the **spgwr** package. GWR is discussed by Schabenberger and Gotway (2005, pp. 316–317) and Waller and Gotway (2004, p. 434), O’Sullivan and Unwin (2010, pp. 226–233), and presented with examples by Lloyd (2007, pp. 79–86). The weaknesses established by Wheeler and Tiefelsdorf (2005) and Wheeler (2007) are in part addressed in the **gwrr** package, which provides diagnostic tools. Páez et al. (2011) report comprehensive simulation results indicating that geographically weighted regression should only be used with caution, and only in certain situations as detailed in their article.

Further ways of using R for applying different methods for modelling areal data are presented in Chap. 10. It is important to remember that the availability of implementations of methods does not mean that any of them are ‘best practice’ as such. It is the analyst who has responsibility for choices of methods and implementations in relation to situation-specific requirements and available data. What the availability of a range of methods in R does make possible is that the analyst has choice and has tools for ensuring that the research outcomes are fully reproducible.

Chapter 10

Disease Mapping

Spatial statistics have been widely applied in epidemiology to the study of the distribution of disease. As we have already shown in Chap. 7, displaying the spatial variation of the incidence of a disease can help us to detect areas where the disease is particularly prevalent, which may lead to the detection of previously unknown risk factors. As a result of the growing interest, Spatial Epidemiology (Elliott et al., 2000) has been established as a new multidisciplinary area of research in recent years.

The importance of this field has been reflected in the appearance of different books and special issues in some scientific journals. To mention a few, recent reviews on the subject can be found in Waller and Gotway (2004), while the special issues of the journal *Statistical Methods in Medical Research* (Lawson, A., editor, 2005) and *Statistics in Medicine* (Lawson, A., Gangnon, R. E. and Wartenburg, D., editors, 2006) also summarise novel developments in disease mapping and the detection of clusters of disease. Walter and Birnie (1991) compared many different atlases of disease and they compile the main issues to pay attention to when reporting disease maps. Banerjee et al. (2004, pp. 88–97, 158–174) also tackle the problem of disease mapping and develop examples that can be reproduced using S-PLUSTM SpatialStats (Kaluzny et al., 1998) and WinBUGS (Spiegelhalter et al., 2003). In addition, some data sets and code with examples available from the book website.¹ Haining (2003) considers different issues in disease mapping, including a Bayesian approach as well and provides examples, data and code to reproduce the examples in the book. Schabenberger and Gotway (2005, pp. 394–399) briefly describe the smoothing of disease rates. Finally, Lawson et al. (2003) provide a practical approach to disease mapping with a number of examples (with full data sets and WinBUGS code) that the reader should be able to reproduce after reading this chapter.

In this chapter we will refer to the analysis of data which have been previously aggregated according to a set of administrative areas. The analysis

¹ <http://www.biostat.umn.edu/~brad/data2.html>

of data available at individual level requires different techniques which have been described in Chap. 7. These kinds of aggregated data are continuously collected by health authorities and usually cover mortality and morbidity counts. Special registers have also been set up in several countries to record the incidence of selected diseases, such as cancer or congenital malformations. Spatial Epidemiology often requires the integration of large amounts of data, statistical methods and geographic information. R offers a unique environment for the development of these types of analysis given its good connectivity to databases and the different statistical methods implemented.

Therefore, the aim of this chapter is not to provide a detailed and comprehensive description of all the methods currently employed in Spatial Epidemiology, but to show those which are widely used. A description as to how they can be computed with R and how to display the results will be provided. From this description, it will be straightforward for the user to adapt the code provided in this chapter to make use of other methods. Other analysis of health data, as well as contents on which this chapter is built, can be found in Chap. 9.

The North Carolina SIDS data, which have already been displayed in Chap. 3 (Fig. 3.6), will be used throughout this chapter in the examples that accompany the statistical methodology described here. The SIDS data set records the number of sudden infant deaths in North Carolina for 1974–1978 and 1979–1984 and some other additional information. It is available as `nc.sids` in package **spdep** and further information is available in the associated manual page. Cressie and Read (1985) and Cressie and Chan (1989), for example, provide a description of the data and study whether there is any clustered pattern of the cases.

10.1 Introduction

The aim of disease mapping is to provide a representation of the spatial distribution of the risk of a disease in the study area, which we will assume is divided into several non-overlapping smaller regions. The risk may reflect actual deaths due to the disease (mortality) or, if it is not fatal, the number of people who suffer from the disease (morbidity) in a certain period of time for the population at risk.

Hence, basic data must include the population at risk and number of cases in each area. These data are usually split according to different variables in a number of groups or strata, which can be defined using sex, age and other important variables. When available, a deprivation index (Carstairs, 2000) is usually employed in the creation of the strata. By considering data in different groups the importance of each variable can be explored and potential confounding factors can be removed (Elliott and Wakefield, 2000) before doing any other analysis of the data. For example, if the age is divided into 13

groups and sex is also considered, this will lead to 26 strata in the population. Note that depending on the type of study the population at risk may be a reduced subset of the total population. For example, in our examples, it is reduced to the number of children born during the period of study.

Following this structure, we will denote by P_{ij} and O_{ij} the population and observed number of cases in region i and stratum j . Summing over all strata j we can get the total population and number of cases per area, which we will denote by P_i and O_i . Summing again over all the regions will give the totals which will be denoted by P_+ and O_+ .

Representing the observed number of cases alone gives no information about the risk of the disease given that the cases are mainly distributed according to the underlying population. In order to obtain an estimate of the risk, the observed number of cases must be compared to an *expected* number of cases.

If P_i and O_i are already available, which is the simplest case, the expected number of cases in region i can be calculated as $E_i = P_i r_+$, where r_+ is the overall incidence ratio equal to $\frac{O_+}{P_+}$. This is an example of the use of *indirect standardisation* (Waller and Gotway, 2004, pp.12–15) to compute the expected number of cases for each area.

When data are grouped in strata, a similar procedure can be employed to take into account the distribution of the cases and population in the different strata. Instead of computing a global ratio $\frac{O_+}{P_+}$ for all regions, a different ratio is computed for each stratum as $r_j = \frac{\sum_i O_{ij}}{\sum_i P_{ij}}$. In other words, we could compute the ratio between the sum of all cases at stratum j over the population at stratum j . In this situation, the expected number of cases in region i is given by $E_i = \sum_j P_{ij} r_j$.

This standardisation is also called *internal standardisation* because we have used the same data to compute reference rates r_j . Sometimes they are known because another reference population has been used. For example, national data can be used to compute the reference rates to be used later in regional studies.

The following code, based on that available in the `nc.sids` manual page, will read the SIDS data, boundaries of North Carolina and the adjacency structure of the North Carolina counties (as in Cressie and Read, 1985) in GAL format (see Chap.9).

```
> library(maptools)
> library(spdep)
> nc_file <- system.file("shapes/sids.shp", package = "maptools")[1]
> llCRS <- CRS("+proj=longlat +datum=NAD27")
> nc <- readShapePoly(nc_file, ID = "FIPSN0", proj4string = llCRS)
> rn <- sapply(slot(nc, "polygons"), function(x) slot(x,
+   "ID"))
> gal_file <- system.file("etc/weights/ncCR85.gal",
+   package = "spdep")[1]
> ncCR85 <- read.gal(gal_file, region.id = rn)
```


By using the argument `region.id` we made sure that the order of the list of neighbours `ncCR85` is the same as the areas in the `SpatialPolygonDataFrame` object `nc`.

10.2 Statistical Models

A common statistical assumption to model the observed number of cases in region i and stratum j is that it is drawn from a Poisson distribution with mean $\theta_i E_{ij}$, with θ_i the relative risk. Thus, a relative risk of 1 means that the risk is as the average in the reference region (from where the rates r_j are obtained) and it will be of interest to locate regions where the relative risk is significantly higher than 1. This basic model is described in Banerjee et al. (2004, pp. 158–159), Haining (2003, pp. 194–199) and Lawson et al. (2003, pp. 2–8).

Note that implicitly we are assuming that there is no interaction between the risk and the population strata, i.e., the relative risk θ_i only depends on the region.

At this point, a basic estimate of the risk in a given region can be computed as $SMR_i = O_i/E_i$, which is known as the *Standardised Mortality Ratio*. This is why the data involving the cases are often referred to as the *numerator* and the data of the population as the *denominator*, because they are used to compute a ratio that estimates the relative risk. Figure 10.1 shows the *SMRs* of the SIDS data for the period 1974–1978. Waller and Gotway (2004, pp. 11–18) describe in detail this and other types of standardisation, together with other risk ratios frequently used in practise.

```
> nc$Observed <- nc$SID74
```

The Population at risk is the number of births:

```
> nc$Population <- nc$BIR74
> r <- sum(nc$Observed)/sum(nc$Population)
> nc$Expected <- nc$Population * r
```

and from that we can compute Standardised Mortality Ratio SMR_i :

```
> nc$SMR <- nc$Observed/nc$Expected
```

Using the fact that O_i is Poisson distributed, we can obtain a confidence interval for each SMR (using function `pois.exact` from package `epitools`). Figure 10.2 displays the 95 % confidence interval of the SMR computed for each area. Highly significant risks (i.e., those whose confidence interval is above one) have been drawn using a dashed line and the name of the county has been added as a label. Anson county, which has been pointed out as a clear extreme value in previous studies (Cressie and Chan, 1989), is the one with the highest confidence interval.

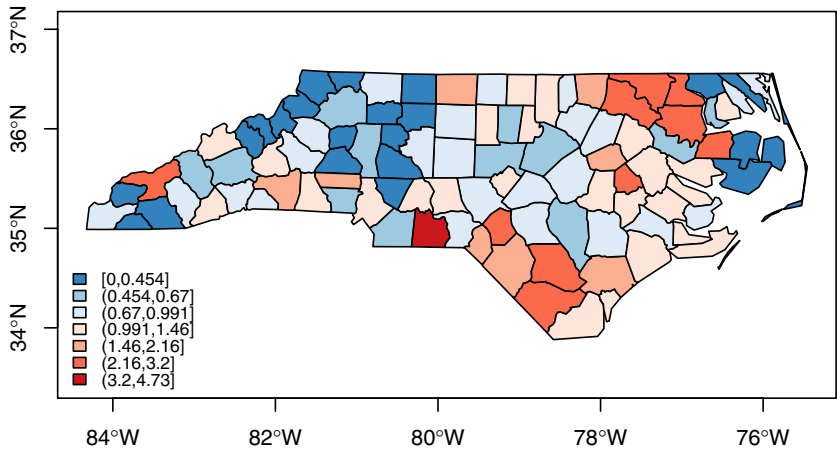


Fig. 10.1 Standardised Mortality Ratio of the North Carolina SIDS data in the period 1974–1978

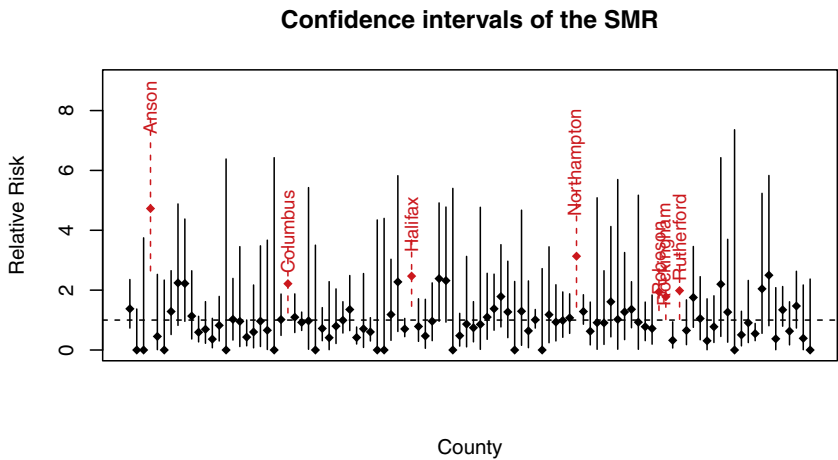


Fig. 10.2 Confidence intervals of the *SMR* obtained with an exact formula. The *dots* represent the *SMR* of each area. The confidence intervals *red dots* and *dashed lines* are significantly higher than 1

10.2.1 Poisson-Gamma Model

Unfortunately, using a Poisson distribution implies further assumptions that may not always hold. One key issue is that for this distribution the mean and the variance of O_i are supposed to be the same. It is often the case that data are “over-dispersed”, meaning that the variance of the data is higher than their mean and the statistical model needs to be expanded. A simple way to allow for a higher variance is to use a Negative Binomial distribution instead of the Poisson.

The Negative Binomial distribution can also be regarded as a mixed model in which a random effect following a Gamma distribution for each region is considered. This formulation is known as the Poisson-Gamma (PG) model because it can be structured as the following two-level model:

$$\begin{aligned} O_i | \theta_i, E_i &\sim Po(\theta_i E_i) \\ \theta_i &\sim Ga(\nu, \alpha) \end{aligned}$$

In this model, we also consider the relative risk θ_i as a random variable which is drawn from a Gamma distribution with mean ν/α and variance ν/α^2 . Note that now the distribution of O_i is conditioned on the value of θ_i . The unconditioned distribution for each O_i is easy to derive and follows a Negative Binomial distribution with size parameter ν and probability $\frac{\alpha}{\alpha + E_i}$.

In addition, the posterior distribution of θ_i , i.e., its distribution given the observed data $\{O_i\}_{i=1}^n$, can also be derived and follows a Gamma distribution with parameters $\nu + O_i$ and $\alpha + E_i$. In other words, the information provided by observing the data has *updated* our prior knowledge or assumptions on θ_i . The posterior expectation of θ_i is

$$E[\theta_i | O_i, E_i] = \frac{\nu + O_i}{\alpha + E_i}$$

which can also be expressed as a compromise between the prior mean of the relative risks and SMR_i so that this is a *shrinkage* estimator:

$$E[\theta_i | O_i, E_i] = \frac{E_i}{\alpha + E_i} SMR_i + (1 - \frac{E_i}{\alpha + E_i}) \frac{\nu}{\alpha}.$$

Two issues should be noted from this estimator. First of all, when E_i is small, as often happens in low population areas, a small variation in O_i can produce dramatic changes in the value of SMR_i . For this reason, according to the previous expectation, the SMR_i will have a low weight, as compared to that of the prior mean. Secondly, information is borrowed from all the areas in order to construct the posterior estimates given that ν and α are the same for every region. This concept of *borrowing strength* can be modified and extended to take into account a different set of areas or neighbours.

Given that ν and α are unknown, we need a procedure to estimate them. They can be easily estimated from the data using the Method of Moments following formulae given by Clayton and Kaldor (1987) to produce Empirical Bayes (EB) estimates, which is implemented in package **DCluster**:

```
> library(DCluster)
> eb <- empbaysmooth(nc$Observed, nc$Expected)
> nc$EBPG <- eb$smthrr
> eb$nu
```

[1] 4.630656

```
> eb$alpha
[1] 4.395646
```

In this example, the values are $\nu = 4.6307$ and $\alpha = 4.3956$, which gives a prior mean of the relative risks of 1.0535 (very close to 1).

Probability maps (Choynowski, 1959) are a convenient way of representing the significance of the observed values. These maps show the probability of a value being higher than the observed data according to the assumption we have made about the model. In other words, probability maps show the p -value of the observed number of cases under the current model. Figure 10.3 represents the probability maps for the Poisson and Poisson-Gamma models. We compare both maps to show how significance varies with the model. We noted that the Poisson-Gamma model was more appropriate in this case due to over-dispersion, and we should try to make inference based on this model. As expected, the p -values for the Poisson-Gamma model are higher because more variability is permitted. Nevertheless, there are still two zones of high risk to the northeast and south.

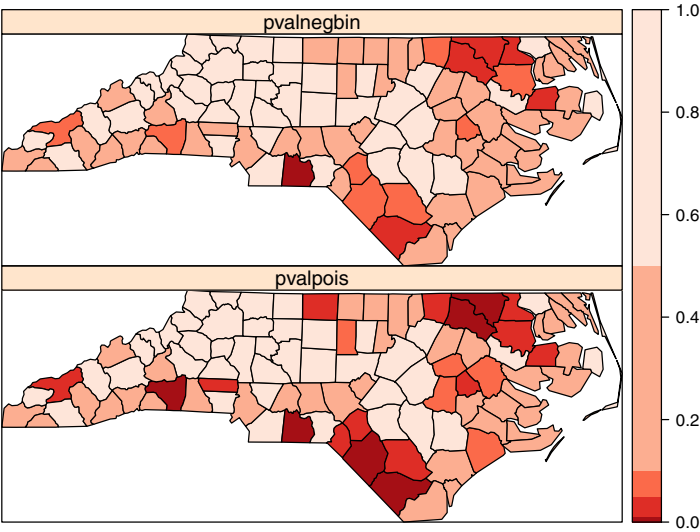


Fig. 10.3 Probability maps for the Poisson and Negative Binomial models

10.2.2 Log-Normal Model

Clayton and Kaldor (1987) proposed another risk estimator based on assuming that the logarithm of the relative risks ($\beta_i = \log(\theta_i)$) follows a multivariate

normal distribution with common mean ϕ and variance σ^2 . In this case, the estimate of the log-relative risk is not taken as $\log(O_i/E_i)$ but $\log((O_i + 1/2)/E_i)$, because the former is not defined if O_i is zero. The EM algorithm is used to obtain estimates of the mean and variance of the model, which can be plugged into the following Empirical Bayes estimator of β_i :

$$\hat{\beta}_i = b_i = \frac{\hat{\phi} + (O_i + \frac{1}{2})\hat{\sigma}^2 \log[(O_i + \frac{1}{2})/E_i] - \hat{\sigma}^2/2}{1 + (O_i + \frac{1}{2})\hat{\sigma}^2}$$

where $\hat{\phi}$ and $\hat{\sigma}^2$ are the estimates of the prior mean and variance, respectively. These are given by

$$\hat{\phi} = \frac{1}{n} \sum_{i=1}^n b_i = \bar{b}$$

and

$$\hat{\sigma}^2 = \frac{1}{n} \left\{ \hat{\sigma}^2 \sum_{i=1}^n [1 + \hat{\sigma}^2(O_i + 1/2)]^{-1} + \sum_{i=1}^n (b_i - \hat{\phi})^2 \right\}$$

Estimates b_i are updated successively using previous formulae until convergence. Hence, the estimator for θ_i is $\hat{\theta}_i = \exp\{\hat{\beta}_i\}$. Note that now information is borrowed by estimating the common parameters φ and σ^2 , and that the resulting estimates are a combination of the local estimate of the log relative risk and φ . Unfortunately, the current estimator is more complex than the previous one and it cannot be reduced to a shrinkage expression. It is evaluated by:

```
> ebln <- lognormalEB(nc$Observed, nc$Expected)
> nc$EBLN <- exp(ebln$smthrr)
```

10.2.3 Marshall's Global EB Estimator

Marshall (1991) developed a new EB estimator assuming that the relative risks θ_i have a common prior mean μ and variance σ^2 , but without specifying any distribution. By using the Method of Moments, he is able to work a new estimator out employing a shrinkage estimator as follows:

$$\hat{\theta}_i = \hat{\mu} + C_i(SMR_i - \hat{\mu}) = (1 - C_i)\hat{\mu} + C_i SMR_i$$

where

$$\hat{\mu} = \frac{\sum_{i=1}^n O_i}{\sum_{i=1}^n E_i}$$

and

$$C_i = \frac{s^2 - \hat{\mu}/\bar{E}}{s^2 - \hat{\mu}/\bar{E} + \hat{\mu}/E_i},$$

where $\bar{E}$ stands for the mean of the E_i 's and s^2 is the usual unbiased estimate of the variance of the SMR_i 's. Unfortunately, this estimator can produce negative estimates of the relative risks when $s^2 < \hat{\mu}/\bar{E}$, in which case $\hat{\theta}_i = \hat{\mu}$ is taken.

The shrinkage of this estimator highly depends (again) on the value of E_i . If it is high, which means that the SMR_i is a reliable estimate, C_i will be close to 1 and the estimator will give more weight to the SMR_i . On the other hand, if E_i is small, more weight is given to the estimate of the prior mean $\hat{\mu}$ because the SMR_i is less reliable and so it borrows more information from other areas. We compute the Marshall risk estimator by:

```
> library(spdep)
> EBMarshall <- EBest(nc$Observed, nc$Expected)
> nc$EBMarshall <- EBMarshall[, 2]
```

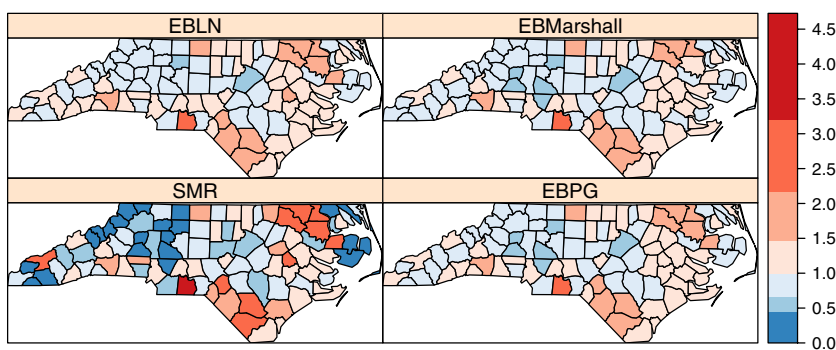


Fig. 10.4 Comparison of different risk estimators. *SMR* displays the Standardised Mortality Ratio, whilst *EBPG*, *EBLN* and *EBMarshall* show different empirical Bayes estimates using the Poisson-Gamma model, the log-normal model and Marshall's global estimator, respectively

Figure 10.4 represents the different estimates obtained by the different estimators described so far. All EB estimators seem to produce very similar estimates in all the areas. By comparing those maps to the map that shows the *SMR* it is possible to see how very extreme values (either high or low) have been shifted towards the global mean. In other words, these values have been *smoothed* by taking into account global information in the computation of the estimate.

To compare the variability of the estimates produced by each method we have created a boxplot of each set of values, which appear in Fig. 10.6. From

the plot it is clear that the *SMR* is the most variable and that the other three have been shrunk towards the global mean, which is approximately 1. Hence, we might expect similar results when using any of the EB estimators. As pointed by Marshall (1991), the estimation procedure based on the Poisson-Gamma model proposed by Clayton and Kaldor (1987) may not converge in some circumstances and another estimator should be used. The EB proposed by Marshall (1991) can also be unfeasible in similar circumstances. Hence, the EB estimator based on the log-Normal model seems to be the most computationally stable and reliable.

All these EB estimators produce smoothed estimates of the risk rates *borrowing information* from the global area but, depending on the size and extension of the total area under study, it could be more reasonable to consider only a small set of areas which are close to each other. A common example is to use only the areas that share a boundary with the current region to compute its risk estimate. Unfortunately, this procedure involves the use of more complex models that require the use of additional software and will be discussed in the following sections.

10.3 Spatially Structured Statistical Models

Although borrowing strength globally can make sense in some cases, it is usually better to consider a reduced set of areas to borrow information from. A sensible choice is to take only neighbouring areas or areas which are within a certain distance from the current area.

Marshall (1991) proposed another estimator that only requires local information to be computed. For each region, a set of neighbours is defined and local means, variances and shrinkage factors are defined in a similar way as in the global estimator, but considering only the areas in the neighbourhood. This produces a local shrinkage for each area, instead of the global shrinkage provided by the previous estimator.

```
> nc$EBMrshloc <- EBlocal(nc$Observed, nc$Expected, ncCR85)$est
```

The way this estimator is computed raises a new question about how areas are related to each other. In the previous models, no account for how areas were distributed in the study region was considered, so that the influence of a region did not depend on its location at all. That is, we would obtain the same estimates if the distribution of the regions were permuted at random. With the new estimator the exact location of the areas is crucial, and different locations of the regions will give different estimates as a result. The way regions are placed in a map can be described by means of its *topology*, which accounts for the neighbours of a given region. See Chap.9 for more details on this and how to obtain it.

Although neighbours are usually defined as two regions that share a common boundary, Cressie and Chan (1989) define two regions as neighbours if the distance between their centroids is within 30 miles. This is not a trivial issue since different definitions of neighbourhood will produce different results.

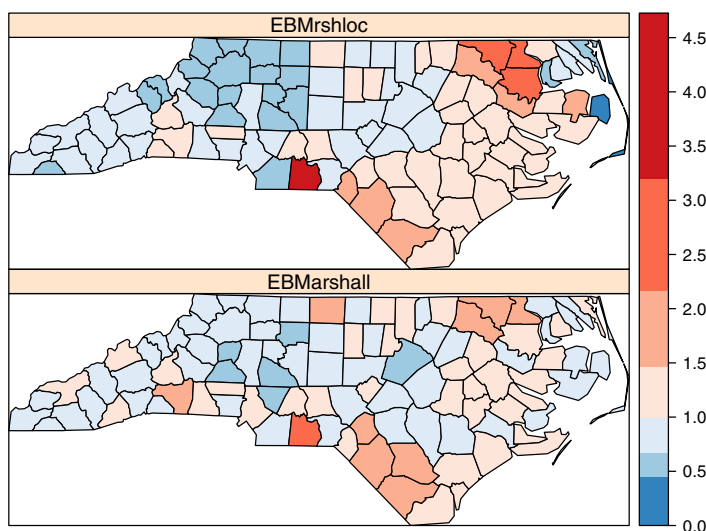


Fig. 10.5 Marshall's EB estimator using local (*top*) and global (*bottom*) information

The two estimators proposed by Marshall have been displayed in Fig. 10.5. The version that uses only local information produces smoothed estimates of the relative risks that are shrunk towards the local mean that turned out to be less shrunk towards the global mean. In addition, the shrinkage produced by the local estimator is in general lower than for the global estimator.

The boxplot presented in Fig. 10.6 compares the different EB estimators discussed so far. Marshall's local estimator also shows a general shift towards the global mean, but it is less severe than for the others because only local information is employed. In general, EB smoothed estimators have been criticised because they fail to cope with the uncertainty of the parameters of the model (Bernardinelli and Montomoli, 1992) and to produce an overshrinkage since the parameters of the prior distributions are estimated from the data and remain fixed. In order to solve this problem several *constrained* EB estimators have been proposed to force the posterior distribution of the smoothed estimates to resemble that of the raw data (Louis, 1984; Devine and Louis, 1994; Devine et al., 1994).

Full Bayes methods allow setting the prior distributions for these parameters and, hence, permit a greater variability and produce more suitable smoothed estimates. More standard smoothed risk estimators that borrow

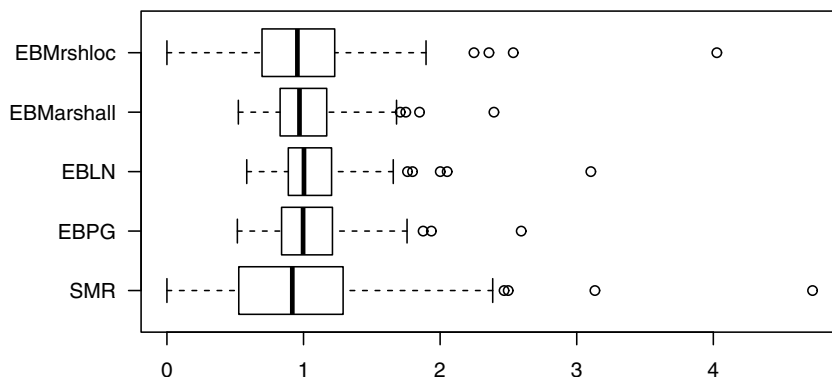


Fig. 10.6 Comparison of raw and EB estimators of the relative risk

information locally can be developed by resorting to Spatial Autoregressive and Conditional Autoregressive specifications (Waller and Gotway, 2004). Basically, these models condition the relative risk in an area to be similar to the values of the neighbouring areas. More details are given in the next sections of this chapter and in Chap. 9 for non-Bayesian models.

10.4 Bayesian Hierarchical Models

Bayesian Hierarchical Models make an appropriate framework for the development of spatially structured models. The model is specified in different layers, so that each one accounts for different sources of variation. For example, they can cope with covariates at the same time as borrowing strength from neighbours to improve the quality of estimates. The use of these models in disease mapping is considered in Haining (2003, pp. 307–311, 367–376), Waller and Gotway (2004, pp. 409–429), Banerjee et al. (2004, pp. 159–169) and Schabenberger and Gotway (2005, pp. 394–399). Lawson et al. (2003) offer a specific volume on the subject with reproducible examples.

Besag et al. (1991), BYM henceforth, introduced in their seminal paper a type of models that split the variability in a region as the sum of a spatially correlated variable (which depends on the values of its neighbours) plus an area-independent effect (which reflects local heterogeneity). Although direct estimates of the variables in the model can seldom be obtained when using Bayesian Hierarchical Models, their posterior distributions can be obtained by means of Markov Chain Monte Carlo (MCMC) techniques. Basically, MCMC methods generate simulations of the parameters of the model which, after a suitable burn-in period, become realisations of their posterior

distributions. An introduction to MCMC and its main applications, including disease mapping, can be found in Gilks et al. (1996).

WinBUGS (Spiegelhalter et al., 2003) is a software that uses MCMC methods (in particular, Gibbs Sampling; Gelman et al., 2003) to simulate from the posterior distributions of the parameters in the model. Starting from a set of initial values, one sample of each variable is simulated at the time using the full conditional distribution of the parameter given the other parameters. After a suitable burn-in period, the simulations generated correspond to the joint posterior distribution.

Although WinBUGS is the main software package, it was previously known as BUGS and currently it comes in different flavours. OpenBUGS, for example, is the open source alternative to WinBUGS and it is actually a fork of the main WinBUGS software. Apart from the advantage of coming with the source code, OpenBUGS can be called from R using package **BRugs**. In addition, some specific plug-ins have been developed for WinBUGS to deal with certain applications. It is worth mentioning GeoBUGS, which provides a graphical interface to the management of maps and compute adjacency relationships within WinBUGS and OpenBUGS, and can create maps with the results. Lawson et al. (2003) have described extensively how to do a disease mapping using Multilevel Models with WinBUGS (and MLwiN), and is a comprehensive reference for those willing to go deeper in this subject.

A package for using WinBUGS from R is **R2WinBUGS** (Sturtz et al., 2005). This package calls WinBUGS using its scripting facilities so that the resulting log file containing all the results can be loaded into R after the computations have finished. **R2WinBUGS** will be the package used in this book. Finally, it is worth noting that Gelman and Hill (2007) provide a good and accessible text on data analysis using Bayesian hierarchical models and, in Chaps. 16 and 17, describe the use of R and WinBUGS via **R2WinBUGS**.

BayesX (Brezger et al., 2005; Belitz et al., 2012) is another interesting software for Bayesian inference that can be used as an standalone software or within R. BayesX is available from <http://www.bayesx.org> and it is aimed at fitting structured additive regression models. Package **BayesXsrc** (Adler et al., 2012) provides the source code, **R2BayesX** provides a link between BayesX and R and **BayesX** includes some functionalities to manipulate data and output (for example, it provides functions to convert to the spatial adjacency format used in BayesX). BayesX has also the possibility of fitting models using MCMC, restricted maximum likelihood (REML) and penalised least squares (PLS).

INLA is an R package for Bayesian inference based on the methods developed in Rue et al. (2009). Here, the authors propose a new method (called the Integrated Nested Laplace Approximation, INLA) to approximate the posterior marginals of the parameters in the model. Although INLA will not provide the full posterior distribution, this is not needed in many applications, such as the estimation of relative risks in disease mapping. Furthermore,

INLA provides accurate estimates in most cases at lower computational time. Package **INLA** is available off-CRAN from <http://www.r-inla.org>.

Finally, package **CARBayes** provides a number of functions for fitting some widely used spatial models using MCMC. Although it is more limited than the previous packages, it is entirely implemented in R and provides a simple alternative to fitting spatial models for disease mapping.

10.4.1 The Poisson-Gamma Model Revisited

The following example shows a full Bayesian Poisson-Gamma formulation (i.e., assigning priors to the parameters ν and α) to produce smoothed estimates of the relative risks that can be run from R using **R2WinBUGS**. In this model, ν and α have been assigned vague gamma priors so that as little prior information as possible is introduced. The WinBUGS code needed to run the Poisson-Gamma model is shown in Fig. 10.7.

```
model
{
  for(i in 1:N)
  {
    observed[i]~dpois(mu[i])
    mu[i]<-theta[i]*expected[i]
    theta[i]~dgamma(nu, alpha)
  }

  nu~dgamma(.01, .01)
  alpha~dgamma(.01, .01)
}
```

Fig. 10.7 Code of the Poisson-Gamma model for WinBUGS

The next chunk of code shows how to convert all the necessary data into the structure used by WinBUGS. In addition, we need to set up the initial values for some of the parameters of the model. Data and initial values must be saved into a separated file.

```
> library(R2WinBUGS)
> N <- length(nc$Observed)
> d <- list(N = N, observed = nc$Observed, expected = nc$Expected)

> pgmodelfile <- paste(getwd(), "/PG-model.txt", sep = "")
> wdir <- paste(getwd(), "/PG", sep = "")
> if (!file.exists(wdir)) {
+   dir.create(wdir)
+ }
> BugsDir <- "/home/asdar2/.wine/dosdevices/c:/Program Files/WinBUGS14"
```

We can then run WinBUGS models by a call to `bugs`:

```
> MCMCres <- bugs(data = d, inits = list(list(nu = 1, alpha = 1)),
+   working.directory = wdir, parameters.to.save = c("theta",
+   "nu", "alpha"), n.chains = 1, n.iter = 20000,
+   n.burnin = 10000, n.thin = 10, model.file = pgmodelfile,
+   bugs.directory = BugsDir, WINEPATH = "/usr/bin/winepath")
```

Briefly, the `bugs` function takes as input data, initial values, model file and other information required, and creates a script that will be run with WinBUGS.² Function `bugs` will create the necessary files (data, initial values and script) that will be placed under `working.directory`. After running the model, the output will be stored here as well. The WinBUGS script will check the syntax of the model, load the data and compile the model. The following step is to read (or generate from the priors) the initial values for the parameters of the model and generate 10,000 simulations of the Markov Chain (keeping just 1 every 10). Note that the burn-in simulations are not saved. Then, we specify that variables “nu”, “alpha” and “theta” will be saved and 10,000 more simulations are generated, of which only 1 of every 10 are saved to avoid autocorrelation and improve mixing and convergence. Finally, the summary statistics and plots are saved into the log files under the working directory. Two such files are created: an ODC file (WinBUGS format) with summary statistics and plots, and an ASCII file with the summary statistics. In addition, a summary of the output is stored as a series of lists in `MCMCres`. The posterior mean and median of the relative risks can be extracted by

```
> nc$PGmean <- MCMCres$mean$theta
> nc$PGmedian <- MCMCres$median$theta
```

Although it will not be described here in detail, it is essential to check that the Markov Chain has converged so that the values that we are using have been drawn from the posterior distribution of the parameters. An example using package `coda` (Best et al., 1995) is shown later when evaluating a more complex model.

As we have obtained samples from the posterior distributions of ν and α , it is possible to compute pointwise estimates and probability intervals for both parameters. For the sake of simplicity, and to be able to compare the values obtained with those from the EB approach, the pointwise estimates (posterior means) of these values were $\hat{\nu} = 6.253$ and $\hat{\alpha} = 5.967$, which are slightly higher than the ones obtained with the EB estimator. Similar estimates can be obtained for the relative risks but note that now they are not based on single values of ν and α , but that the relative risk estimates are *averaged* over different values of those parameters.

Even though point estimates of the relative risks are usually very useful, for most applications it is better to give a credible interval, for it can be used

² Windows users must modify the paths in `working.directory`, `model.file` and `bugs.directory` accordingly, and remove the argument `WINEPATH`, which is not needed.

to detect areas of significantly high risk, if the interval is over 1. Figure 10.8 summarises the 95 % credible intervals for each region. The median has been included (black dot), and the areas whose credible intervals are above 1 have been highlighted using a dashed line and the county name displayed. As we mentioned before, Anson county is of special interest because it shows the highest risk.

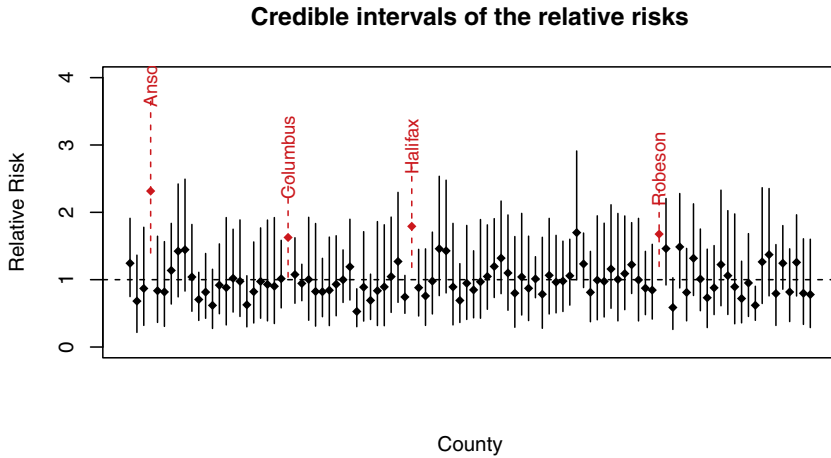


Fig. 10.8 Ninety-five percent credible intervals of the relative risks obtained with WinBUGS using a Full Bayes Poisson-Gamma model

In Fig. 10.9 we have compared the estimates of the relative risks provided by the Poisson-Gamma model using both Empirical Bayes and Full Bayes approaches. Both estimation procedures lead to very similar estimates and they only differ in a few areas. Note how they all provide smoothed estimates of the relative risks, as compared to the raw *SMRs*.

Fitting the Poisson-Gamma model exactly as described in Fig. 10.7 may be difficult with other software. However, the Poisson-Gamma model can be regarded as a model with random effects in the log-scale, i.e., the terms $\log(\theta_i)$ can be seen as a set of random effects with a common prior distribution. For example, we could formulate this model using Gaussian random effects as follows:

$$\begin{aligned} O_i &\sim Po(E_i \theta_i) \\ \log(\theta_i) &= \alpha + u_i \\ \alpha &\sim N(0, \sigma_\alpha^2) \\ u_i &\sim N(0, \sigma_u^2) \end{aligned}$$

where the intercept α is included to model the overall risk and random effects u_i account for differences between the areas. This is a very simple model that we will use to introduce the use of the **BayesX**, **INLA** and **CARBayes**.

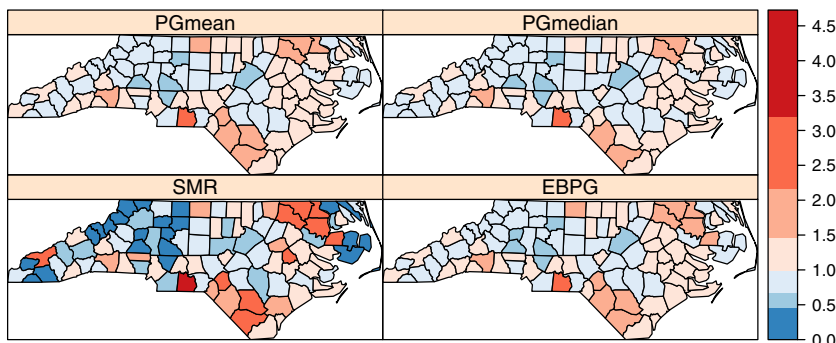


Fig. 10.9 Comparison of Empirical Bayes and Full Bayes estimates of the relative risks using a Poisson-Gamma model

To fit models with **BayesX** we will use function `bayesx`, which works similarly to function `gam`. The model to fit is included in a `formula` where different types of effects can be included. Additive model terms can be included using function `sx` and in the following example it is used to set a different random effect for each area. This has been done by including an index variable in the data set (`AREAID`). A full list of additive models available can be found in the manual page for `sx`.

```
> library(R2BayesX)
> nc$AREAID <- 1:nrow(nc)

> pgbayesx <- bayesx(Observed~sx(AREAID, bs = "re"),
+   offset = log(nc$Expected), family = "poisson", data = as(nc,
+   "data.frame"))
```

INLA can be called using function `inla`. It works in a way similar to `gam` and `bayesx` where models are defined via a `formula` and additive effects are included using function `f`. After loading the package, we'll show its version:

```
> library(INLA)

> inla.version()

INLA build date .....: Wed Feb 13 09:38:42 CET 2013

> pginla <- inla(Observed~offset(log(Expected)) - 1 +
+   f(AREAID, model = "iid"), family = "poisson", data = as(nc,
+   "data.frame"), control.predictor = list(compute = TRUE),
+   control.compute = list(dic = TRUE))
```

CARBayes provides a more limited interface and different functions are needed to fit different types of models, depending on the latent effects and the family used in the model. In this case, we have called function `poisson.independent` to fit a model to Poisson data using independent random effects:

```

> library(CARBayes)

> ncdf <- as(nc, "data.frame")
> attach(ncdf)
> pgcarbayes <- poisson.independent(formula = Observed~
+   offset(log(Expected)), burnin = 5000, n.sample = 10000)
> detach(ncdf)

```

All these functions return results in different data structures. In order to compare the different estimates, we show how to access the results and display the different estimates.

```

> nc$PGBAYESX <- pgbayesx$fitted.values[order(pgbayesx$bayesx.setup$order),
+   2]/nc$Expected
> nc$PGINLA <- pginla$summary.fitted.values$mean/nc$Expected
> nc$PGCARBAYES <- pgcarbayes$fitted.values[, 1]/nc$Expected

```

Figure 10.10 shows the different estimates obtained with the Poisson-Gamma model and the alternative model using Gaussian random effects. Note how similar the estimates are across models and software used.

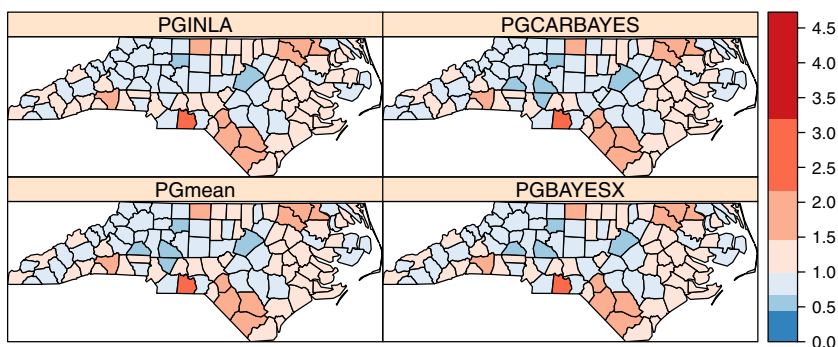


Fig. 10.10 Different estimates of the relative risks using a Poisson-Gamma model (with WinBUGS) and similar models using Gaussian random effects fitted with other software

10.4.2 Spatial Models

Additional spatial structure can be included by considering a CAR model and covariates can be used to explain part of the variability of the relative risks. Cressie and Chan (1989) considered the proportion of non-white births as an important factor related to the incidence of SIDS. A full description of these models can be found in Banerjee et al. (2004, Chap. 5). In general, these models are far more complex than the Poisson-Gamma described before, and

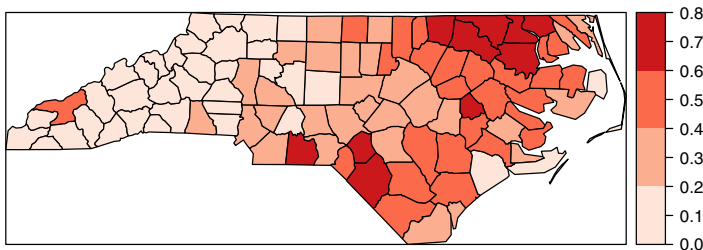


Fig. 10.11 Proportion of non-white births in Carolina, 1974–1978. Notice the similar pattern to the relative risk estimates

they should be used with extreme caution because of the high number of parameters and possible interactions between them.

As described in Sect. 9.4.1.1, the CAR specification for a set of random variables $\{v_i\}_{i=1}^n$ can be written as follows:

$$v_i | v_{-i} \sim N\left(\sum_{j \sim i} \frac{w_{ij} v_j}{\sum_j w_{ij}}, \sigma_v^2 / \sum_j w_{ij}\right)$$

where w_{ij} is a weight that measures the strength of the relationship between (neighbour) regions i and j and σ_v^2 indicates the conditional variance of the CAR specification.

Although the conditional distributions are proper, it is not the case for the joint distribution. Nevertheless, this CAR specification is often used as a prior distribution of the spatial random effects and it can lead to a proper posterior under some constraints (Ghosh et al., 1998).

Given the structure of the CAR specification, it is necessary to know the neighbours of each region. They can be defined in different ways, depending on the type of relationship that exists between the areas. In our example, we will use the same neighbourhood structure as in Cressie and Read (1985) which can be found in package `spdep`. In addition, it is necessary to assign a weight to each pair of neighbours which measure the strength of the interaction. Following Besag et al. (1991) we will set all the weights to 1 if regions are neighbours and 0 otherwise.

The flexibility of the Bayesian Hierarchical Models allows us to perform an Ecologic Regression (English, 1992) at the same time as we consider random and spatial effects. By including covariates in our model we aim to assess and remove the effect of potential confounders or risk factors. The assessment of the importance of a covariate is indicated by the estimated value of its coefficient and its associated probability interval. If, for example, the 95 % credible interval does not contain the value 0, we may assume that the coefficient is significant and, if greater than zero, it will indicate a positive relationship between the risk and the variable.

The results of an Ecologic Regression can be potentially misleading if we try to make inference at the individual level since the effects that operate at that level may not be the same as those reflected at the area level. In the extreme case, the effects might even be reversed. A solution to this is to combine the aggregated data with some individual data from a specific survey, which can be also used to improve the estimation of the effects of the covariates (Jackson et al., 2006).

In our example, we have available the number of non-white births in each county. The variable ethnicity is often used in the United States as a surrogate of the deprivation index (Krieger et al., 1997). Considering this variable in our model may help to explain part of the spatial variability of the risk of SIDS. In order to account for the total number of births, we will use the proportion of non-white births in the area. This will also allow us to compare the values for different counties. Figure 10.11 shows the spatial variation of the proportion of non-white births. Notice how there exists a similar pattern to that shown by the spatial distribution of the *SMR* and the different EB estimates. Finally, the WinBUGS model used in this case can be found in Fig. 10.12. We have used the priors suggested in Best et al. (1999) to allow a better identifiability of the random effects u_i and v_i .

```
model
{
  for(i in 1:N)
  {
    observed[i] ~ dpois(mu[i])
    log(theta[i]) <- alpha + beta*nonwhite[i] + u[i] + v[i]
    mu[i] <- expected[i]*theta[i]

    u[i] ~ dnorm(0, precu)
  }

  v[1:N] ~ car.normal(adj[], weights[], num[], precv)

  alpha ~ dflat()
  beta ~ dnorm(0, 1.0E-5)
  precu ~ dgamma(0.001, 0.001)
  precv ~ dgamma(0.1, 0.1)

  sigmau<-1/precu
  sigmav<-1/precv
}
```

Fig. 10.12 Code of the Besag-York-Mollié model for WinBUGS

The code shown below converts the neighbours of each county as specified in Cressie and Read (1985) into the format required by WinBUGS. Note that these are already available in an R object and that they have been matched

so that the list of neighbours is in the right order. When this is not the case, proper matching must be done.

```
> idx <- match(attr(ncCR85.nb, "region.id"), nc$CNTY_ID)
> nc.nb <- ncCR85
> nc.nb <- nc.nb[order(idx)]
> nc.nb <- lapply(nc.nb, function(X, idx) {
+   idx[X]
+ }, idx = (idx))
> class(nc.nb) <- "nb"
> nc.nb <- nc.nb[(order(idx))]
> nc.nb <- nb2WB(nc.nb)
```

Function `nb2WB` can be used to convert an `nb` object into a list containing the three elements (`adj`, `weights` and `num`) required for a CAR specification in WinBUGS. Similarly, the function `listw2WB` can be used for a `listw` object. The main difference is that `nb2WB` sets all the weights to 1 whilst `listw2WB` keeps the values of the weights as in the `listw` object.

```
> nc.nb <- nb2WB(ncCR85)
```

The last step is to compute the proportion of non-white births in each county and create the R lists with the data and initial values.

```
> nc$nwprop <- nc$NWBIR74/nc$BIR74
> d <- list(N = N, observed = nc$Observed, expected = nc$Expected,
+   nonwhite = nc$nwprop, adj = nc.nb$adj, weights = nc.nb$weights,
+   num = nc.nb$num)
> dwoutcov <- list(N = N, observed = nc$Observed,
+   expected = nc$Expected, adj = nc.nb$adj, weights = nc.nb$weights,
+   num = nc.nb$num)
> inits <- list(u = rep(0, N), v = rep(0, N), alpha = 0,
+   beta = 0, precu = 0.001, precv = 0.001)
```

The procedure to run this model is very similar to the previous one. We only need to change the file names of the model, data and initial values. Notice that not all initial values must be provided and that some can be generated randomly. In this model, we are going to keep the summary statistics for a wide range of variables. In addition to the relative risks θ_i , we want to summarise the values of the intercept (α), the coefficient of the covariate (β) and the values of the random (u_i) and spatial (v_i) effects.

```
> bymmodelfile <- paste(getwd(), "/BYM-model.txt", sep = "")
> wdir <- paste(getwd(), "/BYM", sep = "")
> if (!file.exists(wdir)) {
+   dir.create(wdir)
+ }
> BugsDir <- "/home/asdar2/.wine/dosdevices/c:/Program Files/WinBUGS14"

> MCMCres <- bugs(data = d, inits = list(inits),
+   working.directory = wdir, parameters.to.save = c("theta",
+   "alpha", "beta", "u", "v", "sigmau", "sigmav"),
```

```
+ n.chains = 1, n.iter = 30000, n.burnin = 20000,
+ n.thin = 10, model.file = bymmodelfile, bugs.directory = BugsDir,
+ WINEPATH = "/usr/bin/winepath")
```

After running the model, the summary statistics are added to the spatial object that contains all the information about the North Carolina SIDS data so that it can be displayed easily.

The data obtained by running WinBUGS can be added to `nc`:

```
> nc$BYMmean <- MCMCres$mean$theta
> nc$BYMumean <- MCMCres$mean$u
> nc$BYMvmean <- MCMCres$mean$v
```

Convergence of the Markov Chain must be assessed before attempting any valid inference from the results. Cowles and Carlin (1996) provide a summary of several methods and a useful discussion. They state the difficulty to assess convergence in practice. Some of the criteria discussed in the paper are implemented in package **coda**. These criteria can be applied to the *deviance* of the model to monitor convergence of the joint posterior. Ideally, several chains (each one starting at a sufficiently different point) should be run in parallel so that their traces can be compared (Gelman and Rubin, 1992).

WinBUGS can produce the output in the format required by **coda**. Basically, it will produce an index file (`codaIndex.txt`) plus another file with the values of the variables (`coda1.txt`) that can be read using function `read.coda`. This will create an object of type `mcmc` that contains the simulations from all the variables saved in WinBUGS. Figure 10.13 shows the trace and density of the posterior distribution of the deviance and the parameters α , β and the relative risk of Robeson county (area number 94 and cluster centre in Fig. 10.21).

For a single chain, Geweke's criterion (Geweke, 1992) can be computed to assess convergence. It is a score test based on comparing the means of the first and the last part of the Markov Chain (by default, the 10 % initial values to the 50 % last values). If the chain has converged, both means should be equal. Given that it is a score test, values of the test statistics between -1.96 and 1.96 indicate convergence, whilst more extreme values point to a lack of convergence. For the selected parameters, it seems that convergence has been reached:

```
> geweke.diag(ncoutput[, c("deviance", "alpha", "beta",
+ "theta[94]")])
```

```
Fraction in 1st window = 0.1
```

```
Fraction in 2nd window = 0.5
```

deviance	alpha	beta	theta[94]
1.4639	-0.1123	0.3868	-0.7404

Figure 10.14 shows the *SMR* and the smoothed estimate of the relative risks obtained. When the posterior distribution is very skewed, the posterior median can be a better summary statistic, but it is not the case here.

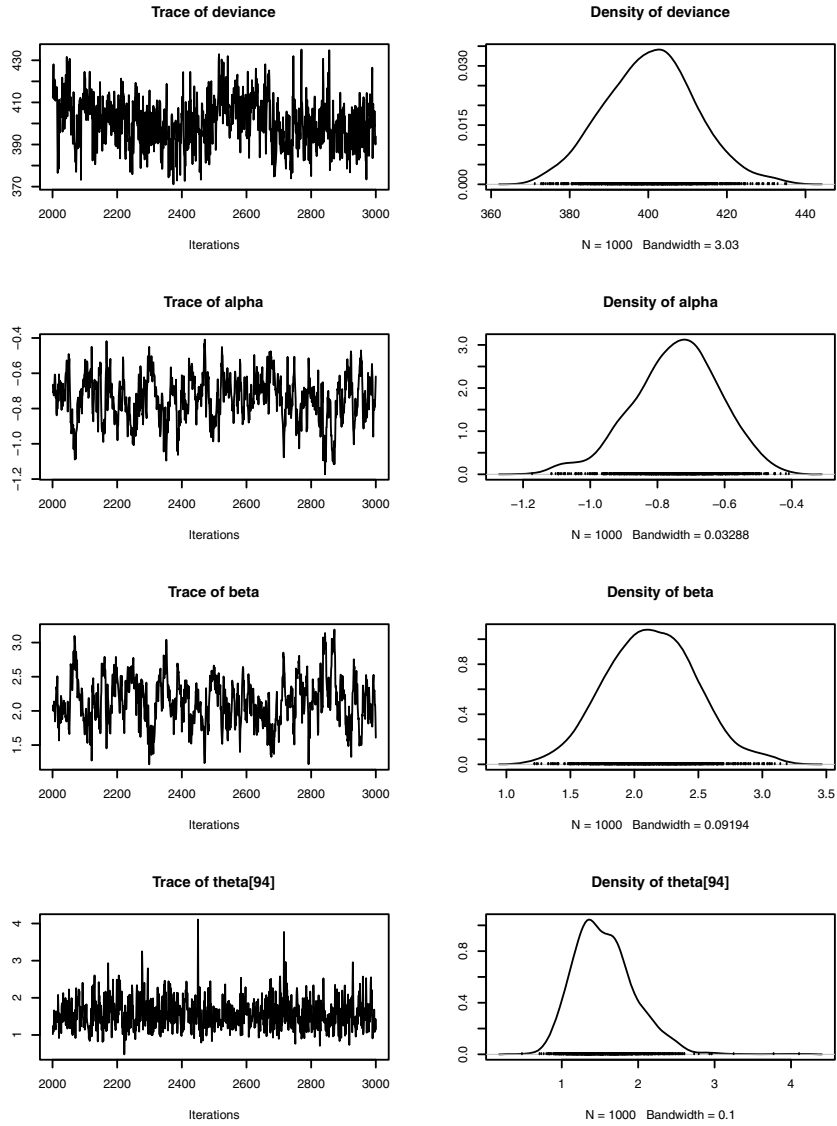


Fig. 10.13 Plots of the posterior distributions of α , β and the deviance of the model

According to the posterior density of β shown in Fig. 10.13, the coefficient of the covariate can be considered as significantly positive given that its posterior mean is greater than 0 and its 95 % credible interval is likely not to contain the value 0. This means that there is an actual risk increase in those regions with a high proportion of non-white births. Point posterior estimates (mean) of the random effects u_i and v_i are shown in Fig. 10.15. They seem to

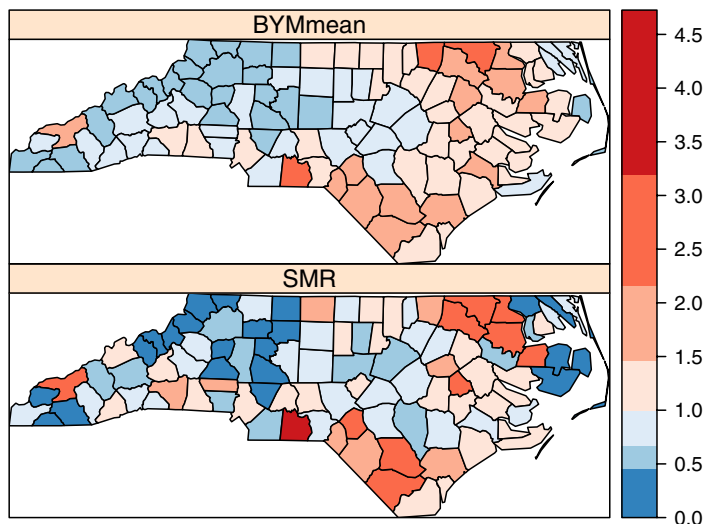


Fig. 10.14 Standardised Mortality Ratio and posterior means of the relative risks obtained with the BYM model

have a very small variation, specially the former, but this is not so because they are in the log-scale.

It should be noted that if the spatial pattern is weak or appropriate covariates are included in the model, the random effects u_i and v_i may become unidentifiable. However, following Besag et al. (1995), valid inference could still be done for the relative risks but care should be taken to avoid having an improper posterior. For this reason, we can monitor $u_i + v_i$ to assess that these values are stable and that they do not have an erratic behaviour that could have an impact on the posterior estimates of the relative risks and the coefficients of the covariates.

The credible intervals of the relative risks have been plotted in Fig. 10.16. The intervals in dashed line show the counties where the relative risk is significantly higher than one. All these regions are among the ones that appear in the two zones of high risk, plus Anson county.

As discussed in Chap. 3, the colours used to produce the maps are based on the palettes developed by Brewer et al. (2003), which are available in package **RColorBrewer**. The research was initiated by Brewer et al. (1997) to produce an atlas of disease in the United States. Brewer and Pickle (2002) study how the variable intervals and colours affect how maps are perceived and Olson and Brewer (1997) developed a useful set of palettes to be used in disease mapping and that are suitable for colour-blind people.

These spatial models can also be fitted with **BayesX**, **INLA** and **CAR-Bayes**. In all cases, we will need a convenient way of converting the **nb** object with the spatial adjacencies into a suitable object. Both **INLA** and **CAR-Bayes** can handle neighbourhood matrices, so we have used function **nb2mat**

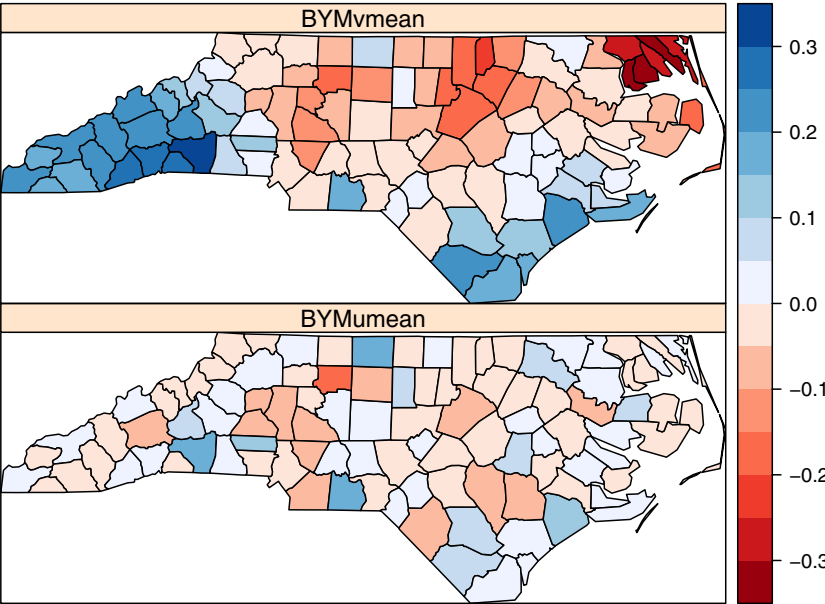


Fig. 10.15 Posterior means of the non-spatial random effects (u_i) and spatial random effects (v_i) estimated with the BYM model

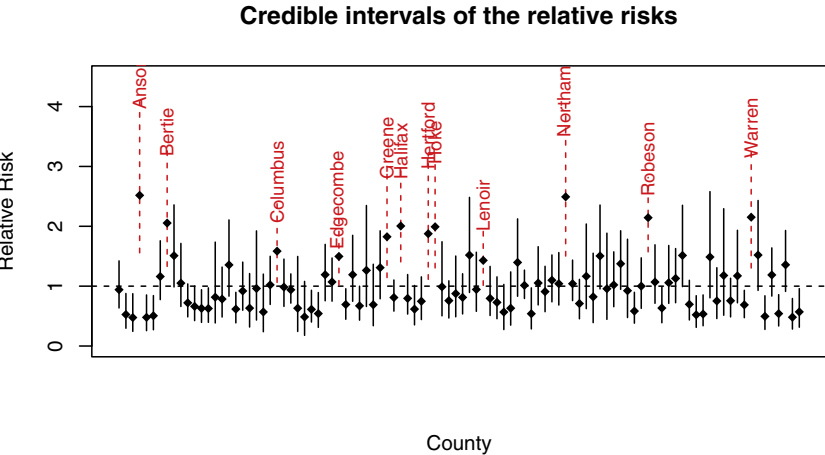


Fig. 10.16 Ninety-five percent credible intervals of the relatives risks obtained with the BYM model

for this. Regarding **BayesX**, function **nb2gra** will convert from a **nb** object into a **gra** object, which is the one required by function **sx** to include spatial adjacency. The following examples show how to fit spatial models with these packages.

With **INLA**, we have used the function `f` twice: `f(FIPS, model="iid")` to include independent Gaussian random effects, and `f(AREAID, model="besag", graph=nb2mat(ncCR85, style="B"))` to include spatial random effects. The first argument refers to an area index used to link each area to a different random effect.

```
> INLA_BYM <- inla(Observed~nwprop + f(FIPS, model = "iid") +
+   f(AREAID, model = "besag", graph = nb2mat(ncCR85,
+     style = "B"))) + offset(log(Expected)), family = "poisson",
+   data = as(nc, "data.frame"), control.predictor = list(compute = TRUE))
```

With **BayesX** we proceeded in a similar way and used function `sx` twice. First, we need to match the neighbourhood object by

```
> ncgra <- nb2gra(ncCR85)
```

Independent random effects were included by `sx(AREAID, bs="re")`, whilst spatial random effects were included by `sx(FIPSNO, bs="spatial", map=ncgra)`. The first argument is an area index used to match each area to a different random effect. Note how functions `f` and `sx` take similar arguments.

```
> bymbayesx <- bayesx(Observed~nwprop + sx(AREAID, bs = "re") +
+   sx(FIPSNO, bs = "spatial", map = ncgra), offset = log(nc$Expected),
+   family = "poisson", data = as(nc, "data.frame"))
```

Finally, **CARBayes** requires the use of function `poisson.bymCAR` to fit this model. Note how the formula only includes the offset and the fixed effects. As stated earlier, **CARBayes** implements different models in different functions, which makes it a less flexible model fitting approach.

```
> ncdf <- as(nc, "data.frame")
> attach(ncdf)
> obj <- poisson.bymCAR(Observed~nwprop + offset(log(Expected)),
+   W = nb2mat(ncCR85, style = "B"), n.sample = 30000,
+   burnin = 20000, thin = 10)
> detach(ncdf)
```

As with the example derived from the Poisson-Gamma model, the posterior means of the relative risks can be obtained from the different data objects returned:

```
> nc$BAYESX <- bymbayesx$fitted.values[order(bymbayesx$bayesx.setup$order),
+   2]/nc$Expected
> nc$INLA <- INLA_BYM$summary.fitted.values[, 1]/nc$Expected
> nc$CARBayes <- obj$fitted.values[, 1]/nc$Expected
```

Posterior means of the relative risks have been displayed in Fig. 10.17. As expected, the point estimates are very similar. It should be noted that some of the default values, including priors for the hyperparameters, may differ across packages. We will not discuss here how different priors can be used in the model, but this information is available in the manual pages of these packages.

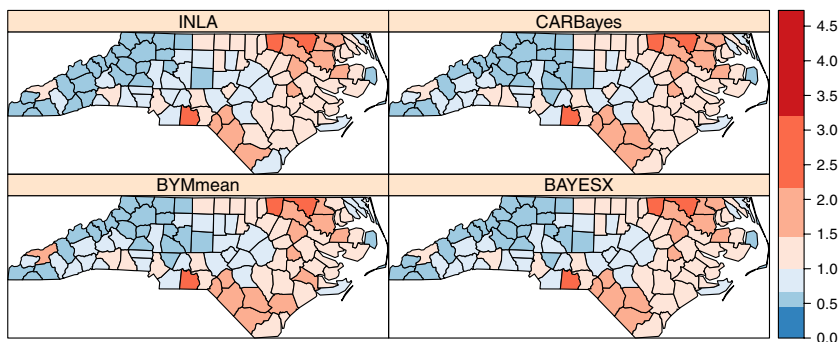


Fig. 10.17 Posterior means of the relative risks obtained with the BYM model using WinBUGS, BayesX, INLA and CARBayes

Furthermore, we have displayed in Fig. 10.18 a summary of results comparing all the four packages discussed in this Section. In particular, we have displayed the posterior marginals of the fixed effects and all packages seem to provide similar estimates. We have also displayed a summary of the posterior means of the spatial and non-spatial random effects. All packages seem to agree on the non-spatial random effects, but we can see some differences in the estimates of the spatial random effects. These differences may be due to different default values in the prior specification.

10.5 Geoadditive Models

BayesX places particular emphasis on the use of geoadditive models and the use of non-linear terms in the linear predictor. As we have already seen, BayesX can handle spatial and independent Gaussian random effects. In addition, **BayesX** allows the specification of different types of smoothing Penalized splines (Ruppert et al., 2003) in the linear predictor by means of function `sx`. P-splines are flexible enough to model non-linear effects and are very popular in Biostatistics (see, for example, Fahrmeir and Kneib, 2011).

In the following example we have fitted two models using P-splines. The first one uses a P-spline (with 10 knots) to fit a non-linear term on a covariate (the proportion of non-white births).

```
> bayesxps <- bayesx(Observed~sx(nwprop, bs = "ps", knots = 10),
+   offset = log(nc$Expected), family = "poisson", data = as(nc,
+   "data.frame"))
```

Secondly, we show how to fit a two dimensional P-spline to model spatial variation using the centroid coordinates of each area. Note that now `sx` takes two arguments.

```
> nc$long <- coordinates(nc)[, 1]
> nc$lat <- coordinates(nc)[, 2]
```

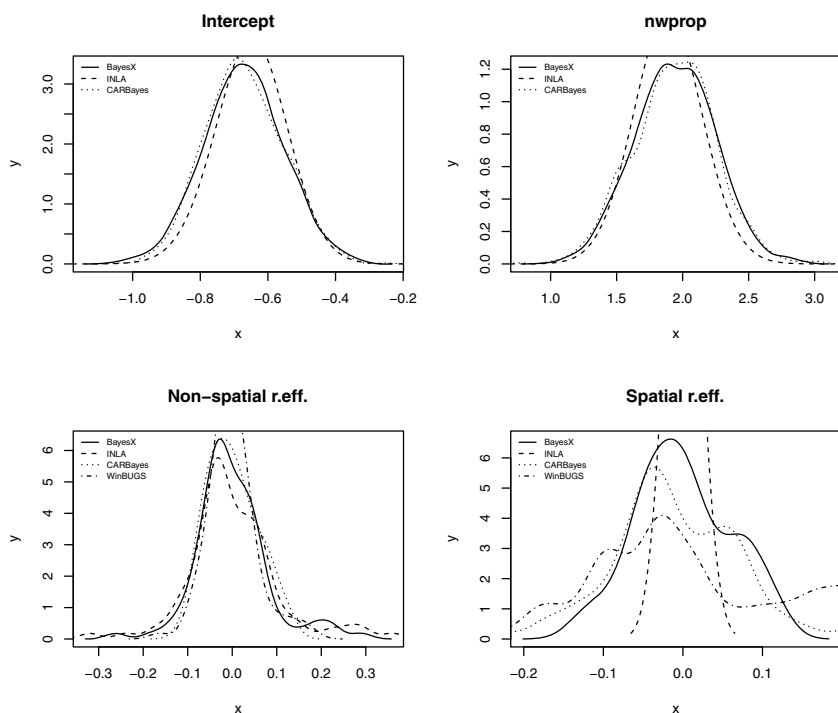



Fig. 10.18 Comparisson of the estimates obtained with different packages of the spatial model

```
> bayesxte <- bayesx(Observed~sx(long, lat, bs = "te"),
+   offset = log(nc$Expected), family = "poisson", data = as(nc,
+   "data.frame"))
```

Figures 10.19 and 10.20 shows the estimated effects for each model. At the top, the non-linear term on the covariate shows that it is probably a good idea to use a simple linear term. At the bottom, we can see the spatial variation in risk modelled by the 2-dimensional P-spline. Note how the areas of high risk (to the south and north-east) are captured by the spatial spline.

Note that, in general, P-splines can be expressed as a mixed-effects model (see, for example, Ruppert et al., 2003, for details). Crainiceanu et al. (2005) describe how to implement P-spline regression with WinBUGS using this representation and a similar implementation could be done with **INLA**. However, developing these approach may be cumbersome and more difficult than by using **BayesX**.

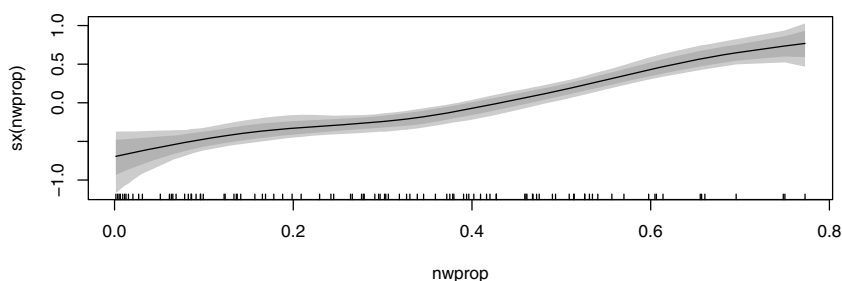


Fig. 10.19 Smoothed effect of a covariate using P-splines with **BayesX**

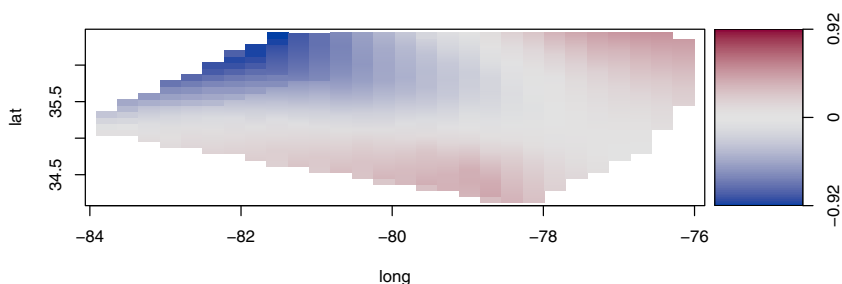


Fig. 10.20 Smoothed effect of a smoothed spatial effect using P-splines with **BayesX**

10.6 Detection of Clusters of Disease

Disease mapping provides a first insight to the spatial distribution of the disease but it may be required to locate the presence of zones where the risk tends to be unusually higher than expected. Besag and Newell (1991) distinguish between methods for clustering and the assessment of risk around putative pollution sources. The former tackle the problem of assessing the presence of clusters, whilst the latter evaluate the risk around a pre-specified source. A third type of methods is related to the location of the clusters themselves, which usually involve the examination of small portions of the whole study area each at a time.

Wakefield et al. (2000) provide a review of some classic methods for the detection of clusters of disease. Haining (2003, pp. 237–264) summarises a good number of well-known methods. Waller and Gotway (2004) also cover in detail most of the methods described in this chapter and many others, providing a discussion on the statistical performance of the tests (pp. 259–263). Lawson et al. (2003, Chap. 7) describe the use of Hierarchical Bayesian models for the analysis of risk around pollution sources.

Some of these methods have been implemented in package **DCluster** (Gómez-Rubio et al., 2005), which uses different models and bootstrap (Davison and Hinkley, 1997) to compute the significance of the observed values.

This can be done in a general way by resampling the observed number of cases in each area and re-computing the value of the test statistic for each of the simulated data sets. Then, a p -value can be computed by ranking the observed value of the test statistic among the values obtained from the simulations.

Under the usual assumption that O_i is drawn from a Poisson with mean $\theta_i E_i$ conditioning on the total number of cases, the distribution of $(O_1, \dots, O_n)$ is Multinomial with probabilities $(E_1/E_+, \dots, E_n/E_+)$. In addition to the multinomial model, **DCluster** offers the possibility of sampling using a non-parametric bootstrap, or from a Poisson (thus, not conditioning on O_+) or Negative Binomial distribution, to account for over-dispersion in the data. As discussed below, over-dispersion may affect the p -value of the test and when data are highly over-dispersed it may be worth re-running the test sampling from a Negative Binomial distribution.

10.6.1 Testing the Homogeneity of the Relative Risks

Before conducting any analysis of the presence of clusters, the heterogeneity of the relative risks must be assessed. In this way, we can test whether there are actual differences among the different relative risks. The reasons for this heterogeneity may be related to many different factors, such as the presence of a pollution source in the area which may lead to an increase in the risk around it. Other times the heterogeneity is due to a spatially varying risk factor, and higher risks are related to a higher exposure to this risk factor.

Given that for each area we have computed its expected and observed number of cases, a chi-square test can be carried out to test for (global) significant differences between these two quantities. The statistic is defined by the following formula:

$$\chi^2 = \sum_{i=1}^n \frac{(O_i - \theta E_i)^2}{\theta E_i}$$

where θ is the global $SMR = \sum_i O_i / \sum_i E_i$ and, asymptotically, it follows a chi-square distribution with n degrees of freedom. If internal standardisation has been used to obtain E_i , then θ is equal to one and the number of degrees of freedom are reduced to $n - 1$ because the additional constraint $\sum_{i=1}^n O_i = \sum_{i=1}^n E_i$ holds (Wakefield et al., 2000).

```
> chctest <- achisq.test(Observed~offset(log(Expected)),
+   as(nc, "data.frame"), "multinom", 999)
> chctest
```

Chi-square test for overdispersion

Type of boots.: parametric

```

Model used when sampling: Multinomial
Number of simulations: 999
Statistic: 225.5723
p-value : 0.001

```

Note that in this case we know that the asymptotic distribution of the test statistic is a chi-square with $n - 1$ degrees of freedom and that an exact test can be done instead of re-sampling (however, it may still be useful for small samples and recall that we may be interested in a Monte Carlo Test using a Negative Binomial).

```

> 1 - pchisq(chtest$t0, 100 - 1)

[1] 7.135514e-12

```

Potthoff and Whittinghill (1966) proposed another test of homogeneity of the means of different Poisson distributed variables which can be used to test the homogeneity of the relative risks (Wakefield et al., 2000). The alternative hypothesis is that the relative risks are drawn from a gamma distribution with mean λ and variance σ^2 :

$$\begin{aligned}
 H_0 : \theta_1 = \dots = \theta_n = \lambda \\
 H_1 : \theta_i \sim Ga(\lambda^2/\sigma^2, \lambda/\sigma^2)
 \end{aligned}$$

The test statistic is given by

$$PW = E_+ \sum \frac{O_i(O_i - 1)}{E_i} \quad (10.1)$$

The alternative hypothesis of this test is that the O_i are distributed following a Negative Binomial distribution, as explained before and, therefore, this test can also be considered as a test of over-dispersion.

```

> pwtest <- pottwhitt.test(Observed~offset(log(Expected)),
+   as(nc, "data.frame"), "multinom", 999)

```

The asymptotic distribution of this statistic is Normal with mean $O_+(O_+ - 1)$ and variance $2nO_+(O_+ - 1)$, so a one-side test can be done as follows:

```

> Oplus <- sum(nc$Observed)
> 1 - pnorm(pwtest$t0, Oplus * (Oplus - 1), sqrt(2 * 100 *
+   Oplus * (Oplus - 1)))

[1] 0

```

Other tests for over-dispersion included in **DCluster** are the likelihood ratio test and some of the score tests proposed in Dean (1992). Although they are not described here, all these tests agree with the previous results obtained before and support the fact that the relative risks are not homogeneous and the observed cases are over-dispersed. Therefore, we have preferred the Negative Binomial to produce the simulations needed to assess the significance

of some of the methods described in the remainder of this section. McMillen (2003) has addressed the importance of choosing the right model in a statistical analysis, and how autocorrelation can appear as a result of a wrong specification of the model.

In addition, Loh and Zhou (2007) discuss the effect of not accounting for extra-Poisson variation and sampling from the wrong distribution when detection of clusters of disease employs the spatial scan statistic (see Sect. 10.6.6 below). Loh and Zhou (2007) propose a correction based on estimating the distribution of the test statistics by sampling from a distribution that accounts for spatial correlation and other factors (for example, covariates). This approach produces more reliable p -values than the original test. Cressie and Read (1989) already mentioned that the Poisson model was not appropriate for the SIDS data due to the presence of over-dispersion and that other models that take it into account would be more appropriate.

In case of doubt, the reader is advised to assess the significance of a given test by using the Multinomial distribution. This is the standard procedure to assess the significance of the test statistic by Monte Carlo in this scenario. See Waller and Gotway (2004, pp. 202–203) for a discussion on this issue.

A first evaluation of the presence of clusters in the study region can be obtained by checking the spatial autocorrelation. Note that using the chi-square test, for example, we can only detect that there are clear differences among the relative risks but not if there is any spatial structure in these differences. In other words, if neighbours tend to have similar (and higher) values. Note that a possible scenario is that of regions having significantly different (low and high) relative risks but with no spatial structure, in which the chi-square test will be significant but there will not be any spatial autocorrelation. This can happen if the scale of aggregation of the data is not taken properly into account or the scale of the risk factors does not exceed the scale of aggregation.

10.6.2 Moran's I Test of Spatial Autocorrelation

We have already discussed the use of Moran's I statistic to assess the presence of spatial autocorrelation. Here we apply Moran's I statistic to the *SMR* to account for the spatial distribution of the population. If we computed Moran's statistic for the O_i we could find spatial autocorrelation only due to the spatial distribution of the underlying population, because it is well known that the higher the population, the higher the number of cases. Binary weights are used depending on whether two regions share a common boundary or not. Spatial autocorrelation is still found even after accounting for over-dispersion.

```
> col.W <- nb2listw(ncCR85, zero.policy = TRUE)
> moranI.test(Observed~offset(log(Expected)), as(nc,
```

```
+ "data.frame"), "negbin", 999, listw = col.W, n = length(ncCR85),
+ S0 = Szero(col.W))
```

Moran's I test of spatial autocorrelation

```
Type of boots.: parametric
Model used when sampling: Negative Binomial
Number of simulations: 999
Statistic: 0.2385172
p-value : 0.001
```

10.6.3 Tango's Test of General Clustering

Tango (1995) proposed a similar test of global clustering by comparing the observed and expected number of cases in each region. He points out that different types of interactions between neighbouring regions can be considered and he proposes a measure of strength based on a decaying function of the distance between two regions. The statistic proposed by Tango is

$$T = (r - p)^T A (r - p) \quad \begin{cases} r^T = [O_1/O_+, \dots, O_n/O_+] \\ p^T = [E_1/E_+, \dots, E_n/E_+] \\ A = (a_{ij}) \text{ closeness matrix} \end{cases} \quad (10.2)$$

where $a_{ij} = \exp\{-d_{ij}/\phi\}$ and d_{ij} is the distance between regions i and j , measured as the distance between their centroids. ϕ is a (positive) constant that reflects the strength of the dependence between areas and the scale at which the interaction occurs.

In our example, we construct the dependence matrix as suggested by Tango and, in addition, we take $\phi = 100$ to simulate a smooth decrease of the relationship between two areas as their relative distance increases. It is advisable to try different values of ϕ because this can have an important impact on the results and the significance of the test. Constructing this matrix in R is straightforward using some functions from package **spdep**, as shown in the following in the code below. In the computations the weights are globally re-scaled, but this does not affect the significance of the test since they all have simply been divided by the same constant. Furthermore, we have taken the approximate location of the county seats from `nc.sids` (columns `x` and `y`), which are in UTM (zone 18) projection. Note that using the centroids as the county seats – as obtained by `coordinates(nc)` – may lead to slightly different coordinates and this may have an impact on the results of this and other tests.

```
> data(nc.sids)
> idx <- match(nc$NAME, rownames(nc.sids))
> nc$x <- nc.sids$x[idx]
> nc$y <- nc.sids$y[idx]
> coords <- cbind(nc$x, nc$y)
```

```

> dlist <- dnearneigh(coords, 0, Inf)
> dlist <- include.self(dlist)
> dlist.d <- nbdists(dlist, coords)
> phi <- 100
> col.W.tango <- nb2listw(dlist, glist = lapply(dlist.d,
+       function(x, phi) {
+         exp(-x/phi)
+       }, phi = phi), style = "C")

```

After computing the adjacency matrix we are ready to compute Tango's test of general clustering, which points out the presence of global clustering:

```

> tango.test(Observed~offset(log(Expected)), as(nc, "data.frame"),
+       "negbin", 999, listw = col.W.tango, zero.policy = TRUE)

```

Tango's test of global clustering

```

Type of boots.: parametric
Model used when sampling: Negative Binomial
Number of simulations: 999
Statistic: 0.000483898
p-value : 0.049

```

10.6.4 Detection of the Location of a Cluster

So far we have considered methods that only assess the presence of heterogeneity of risks in the study area and give a general evaluation of the presence of clusters. In order to detect the actual location of the clusters present in the area a different approach must be followed. A useful family of methods that can help in this purpose are *scan statistics* (Hjalmars et al., 1996). These methods are based on a moving window that only covers a few areas each time and for which a test of clustering is carried out locally. By repeating this procedure throughout the study area it will be possible to detect the locations of clusters of disease.

Scan methods usually differ in the way the window is defined, how it is moved over the area and how the local test of clustering is carried. A recent review of these methods has appeared in *Statistics in Medicine* (Lawson, A., Gangnon, R. E. and Wartenburg, D., editors, 2006). In this section we will only refer to Openshaw's Geographical Analysis Machine (Openshaw et al., 1987) and Kulldorff's statistic (Kulldorff and Nagarwalla, 1995) because the latter is probably the first scan method proposed and the former is a widely established (and used) methodology.

10.6.5 Geographical Analysis Machine

Openshaw's Geographical Analysis Machine considers a regular grid of points $\{(x_i, y_i)\}_{k=1}^p$ over the study region at which a circular window is placed in turn. The test only considers the regions whose centroids are inside the window and it is based on comparing the total number of observed cases in the window (O_{k+}) to the total of expected cases in the window (E_{k+}) to assess if the latter is significantly high. Openshaw et al. (1987) define this test as the (one tailed) p -value of O_{k+} assuming that it follows a Poisson distribution with mean E_{k+} . This procedure can be generalised and, if we have signs that the observed number of cases does not follow a Poisson distribution, the p -value can be obtained by simulation (Gómez-Rubio et al., 2005). Finally, if the current test is significant, the circle is plotted on the map. Alternatively, only the centre of each significant cluster can be plotted for the sake of simplicity and visualisation. Note also that we need to project the cluster centres back to longitude/latitude to be able to plot them on the map of North Carolina.

```
> sidsgam <- opgam(data = as(nc, "data.frame"), radius = 30,
+   step = 10, alpha = 0.002)
> gampoints <- SpatialPoints(sidsgam[, c("x", "y")] * 1000,
+   CRS("+proj=utm +zone=18 +datum=NAD27"))

> library(rgdal)
> ll <- CRS("+proj=longlat +datum=NAD27")
> gampoints <- spTransform(gampoints, ll)
> gam.layout <- list("sp.points", gampoints)
```

When the complete area has been screened, we will probably have found several places where many overlapping clusters have been found, as shown in Fig. 10.21, where the centres of the clusters found have been plotted. This is due to the fact that the tests performed are not independent and, hence, very similar clusters (i.e., most of their regions are the same) are tested. That is the reason why Openshaw's GAM has been highly criticised by the statistical community and why, in order to maintain global significance, the significance level of the local tests should be corrected. Despite this, the GAM is still helpful as an exploratory method and to generate epidemiological hypotheses (Cromley and McLafferty, 2002).

10.6.6 Kulldorff's Statistic

To overcome this and other problems, Kulldorff and Nagarwalla (1995) developed a new test for the detection of clusters based on a window of variable size that only considers the most likely cluster around a given region. Kulldorff's statistic works with the regions within a given circular window

and the overall relative risk in the regions inside the window is compared to that of the regions outside the window. This scan method is available in the SatScan™ software (<http://www.satscan.org/>), which includes enhancements to handle covariates, detect space-time clusters and some other functionalities.

The null hypothesis, of no clustering, is that the two relative risks are equal, while the alternative hypothesis (clustering) is that the relative risk inside the window is higher. This is resolved by means of a likelihood ratio test, which has two main advantages. Firstly, the most likely cluster can be detected as the window with the highest value of the likelihood ratio and, secondly, there is no need to correct the p -value because the simulations for different centres are independent (Waller and Gotway, 2004, p. 220). For a Poisson model, the expression of the test statistic is as follows:

$$\max_{z \in Z_i} \left(\frac{O_z}{E_z} \right)^{O_z} \left(\frac{O_+ - O_z}{E_+ - E_z} \right)^{O_+ - O_z} \quad (10.3)$$

where z is an element of Z_i , the set of all circles centred at region i . These circles are constructed so that only those that contain up to a fixed proportion of the total population are considered.

Note that, even though we select the most likely cluster around each region, it might not be significant. On the other hand, we may have more than one significant cluster, around two or more different regions, and that some clusters may overlap each other. When more than one cluster is found, we can consider the cluster with the lowest p -value as the *primary* or most prominent in the study region. *Secondary* clusters, that do not overlap with the former, may be considered too.

Loh and Zhou (2007) show that when data are over-dispersed the *classical* spatial scan statistic will produce more false positives than the nominal significance level. To correct for this, they propose sampling from a different distribution that accounts for spatial correlation. The Negative Binomial can be used to account for the extra-variability, which may be caused by spatial autocorrelation coming from unmeasured covariates, and estimate the distribution of the test statistic under over-dispersion.

```
> mle <- calculate.mle(as(nc, "data.frame"), model = "negbin")
> thegrid <- as(nc, "data.frame")[, c("x", "y")]
> knresults <- opgam(data = as(nc, "data.frame"),
+   thegrid = thegrid, alpha = 0.05, iscluster = kn.iscluster,
+   fractpop = 0.15, R = 99, model = "negbin",
+   mle = mle)
```

The most likely cluster for the SIDS data set is shown in Fig. 10.21. The p -value is 0.04, which means that the cluster is significant.

The general procedure of application of this method includes testing each area as the centre of a possible cluster, although it can only be used on a single point to test whether it is the centre of a cluster. This is specially helpful, for

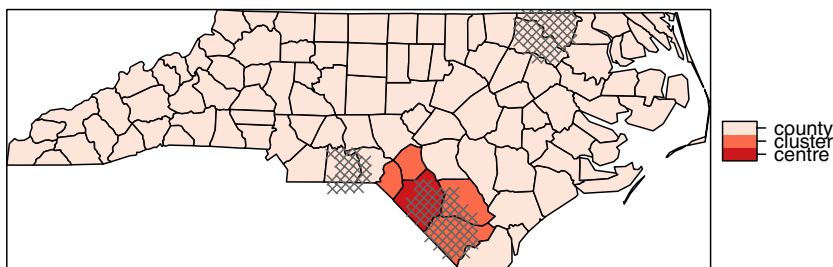


Fig. 10.21 Results of Openshaw's GAM and Kulldorff's test. The crosses show the GAM results, and the colour key the most likely Kulldorff cluster

example, to assess the risk around putative pollution sources. Note that no assumption about the variation of the risk around the source is made. This is discussed in the next section.

10.6.7 Stone's Test for Localised Clusters

As an alternative to the detection of clusters of disease, we may have already identified a putative pollution source and wish to investigate whether there is an increased risk around it. Stone (1988) developed a test that considers the alternative hypothesis of a descending trend around the pollution source. Basically, if we consider $\theta_{(1)}, \dots, \theta_{(n)}$, the ordered relative risks of the regions according to their distances to the source, the test is as follows:

$$\begin{aligned} H_0 : \theta_{(1)} &= \dots = \theta_{(n)} = \lambda \\ H_1 : \theta_{(1)} &\geq \dots \geq \theta_{(n)} \end{aligned}$$

λ is the overall relative risk, which may be one if internal standardisation has been used. The test statistic proposed by Stone is the maximum accumulated risk up to a certain region:

$$\max_i \frac{\sum_{j=1}^i O_j}{\sum_{j=1}^i E_j}$$

A word of caution must be given here because, as already discussed by many authors (Hills and Alexander, 1989, for example), focussed tests should be employed before checking the data, because a bias is introduced when we try to use these tests on regions where an actual increased risk has been observed. In those cases, it will be more likely to detect a cluster than usual.

As an example, we will try to assess whether there is an increased risk around Anson county, which has been spotted as an area of high risk. A call

to `stone.stat` will give us the value of the test statistic and the number of regions for which the maximum accumulated risk is achieved. Later, we can use `stone.test` to compute the significance of this value.

```
> stone.stat(as(nc, "data.frame"), region = which(nc$NAME ==
+           "Anson"))

           region
4.726392 1.000000

> st <- stone.test(Observed~offset(log(Expected)), as(nc,
+           "data.frame"), model = "negbin", 99, region = which(nc$NAME ==
+           "Anson"))
> st
```

Stone's Test for raised incidence around locations

```
Type of boots.: parametric
Model used when sampling: Negative Binomial
Number of simulations: 99
Statistic: 4.726392
p-value : 0.01
```

As the results show, the size of the cluster is 1 (just Anson county) which turns out to be highly significant.

10.7 Spatio-Temporal Disease Mapping

10.7.1 Introduction

So far, we have only considered the case of spatial patterns of disease. However, it is often the case that Public Health data are collected not only over a different set of areas but also over different time periods. If time trends are thought not to be present in the data a purely spatial analysis can be conducted, but considering different time periods in the model may prove important.

As in the spatial case, we will use a Poisson distribution with mean $\mu_{i,t}$ to model the number of occurrences of a disease, $O_{i,t}$, in area i at time t . For each area and time period we can compute an expected number of cases $E_{i,t}$, so that $\mu_{i,t}$ is written down as $\mu_{i,t} = E_{i,t}\theta_{i,t}$, where $\theta_{i,t}$ is the relative risk in area i at time t .

Modelling disease rates in space and time is a complex issue. Knorr-Held (2000) provides a summary of different ways of modelling space-time interaction. Schrödle and Held (2011) make a summary of different models for spatio-temporal disease mapping and discuss an implementation using **INLA**. Some of these models have a large number of parameters and constraints need to

be imposed in order to make all the effects identifiable, similarly as the sum-to-zero-constraint used in the intrinsic CAR specification for spatial random effects.

A simpler approach is to assume that the spatial and temporal effects can be separated. For example, a parametric trend can be considered for the time whilst spatial interaction can be modelled using any of the spatial random effects studied before.

In order to show how to deal with spatio-temporal disease data we will use an example on cases of brain cancer in the state of New Mexico (U.S.A.) in the period 1973–1991. The original data set has been downloaded from the SatScan™ web site (<http://www.satscan.org>) and it has been completed with county boundaries obtained from the U.S. Census Bureau web site (<http://www.census.gov/geo/www/cob/cs2000.html>).

Counts are available at the county level and the expected number of cases have been computed using standardisation by age, race and sex. In addition, the Cibola and Valencia counties has been merged together as data from Cibola county are only available from 1981. Kulldorff et al. (1998) have analysed this data set using a spatio-temporal scan statistic and have reported a cluster around Los Alamos National Laboratory (LANL) in the period 1986–1989. For this reason, we have computed the (inverse) distance to LANL and included it as a covariate in the analysis. Note that, unlike the spatio-temporal scan statistic, including this covariate will assess any increased risk around LANL for all the years in our study period and not a particular time frame.

These data are provided in file `brainNM.RData`, which contains a `STFDF` object named `brainst` with the spatio-temporal data and a `SpatialPolygonsDataFrame` object named `nmf` that we will use to create the spatial adjacency matrix, and a `SpatialPoints` object named `losalamos` holding a single point with the location of LANL. Figures 10.22 and 10.23 shows the *SMR* per county and year and the overall *SMR* per year.

10.7.2 Spatio-Temporal Modelling of Disease

In order to analyse this data we will consider a separable spatio-temporal model. Spatial dependence will be modelled using an intrinsic CAR specification and temporal dependence will be modelled using a first-order random walk to provide a flexible estimation of the temporal trend. Finally, we have included the inverse distance to LANL as a covariate. As this covariate has large values, it has been re-scaled dividing by its mean value.

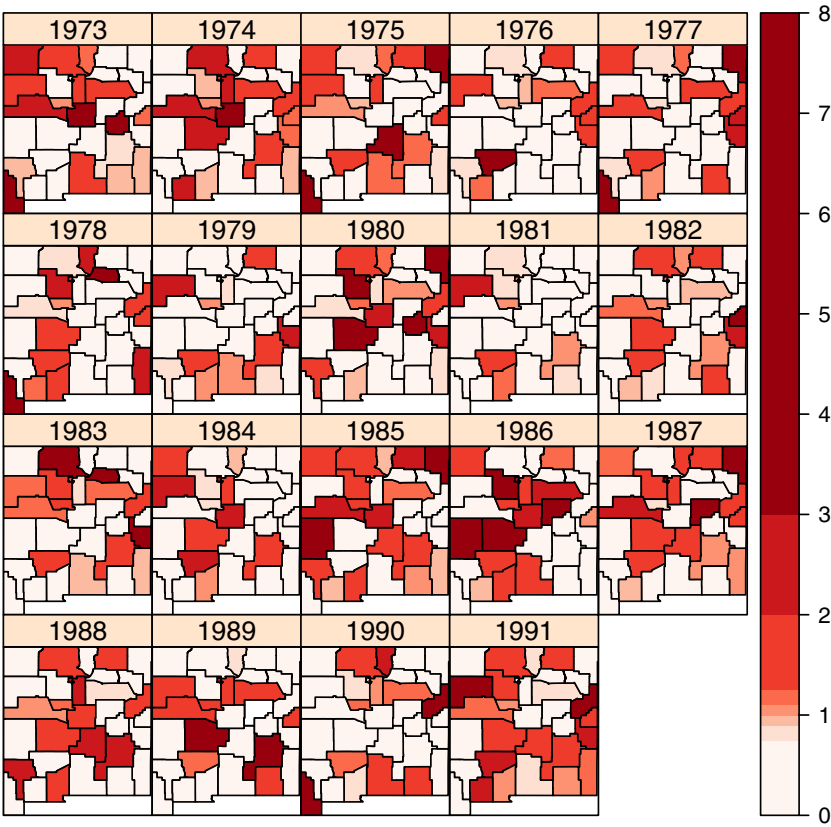


Fig. 10.22 *SMR by county and year (top) and time series plot of yearly SMR (bottom)*

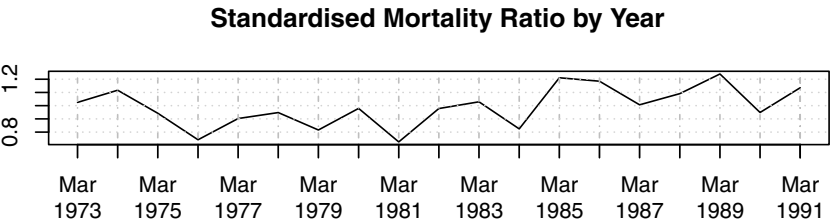


Fig. 10.23 *SMR by county and year (top) and time series plot of yearly SMR (bottom)*

This model can be summarised as follows:

$$\begin{aligned} O_{i,t} &\sim Po(E_{i,t}\theta_{i,t}) \\ \log(\theta_{i,t}) &= \alpha + \beta x_i + v_i + w_t \\ v_i &\sim CAR(\sigma_v^2) \\ w_t &\sim RW(1, \sigma_w^2) \end{aligned}$$

Here x_i represents the values of the covariate, v_i the spatial random effects and w_t the temporal effects. We have not included the prior on the hyperparameters σ_v^2 and σ_w^2 , but this are often assigned inverted Gamma distributions.

We have conducted this analysis using **INLA** because it provides a fast way of fitting these models. First of all, we have created an **nb** object from the **SpatialPolygons** object **nmf** using function **poly2nb**. This will provide the adjacencies that we will use when defining the spatial effect in our model.

```
> library(spdep)
> neib <- poly2nb(nmf, row.names = 1:length(nmf))
```

Next, we have defined the formula with the different effects that we want to include in our model: linear term on the covariate (**IDLANLre**), first-order random walk on the variable **Year** and intrinsic CAR using an adjacency matrix (which is defined using area index **ID**). Note also how we can use **brainst** directly in the call to **inla**. Note that here we have used argument **hyper** when defining the random effects to tune the priors of the variances of the random effects. In particular, we have considered inverted Gammas with parameters 0.001 and 0.001 in both cases to use the same default priors as in **BayesX**.

```
> library(INLA)
> hyper1 <- list(prec = list(param = c(0.001, 0.001)))
> form <- Observed~1 + IDLANLre + f(Year, model = "rw1",
+   hyper = list(prec = list(param = c(0.001, 0.001)))) +
+   f(ID, model = "besag", graph = nb2mat(neib), hyper = hyper1)
> inlares <- inla(form, family = "poisson", data = slot(brainst,
+   "data"), E = Expected, control.predictor = list(compute = TRUE),
+   control.results = list(return.marginals.predictor = TRUE))
```

The same model can easily be fitted with **BayesX** to compare the approximate inference provided by **INLA** to a full MCMC approach. We have set the priors of the variances of the random effects to be the same as we have obtained different results when the default values were used. Note that this means that, in this particular case, the choice of priors may have an impact on the results obtained.

```
> nmgra <- nb2gra(neib)
> nmbayesx <- bayesx(Observed~IDLANLre + sx(Year, bs = "rw1") +
+   sx(ID, bs = "spatial", map = nmgra), offset = log(brainst$Expected),
+   family = "poisson", data = as(brainst, "data.frame"))
```

Figure 10.24 shows the posterior marginals of the coefficients of the fixed effects. It should be noted how both **INLA** and **BayesX** report very similar marginal distributions. The covariate that we have included in the model seems to have no influence on the number of cases. The 95 % credible interval is $(-0.0164, 0.0262)$, which includes zero, meaning that the covariate has no significant effect on the cases of brain cancer.

In order to display the posterior means of the spatial random these have been added to the **nmf** object from the fitted models:

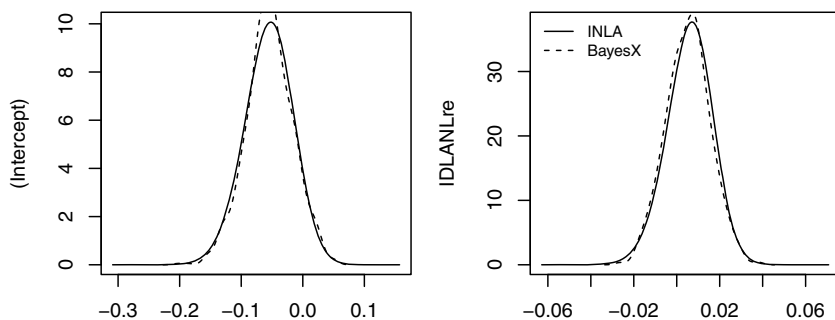


Fig. 10.24 Posterior marginals of the fixed effects computed with **INLA** and **BayesX**

```
> nmf$SPINLA <- inlares$summary.random$ID$mean
> nmf$SPBAYESX <- nmbayesx$effects[["sx(ID)"]] $Mean$ 
```

The temporal and spatial trends have been displayed in Figs. 10.25 and 10.26. Although there seems to be an increasing temporal trend it is not significant considering that all 95 % credible intervals contain zero. However, note how the temporal effect becomes almost significant in the period 1986–1989. This is consistent with the findings reported by Kulldorff et al. (1998). Regarding the spatial variation, we find a high level of agreement between the estimates provided by **INLA** and **BayesX**. However, the posterior means of the spatial random effects are very small, so we conclude that there is no apparent spatial variation in the data.

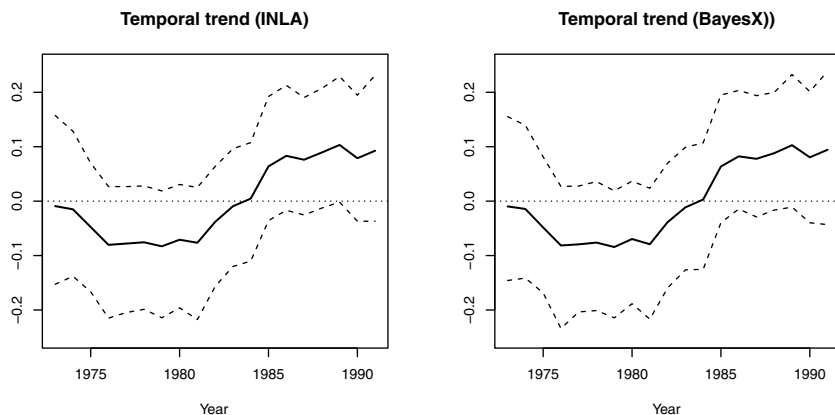


Fig. 10.25 Posterior means of the temporal trend, *dashed lines* mark 95 % credible intervals

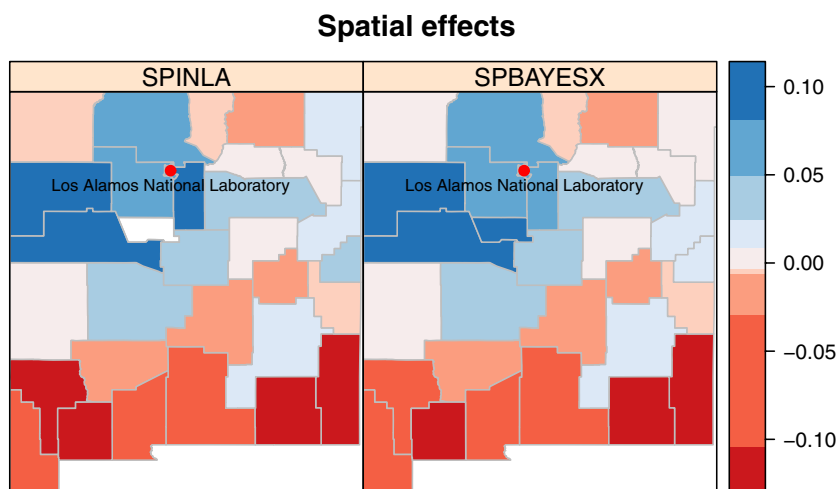


Fig. 10.26 Posterior means of the spatial effects

10.8 Other Topics in Disease Mapping

Although we have tried to cover a wide range of analyses in this chapter, we have not been able to include other important topics, such as the detection of non-circular clusters (see, for example, Tango and Takahashi, 2005) the joint modelling of several diseases (Held et al., 2005) or the disease mapping of rare diseases (Gómez-Rubio and López-Quílez, 2010), which requires the use of zero-inflated models. Gelfand et al. (2010) discuss some of these issues, as well as many other spatial and spatio-temporal models that could be used for disease mapping. Cressie and Wikle (2011) also cover a good number of model that can be used for disease mapping and provide several examples.

Other data sets and models could be used by making the corresponding modifications to the code shown here. Some examples are available in Lawson et al. (2003). Furthermore, Banerjee et al. (2004) describe a number of other possible Bayesian analyses of spatial data and provide data and WinBUGS code in the associated website which the reader should be able to reproduce using the guidelines provided in this chapter.

Afterword

Both parts of this book have quite consciously tried not to give authoritative advice on choices of methods or techniques.¹ The handling and analysis of spatial data with R continues to evolve – this is implicit in open source software development. It is also an important component attempting to offer applied researchers access to accepted and innovative alternatives for data analysis, and applied statisticians with representations of spatial data that make it easier to test and develop new analytical tools.

A further goal has been to provide opportunities for bringing together the various camps and traditions analysing spatial data, to make it somewhat easier to see that their ways of conducting their work are not so different from one another in practise. It has always been worrying that fields like disease mapping or spatial econometrics, with very similar data scenarios, make different choices with regard to methods, and treatments of the assumptions underlying those methods, in their research practice. Research practice evolves, and learning from a broader spread of disciplines must offer the chance to avoid choices that others have found less satisfactory, to follow choices from which others have benefited and to participate in innovation in methods.

This makes participation in the R community, posting questions or suggestions, reporting apparent bugs not only a practical activity, but also an affirmation that science is fostered more by openness than the unwarranted restriction of findings. In the context of this book, and as we said in the preface, we would be grateful for messages pointing out errors; errata will be posted on the book website (<http://www.asdar-book.org>).

¹ An illustration from an email exchange between the authors: “I think we are trying to enable people to do what they want, even if they shoot themselves in the feet (but in a reproducible way)!”

R and Package Versions Used

- R version 3.0.0 (2013-04-03), `x86_64-unknown-linux-gnu`
- Base packages: `base`, `datasets`, `graphics`, `grDevices`, `grid`, `methods`, `splines`, `stats`, `stats4`, `utils`
- Other packages: `akima` 0.5-10, `BayesX` 0.2-6, `BayesXsrc` 2.1-1, `bdsmatrix` 1.3, `bitops` 1.0-5, `boot` 1.3-9, `CARBayes` 1.3, `class` 7.3-7, `classInt` 0.1-19, `coda` 0.16-1, `colorspace` 1.2-1, `cubature` 1.1-2, `DCluster` 0.2-6, `deldir` 0.0-21, `e1071` 1.6-1, `epitools` 0.5-7, `evd` 2.3-0, `fields` 6.7, `foreign` 0.8-53, `Formula` 1.1-0, `geoR` 1.7-4, `geosphere` 1.2-28, `ggplot2` 0.9.3.1, `graph` 1.37.7, `gstat` 1.0-16, `gtools` 2.7.0, `INLA` 0.0, `lattice` 0.20-14, `latticeExtra` 0.6-24, `lmtest` 0.9-30, `locfit` 1.5-8, `maps` 2.3-2, `maptools` 0.8-23, `MASS` 7.3-26, `Matrix` 1.0-11, `MCMCpack` 1.2-4, `McSpatial` 1.1.1, `mgcv` 1.7-22, `nlme` 3.1-109, `osmar` 1.1-5, `pgirmess` 1.5.6, `pixmap` 0.4-11, `pkgDepTools` 1.25.0, `plm` 1.3-1, `quantreg` 4.96, `R2BayesX` 0.1-2, `R2WinBUGS` 2.1-18, `RandomFields` 2.0.66, `RANN` 2.2.1, `raster` 2.1-16, `RBGL` 1.35.0, `RColorBrewer` 1.0-5, `RCurl` 1.95-4.1, `rgdal` 0.8-5, `rgeos` 0.2-13, `sandwich` 2.2-9, `shapefiles` 0.7, `sp` 1.0-6, `spacetime` 1.0-4, `spam` 0.29-2, `SparseM` 0.96, `spatstat` 1.31-1, `spdep` 0.5-56, `spgrass6` 0.7-15, `sphet` 1.2-00, `splancs` 2.01-32, `truncdist` 1.0-1, `XML` 3.96-0.1, `xtable` 1.7-1, `xts` 0.9-3, `zoo` 1.7-9
- Loaded via a namespace (and not attached): `BiocGenerics` 0.5.6, `dichromat` 2.0-0, `digest` 0.6.3, `gtable` 0.1.2, `intervals` 0.14.0, `labeling` 0.1, `LearnBayes` 2.12, `munsell` 0.4, `parallel` 3.0.0, `plyr` 1.8, `proto` 0.3-10, `reshape2` 1.2.2, `scales` 0.2.3, `stringr` 0.6.2, `tools` 3.0.0

Data Sets Used

- Auckland 90 m Shuttle Radar Topography Mission: downloaded on 26 September 2006 from the US Geological Survey, National Map Seamless Server <http://seamless.usgs.gov/>, now <http://earthexplorer.usgs.gov/>, GeoTiff file, 3 arcsec ‘Finished’ (90 m) data; file 70042108.zip on book website.
- Auckland shoreline: downloaded on 7 November 2005 from the National Geophysical Data Center coastline extractor <http://www.ngdc.noaa.gov/mgg/shorelines/shorelines.html>; file `auckland_mapgen.dat` on book website.
- Biological cell centres: available as `data(cells)` from `spatstat`, documented in Ripley (1977).
- Broad Street cholera mortalities: original files provided by Jim Detwiler, who had collated them for David O’Sullivan for use on the cover of O’Sullivan and Unwin (2003), based on earlier work by Waldo Tobler and others; this version is available as a compressed archive of a GRASS

location in file `snow_location.tgz`, and a collection of GeoTiff and shapefiles exported from this location in file `snow_files.zip` on the book website.

- California redwood trees: available as `data(redwoodfull)` from **spatstat**, documented in [Strauss \(1975\)](#).
- Cars: available as `data(cars)` from **datasets**.
- CRAN mirrors: locations of CRAN mirrors 1 October 2005; file on book website `CRAN051001a.txt`.
- Eurasian Collared Dove, *Streptopelia decaocto*, for the years 1986–2003, data used were originally obtained from the North American Breeding Bird Survey, published by [Cressie and Wikle \(2011\)](#); downloaded from the book web site, ftp://ftp.wiley.com/public/sci_tech_med/spatio_temporal_data/.
- Japan shoreline: available in the ‘world’ database provided by **maps**.
- Japanese black pine saplings: available as `data(japanesepines)` from **spatstat**, documented in [Numata \(1961\)](#).
- Lansing Woods maple trees: available as `data(lansing)` from **spatstat**, documented in [Gerard \(1969\)](#).
- Loggerhead turtle: downloaded on 2 November 2005 with permission from SEAMAP, ([Read et al., 2003](#)), data set 105; data described in [Nichols et al. \(2000\)](#); file `seamap105_mod.csv` on book website.
- Manitooulin Island: created using `Rgshhs` in **maptools** from the GSHHS high resolution file `gshhs_h.b`, version 1.5, of 3 April 2007, downloaded from <ftp://ftp.soest.hawaii.edu/pwessel/gshhs>.
- Maunga Whau volcano: available as `data(volcano)` from **datasets**.
- Meuse bank: available as `data(meuse)` from **sp**, supplemented by `data(meuse.grid)` and `data(meuse.riv)`, and documented in [Rikken and Van Rijn \(1993\)](#) and [Burrough and McDonnell \(1998\)](#).
- New Mexico brain cancer: The original data set for 1973–1991 has been downloaded from the SatScan™ web site (<http://www.satscan.org>) and it has been completed with county boundaries obtained from the U.S. Census Bureau web site (<http://www.census.gov/geo/www/cob/cs2000.html>).
- New York leukemia: used and documented extensively in [Waller and Gotway \(2004\)](#) and with data made available in Chap.9 of <http://www.sph.emory.edu/~lwaller/WGindex.htm>; the data import process is described in the help file of `NY_data` in **spdep**; geometries downloaded from the CIESIN server at <ftp://ftp.ciesin.columbia.edu>, file `/pub/census/usa/tiger/ny/bna_st/t8_36.zip`, and extensively edited; a zip archive `NY_data.zip` of shapefiles and a GAL format neighbours list is on the book website.
- North Carolina SIDS: shapefile `sids.shp` (based on geometries downloaded from <http://sal.agecon.uiuc.edu/datasets/sids.zip>; currently at <http://geodacenter.org/downloads/data-files/sids.zip>) and GAL format neighbour lists `ncCC89.gal` and `ncCR85.gal` distributed with **spdep**, data from [Cressie \(1993\)](#), neighbour lists from [Cressie and](#)

Chan (1989) and Cressie and Read (1985), documented in the `nc.sids` help page.

- North Derbyshire asthma study: the data has been studied by Diggle and Rowlingson (1994), Singleton et al. (1995), and Diggle (2003); the data are made available in anonymised form by permission from Peter Diggle as shapefiles in a zip archive `north_derby_asthma.zip` on the book website.
- Olinda 2010 population census, enumeration districts and remotely sensed data: Shapefile, raster files and data modified from downloads from <http://censo2010.ibge.gov.br/en/resultados>, <http://www.ibge.gov.br/home/estatistica/populacao/censo2010>, <http://www.dgi.inpe.br/CDSR/>, <http://earthexplorer.usgs.gov/>, ftp://geofftp.ibge.gov.br/malhas_digitais/censo_2010/setores_censitarios/shape/pe_v1.2.zip and ftp://ftp.ibge.gov.br/Censos/Censo_Demografico_2010/Sinopse/Agregados_por_Setores_Censitarios/Base_informacoes_setores2010_sinopse_PE.zip; the two latter are now: ftp://geofftp.ibge.gov.br/malhas_digitais/censo_2010/setores_censitarios/pe.zip and ftp://ftp.ibge.gov.br/Censos/Censo_Demografico_2010/Resultados_do_Universo/Agregados_por_Setores_Censitarios/Base_informacoes_setores2010_universo_PE.zip. These are stored in a zip archive `Olinda_data.zip` on the book website.
- Product data in `plm`: A panel of 48 observations (one for each US state) from 1970 to 1986; online complement to Baltagi (2001), <http://www.wiley.com/legacy/wileychi/baltagi/>, <http://www.wiley.com/legacy/wileychi/baltagi/supp/PRODUC.prn>
- Scottish lip cancer: Shapefile and data file downloaded from the book website of Waller and Gotway (2004), <http://www.sph.emory.edu/~lwaller/WGindex.htm>, Chaps. 2 and 9.
- Spearfish: downloaded as GRASS location from http://grass.itc.it/sampleddata/spearfish_grass60data-0.3.tar.gz, now http://grass.osgeo.org/sampleddata/spearfish_grass60data.tar.gz; this data set has been the standard GRASS location for tutorials and is documented in Neteler and Mitasova (2004).
- US 1999 SAT scores: state boundaries available in the ‘state’ database provided by `maps`, original attribute data downloaded on 2 November 2005 from <http://www.biostat.umn.edu/~melanie/Data/> and supplemented with variable names and state names; the data set is also available from the website of Banerjee et al. (2004), <http://www.biostat.umn.edu/~brad/data/state-sat.dat>, and the modified version as file `state.sat.data_mod.txt` from the book website.
- World volcano locations: downloaded from the National Geophysical Data Center <http://www.ngdc.noaa.gov/hazard/volcano.shtml>, available as file `data1964a1.xy` from book website.

References

- Abrahamsen, P. and Benth, F. E. (2001). Kriging with inequality constraints. *Mathematical Geology*, 33:719–744. [253]
- Adler, D., Kneib, T., Lang, S., Umlauf, N., and Zeileis, A. (2012). *BayesXsrc: R Package Distribution of the BayesX C++ Sources*. R package version 2.1-1. [331]
- Adler, J. (2010). *R in a Nutshell*. O'Reilly, Sebastopol, CA. [25]
- Akima, H. (1978). A method of bivariate interpolation and smooth surface fitting for irregularly distributed data points. *ACM Transactions on Mathematical Software*, 4:148–159. [258]
- Altman, M. and McDonald, M. P. (2011). BARD: Better automated redistricting. *Journal of Statistical Software*, 42(4):1–28. [128]
- Andrade Neto, P. R. and Ribeiro Jr., P. J. (2005). A process and environment for embedding the R software into TerraLib. In *VII Brazilian Symposium on Geoinformatics, Campos do Jordão*. [123]
- Anselin, L. (1988). *Spatial Econometrics: Methods and Models*. Kluwer, Dordrecht. [303, 304]
- Anselin, L. (2002). Under the hood: Issues in the specification and interpretation of spatial regression models. *Agricultural Economics*, 27:247–267. [303, 304]
- Anselin, L., Bera, A. K., Florax, R., and Yoon, M. J. (1996). Simple diagnostic tests for spatial dependence. *Regional Science and Urban Economics*, 26:77–104. [304]
- Anselin, L. and Lozano-Gracia, N. (2008). Errors in variables and spatial effects in hedonic house price models of ambient air quality. *Empirical Economics*, 34:5–34. [308]
- Anselin, L., Syabri, I., and Kho, Y. (2006). GeoDa: An introduction to spatial data analysis. *Geographical Analysis*, 38:5–22. [273]
- Aquino, J. (2003). JTS topology suite developer's guide, version 1.4. Technical report, Vivid Solutions Inc. [131]

- Arraiz, I., Drukker, D. M., Kelejian, H. H., and Prucha, I. R. (2010). A spatial Cliff-Ord-type model with heteroskedastic innovations: small and large sample results. *Journal of Regional Science*, 50:592–614. [311]
- Assunção, R. and Reis, E. A. (1999). A new proposal to adjust Moran's I for population density. *Statistics in Medicine*, 18:2147–2162. [282]
- Baddeley, A., Gregori, P., Mateu, J., Stoica, R., and Stoyan, D., editors (2005). *Case Studies in Spatial Point Process Modeling*. Lecture Notes in Statistics, 185. Springer-Verlag, Berlin. [210]
- Baddeley, A., Möller, J., and Waagepetersen, R. (2000). Non- and semi-parametric estimation of interaction in inhomogeneous point patterns. *Statistica Neerlandica*, 54:329–350. [192, 206, 208]
- Baddeley, A. and Turner, R. (2005). Spatstat: an R package for analyzing spatial point patterns. *Journal of Statistical Software*, 12(6):1–42. [174]
- Baddeley, A. J. and Silverman, B. W. (1984). A cautionary example on the use of second-order methods for analysing point patterns. *Biometrics*, 40: 1089–1093. [205]
- Bailey, T. C. and Gatrell, A. C. (1995). *Interactive Spatial Data Analysis*. Longman, Harlow. [14]
- Baltagi, B. (2001). *Econometric Analysis of Panel Data*, 3rd edition. John Wiley & Sons, New York. [164, 366]
- Banerjee, S., Carlin, B. P., and Gelfand, A. E. (2004). *Hierarchical Modeling and Analysis for Spatial Data*. Chapman & Hall/CRC, Boca Raton/London. [7, 13, 14, 266, 276, 290, 314, 319, 322, 330, 336, 361, 366]
- Bastin, L., Cornford, D., Jones, R., Heuvelink, G. B., Pebesma, E., Stasch, C., Nativi, S., Mazzetti, P., and Williams, M. (2013). Managing uncertainty in integrated environmental modelling: The uncertweb framework. *Environmental Modelling and Software*, 39:116–134. [6]
- Bavaud, F. (1998). Models for spatial weights: a systematic look. *Geographical Analysis*, 30:153–171. [269]
- Beale, C. M., Lennon, J. J., Elston, D. A., Brewer, M. J., and Yearsley, J. M. (2007). Red herrings remain in geographical ecology: a reply to Hawkins et al. (2007). *Ecography*, 30:845–847. [12]
- Becker, R. A., Chambers, J. M., and Wilks, A. R. (1988). *The New S Language*. Chapman & Hall, London. [2, 37]
- Belitz, C., Brezger, A., Kneib, T., Lang, S., and Umlauf, N. (2012). *BayesX: Software for Bayesian Inference in Structured Additive Regression Models. Version 2.1*. [331]
- Berman, M. and Diggle, P. J. (1989). Estimating weighted integrals of the second-order intensity of a spatial point process. *Journal of the Royal Statistical Society B*, 51:81–92. [184, 185]
- Bernardinelli, L. and Montomoli, C. (1992). Empirical Bayes versus fully Bayesian analysis of geographical variation in disease risk. *Statistics in Medicine*, 11:983–1007. [329]
- Besag, J., Green, P., Higdon, D., and Mengersen, K. (1995). Bayesian computation and stochastic systems. *Statistical Science*, 10:3–41. [342]

- Besag, J. and Newell, J. (1991). The detection of clusters in rare diseases. *Journal of the Royal Statistical Society, Series A*, 154:143–155. [347]
- Besag, J., York, J., and Mollie, A. (1991). Bayesian image restoration, with two applications in spatial statistics. *Annals of the Institute of Statistical Mathematics*, 43:1–59. [330, 337]
- Best, N., Cowles, M. K., and Vines, K. (1995). CODA: Convergence diagnosis and output analysis software for Gibbs sampling output, Version 0.30. Technical report, MRC Biostatistics Unit, Cambridge. [309, 333]
- Best, N. G., Waller, L. A., Thomas, A., Conlon, E. M., and Arnold, R. A. (1999). Bayesian models for spatially correlated diseases and exposure data. In Bernardo, J., Berger, J. O., Dawid, A. P., and Smith, A. F. M., editors, *Bayesian Statistics 6*, pages 131–156. Oxford University Press, Oxford. [338]
- Bivand, R. S. (2000). Using the R statistical data analysis language on GRASS 5.0 GIS data base files. *Computers and Geosciences*, 26:1043–1052. [112]
- Bivand, R. S. (2002). Spatial econometrics functions in R: Classes and methods. *Journal of Geographical Systems*, 4:405–421. [169, 303]
- Bivand, R. S. (2006). Implementing spatial data analysis software tools in R. *Geographical Analysis*, 38:23–40. [303]
- Bivand, R. S. (2008). Implementing representations of space in economic geography. *Journal of Regional Science*, 48:1–27. [13, 276]
- Bivand, R. S. (2010). Exploratory spatial data analysis. In Fischer, M. and Getis, A., editors, *Handbook of Applied Spatial Analysis*, pages 219–254. Springer, Heidelberg. pp. 36. [284]
- Bivand, R. S., Hauke, J., and Kossowski, T. (2013). Computing the Jacobian in Gaussian spatial autoregressive models: an illustrated comparison of available methods. *Geographical Analysis*, 45:150–179. [300, 301]
- Bivand, R. S., Müller, W., and Reder, M. (2009). Power calculations for global and local Moran's I . *Computational Statistics and Data Analysis*, 53:2859–2872. [281]
- Bivand, R. S. and Portnov, B. A. (2004). Exploring spatial data analysis techniques using R: the case of observations with no neighbours. In Anselin, L., Florax, R. J. G. M., and Rey, S. J., editors, *Advances in Spatial Econometrics: Methodology, Tools, Applications*, pages 121–142. Springer, Berlin. [272]
- Bivand, R. S. and Szymanski, S. (1997). Spatial dependence through local yardstick competition: theory and testing. *Economics Letters*, 55:257–265. [13]
- Borcard, D., Gillet, F., and Legendre, P. (2011). *Numerical Ecology with R*. Springer, New York. [283, 317]
- Bordignon, M., Cerniglia, F., and Revelli, F. (2003). In search of yardstick competition: a spatial analysis of Italian municipality property tax setting. *Journal of Urban Economics*, 54:199–217. [13]

- Brenning, A. (2009). Benchmarking classifiers to optimally integrate terrain analysis and multispectral remote sensing in automatic rock glacier detection. *Remote Sensing of Environment*, 113(1):239–247. [124]
- Brewer, C. A., Hatchard, G. W., and Harrower, M. A. (2003). ColorBrewer in print: a catalog of color schemes for maps. *Cartography and Geographic Information Science*, 30:5–32. [79, 342]
- Brewer, C. A., MacEachren, A. M., Pickle, L. W., and Herrmann, D. J. (1997). Mapping mortality: Evaluating color schemes for choropleth maps. *Annals of the Association of American Geographers*, 87:411–438. [342]
- Brewer, C. A. and Pickle, L. (2002). Comparison of methods for classifying epidemiological data on choropleth maps in series. *Annals of the Association of American Geographers*, 92:662–681. [342]
- Brezger, A., Kneib, T., and Lang, S. (2005). Bayesx: Analyzing bayesian structural additive regression models. *Journal of Statistical Software*, 14(11):1–22. [331]
- Brody, H., Rip, M. R., Vinten-Johansen, P., Paneth, N., and Rachman, S. (2000). Map-making and myth-making in Broad Street: the London cholera epidemic, 1854. *Lancet*, 356:64–68. [118, 119]
- Burrough, P. A. and McDonnell, R. A. (1998). *Principles of Geographical Information Systems*. Oxford University Press, Oxford. [5, 6, 130, 213, 365]
- Calenge, C. (2006). The package adehabitat for the R software: a tool for the analysis of space and habitat use by animals. *Ecological Modelling*, 197:516–519. [122]
- Carrera-Hernández, J. and Gaskin, S. (2008). The basin of mexico hydrogeological database (BMHDB): Implementation, queries and interaction with open source software. *Environmental Modelling & Software*, 23(10–11):1271–1279. [118]
- Carstairs, V. (2000). Socio-economic factors at areal level and their relationship with health. In Elliot, P., Wakefield, J., Best, N., and Briggs, D., editors, *Spatial Epidemiology: Methods and Applications*, pages 51–67. Oxford University Press, Oxford. [320]
- Chambers, J. M. (1998). *Programming with Data*. Springer, New York. [3, 27]
- Chambers, J. M. (2008). *Software for Data Analysis: Programming with R*. Springer, New York. [27]
- Chambers, J. M. and Hastie, T. J. (1992). *Statistical Models in SS*. Chapman & Hall, London. [24, 25, 26]
- Chilès, J. and Delfiner, P. (2012). *Geostatistics: Modeling Spatial Uncertainty* (second edition). Wiley, New York. [213]
- Choynowski, M. (1959). Map based on probabilities. *Journal of the American Statistical Society*, 54:385–388. [325]
- Chrisman, N. (2002). *Exploring Geographic Information Systems*. Wiley, New York. [6, 8]

- Christensen, R. (1991). *Linear Models for Multivariate, Time Series, and Spatial Data*. Springer, New York. [213]
- Chun, Y. and Griffith, D. A. (2013). *Spatial Statistics & Geostatistics*. Sage, Thousand Oaks, CA. [266, 276, 284]
- Clark, A. B. and Lawson, A. B. (2004). An evaluation of non-parametric relative risk estimators for disease mapping. *Computational Statistics and Data Analysis*, 47:63–78. [185]
- Clayton, D. and Kaldor, J. (1987). Empirical Bayes estimates of age-standardized relative risks for use in disease mapping. *Biometrics*, 43:671–681. [93, 324, 325, 328]
- Cleveland, W. S. (1993). *Visualizing Data*. Hobart Press, Summit, NJ. [59, 69, 214]
- Cleveland, W. S. (1994). *The Elements of Graphing Data*. Hobart Press, Summit, NJ. [59, 69]
- Cliff, A. D. and Ord, J. K. (1973). *Spatial Autocorrelation*. Pion, London. [274]
- Cliff, A. D. and Ord, J. K. (1981). *Spatial Processes*. Pion, London. [13, 271]
- Cowles, M. K. and Carlin, B. P. (1996). Markov Chain Monte Carlo convergence diagnostics: A comparative review. *Journal of the American Statistical Association*, 91:883–904. [340]
- Cox, C. R. (1955). Some statistical methods connected with series of events (with discussion). *Journal of the Royal Statistical Society, Series B*, 17:129–164. [208]
- Crainiceanu, C. M., Ruppert, D., and Wand, M. P. (2005). Bayesian analysis for penalized spline regression using winbugs. *Journal of Statistical Software*, 14(14):1–24. [346]
- Cressie, N. (1985). Fitting variogram models by weighted least squares. *Mathematical Geology*, 17:563–586. [225]
- Cressie, N. (1993). *Statistics for Spatial Data, Revised Edition*. John Wiley & Sons, New York. [13, 16, 170, 213, 221, 266, 276, 290, 295, 365]
- Cressie, N. and Chan, N. H. (1989). Spatial modeling of regional variables. *Journal of the American Statistical Association*, 84:393–401. [320, 322, 329, 336, 365]
- Cressie, N. and Read, T. R. C. (1985). Do sudden infant deaths come in clusters? *Statistics and Decisions*, 3:333–349. [320, 321, 337, 338, 366]
- Cressie, N. and Read, T. R. C. (1989). Spatial data analysis of regional counts. *Biometrical Journal*, 31:699–719. [350]
- Cressie, N. and Wikle, C. (2011). *Statistics for Spatio-temporal Data*. John Wiley & Sons, New York. [ix, 14, 156, 174, 193, 361, 365]
- Croissant, Y. and Millo, G. (2008). Panel data econometrics in R: The plm package. *Journal of Statistical Software*, 27:1–43. [313]
- Cromley, E. K. and McLafferty, S. L. (2002). *GIS and Public Health*. Guilford Press, New York. [353]
- Dalgaard, P. (2008). *Introductory Statistics with R, (second edition)*. Springer, New York. [25, 170]

- Davis, M. and Aquino, J. (2003). JTS topology suite technical specifications, version 1.4. Technical report, Vivid Solutions Inc. [131]
- Davison, A. C. and Hinkley, D. V. (1997). *Bootstrap Methods and Their Application*. Cambridge University Press, Cambridge. [347]
- Dean, C. B. (1992). Testing for overdispersion in Poisson and Binomial regression models. *Journal of the American Statistical Association*, 87:451–457. [349]
- Deutsch, C. and Journel, A. (1992). *GSLIB: Geostatistical Software Library and User's Guide*. Oxford University Press, New York. [213]
- Devine, O. J. and Louis, T. A. (1994). A constrained empirical Bayes estimator for incidence rates in areas with small populations. *Statistics in Medicine*, 13:1119–1133. [329]
- Devine, O. J., Louis, T. A., and Halloran, M. E. (1994). Empirical Bayes estimators for spatially correlated incidence rate. *Environmetrics*, 5:381–398. [329]
- Dietrich, C. and Newsam, G. (1993). A fast and exact method for multi-dimensional gaussian stochastic simulations. *Water Resources Research*, 29:2861–2869. [252]
- Diggle, P. J. (1985). A kernel method for smoothing point process data. *Applied Statistics*, 34:138–147. [184, 185]
- Diggle, P. J. (1990). A point process modelling approach to raised incidence of a rare phenomenon in the vicinity of a prespecified point. *Journal of the Royal Statistical Society, Series A*, 153:349–362. [193, 202]
- Diggle, P. J. (2000). Overview of statistical methods for disease mapping and its relationship to cluster detection. In Elliott, P., Wakefield, J., Best, N., and Briggs, D., editors, *Spatial Epidemiology: Methods and Applications*, pages 87–103. Oxford University Press, Oxford. [193, 204]
- Diggle, P. J. (2003). *Statistical Analysis of Spatial Point Patterns*. Arnold, London, second edition. [173, 175, 176, 180, 182, 183, 185, 188, 189, 191, 192, 204, 210, 366]
- Diggle, P. J. (2006). Spatio-temporal point processes: Methods and applications. In Finkenstadt, B., Held, L., and Isham, V., editors, *Statistical Methods for Spatio-Temporal Systems*, pages 1–46. CRC Press, Boca Raton. [210]
- Diggle, P. J. and Chetwynd, A. (1991). Second-order analysis of spatial clustering for inhomogeneous populations. *Biometrics*, 47:1155–1163. [193, 204, 205]
- Diggle, P. J., Elliott, P., Morris, S., and Shaddick, G. (1997). Regression modelling of disease risk in relation to point sources. *Journal of the Royal Statistical Society, Series A*, 160:491–505. [204]
- Diggle, P. J., Gómez-Rubio, V., Brown, P. E., Chetwynd, A., and Gooding, S. (2007). Second-order analysis of inhomogeneous spatial point processes using case-control data. *Biometrics*, 63:550–557. [193, 195, 206, 207, 208]
- Diggle, P. J., Morris, S., and Wakefield, J. (2000). Point-source modelling using case-control data. *Biostatistics*, 1:89–105. [194]

- Diggle, P. J. and Ribeiro Jr., P. J. (2007). *Model-Based Geostatistics*. Springer, New York. [259]
- Diggle, P. J. and Rowlingson, B. (1994). A conditional approach to point process modelling of elevated risk. *Journal of the Royal Statistical Society, Series A*, 157:433–440. [176, 178, 197, 202, 203, 204, 366]
- Diggle, P. J., Tawn, J. A., and Moyeed, R. A. (1998). Model-based geostatistics. *Applied Statistics*, pages 299–350. [256]
- Diniz-Filho, J. A., Bini, L. M., and Hawkins, B. A. (2003). Spatial autocorrelation and red herrings in geographical ecology. *Global Ecology & Biogeography*, 12:53–64. [12]
- Diniz-Filho, J. A., Hawkins, B. A., Bini, L. M., De Marco Jr., P., and Blackburn, T. M. (2007). Are spatial regression methods a panacea or a Pandora’s box? a reply to Beale et al. (2007). *Ecography*, 30:848–851. [12]
- Dormann, C., McPherson, J., Araújo, M., Bivand, R., Bolliger, J., Carl, G., Davies, R., Hirzel, A., Jetz, W., Kissling, D., Kühn, I., Ohlemüller, R., Peres-Neto, P., Reineking, B., Schröder, B., Schurr, F., and Wilson, R. (2007). Methods to account for spatial autocorrelation in the analysis of species distributional data: a review. *Ecography*, 30:609–628. [290, 314, 317]
- Dray, S., Legendre, P., and Peres-Neto, P. R. (2006). Spatial modeling: a comprehensive framework for principle coordinate analysis of neighbor matrices (PCNM). *Ecological Modelling*, 196:483–493. [317]
- Dray, S., Pélissier, R., Couteron, P., Fortin, M., Legendre, P., Peres-Neto, P. R., Bellier, E., Bivand, R., Blanchet, F. G., De Cáceres, M., Dufour, A., Heegaard, E., Jombart, T., Munoz, F., Oksanen, J., Thioulouse, J., and Wagner, H. H. (2012). Community ecology in the age of multivariate multiscale spatial analysis. *Ecological Monographs*, 82:257–275. [12, 264, 317]
- Elhorst, J. P. (2010). Applied spatial econometrics: Raising the bar. *Spatial Economic Analysis*, 5:9–28. [308]
- Elliott, P., Wakefield, J., Best, N., and Briggs, D., editors (2000). *Spatial Epidemiology. Methods and Applications*. Oxford University Press, Oxford. [192, 319]
- Elliott, P. and Wakefield, J. C. (2000). Bias and confounding in spatial epidemiology. In Elliott, P., Wakefield, J., Best, N., and Briggs, D., editors, *Spatial Epidemiology: Methods and Applications*, pages 68–84. Oxford University Press, Oxford. [320]
- English, D. (1992). Geographical epidemiology and ecological studies. In Elliott, P., Cuzick, J., English, D., and Stern, R., editors, *Geographical and Environmental Epidemiology. Methods for Small-Area Studies*, pages 3–13. Oxford University Press, Oxford. [337]
- Erle, S., Gibson, R., and Walsh, J. (2005). *Mapping Hacks*. O’Reilly, Sebastopol, CA. [7]
- Fahrmeir, L. and Kneib, T. (2011). *Bayesian Smoothing and Regression for Longitudinal, Spatial and Event History Data*. Oxford University Press, New York. [345]

- Faraway, J. J. (2004). *Linear Models with R*. Chapman & Hall/CRC, Boca Raton. [170]
- Faraway, J. J. (2006). *Extending Linear Models with R: Generalized Linear, Mixed Effects and Nonparametric Regression Models*. Chapman & Hall/CRC, Boca Raton. [170]
- Finkenstadt, B., Held, L., and Isham, V. (2006). *Statistical Methods for Spatio-Temporal Systems*. CRC Press, Boca Raton. [14]
- Folmer, E. O., Olf, H., and Piersma, T. (2012). The spatial distribution of flocking foragers: disentangling the effects of food availability, interference and conspecific attraction by means of spatial autoregressive modeling. *Oikos*, 121:551–561. [308]
- Fortin, M.-J. and Dale, M. (2005). *Spatial Analysis: A Guide for Ecologists*. Cambridge University Press, Cambridge. [14, 266, 276, 284, 290]
- Fotheringham, A. S., Brunson, C., and Charlton, M. E. (2002). *Geographically Weighted Regression: The Analysis of Spatially Varying Relationships*. Wiley, Chichester. [317]
- Fox, J. and Weisberg, S. (2011). *An R Companion to Applied Regression*. Sage Publications, Thousand Oaks, CA, USA, second edition. [170]
- Gabriel, E. and Diggle, P. J. (2009). Second-order analysis of inhomogeneous spatio-temporal point process data. *Statistica Neerlandica*, 63(1):43–51. [210]
- Gaetan, C. and Guyon, X. (2010). *Spatial Statistics and Modeling*. Springer, New York. [14, 174]
- Galton, A. (2004). Fields and objects in space, time, and space-time. *Spatial Cognition and Computation*, 4:39–68. [9, 165]
- Gatrell, A. C., Bailey, T. C., Diggle, P. J., and Rowlingson, B. S. (1996). Spatial point pattern analysis and its application in geographical epidemiology. *Transactions of the Institute of British Geographers*, 21:256–274. [192]
- Gelfand, A. E., Diggle, P. J., Guttorp, P., and Fuentes, M., editors (2010). *Handbook of Spatial Statistics*. Chapman & Hall/CRC Press. [14, 174, 193, 210, 361]
- Gelman, A., Carlin, J. B., Stern, H. S., and Rubin, D. B. (2003). *Bayesian Data Analysis*. CRC Press, Boca Raton. [331]
- Gelman, A. and Hill, J. (2007). *Data Analysis Using Regression and Multi-level/Hierarchical Models*. Cambridge University Press, Cambridge. [331]
- Gelman, A. and Rubin, D. B. (1992). Inference from iterative simulation using multiple sequences (with discussion). *Statistical Science*, 7:457–472. [340]
- Gerard, D. J. (1969). Competition quotient: a new measure of the competition affecting individual forest trees. Research Bulletin 20, Agricultural Experiment Station, Michigan State University. [189, 365]
- Geweke, J. (1992). Evaluating the accuracy of sampling-based approaches to calculating posterior moments. In Bernardo, J. M., Berger, J. O., Dawid, A. P., and Smith, A. F. M., editors, *Bayesian Statistics 4*, pages 169–194. Oxford University Press, Oxford. [340]

- Ghosh, M., Natarajan, K., Stroud, T. W. F., and Carlin, B. P. (1998). Generalized linear models for small-area estimation. *Journal of the American Statistical Association*, 93:273–282. [337]
- Gilks, W. R., Richardson, S., and Spiegelhalter, D. J., editors (1996). *Markov Chain Monte Carlo in Practice*. Chapman & Hall, London. [331]
- Goetz, J., Guthrie, R., and Brenning, A. (2011). Integrating physical and empirical landslide susceptibility models using generalized additive models. *Geomorphology*, 129(3–4):376–386. [124]
- Gómez-Rubio, V., Ferrándiz-Ferragud, J., and López-Quílez, A. (2005). Detecting clusters of disease with R. *Journal of Geographical Systems*, 7:189–206. [347, 353]
- Gómez-Rubio, V. and López-Quílez, A. (2005). RArcInfo: Using GIS data with R. *Computers and Geosciences*, 31:1000–1006. [91, 99]
- Gómez-Rubio, V. and López-Quílez, A. (2010). Statistical methods for the geographical analysis of rare diseases. *Advances in experimental medicine and biology*, 686:151–171. [361]
- Goodchild, M. F. (1992). Geographical information science. *International journal of geographical information systems*, 6(1):31–45. [9]
- Goovaerts, P. (1997). *Geostatistics for Natural Resources Evaluation*. Oxford University Press, Oxford. [213, 242, 252]
- Gotway, C. A. and Young, L. J. (2002). Combining incompatible spatial data. *Journal of the American Statistical Association*, 97:632–648. [128]
- GRASS Development Team (2012). *Geographic Resources Analysis Support System (GRASS GIS) Software*. Open Source Geospatial Foundation, USA. [91]
- Griffith, D. A. (1995). Some guidelines for specifying the geographic weights matrix contained in spatial statistical models. In Arlinghaus, S. L. and Griffith, D. A., editors, *Practical Handbook of Spatial Statistics*, pages 65–82. CRC Press, Boca Raton. [269]
- Griffith, D. A. and Peres-Neto, P. R. (2006). Spatial modeling in ecology: the flexibility of eigenfunction spatial analyses. *Ecology*, 87:2603–2613. [317]
- Grohmann, C. and Steiner, S. (2008). Srtm resample with short distance-low nugget kriging. *International Journal of Geographical Information Science*, 22(8):895–906. [118]
- Güting, R. H. and Schneider, M. (2005). *Moving Objects Databases*. Morgan Kaufmann. [153]
- Guttorp, P. (2003). Environmental statistics — a personal view. *International Statistical Review*, 71:169–180. [128]
- Haining, R. P. (2003). *Spatial Data Analysis: Theory and Practice*. Cambridge University Press, Cambridge. [14, 169, 295, 319, 322, 330, 347]
- Hall, G. B. and Leahy, M., editors (2008). *Open Source Approaches in Spatial Data Handling*. Springer-Verlag, Berlin. [83]
- Härdle, W., Müller, M., Sperlich, S., and Werwatz, A. (2004). *Nonparametric and Semiparametric Models*. Springer-Verlag, Berlin. [187]

- Hastie, T. and Tibshirani, R. (1990). *Generalised Additive Models*. Chapman & Hall, London. [314]
- Hawkins, B. A., Diniz-Filho, J. A., Bini, L. M., De Marco Jr., P., and Blackburn, T. M. (2007). Red herrings revisited: spatial autocorrelation and parameter estimation in geographical ecology. *Ecography*, 30:375–384. [12]
- Haywood, A. and Stone, C. (2011). Mapping eucalypt forest susceptible to dieback associated with bell miners (*Manorina melanophrys*) using laser scanning, Spot 5 and ancillary topographical data. *Ecological Modelling*, 222(5):1174–1184. [118]
- Held, L., Natário, I., Fento, S. E., Rue, H., and Becke, N. (2005). Towards joint disease mapping. *Statistical Methods in Medical Research*, 14:61–82. [361]
- Hengl, T., Bajat, B., Blagojević, D., and Reuter, H. (2008). Geostatistical modeling of topography using auxiliary maps. *Computers & Geosciences*, 34(12):1886–1899. [124]
- Hengl, T., Heuvelink, G., and van Loon, E. (2010). On the uncertainty of stream networks derived from elevation data: the error propagation approach. *Hydrology and Earth System Sciences*, 14(7):1153–1165. [124]
- Hepple, L. W. (1998). Exact testing for spatial autocorrelation among regression residuals. *Environment and Planning A*, 30:85–108. [281]
- Herring, J. R. (2011). OpenGIS® implementation standard for geographic information - simple feature access - part 1: Common architecture. Technical Report 1.2.1, OGC 06-103r4, Open Geospatial Consortium Inc. [30, 31, 38, 43, 48, 131]
- Heuvelink, G. B. M. (1998). *Error Propagation in Environmental Models with GIS*. Taylor & Francis, London. [129]
- Heywood, I., Cornelius, S., and Carver, S. (2006). *An Introduction to Geographical Information Systems*. Pearson Education, Harlow, England. [7]
- Hijmans, R. J. (2012a). Introduction to the “geosphere” package. Technical report, **geosphere** vignette. [130]
- Hijmans, R. J. (2012b). Introduction to the ‘raster’ package. Technical report, **raster** vignette. [54, 130]
- Hijmans, R. J. (2012c). Writing functions with the “raster” package. Technical report, **raster** vignette. [54, 55]
- Hills, M. and Alexander, F. (1989). Statistical methods used in assessing the risk of disease near a source of possible environmental pollution: a review. *Journal of the Royal Statistical Society, Series A*, 152:353–363. [355]
- Hjalmars, U., Kulldorff, M., Gustafsson, G., and Nagarwalla, N. (1996). Childhood leukaemia in Sweden: using GIS and a spatial scan statistic for cluster detection. *Statistics in Medicine*, 15:707–715. [352]
- Hjaltason, G. and Samet, H. (1995). Ranking in spatial databases. In Egenhofer, M. J. and Herring, J. R., editors, *Advances in Spatial Databases - 4th Symposium, SSD’95*, number 951 in Lecture Notes in Computer Science, pages 83–95. Springer-Verlag, Berlin. [238]

- Hoef, J. M. V. and Cressie, N. A. C. (1993). Multivariable spatial prediction. *Mathematical Geology*, 25:219–240. [233]
- Hothorn, T., Müller, J., Schröder, B., Kneib, T., and Brandl, R. (2011). Decomposing environmental, spatial, and spatiotemporal components of species distributions. *Ecological Monographs*, 81:329–347. [314]
- Hovmöller, E. (1949). The Trough-and-Ridge diagram. *Tellus*, 1(2):62–66. [162]
- Illian, J., Penttinen, A., Stoyan, H., and Stoyan, D. (2008). *Statistical Analysis and Modelling of Spatial Point Patterns*. Wiley, New York. [174, 193]
- Isaaks, E. and Strivastava, R. (1989). *An Introduction to Applied Geostatistics*. Oxford University Press, Oxford. [213]
- Jackson, C., Best, N., and Richardson, S. (2006). Improving ecological inference using individual-level data. *Statistics in Medicine*, 25(12):2136–2159. [338]
- Jarner, M. F., Diggle, P., and Chetwynd, A. G. (2002). Estimation of spatial variation in risk using matched case-control data. *Biometrical Journal*, 44:936–945. [193, 194]
- Johnston, J. and DiNardo, J. (1997). *Econometric Methods*. McGraw Hill, New York. [304]
- Jolma, A., Ames, D. P., Horning, N., Mitasova, H., Neteler, M., Racicot, A., and Sutton, T. (2012). Open-source tools for environmental modeling. In Kresse, W. and Danko, D. M., editors, *Springer Handbook of Geographic Information*, pages 967–984. Springer, Berlin Heidelberg. [112]
- Journel, A. G. and Huijbregts, C. J. (1978). *Mining Geostatistics*. Academic Press, London. [213, 238]
- Kaluzny, S. P., Vega, S. C., Cardoso, T. P., and Shelly, A. A. (1998). *S+SpatialStats, User Manual for Windows and UNIX*. Springer-Verlag, Berlin. [14, 319]
- Kelejian, H. H., Murrell, P., and Shepotylo, O. (2013). Spatial spillovers in the development of institutions. *Journal of Development Economics*, 101:297–315. [308]
- Kelejian, H. H. and Prucha, I. R. (2007). HAC estimation in a spatial framework. *Journal of Econometrics*, 140:131–154. [311]
- Kelejian, H. H. and Prucha, I. R. (2010). Specification and estimation of spatial autoregressive models with autoregressive and heteroskedastic disturbances. *Journal of Econometrics*, 157:53–67. [311]
- Kelejian, H. H., Tavlas, G., and Hondroyannis, G. (2006). A spatial modelling approach to contagion among emerging economies. *Open Economies Review*, 17:423–441. [308]
- Kelsall, J. E. and Diggle, P. J. (1995a). Kernel estimation of relative risk. *Bernoulli*, 1:3–16. [185, 193, 194, 196]
- Kelsall, J. E. and Diggle, P. J. (1995b). Non-parametric estimation of spatial variation in relative risk. *Statistics in Medicine*, 14:559–573. [185, 193, 194, 196]

- Kelsall, J. E. and Diggle, P. J. (1998). Spatial variation in risk: a non-parametric binary regression approach. *Applied Statistics*, 47:559–573. [185, 193, 198, 199, 200]
- Kneib, T., Hothorn, T., and Tutz, G. (2009). Variable selection and model choice in geoadditive regression models. *Biometrics*, 65(2):626–634. [314]
- Knorr-Held, L. (2000). Bayesian modelling of inseparable space-time variation in disease risk. *Statistics in Medicine*, 19(17–18):2555–2567. [356]
- Kopczewska, K. (2006). *Ekometria i Statystyka Przestrzenna*. CeDeWu, Warszawa. [xii]
- Kresse, W., Danko, D. M., and Fadaie, K. (2012). Standardization. In Kresse, W. and Danko, D. M., editors, *Springer Handbook of Geographic Information*, pages 393–565. Springer, Berlin Heidelberg. [9, 22, 30, 31, 38, 43, 48, 131]
- Krieger, N., Williams, D. R., and Moss, N. E. (1997). Measuring social class in US Public Health research: Concepts, methodologies, and guidelines. *Annual Review of Public Health*, 18:341–378. [338]
- Krivoruchko, K. (2011). *Spatial Statistical Data Analysis for GIS Users*. ESRI Press, Redlands, CA. DVD. [7, 124, 174]
- Kulldorff, M., Athas, W. F., Feuer, E. J., Miller, B. A., and Key, C. R. (1998). Evaluating cluster alarms: A space-time scan statistic and brain cancer in los alamos, new mexico. *American Journal of Public Health*, 88(9):1377–1380. [357, 360]
- Kulldorff, M. and Nagarwalla, N. (1995). Spatial disease clusters: Detection and inference. *Statistics in Medicine*, 14:799–810. [352, 353]
- Laurent, T., Ruiz-Gazen, A., and Thomas-Agnan, C. (2012). Geoxp: An R package for exploratory spatial data analysis. *Journal of Statistical Software*, 47(2):1–23. [123]
- Lawson, A. B., Browne, W. J., and Rodeiro, C. L. V. (2003). *Disease Mapping with WinBUGS and MLwiN*. Wiley, Chichester. [319, 322, 330, 331, 347, 361]
- Lawson, A., editor (2005). SMMR special issue on disease mapping. *Statistical Methods in Medical Research*, 14(1). [319]
- Lawson, A., Gangnon, R. E. and Wartenburg, D., editors (2006). Special issue: Developments in disease cluster detection. *Statistics in Medicine*, 25(5). [319, 352]
- Leisch, F. (2002). Sweave: Dynamic generation of statistical reports using literate data analysis. In Härdle, W. and Rönz, B., editors, *Compstat 2002 — Proceedings in Computational Statistics*, pages 575–580, Heidelberg. Physica Verlag. [xi]
- Leisch, F. and Rossini, A. J. (2003). Reproducible statistical research. *Chance*, 16(2):46–50. [xi]
- Lennon, J. J. (2000). Red-shifts and red herrings in geographical ecology. *Ecography*, 23:101–113. [12]

- LeSage, J. and Fischer, M. (2008). Spatial growth regression: Model specification, estimation and interpretation. *Spatial Economic Analysis*, 3:275–304. [308]
- LeSage, J. and Pace, R. (2009). *Introduction to Spatial Econometrics*. CRC Press, Boca Raton, FL. [303, 308, 309, 311]
- Leung, Y., Ma, J.-H., and Goodchild, M. F. (2004). A general framework for error analysis in measurement-based GIS Part 1: The basic measurement-error model and related concepts. *Journal of Geographical Systems*, 6:325–354. [129]
- Leutenegger, S. T., Edgington, J. M., and Lopez, M. A. (1997). STR: A simple and efficient algorithm for R-tree packing. In Gray, W. A. and Larson, P.-Å., editors, *Proceedings of the Thirteenth International Conference on Data Engineering*, pages 497–506. IEEE Computer Society. [138]
- Li, H., Calder, C. A., and Cressie, N. (2007). Beyond Moran's I: testing for spatial dependence based on the spatial autoregressive model. *Geographical Analysis*, 39:357–375. [292]
- Li, H., Calder, C. A., and Cressie, N. (2012). One-step estimation of spatial dependence parameters: Properties and extensions of the APLE statistic. *Journal of Multivariate Analysis*, 105:68–84. [292]
- Lloyd, C. D. (2007). *Local Models for Spatial Analysis*. CRC Press, Boca Raton. [284, 318]
- Loh, J. M. and Zhou, Z. (2007). Accounting for spatial correlation in the scan statistic. *The Annals of Applied Statistics*, 1:560–584. [350, 354]
- Longley, P. A., Goodchild, M. F., Maguire, D. J., and Rhind, D. W. (2005). *Geographic Information Systems and Science*. Wiley, Chichester. [7, 9]
- Louis, T. A. (1984). Estimating a population of parameter values using Bayes and empirical Bayes methods. *Journal of the American Statistical Society*, 79:393–398. [329]
- Marshall, R. J. (1991). Mapping disease and mortality rates using Empirical Bayes estimators. *Applied Statistics*, 40:283–294. [326, 328]
- Mateu, J. and Müller, W. G. (2013). *Spatio-temporal design: advances in efficient data acquisition*. Wiley, New York. [256]
- McMillen, D. P. (2003). Spatial autocorrelation or model misspecification? *International Regional Science Review*, 26:208–217. [350]
- McMillen, D. P. (2012). Perspectives on spatial econometrics: Linear smoothing with structured models. *Journal of Regional Science*, 52:192–209. [313]
- McMillen, D. P. (2013). *Quantile Regression for Spatial Data*. Springer, Heidelberg. [313]
- Millo, G. and Piras, G. (2012). splm: Spatial panel data models in R. *Journal of Statistical Software*, 47(1):1–38. [313]
- Mitchell, T. (2005). *Web Mapping Illustrated: Using Open Source GIS Toolkits*. O'Reilly, Sebastopol, CA. [7, 83]
- Möller, J. and Waagepetersen, R. (2003). *Statistical Inference and Simulation for Spatial Point Processes*. Chapman & Hall/CRC, Boca Raton. [173, 182, 183, 191, 210]

- Murrell, P. (2011). *R Graphics*. Chapman & Hall/CRC, Boca Raton. [37, 59]
- Murrell, P. (2012). It's Not What You Draw, It's What You Don't Draw. *The R Journal*, 4(2):13–18. [48]
- Neteler, M., Beaudette, D., Cavallini, P., Lami, L., and Cepicky, J. (2008). GRASS GIS. In Hall, G. B. and Leahy, M., editors, *Open Source Approaches in Spatial Data Handling*, pages 171–199. Springer-Verlag, Berlin. [112]
- Neteler, M., Bowman, M. H., Landa, M., and Metz, M. (2012). GRASS GIS: A multi-purpose open source GIS. *Environmental Modelling & Software*, 31(1):124–130. [112]
- Neteler, M. and Mitasova, H. (2004). *Open Source GIS: A GRASS GIS Approach, Second Edition*. Kluwer, Boston. [112, 366]
- Neteler, M. and Mitasova, H. (2008). *Open Source GIS: A GRASS GIS Approach, Third Edition*. Springer, New York. [7, 112, 136]
- Nichols, W., Resendiz, A., J.A.Seminoff, and Resendiz, B. (2000). Transpacific migration of a loggerhead turtle monitored by satellite telemetry. *Bulletin of Marine Science*, 67:937–947. [37, 365]
- Numata, M. (1961). Forest vegetation in the vicinity of Choshi. Coastal flora and vegetation at Choshi, Chiba Prefecture IV. *Bulletin of Choshi Marine Laboratory, Chiba University*, 3:28–48 [in Japanese]. [175, 177, 365]
- Nüst, D., Stasch, C., and Pebesma, E. (2011). Connecting R to the sensor web. In Geertman, S., Reinhardt, W., and Toppen, F., editors, *Advancing Geoinformation Science for a Changing World*, volume 1 of *Lecture Notes in Geoinformation and Cartography*, Berlin, Germany. Springer. [6]
- Olson, J. M. and Brewer, C. A. (1997). An evaluation of color selections to accommodate map users with color-vision impairments. *Annals of the Association of American Geographers*, 87:103–134. [342]
- Openshaw, S., Charlton, M., Wymer, C., and Craft, A. W. (1987). A Mark I geographical analysis machine for the automated analysis of point data sets. *International Journal of Geographical Information Systems*, 1:335–358. [352, 353]
- Ord, J. K. (1975). Estimation methods for models of spatial interaction. *Journal of the American Statistical Association*, 70:120–126. [301]
- O'Sullivan, D. and Unwin, D. J. (2003). *Geographical Information Analysis*. Wiley, Hoboken, NJ. [118, 364]
- O'Sullivan, D. and Unwin, D. J. (2010). *Geographical Information Analysis*. Wiley, Hoboken, NJ. [14, 130, 174, 179, 193, 266, 271, 276, 284, 318]
- Páez, A., Farber, S., and Wheeler, D. C. (2011). A simulation-based study of geographically weighted regression as a method for investigating spatially varying relationships. *Environment and Planning A*, 43:2992–3010. [318]
- Pebesma, E. (2012a). Map overlay and spatial aggregation in **sp**. Technical report, **sp** vignette. [141, 161]
- Pebesma, E. (2012b). spacetime: Spatio-temporal data in R. *Journal of Statistical Software*, 51(7):1–30. [151, 164]
- Pebesma, E., Cornford, D., Dubois, G., Heuvelink, G. B., Hristopulos, D., Pilz, J., Stöhlker, U., Morin, G., and Skøien, J. O. (2011). INTAMAP: The

- design and implementation of an interoperable automated interpolation web service. *Computers and Geosciences*, 37(3):343–352. [6]
- Pebesma, E., Nüst, D., and Bivand, R. (2012). The R software environment in reproducible geoscientific research. *Eos Trans. AGU*, 93(16). [5]
- Pebesma, E. J. (2004). Multivariable geostatistics in S: the gstat package. *Computers and Geosciences*, 30:683–691. [233]
- Pebesma, E. J. and Bivand, R. S. (2005). Classes and methods for spatial data in R. *R News*, 5(2):9–13. [3]
- Pinheiro, J. C. and Bates, D. M. (2000). *Mixed-Effects Models in S and S-Plus*. Springer, New York. [317]
- Piras, G. (2010). sphet: Spatial models with heteroskedastic innovations in R. *Journal of Statistical Software*, 35:1–21. [311]
- Potthoff, R. F. and Whittinghill, M. (1966). Testing for homogeneity: II. The Poisson distribution. *Biometrika*, 53:183–190. [349]
- Prince, M. I., Chetwynd, A., Diggle, P. J., Jarner, M., Metcalf, J. V., and James, O. F. (2001). The geographical distribution of primary biliary cirrhosis in a well-defined cohort. *Hepatology*, 34:1083–1088. [193]
- R Core Team (2013). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. [vii, 2]
- R Development Core Team (2008). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. [xi]
- Read, A. J., Halpin, P. N., Crowder, L. B., Hyrenbach, K. D., Best, B. D., and Freeman, S. A. (2003). *OBIS-SEAMAP: mapping marine mammals, birds and turtles*. Duke University. World Wide Web electronic publication. <http://seamap.env.duke.edu>, Accessed on April 01, 2008. [37, 365]
- Revelli, F. (2003). Reaction or interaction? Spatial process identification in multi-tiered government structures. *Journal of Urban Economics*, 53:29–53. [13]
- Revelli, F. and Tovmo, P. (2007). Revealed yardstick competition: local government efficiency patterns in Norway. *Journal of Urban Economics*, 62:121–134. [13]
- Rikken, M. G. J. and Van Rijn, R. P. G. (1993). Soil pollution with heavy metals - an inquiry into spatial variation, cost of mapping and the risk evaluation of copper, cadmium, lead and zinc in the floodplains of the meuse west of stein. Technical report, Dept. of Physical Geography, Utrecht University. [365]
- Ripley, B. D. (1976). The second order analysis of stationary point processes. *Journal of Applied Probability*, 13:255–266. [191]
- Ripley, B. D. (1977). Modelling spatial patterns (with discussion). *Journal of the Royal Statistical Society, Series B*, 39:172–212. [175, 177, 191, 364]
- Ripley, B. D. (1981). *Spatial Statistics*. Wiley, New York. [13, 146]
- Ripley, B. D. (1988). *Statistical Inference for Spatial Processes*. Cambridge University Press, Cambridge. [13]
- Ripley, B. D. (2001). Spatial statistics in R. *R News*, 1(2):14–15. [14]

- Roberts, J. J., Best, B. D., Dunn, D. C., Trembl, E. A., and Halpin, P. N. (2010). Marine Geospatial Ecology Tools: An integrated framework for ecological geoprocessing with ArcGIS, Python, R, MATLAB, and C++. *Environmental Modelling & Software*, 25(10):1197–1207. [125]
- Rowlingson, B. and Diggle, P. J. (1993). Splancs: spatial point pattern analysis code in S-PLUS™. *Computers and Geosciences*, 19:627–655. [174]
- Rue, H., Martino, S., and Chopin, N. (2009). Approximate Bayesian inference for latent Gaussian models by using integrated nested Laplace approximations. *Journal of the Royal Statistical Society, Series B*, 71(Part 2):319–392. [ix, 331]
- Ruppert, D., Wand, M. P., and Carroll, R. J. (2003). *Semiparametric Regression*. Cambridge University Press, Cambridge, UK. [345, 346]
- Sarkar, D. (2008). *Lattice: Multivariate Data Visualization with R*. Springer, New York. [59, 70]
- Schabenberger, O. and Gotway, C. A. (2005). *Statistical Methods for Spatial Data Analysis*. Chapman & Hall/CRC, Boca Raton/London. [13, 128, 151, 174, 179, 182, 183, 191, 193, 210, 266, 275, 277, 284, 290, 299, 314, 317, 318, 319, 330]
- Schrödle, B. and Held, L. (2011). Spatio-temporal disease mapping using INLA. *Environmetrics*, 22(6):725–734. [356]
- Shekar, S. and Xiong, H., editors (2008). *Encyclopedia of GIS*. Springer, New York. [7]
- Sibson, R. (1981). A brief description of natural neighbor interpolation. In Barnett, V., editor, *Interpreting Multivariate Data*, pages 21–36. John Wiley & Sons, Chichester. [258]
- Silverman, B. W. (1986). *Density Estimation for Statistics and Data Analysis*. Chapman & Hall, London. [184, 185]
- Singleton, C. D., Gatrell, A. C., and Briggs, J. (1995). Prevalence of asthma and related factors in primary school children in an industrial part of England. *Journal of Epidemiology and Community Health*, 49:326–327. [176, 366]
- Slocum, T. A., McMaster, R. B., Kessler, F. C., and Howard, H. H. (2005). *Thematic Cartography and Geographical Visualization*. Pearson Prentice Hall, Upper Saddle River, NJ. [29, 59, 80]
- Spiegelhalter, D., Thomas, A., Best, N., and Lunn, D. (2003). *WinBUGS Version 1.4 User's Manual*. MRC Biostatistics Unit, Cambridge. <http://www.mrc-bsu.cam.ac.uk/bugs>. [319, 331]
- Stein, M. (1999). *Interpolation of Spatial Data: Some Theory for Kriging*. Springer, New York. [221]
- Stineman, R. (1980). A consistently well behaved method of interpolation. *Creative Computing*, 6:54–57. [258]
- Stone, R. A. (1988). Investigating of excess environmental risks around putative sources: Statistical problems and a proposed test. *Statistics in Medicine*, 7:649–660. [355]

- Strauss, D. J. (1975). A model for clustering. *Biometrika*, 62:467–475. [175, 177, 365]
- Sturtz, S., Ligges, U., and Gelman, A. (2005). R2WinBUGS: A package for running WinBUGS from R. *Journal of Statistical Software*, 12(3):1–16. [331]
- SzaLay, A. S. and Blakeley, J. A. (2009). Gray’s laws: Database-centric computing in science. In Hey, T., Tansley, S., and Tolle, K., editors, *The Fourth Paradigm: Data-Intensive Scientific Discovery*, pages 5–11. Microsoft, Redmond. [165]
- Tait, N., Durr, P. A., and Zheng, P. (2004). Linking R and ArcGIS: Developing a spatial statistical toolkit for epidemiologists. In *Proceedings of GISVET’04, Guelph, Canada*. [124]
- Tango, T. (1995). A class of tests for detecting general and focused clustering of rare diseases. *Statistics in Medicine*, 14:2323–2334. [351]
- Tango, T. and Takahashi, K. (2005). A flexibly shaped spatial scan statistic for detecting clusters. *International Journal of Health Geographics*, 4:1–15. [361]
- Teator, P. (2011). *R Cookbook*. O’Reilly, Sebastopol, CA. [23, 25]
- Theus, M. (2002). Interactive data visualization using Mondrian. *Journal of Statistical Software*, 7(11):1–9. [123]
- Theus, M. and Urbanek, S. (2009). *Interactive graphics for data analysis: principles and examples*. CRC Press, Boca Raton, FL. [123]
- Tiefelsdorf, M. (1998). Some practical applications of Moran’s I ’s exact conditional distribution. *Papers in Regional Science*, 77:101–129. [281]
- Tiefelsdorf, M. (2000). *Modelling Spatial Processes: The Identification and Analysis of Spatial Relationships in Regression Residuals by Means of Moran’s I*. Springer, Berlin. [281]
- Tiefelsdorf, M. (2002). The saddlepoint approximation of Moran’s I and local Moran’s I_i reference distributions and their numerical evaluation. *Geographical Analysis*, 34:187–206. [281]
- Tiefelsdorf, M. and Griffith, D. A. (2007). Semiparametric filtering of spatial autocorrelation: The eigenvector approach. *Environment and Planning A*, 39:1193–1221. [317]
- Tiefelsdorf, M., Griffith, D. A., and Boots, B. (1999). A variance-stabilizing coding scheme for spatial link matrices. *Environment and Planning A*, 31:165–180. [269]
- Tukey, J. W. (1977). *Exploratory Data Analysis*. Addison-Wesley, Reading, MA. [169]
- Unwin, D. J. (1996). Integration through overlay analysis. In Fischer, M. M., Scholten, H. J., and Unwin, D., editors, *Spatial Analytical Perspectives on GIS*, pages 129–138. Taylor & Francis, London. [130]
- van Etten, J. (2012). R package **gdistance**: distances and routes on geographical grids. Technical report, **gdistance** vignette. [130]

- Venables, W. N. and Dichmont, C. M. (2004). A generalised linear model for catch allocation: an example from Australia's northern prawn fishery. *Fisheries Research*, 70:409–426. [170]
- Venables, W. N. and Ripley, B. D. (2000). *S Programming*. Springer, New York. [27]
- Venables, W. N. and Ripley, B. D. (2002). *Modern Applied Statistics with S. Fourth Edition*. Springer, New York. [12, 170, 174, 259]
- Venables, W. N., Smith, D. M., and the R Development Core Team (2013). *An Introduction to R*. R Foundation for Statistical Computing, Vienna, Austria. [23, 25, 26]
- Wakefield, J. C., Kelsall, J. E., and Morris, S. E. (2000). Clustering, cluster detection and spatial variation in risk. In Elliott, P., Wakefield, J., Best, N., and Briggs, D., editors, *Spatial Epidemiology: Methods and Applications*, pages 128–152. Oxford University Press, Oxford. [347, 348, 349]
- Wall, M. M. (2004). A close look at the spatial structure implied by the CAR and SAR models. *Journal of Statistical Planning and Inference*, 121:311–324. [46, 290]
- Waller, L. A. and Gotway, C. A. (2004). *Applied Spatial Statistics for Public Health Data*. John Wiley & Sons, Hoboken, NJ. [7, 14, 59, 84, 93, 95, 128, 174, 179, 182, 183, 191, 192, 193, 194, 263, 265, 266, 267, 276, 279, 282, 284, 286, 288, 290, 291, 294, 300, 314, 317, 318, 319, 321, 322, 330, 347, 350, 354, 365, 366]
- Walter, S. D. and Birnie, S. E. (1991). Mapping mortality and morbidity patterns: An international comparison. *International Journal of Epidemiology*, pages 678–689. [319]
- Ward, M. D. and Gleditsch, K. S. (2008). *Spatial regression models*. Sage, Thousand Oaks, CA. [266, 276, 284, 308]
- Wheeler, D. and Tiefelsdorf, M. (2005). Multicollinearity and correlation among local regression coefficients in geographically weighted regression. *Journal of Geographical Systems*, 7:161–187. [318]
- Wheeler, D. C. (2007). Diagnostic tools and a remedial method for collinearity in geographically weighted regression. *Environment and Planning A*, 39:2464–2481. [318]
- Wickham, H. (2009). *ggplot2: Elegant Graphics for Data Analysis*. Springer, New York. [59, 75]
- Wikle, C. K. (2003). Hierarchical models in environmental science. *International Statistical Review*, 71:181–200. [128]
- Wilkinson, L. (2005). *The Grammar of Graphics*. Springer, New York. [59, 76]
- Wise, S. (2002). *GIS Basics*. Taylor & Francis, London. [7]
- Wood, S. (2006). *Generalized Additive Models: An Introduction with R*. Chapman & Hall/CRC, Boca Raton. [200, 258, 314]
- Worboys, M. F. and Duckham, M. (2004). *GIS: A Computing Perspective — Second Edition*. CRC Press, Boca Raton. [7, 129]

- Yao, T. and Journal, A. G. (1998). Automatic modeling of (cross) correlogram tables using fast Fourier transform. *Mathematical Geology*, 30:589–615. [224]
- Zeileis, A. (2004). Econometric computing with HC and HAC covariance matrix estimators. *Journal of Statistical Software*, 11(10):1–17. [304]
- Zuur, A., Ieno, E. N., Walker, N., Saveiliev, A. A., and Smith, G. M. (2009). *Mixed Effects Models and Extensions in Ecology with R*. Springer, New York. [317]
- Zuur, A., Saveiliev, A. A., and N., E. (2012). *Zero inflated models and generalized linear mixed models with R*. Highland Statistics Ltd, Newburgh, UK. [317]

Subject Index

- \$, *see* Methods, \$
- [, *see* Methods, [
[[, *see* Methods, [[
- ade4**, *see* CRAN, **ade4**
- adehabitatHR**, *see* CRAN,
 adehabitatHR
- adehabitatHS**, *see* CRAN,
 adehabitatHS
- adehabitatLT**, *see* CRAN,
 adehabitatLT
- adehabitatMA**, *see* CRAN,
 adehabitatMA
- aerial photogrammetry, 21
- aggregate**, *see* Methods,
 aggregate
- Akaike's Information Criterion
 (AIC), 298
- akima**, *see* CRAN, **akima**
- ArcGIS™, 6, 7, 91, 99, 104, 108,
 124, 125
 coverage, 91, 99
- areal aggregates, 264
- areal data, 263, 264
- aRT** package, 123, 124
- as**, *see* Methods, **as**
- ASPRS Grids & Datums, 87
- azimuth, 88, 149
- Bayesian Hierarchical Models, ix,
 330–334, 337–340, 361
- BayesX, ix, 331
- BayesX**, *see* CRAN, **BayesX**
- BayesXsrc**, *see* CRAN, **BayesXsrc**
- bbox**, *see* Methods, **bbox**
- Bioconductor
 - EBImage**, 100
 - pkgDepTools**, viii
- biOps**, *see* CRAN, **biOps**
- boot**, *see* CRAN, **boot**
- boundaries
 - crisp, 22
 - indeterminate, 22
- Broad Street pump, 118
- BRugs**, *see* CRAN, **BRugs**
- CARBayes, 332
- CARBayes**, *see* CRAN,
 CARBayes
- Class, 24, 27
 - CRS, **sp**, 29, 87–89
 - CsparseMatrix, **Matrix**, 309
 - DMS, **sp**, 88, 90
 - GDALDataset, **rgdal**, 101
 - GDALDriver, **rgdal**, 101
 - GridTopology, **sp**, 48, 49
 - im, **spatstat**, 187
 - Line, **sp**, 38
 - Lines, **sp**, 38
 - listw, **spdep**, 269–271,
 273–275, 339

- nb, **spdep**, 137, 266–269, 273, 274, 339
- owin, **spatstat**, 175
- Polygon, **sp**, 42
- Polygons, **sp**, 43, 132, 133, 135, 138, 139, 145, 146, 148
- POSIXlt, **base**, 37
- ppp, **spatstat**, 175, 176
- Raster, **raster**, 54
- RasterLayer, **raster**, 54
- RsparseMatrix, **Matrix**, 273
- Spatial, **sp**, 28, 29
- SpatialGrid, **sp**, 49, 50
- SpatialGridDataFrame, **sp**, 50, 52, 195
- SpatialLines, **sp**, 39, 61
- SpatialLinesDataFrame, **sp**, 39
- SpatialPixels, **sp**, 51, 53, 61
- SpatialPixelsDataFrame, **sp**, 51, 52, 195
- SpatialPoints, **sp**, 30, 31, 174
- SpatialPointsDataFrame, **sp**, 33, 60, 174
- SpatialPolygons, **sp**, 43, 61, 133, 134, 142, 145, 148, 195, 268
- SpatialPolygonsDataFrame, **sp**, 44, 143
- ST, **spacetime**, 154
- STF, **spacetime**, 154
- STFDF, **spacetime**, 154
- STI, **spacetime**, 154
- STIDF, **spacetime**, 154
- STS, **spacetime**, 154
- STSDF, **spacetime**, 154
- STT, **spacetime**, 154
- STTDF, **spacetime**, 154
- class intervals, 79–81
 - Fisher-Jenks, 80
 - natural breaks, 80, 81
 - quantiles, 80, 81
- classInt**, *see* CRAN, **classInt**
- cluster, *see* disease cluster
- coda**, *see* CRAN, **coda**, *see* CRAN, **coda**
- coerce, *see* Methods, **as**
- cokriging, *see* geostatistics, prediction, multivariable
- Color Brewer, 79, 342
- colour palettes, 79, 342
 - bpy.colors, 79
 - cm.colors, 79
 - grey.colors, 79
 - heat.colors, 79
 - rainbow, 79
 - terrain.colors, 79
 - topo.colors, 79
- Complete Spatial Randomness, 179–181, 192
- Comprehensive R Archive Network (CRAN), 3
- computational geometry, 21, 131
 - area measurement, 133, 139
 - line measurement, 136–139
 - point-in-polygon problem, 140, 148
 - scaling, 131, 135
- Sort-Tile-Recursive tree, 138, 139
- topological operations, 134, 135, 137, 139, 140
 - buffering, 139
 - intersection, 137, 139, 140
 - line merge, 137
 - union, 134, 135
- topological predicates, 135, 137, 148
 - contains, 148
 - overlaps, 135
 - touches, 137
- coordinate reference systems, 8, 22, 84, 130
 - datum, 85–87
 - ellipsoid, 85
 - prime meridian, 85
- coordinates, 8, 30, 214

- geographical, 29, 30, 86, 88, 90
- projected, 89, 130
- coordinates, *see* Methods, coordinates
- coordinates<-, *see* Methods, coordinates<-
- CRAN
 - ade4, 266
 - adehabitatHR, 122
 - adehabitatHS, 122
 - adehabitatLT, 122
 - adehabitatMA, 122
 - akima, 258
 - BayesX, 274, 331, 334, 335, 343–346, 359, 360
 - BayesXsrc, 331
 - biOps, 100
 - boot, 282
 - BRugs, 331
 - CARBayes, 332, 334–336, 343, 344
 - classInt, 80
 - coda, 310, 333, 334, 340, 342
 - cubature, 189
 - DCluster, 95, 324, 326, 349, 350, 352–354, 356
 - fields, 252, 257, 258, 260
 - foreign, 99
 - gdistance, 122, 130, 149
 - geoR, 227–229, 259
 - geoRglm, 259
 - geosphere, 130, 149
 - GeoXp, 123
 - ggplot2, 59, 75
 - GRASS, 112
 - gstat, 151, 165, 214, 216, 217, 219–226, 228–234, 236–245, 248, 250–255, 257, 261
 - gwrr, 318
 - intervals, 161
 - latticeExtra, 76
 - LearnBayes, 311
 - lgcp, 211
 - lmtest, 303
 - mapproj, 86
 - maps, 39, 41, 45, 46, 91, 158
 - maptools, 41, 44, 95, 99, 122, 123, 142, 174–176, 187
 - MASS, 242
 - Matrix, 273, 301, 302, 309
 - McSpatial, 307, 313
 - mgcv, 200, 202, 258, 314, 315
 - ncdf, 165
 - ncdf4, 165
 - pbdNCDF4, 165
 - PBSmapping, 122
 - pgirmess, 284
 - pixmap, 108
 - plm, 164
 - plotKML, 151, 164
 - R2BayesX, 331
 - R2WinBUGS, 331–334, 339
 - RandomFields, 228, 252, 259, 261
 - RArcInfo, 41, 91, 99
 - raster, viii, ix, 54, 76, 130, 142, 165
 - rasterVis, 76, 165
 - RColorBrewer, 79, 163, 342
 - ReadImages, 100
 - rgdal, 84, 86, 88, 92, 100, 125, 132, 136, 138, 177, 353
 - installation, 125
 - rgeos, ix, 41, 48, 97, 120, 130–133, 135, 137–140
 - RNetCDF, 165
 - RPyGeo, 125
 - RSAGA, 124
 - sandwich, 304
 - shapefiles, 99
 - sos4R, 165
 - sp, vii–ix, 4, 28, 91, 122, 142, 151, 214, 240, 244
 - spacetime, ix, 97, 151, 154, 156, 165, 211, 261
 - spacetime, 156
 - spatial, 174, 217, 259

- Spatial Task View, 15
- spatialkernel**, 174
- SpatioTemporal**, 261
- spatstat**, 122, 174–176, 180, 181, 186, 187, 189, 191, 195, 197, 199, 205, 206, 208, 209, 211
- spdep**, ix, 95, 137, 266–276, 278–283, 285–287, 292, 294, 296, 299–302, 305–307, 309–311, 315, 317, 339, 350–352
- spgrass6**, 113, 114, 118–120, 136, 137
- spgwr**, 318
- sphet**, 311–313
- splancs**, 124, 174, 186, 187, 189, 195, 203, 204, 211
- splm**, 313
- spTimer**, 261
- stinepack**, 258
- stpp**, 164, 211
- surveillance**, 165
- testthat**, 132
- XML**, 113
- xts**, 156
- zoo**, 156, 161
- crepuscule**, *see* Methods, **crepuscule**
- CRS**, *see* Class, **CRS**
- CsparseMatrix**, *see* Class, **CsparseMatrix**
- CSR**, *see* Complete Spatial Randomness
- cubature**, *see* CRAN, **cubature**
- Data formats
 - GAL, 273
 - GDAL RData, 105
 - Geography Markup Language (GML), 96
 - GeoTiff, 102, 103, 136
 - GPS Exchange Format (GPX), 98, 99
 - Keyhole Markup Language (KML), 96, 99, 110
 - Mapgen, 41
 - Portable Network Graphics (PNG), 110, 111
 - PostGIS, 93
 - raster, 100
 - shapefile, 92–94, 96, 99, 132
 - vector, 91
 - Web Feature Service (WFS), 96
 - Web Map Service (WMS), 106
 - WKT, 132
- data frames, 25, 35
- Data set
 - Auckland 90 m Shuttle Radar Topography Mission, 364
 - Auckland 90m Shuttle Radar Topography Mission, 50–54, 100, 101
 - Auckland shoreline, 41, 43, 44, 364
 - Biological cell centres, 176, 180, 181, 191, 192, 364
 - Broad Street cholera mortalities, 118–120, 122, 365
 - California redwood trees, 176, 180, 181, 186, 187, 191, 192, 365
 - cars, 24–27, 365
 - CRAN mirrors, 30–37, 365
 - Eurasian Collared Dove, 156, 365
 - Japan shoreline, 39, 40, 365
 - Japanese black pine saplings, 175, 176, 180, 181, 191, 192, 365
 - Lansing Woods maple trees, 189, 190, 365
 - Loggerhead turtle, 37
 - loggerhead turtle, 365
 - Manitoulin Island, 47–50, 365

- Maunga Whau volcano, 8, 9, 40, 128, 129, 365
- Meuse bank, 53, 60–62, 64, 65, 67, 69–73, 77–82, 103, 104, 110, 111, 141, 142, 214–217, 219–226, 228–234, 236, 237, 239–244, 248, 250, 251, 253–255, 257, 365
- New Mexico brain cancer, 357, 365
- New York leukemia, 265, 267–275, 277, 279–288, 291, 292, 294–296, 299–302, 304–307, 309–315, 365
- North Carolina SIDS, 65, 78, 322, 324–329, 332–334, 337–340, 342–344, 349–354, 356, 366
- North Derbyshire asthma study, 177, 195–200, 202–206, 208, 209, 366
- Olinda, 132–140, 142–148, 366
- Produc panel data, 164, 366
- Scottish lip cancer, 93–96, 366
- Spearfish, 113–117, 366
- US 1999 SAT scores, 45, 46, 366
- world volcano locations, 8, 366
- `data.frame`, 25, 35
- DCluster**, *see* CRAN, **DCluster**
- dimensions, 22
 - 2.5D, 23
 - 3D, 22
- disease cluster, 347
 - detection, 347
 - testing, 348, 352
 - chi-square test, 348
 - general clustering, 351
 - Geographical Analysis Machine, 353
 - homogeneity, 348
 - Kulldorff's statistic, 353, 354
 - localised, 355, 356
 - Moran's *I* test, 350
 - Potthoff-Whittinghill test, 349
 - scan statistic, 353, 354
 - spatial autocorrelation, 350
 - Stone's test, 355, 356
 - Tango's test, 351
- disease mapping, 319
- DMS, *see* Class, DMS
- EB, *see* Empirical Bayes
- EBImage**, *see* Bioconductor, **EBImage**
- `elide`, *see* Methods, `elide`
- ellipsoid, 30
 - WGS84, 30
- Empirical Bayes
 - estimation, 324, 326–329
 - local estimation, 328, 329
 - log-normal model, 326
 - Poisson-Gamma model, 324
- empirical cumulative distribution function, 80
- ENVI™, 104
- epidemiology
 - spatial, 192
- EPSG geodetic parameter data set, 85, 86
- error measurement, 128
- error propagation, 129
- European Petroleum Survey Group (EPSG), 85
- fields**, *see* CRAN, **fields**
- foreign**, *see* CRAN, **foreign**
- `gcDestination`, 149
- `GDALDataset`, *see* Class, `GDALDataset`
- `GDALDriver`, *see* Class, `GDALDriver`
- `gdistance`, *see* CRAN, `gdistance`

- Geary's C test, *see* spatial autocorrelation, tests, Geary's C
- generalised additive model, 200, 202, 314, 315
- generalised linear model, 208, 315
- generic functions, *see* Methods
- GeoDa, 273
- geographical coordinates, *see* coordinates, geographical
- Geographical Information Systems (GIS), 5–8, 83, 91, 100, 112, 123
- Geometry Engine – Open Source, 131
- Geometry Engine — Open Source, 47
- geoR**, *see* CRAN, **geoR**
- geoRglm**, *see* CRAN, **geoRglm**
- GEOS, *see* Geometry Engine — Open Source
- Geospatial Data Abstraction Library (GDAL), 83, 100, 125
- OGR, 92, 96, 98, 110, 138
- geosphere**, *see* CRAN, **geosphere**
- geostatistics, 10, 213, 214, 217, 252
 - anisotropy, 221, 223, 228, 229
 - conditional simulation, 214, 252
 - circulant embedding, 252
 - sequential, 252–255
 - covariance, 214
 - isotropy, 218
 - model diagnostics, 247
 - cross validation, 247–251
 - model-based, 214, 256
 - monitoring networks, 213, 256, 257
 - multivariable, 214, 229, 230
 - prediction, 214, 232–234, 236–238
 - block kriging, 214, 215, 217, 238, 239
 - domain stratification, 240
 - indicator kriging, 214, 243, 255
 - multivariable, 214, 230, 234, 236
 - ordinary, 233
 - ordinary kriging, 233
 - simple, 233
 - simple kriging, 214, 233
 - singular matrix errors, 243–245
 - universal, 233
 - universal kriging, 214, 233
- semivariance, 214, 217, 218, 221
- stationarity, 218, 221
- variable transformation, 242, 243
- variogram, 214, 217–228
 - cloud, 220
 - cross, 229, 230
 - cutoff, 222, 223
 - direction, 222, 223, 228
 - exploratory, 219–221
 - lag width, 222, 223
 - model, 224–229, 231, 232, 255
 - nugget, 226
 - partial sill, 226
 - range, 226
 - residual, 230–232
- GeoXp**, *see* CRAN, **GeoXp**
- Getis-Ord G test, *see* spatial autocorrelation, tests, Getis-Ord G
- ggplot2**, *see* CRAN, **ggplot2**
- GIS
 - data models, 8, 22, 91, 100
 - raster, 48
- Global Positioning System (GPS), 22, 84
- Global Self-consistent Hierarchical

- High-resolution Shoreline Database (GSHHS), 47, 91
- Google Earth™, viii, 6, 21, 110, 111, 164
 - image overlay, 110, 111
 - KML, 110
- Google Maps™, viii, 21, 108
- GRASS
 - location, 113
 - Soho, 118
 - Spearfish, 114
 - mapset, 113
 - version 5, 112
 - version 6, 112, 113, 136
 - OSX, 113
 - Windows, 113
 - window, 113
- GRASS**, *see* CRAN, **GRASS**
- GRASS GIS, 112, 113, 118, 136
- Great Circle distance, 88, 149
- grid, 9, 48
- GridTopology, *see* Class, GridTopology
- ground control points, 22
- gstat**, *see* CRAN, **gstat**
- gwrr**, *see* CRAN, **gwrr**
- habitat, 122
- ID matching, 34, 44, 46, 322
- im, *see* Class, im
- image, *see* Methods, image
- impacts, 308–311
 - direct and indirect, 308–311
 - summary measures, 308–310
- impacts, *see* Methods, impacts
- INLA, ix, 332, 334, 335, 343, 344, 346, 359–361
- INLA**, 274
- interpolation, 215
 - geostatistical, *see* geostatistics, prediction
 - geostatistics, prediction
 - inverse distance weighted, 216, 240
 - linear regression, 217
 - trend surface, 217
- ISO 8601, *see* spatio-temporal data
- John Snow, 118
- join count test, *see* spatial autocorrelation, tests, join count
- JTS, *see* JTS Topology Suite
- JTS Topology Suite, 131
- kriging, *see* geostatistics, prediction
- lag
 - spatial, 275
- lattice graphics, 69
- LearnBayes**, *see* CRAN, **LearnBayes**
- levelplot, 70
- lgcp**, *see* CRAN, **lgcp**
- Line**, *see* Class, **Line**
- line generalisation, 41
- linear model, 280, 286, 291, 292, 295, 305, 314
 - heteroskedasticity, 303, 304
 - multicollinearity, 292, 305
 - residuals, 280, 286, 291, 292, 295, 296, 305
 - weighted, 295, 296
- Lines**, *see* Class, **Lines**
- lines, 9, 38
- lines, *see* Methods, lines
- listw, *see* Class, listw
- lmtest**, *see* CRAN, **lmtest**
- longlat, 29, 31, 86
- mailing list, *see* R-Sig-Geo mailing list, *see* R-Sig-Geo mailing list
- Mantel general cross product test, *see* spatial autocorrelation, tests, Mantel
- map class intervals, 79
- map colours, 68, 79, 342

- map grids, 66
- map north arrow, 65
- map plotting, 59
- map scale bar, 65
- map symbols, 68
- Mapgen, *see* Data formats, Mapgen
- mapproj**, *see* CRAN, **mapproj**
- maps**, *see* CRAN, **maps**
- maptools**, *see* CRAN, **maptools**
- MASS**, *see* CRAN, **MASS**
- mathematical geography, 21
- Matlab™, 274
- Matrix**, *see* CRAN, **Matrix**
- MCMCsamp**, *see* Methods, MCMCsamp
- McSpatial**, *see* CRAN, **McSpatial**
- memory management, 11
- Methods, 24
 - aggregate, 156
 - aggregate, 142, 159
 - as, 28, 52, 57, 60, 72, 88, 91, 110, 122, 175, 176, 195, 198, 309
 - bbox, 31
 - coordinates, 32
 - coordinates<-, 36, 60, 61
 - crepuscule, 149
 - elide, 149, 176
 - image, 59, 61, 62, 69, 81, 111
 - impacts, 309, 310
 - lines, 59
 - MCMCsamp, 311
 - na.approx, 156
 - na.locf, 156
 - na.spline, 156
 - names, 35
 - over, 156
 - over, 33, 111, 119, 140–142, 159, 240
 - plot, 33, 59–62, 68, 81
 - points, 59
 - predict, 217
 - print, 33
 - proj4string, 32
 - proj4string<-, 32
 - solarnoon, 149
 - solarpos, 149
 - spCbind, 95
 - spplot, 60, 70–72, 78, 82, 214
 - spsample, 33, 146, 239
 - spTransform, 66, 88, 95, 110, 132, 353
 - stplot, 156
 - stplot, 161, 163
 - subset, 267, 273
 - summary, 33
 - sunriset, 149
- mgcv**, *see* CRAN, **mgcv**
- missing values, 273
- misspecification, 264, 265, 276, 277, 280, 288, 289, 292
- modifiable areal unit problem, 263
- Mondrian, 123
- Moran's *I* test, *see* spatial autocorrelation, tests, Moran's *I*
- moving objects, *see* spatio-temporal data
- names**, *see* Methods, **names**
- nb**, *see* Class, **nb**
- neighbours
 - spatial, 137, 264–269, 273, 276
 - points, 268, 269
 - polygons, 268
 - sets, 264
- objects and fields, 10
- observation window, 175
- Oil & Gas Producers (OGP) Surveying & Positioning Committee, 85
- Open Source Geospatial Foundation (OSGeo), 83, 112

- OpenGIS®
 - simple features, 43, 48, 92, 131, 132
 - Web Feature Service (WFS), 96
- OpenStreetMap, 106–108
- osmar**, *see* CRAN, **osmar**
- over**, *see* Methods, **over**
- overlay**, *see* Methods, **over**
- owin**, *see* Class, **owin**
- P-splines, 345
- PBSmapping**, *see* CRAN, **PBSmapping**
- pgirmess**, *see* CRAN, **pgirmess**
- pixmap**, *see* CRAN, **pixmap**
- pkgDepTools**, *see* Bioconductor, **pkgDepTools**
- plot**, *see* Methods, **plot**
- plotKML**, *see* CRAN, **plotKML**
- plotting maps, 59
 - axes, 62, 64, 65
- point pattern
 - binary regression estimator, 198, 200
 - bounding region, 173
 - case-control, 193–196, 198, 200, 206, 207, 209
 - clustered, 179, 204
 - definition, 173
 - intensity, 182, 184
 - kernel bandwidth, 185–187, 194, 195, 197, 199
 - kernel density, 184, 185, 187, 194, 195
 - kernel density ratio, 194, 195, 197, 199
 - marked, 176, 193
 - regular, 179
- point pattern analysis, 10, 173
- point process
 - F function, 181
 - G function, 179, 180
 - K function, 191, 192, 205
 - inhomogeneous, 192, 193, 206–209
 - definition, 173
 - homogeneous, 183
 - inhomogeneous, 184
 - isotropic, 183
 - K function, 191
 - likelihood, 188
 - second-order properties, 190
 - stationary, 183
- point source pollution, 202–204
- points, 9, 30, 214
 - 2D, 30
 - 3D, 31
 - neighbours, 268, 269
 - k -nearest, 268
 - distance bands, 269
- points**, *see* Methods, **points**
- Poisson-Gamma model, 323, 324, 332–334
- Polygon**, *see* Class, **Polygon**
- Polygons**, *see* Class, **Polygons**
- polygons, 9, 42, 263, 264
 - contiguous neighbours, 268
 - queen, 268
 - hole, 42, 46, 47, 132, 263
 - plot order, 43, 48
 - ring direction, 42, 46, 47
 - topology, 47
- POSIXlt**, *see* Class, **POSIXlt**
- ppp**, *see* Class, **ppp**
- predict**, *see* Methods, **predict**
- print**, *see* Methods, **print**
- probability map, 325
- PROJ.4 Cartographic Projections
 - library, 83, 85, 86, 125
- tags, 86
 - ellps, 87, 89
 - init, 87
 - proj, 87, 89
 - towgs84, 87
- proj4string**, *see* Methods, **proj4string**
- proj4string<-**, *see* Methods, **proj4string<-**

- Python, 125
 - MGET, 125
- quadtree, 238
- R-Sig-Geo mailing list, viii, 15
- R2BayesX**, *see* CRAN, **R2BayesX**
- R2WinBUGS**, *see* CRAN, **R2WinBUGS**
- raised incidence, 203, 204
- random fields, 214, 252
- RandomFields**, *see* CRAN, **RandomFields**
- RArcInfo**, *see* CRAN, **RArcInfo**
- RColorBrewer**, *see* CRAN, **RColorBrewer**
- ReadImages**, *see* CRAN, **ReadImages**
- remote sensing, 21, 48
 - multi-spectral images, 21
- Reproducible research, xi, 4, 5
- rgdal**, *see* CRAN, **rgdal**
- rgeos**, *see* CRAN, **rgeos**, *see* CRAN, **rgeos**
- RgoogleMaps**, *see* CRAN, **RgoogleMaps**
- row names, 26
- RPyGeo**, *see* CRAN, **RPyGeo**
- RSAGA**, *see* CRAN, **RSAGA**
- RsparseMatrix**, *see* Class, **RsparseMatrix**
- sampling
 - spatial, *see* spatial sampling
- sandwich**, *see* CRAN, **sandwich**
- shapefiles**, *see* CRAN, **shapefiles**
- Shuttle Radar Topography
 - Mission, 21, 50
- simultaneous autoregression, 274
 - simulation, 274
- solar noon, 149
- solar position, 149
- solarnoon**, *see* Methods, **solarnoon**
- solarpos**, *see* Methods, **solarpos**
- Sort-Tile-Recursive tree, *see* computational geometry, Sort-Tile-Recursive tree
- sp.layout**, argument to **spplot**, 73
- space-time layout, *see* spatio-temporal data
- spacetime**, *see* CRAN, **spacetime**, *see* CRAN, **spacetime**
- Spatial**, *see* Class, **Spatial**
- spatial**, *see* CRAN, **spatial**
- spatial autocorrelation, 264, 274, 289, 290
 - approximate
 - profile-likelihood estimator, 292
 - correlogram, 283, 284
 - local tests, 284–288
 - Moran's I , 286–288
 - misspecification, 264, 265, 276, 277, 280, 288, 289, 292
- Moran scatterplot, 285
- over-dispersion, 286
- tests, 265, 276–285, 287, 288, 292, 296
 - Empirical Bayes Moran's I , 282, 283
 - exact, 281, 286
 - Geary's C , 278
 - Getis-Ord G , 278
 - join count, 278
 - Lagrange Multiplier, 304–306
 - Mantel, 278
 - Monte Carlo, 278, 281, 282
 - Moran's I , 276, 277, 279–281, 292, 296, 350
 - Normality assumption, 278, 280, 286
 - parametric bootstrap, 278, 282, 287, 288

- permutation bootstrap, 278, 281, 282
- Randomisation
 - assumption, 279, 286
 - rates, 282, 283, 286–288
- Saddlepoint
 - approximation, 281, 286
- Spatial Econometrics Library, 274
- spatial epidemiology, 319
- spatial lag, *see* lag, spatial
- spatial models, 289–293, 295, 314, 315, 317
 - Common Factor, 310
 - conditional autoregressive (CAR), 290, 298–300, 337–340, 344, 357, 359–361
 - generalised additive model, 314, 315
 - generalised estimating equations, 317
 - generalised linear
 - mixed-effect model, 317
 - generalised linear model, 314, 315
 - geoadditive model, 314, 345
 - geographically weighted
 - regression, 318
 - Jacobian, 301, 302
 - eigenvalues, 301
 - sparse matrix
 - representation, 301, 302
 - likelihood ratio test, 295
 - log likelihood, 302
 - Moran eigenvector, 317
 - P-splines, 346
 - simultaneous autoregressive (SAR), 290, 292–295, 297, 302
 - components of fitted values, 295, 298
 - simultaneous moving average (SMA), 302
 - spatial Durbin, 305, 307–310
 - spatial Durbin error, 311
 - spatial econometrics, 303, 304
 - spatial error, 305, 310, 311
 - GM estimator, 312, 313
 - spatial filtering, 317
 - spatial lag, 305–310
 - 2SLS, 311
 - spatial quantile, 313
 - spatial neighbours, *see* neighbours, spatial
 - spatial queries, 140
 - spatial sampling, 146, 239
 - Spatial Task View, *see* CRAN, Spatial Task View
 - spatial weights, *see* weights, spatial
 - SpatialGrid**, *see* Class, **SpatialGrid**
 - SpatialGridDataFrame**, *see* Class, **SpatialGridDataFrame**
 - spatialkernel**, *see* CRAN, **spatialkernel**
 - SpatialLines**, *see* Class, **SpatialLines**
 - SpatialLinesDataFrame**, *see* Class, **SpatialLinesDataFrame**
 - SpatialPixels**, *see* Class, **SpatialPixels**
 - SpatialPixelsDataFrame**, *see* Class, **SpatialPixelsDataFrame**
 - SpatialPoints**, *see* Class, **SpatialPoints**
 - SpatialPointsDataFrame**, *see* Class, **SpatialPointsDataFrame**
 - SpatialPolygons**, *see* Class, **SpatialPolygons**
 - SpatialPolygonsDataFrame**, *see* Class, **SpatialPolygonsDataFrame**
 - spatio-temporal data, 151
 - aggregation, 159

- construction, 156
 - interpolation, 261
 - ISO 8601, 156
 - moving objects, 153
 - overlay, 159
 - replacement, 158
 - selection, 158
 - space-time layout, 152
 - spatio-temporal fields, 152
 - spatio-temporal point
 - pattern, 152
 - support, 152
 - time instance or interval,
 - 152
 - trajectories, 153
 - visualising, 161, 162
- spatio-temporal models, 356
- spatstat**, *see* CRAN, **spatstat**, *see*
 - CRAN, **spatstat**
- spdep**, *see* CRAN, **spdep**
- spgrass6**, *see* CRAN, **spgrass6**
- spgwr**, *see* CRAN, **spgwr**
- sphet**, *see* CRAN, **sphet**
- splanacs**, *see* CRAN, **splanacs**, *see*
 - CRAN, **splanacs**
- splm**, *see* CRAN, **splm**
- spplot**, *see* Methods, **spplot**
- spsample**, *see* Methods,
 - spsample**
- spTransform**, *see* Methods,
 - spTransform**
- standardisation
 - indirect, 321
 - internal, 321
- Standardised Mortality Ratio,
 - 322, 357
- Stata™, 123, 274
 - tmap, 123
- StatConnector (D)COM
 - mechanism, 124
- stinepack**, *see* CRAN, **stinepack**
- stpp**, *see* CRAN, **stpp**
- subset**, *see* Methods, **subset**
- summary**, *see* Methods, **summary**
- sunrise, 149
- sunrisset**, *see* Methods,
 - sunrisset**
- sunset, 149
- support, 127, 129, 263
 - change of, 128, 238, 239
- temporal models
 - random walk, 357, 359–361
- TerraLib, 123, 124
 - R interface, 123, 124
- testthat**, *see* CRAN, **testthat**
- thematic maps, 68
- trajectories, *see* spatio-temporal
 - data
- trellis graphics, *see* lattice
 - graphics
- triangulation, 85
- uncertainty, 22, 129, 252
- visualisation, 11
- visualising spatial data, 59
- weights
 - spatial, 264, 265, 269–271,
 - 273, 274, 276, 339
 - asymmetric, 301
 - binary, 269, 270
 - generalised, 271
 - no-neighbour areal entities,
 - 271, 272, 279
 - row standardised, 270
 - similar to symmetric, 301
 - sparse matrix
 - representation, 301
 - styles, 270
 - symmetric, 299, 301
 - unknown, 274
 - zero policy, 271, 272
- WinBUGS, ix, 331–334, 337–340,
 - 346, 361
- GeoBUGS polygon import,
 - 123

GeoBUGS weights import,
274
window, 175
WKT, *see* Data formats, WKT

XML, *see* CRAN, **XML**

zero policy, *see* weights, spatial,
zero policy

Functions Index

- achisq.test, **DCluster**, 349
- adaptIntegrate, **cubature**, 189
- aes, **ggplot2**, 75
- aggregate method, **sp**, 142
- anova.sarlm, **spdep**, 307
- aple, **spdep**, 292
- as.geodata, **geoR**, 227
- as.vgm.variomodel, **gstat**, 227
- as_dgRMatrix_listw, **spdep**,
273, 309
- as_sp, **osmar**, 109
- axis, **graphics**, 62

- bayesx, **BayesX**, 335, 344–346
- bbox method, **sp**, 31
- blockSize, **raster**, 55
- boot, **boot**, 282
- boxcox, **MASS**, 242
- bptest, **lmtest**, 303
- bptest.sarlm, **spdep**, 306
- bpy.colors, **sp**, 79
- brewer.pal, **RColorBrewer**, 79,
80
- bugs, **R2WinBUGS**, 332–334, 339
- bw.diggle, **spatstat**, 186
- bw.relrisk, **spatstat**, 199

- calculate.mle, **DCluster**, 354
- card, **spdep**, 266, 269
- cellStats, **raster**, 55
- char2dms, **sp**, 88, 90

- classIntervals, **classInt**, 80
- cm.colors, **grDevices**, 79
- coefest, **lmtest**, 304
- colorRampPalette, **grDevices**, 79
- ContourLines2SLDF, **maptools**,
40, 72, 198
- coord_equal, **ggplot2**, 75
- coordinates method, **sp**, 32
- cor, **stats**, 230
- correlog, **pgirmess**, 284
- cover.design, **fields**, 257
- create2GDAL, **rgdal**, 103
- createSPComment, **rgeos**, 48
- crepuscule method, **maptools**,
149
- CRS, **sp**, 29, 31, 32, 66, 87–89, 94,
95, 103, 110, 111, 132,
136, 137

- dd2dms, **sp**, 88, 90
- default.stringsAsFactors,
base, 94
- degAxis, **sp**, 65
- density, **spatstat**, 187, 195, 197
- dnearneigh, **spdep**, 269, 284, 351
- doGRASS, **spgrass6**, 113

- EBest, **spdep**, 327
- EBImoran.mc, **spdep**, 282
- EBlocal, **spdep**, 328
- ecdf, **stats**, 80

- eigen, **base**, 301
- eigenw, **spdep**, 301, 302
- elide method, **maptools**, 149, 176
- empbaysmooth, **DCluster**, 95, 324
- envelope, **spatstat**, 180, 181, 191, 205, 209
- errorsarlm, **spdep**, 310, 311
- eval.im, **spatstat**, 197
- execGRASS, **spgrass6**, 113, 116, 136
- eyefit, **geoR**, 227
- f, **INLA**, 335, 344, 359
- Fest, **spatstat**, 181
- findColours, **classInt**, 81
- fit.lmc, **gstat**, 229, 230
- fit.variogram, **gstat**, 226, 228, 231, 232, 248, 255
- fit.variogram.reml, **gstat**, 228
- fitvario, **RandomFields**, 228
- fortify, **ggplot2**, 75
- fv, **spatstat**, 209
- gam, **mgcv**, 200, 202, 314, 315
- gArea, **rgeos**, 133, 135, 140
- gBinarySTRtreeQuery, **rgeos**, 139
- gBuffer, **rgeos**, 120
- gBuffer, **rgeos**, 139
- gcDestination, **maptools**, 149
- GDAL.close, **rgdal**, 101
- GDAL.open, **rgdal**, 101
- GDALinfo, **rgdal**, 101, 103
- GE_SpatialGrid, **maptools**, 111
- geary.test, **spdep**, 278
- geom_point, **ggplot2**, 75
- Gest, **spatstat**, 180
- get_osm, **osmar**, 109
- getDriverLongName, **rgdal**, 101
- getinfo.shape, **maptools**, 99
- GetMap, **RgoogleMaps**, 108
- GetMap.OSM, **RgoogleMaps**, 108
- getScale, **rgeos**, 131, 135
- getValues, **raster**, 55
- geweke.diag, **coda**, 340
- ggplot, **ggplot2**, 75
- gIntersection, **rgeos**, 97
- gIntersection, **rgeos**, 138–140
- gLength, **rgeos**, 137–139
- gLineMerge, **rgeos**, 137
- glm, **stats**, 208, 282, 315
- globalG.test, **spdep**, 278
- gmeta2grd, **spgrass6**, 120
- gmeta2grd, **spgrass6**, 118
- gOverlaps, **rgeos**, 135
- grey.colors, **grDevices**, 79
- grid.locator, **grid**, 78
- gridat, **sp**, 66
- gridlines, **sp**, 66
- GridTopology, **sp**, 48, 49
- gstat, **gstat**, 229–232, 236, 238, 241, 242
- gstat.cv, **gstat**, 250
- gTouches, **rgeos**, 137
- gUnaryUnion, **rgeos**, 135
- gzAzimuth, **maptools**, 88, 149
- heat.colors, **grDevices**, 79
- hscat, **gstat**, 219
- I, **base**, 217
- identify, **graphics**, 77
- idw, **gstat**, 111, 216, 240
- image method, **sp**, 59, 61, 62, 69, 81, 111
- include.self, **spdep**, 351
- influence.measures, **stats**, 285
- initGRASS, **spgrass6**, 113, 136
- inla, **INLA**, 274, 335, 344, 359
- inMemory, **raster**, 55
- interp.im, **spatstat**, 208, 209
- invIrW, **spdep**, 274
- joincount.multi, **spdep**, 278
- joincount.test, **spdep**, 278
- Kest, **spatstat**, 191, 206
- Kinhom, **spatstat**, 208, 209
- kmlLine, **maptools**, 110
- kmlOverlay, **maptools**, 111
- kmlPoints, **maptools**, 110
- kmlPolygon, **maptools**, 110
- knearneigh, **spdep**, 269

- knn2nb, **spdep**, 269
- krige, **gstat**, 217, 233, 238–240, 243–245, 248, 251, 253–255, 257
- krige.cv, **gstat**, 248, 251
- lag.listw, **spdep**, 275, 286, 287
- lagsarlm, **spdep**, 306, 307
- layout, **graphics**, 68
- layout.north.arrow, **sp**, 65
- layout.scale.bar, **sp**, 65
- legend, **graphics**, 69
- levelplot, **lattice**, 70
- likfit, **geoR**, 228, 229
- Line, **sp**, 38
- Lines, **sp**, 38
- lines method, **sp**, 59
- listw2mat, **spdep**, 274
- listw2sn, **spdep**, 274
- listw2U, **spdep**, 273
- listw2WB, **spdep**, 274, 339
- lm, **stats**, 214, 217, 280, 291, 292, 295, 305
- lm.LMtests, **spdep**, 305
- lm.morantest, **spdep**, 276, 280, 292, 296, 315
- lm.morantest.exact, **spdep**, 281
- lm.morantest.sad, **spdep**, 281
- localmoran, **spdep**, 286
- localmoran.exact, **spdep**, 286
- localmoran.sad, **spdep**, 286
- locator, **graphics**, 69, 77, 78
- lognormalEB, **DCluster**, 326
- make_EPSG, **rgdal**, 86
- map, **maps**, 39, 45, 66
- map2SpatialLines, **maptools**, 39, 66
- map2SpatialPolygons, **maptools**, 45, 91, 158
- MapGen2SL, **maptools**, 41
- marks, **spatstat**, 209
- mat2listw, **spdep**, 274
- MCMCsamp.sarlm, **spdep**, 311
- ME, **spdep**, 317
- moran, **spdep**, 282
- moran.mc, **spdep**, 281
- moran.plot, **spdep**, 285
- moran.test, **spdep**, 276, 279, 280, 283, 284
- moranI.test, **DCluster**, 350
- mse2d, **splancs**, 186
- n.comp.nb, **spdep**, 137
- nb2gra, **BayesX**, 274, 343, 344, 359
- nb2INLA, **spdep**, 274
- nb2lines, **spdep**, 274
- nb2listw, **spdep**, 269–271, 274, 276, 279–282, 286, 292, 294, 305, 350, 352
- nb2mat, **spdep**, 343
- nb2WB, **spdep**, 274, 339
- nbdists, **spdep**, 269, 271, 351
- nblag, **spdep**, 283
- nclass.Sturges, **grDevices**, 80
- neig2nb, **ade4**, 266
- npoints, **spatstat**, 195
- ogr, **rgdal**, 96
- ogrDrivers, **rgdal**, 92, 96
- ogrInfo, **rgdal**, 93, 94
- opgam, **DCluster**, 353, 354
- optim, **stats**, 189
- optimize, **stats**, 300
- osmsource_api, **osmar**, 109
- over, **sp**, 120
- over method, **sp**, 78, 111, 119, 140–142, 240
- pal2SpatialPolygons, **maptools**, 91
- panel.identify, **lattice**, 78
- par, **graphics**, 62, 66, 67
- parseGRASS, **spgrass6**, 113
- plot method, **sp**, 59–62, 68, 81
- plot method, **classInt**, 80
- points method, **sp**, 59
- poisson.bymCAR, **CARBayes**, 344
- poisson.independent, **CARBayes**, 335, 336
- poly, **stats**, 217

- `poly2nb`, **spdep**, 359
- `Polygon`, **sp**, 42
- `Polygons`, **sp**, 43
- `pottwhitt.test`, **DCluster**, 349
- `ppm`, **spatstat**, 189
- `predict` method, **gstat**, 234, 236, 238, 241, 242, 255
- `probmap`, **spdep**, 95
- `proj4string` method, **sp**, 32
- `pruneMap`, **maptools**, 66
-
- `gregspiv`, **McSpatial**, 313
-
- `rainbow`, **grDevices**, 79
- `raster`, **raster**, 54
- `read.coda`, **coda**, 340
- `read.dat2listw`, **spdep**, 274
- `read.gal`, **spdep**, 267, 273, 322
- `read.gwt2nb`, **spdep**, 273
- `readAsciiGrid`, **maptools**, 108
- `readGDAL`, **rgdal**, 100, 101, 136
- `readOGR`, **rgdal**, 96, 99
- `readOGR`, **rgdal**, 93, 94, 132, 138, 177, 267
- `readRAST6`, **spgrass6**, 114, 119
- `readShapePoly`, **maptools**, 78, 322
- `readShapeSpatial`, **maptools**, 99
- `readVECT6`, **spgrass6**, 116, 119, 120, 137
- `relrisk`, **spatstat**, 195, 199
- `Rgshhs`, **maptools**, 91
- `rlabel`, **spatstat**, 197, 205, 206
- `rSPDistance`, **gdistance**, 122
- `rwmetrop`, **LearnBayes**, 311
-
- `sarml`, **McSpatial**, 307
- `set.ZeroPolicyOption`, **spdep**, 272
- `setScale`, **rgeos**, 135
- `Sobj_SpatialGrid`, **maptools**, 187
- `solarnoon` method, **maptools**, 149
- `solarpos` method, **maptools**, 149
- `sp.correlogram`, **spdep**, 283
-
- `sp.mantel.mc`, **spdep**, 278
- `sp2Mondrian`, **maptools**, 123
- `sp2tmap`, **maptools**, 123
- `sp2WB`, **maptools**, 123
- `Spatial`, **sp**, 29
- `SpatialFiltering`, **spdep**, 317
- `SpatialGrid`, **sp**, 49, 50
- `SpatialGridDataFrame`, **sp**, 50, 195
- `SpatialLines`, **sp**, 39, 61
- `SpatialLines2PolySet`, **maptools**, 122
- `SpatialLinesDataFrame`, **sp**, 39
- `SpatialLinesLengths`, **sp**, 137
- `SpatialPixels`, **sp**, 51, 53, 61
- `SpatialPixelsDataFrame`, **sp**, 51
- `SpatialPoints`, **sp**, 31, 88, 353
- `SpatialPointsDataFrame`, **sp**, 33
- `SpatialPolygons`, **sp**, 43, 61
- `SpatialPolygons2PolySet`, **maptools**, 122
- `SpatialPolygonsDataFrame`, **sp**, 44, 45
- `spautolm`, **spdep**, 292, 294, 299, 300, 302
- `spCbind` method, **maptools**, 95
- `spDistsN1`, **sp**, 88
- `spkernel2d`, **splancs**, 187, 195
- `spmap.to.lev`, **sp**, 70
- `spplot` method, **sp**, 60, 70–72, 78, 82, 214
- `spplot.locator`, **sp**, 78
- `spreg`, **sphet**, 311–313
- `spsample` method, **sp**, 146, 239
- `spTransform` method, **rgdal**, 66, 88, 95, 110, 111, 132
- `stConstruct`, **spacetime**, 156
- `stConstruct`, **spacetime**, 156
- `STIDF`, **spacetime**, 97
- `stone.stat`, **DCluster**, 356
- `stone.test`, **DCluster**, 356
- `stplot`, **spacetime**, 97
- `subset.listw`, **spdep**, 273
- `subset.nb`, **spdep**, 267, 273
- `sunriset` method, **maptools**, 149

- `surf.ls`, **spatial**, 217
- `Sweave`, **utils**, xi
- `sx`, **BayesX**, 344–346, 359
- `sx`, `sx`, 335
- `system`, **base**, 113, 125
- `Szero`, **spdep**, 282

- `tango.test`, **DCluster**, 352
- `terrain.colors`, **grDevices**, 79
- `test_package`, **testthat**, 132
- `topo.colors`, **grDevices**, 79
- `Tps`, **fields**, 258
- `transition`, **gdistance**, 122
- `tribble`, **splancs**, 203, 204
- `trW`, **spdep**, 309

- `unionSpatialPolygons`,
 maptools, 110
- `unmark`, **spatstat**, 195, 197, 209

- `variog`, **geoR**, 227
- `variogram`, **gstat**, 220, 221, 223,
 225, 228–232, 255

- `vcovHC`, **sandwich**, 304
- `vect2neigh`, **spgrass6**, 118
- `vgm`, **gstat**, 224–226, 228, 229,
 231, 232, 248, 251, 255
- `vInfo`, **spgrass6**, 116

- `write.nb.gal`, **spdep**, 273
- `write.sn2dat`, **spdep**, 274
- `write.sn2gwt`, **spdep**, 274
- `writeAsciiGrid`, **maptools**, 108
- `writeGDAL`, **rgdal**, 103
- `writeOGR`, **rgdal**, 96, 98, 110
- `writeRAST6`, **spgrass6**, 116, 136
- `writeSpatialShape`, **maptools**,
 99
- `writeStart`, **raster**, 55
- `writeStop`, **raster**, 55
- `writeValues`, **raster**, 55
- `writeVECT6`, **spgrass6**, 116

- `zerodist`, **sp**, 244