DSC 2003 Working Papers
(Draft Versions)

http://www.ci.tuwien.ac.at/Conferences/DSC-2003/

Approaches to classes for spatial data in R

Roger Bivand[*]

Abstract

Access to well-structured and sometimes self-describing spatial position data with associated data attributes in geographical scales domains is increasing, and is expected to increase further. Until recently, it has often been sufficient to treat data sets as autonomous, dropping positional metadata attributes for analysis and visualization. It may be argued that this is short-sighted, because positional data from different sources may not then be readily co-registered. This contribution will survey the representation of positional spatial data in contributed packages to R, and suggest which alternatives exist, or might be implemented, to provide underpinnings that, on the one hand, should make it more convenient to ingest positional and attribute data for analysis, and, on the other, to attempt to retain positional metadata so that the results of analysis can also be utilized in other software contexts.

1 Introduction

There is now a widening range of contributed packages for analysing spatial data in R, including interface packages to geographical information systems. This raises the question of whether some interface data structures would be useful to ensure coherent preservation of spatial metadata. At present, some do not use classes — assuming specific list structures for spatial data, others use old-style classes specific to the package (also known as S3 classes, described in Chambers and Hastie (1993), appendix A, pp. 455–480), and a few have begun to use new-style classes (S4 classes, described in Chambers (1998), chapters 7–8 and Venables and Ripley (2000), chapter 5). It is perhaps unfair to set firm boundaries around spatial statistical analysis, which is more general than the geographical data setting used here. This focus concerns data with a location in geographical space on or near the surface of the earth, and where the distribution of observations is at scales typically considered geographical.

This survey will first describe how the available contributed analysis and interface packages handle spatial data, including metadata, and discuss the simplest steps that might be taken to help them inter-operate.

[*]Economic Geography Section, Department of Economics, Norwegian School of Economics and Business Administration, Bergen, Norway

This will be extended to examine which kinds of design criteria might be used for spatial data classes, for point data, raster data, and finally vector data. In particular, the needs of data in which the three spatial dimensions and time are acknowledged will be highlighted, so that the fundamental classes build on a 3D+time view of spatial phenomena, rather than starting from a static 2D view. This admittedly does involve redundancy where the data are just 2D and when time is unimportant for analysis, but may help admit work in domains such as water bodies and the atmosphere.

Two further goals will be to set the class design criteria in the context of possible requirements for changes in broadly-understood projection (projection, spheroid, datum), and for line generalization in connection with visualization. Both of these are components in a `map()` function, but ought to be resolved in a more general way. Techniques for constructing vector topologies (in 2D) on the fly for visualization and other purposes are also relevant here. There is considerable interest in these topics, but some forms of collaborative harmonization of solutions appear desirable, to which this paper should contribute.

2 Existing packages

Current contributed packages with spatial statistics applications include:

point patterns: **VR:spatial** (CRAN), **splancs** (CRAN), **spatstat** (CRAN)

geostatistics: **VR:spatial** (CRAN), **sgeostat** (CRAN), **fields** (CRAN), **geoR** (CRAN), **geoRglm** (CRAN), **gstat** (CRAN), **RandomFields** (CRAN)

lattice/area data: **spdep** (CRAN)

others: **GDAL**, **pixmap** (CRAN), **GRASS** (CRAN), **maps**, **RArcInfo** (CRAN), **Rmap**, **maptools**, **akima** (CRAN), **tripack** (CRAN), **deldir** (CRAN)

of which a few are not currently on CRAN, either because they are under development, or depend on libraries that are distributed separately. They vary a good deal in maturity, background and in level of documentation. There are a number of broad categories, both as given above, and in relation to the use of classes in these contributions. Class use appears to be pre-S3 (no classes), pre-S3 with retrofitted classes to some of the code, S3 style classes by design, or by design and accident, and S4 classes by design. Many packages appear to have been written for teaching purposes, and/or as expositions of methods of analysis, but have been adapted to applied research with maturity.

A further classification is between packages that are ports of legacy compiled code to R, of ports of S or S-PLUS code and accompanying legacy compiled code to R, those that have been written for R with no reference to other implementations of S, and those which are primarily interfaces to libraries of compiled code. Each of these will have different approaches to object classes, particularly if the work upon which they build uses a class abstraction of some kind. A general aside: looking at code in this way is only possible when it is actually available as source, open for review.

2.1 Point pattern analysis

The compiled and interpreted code in **spatial**[1], and in **splancs**[2] was written before S3 classes were introduced, and for this reason use no classes by design. The second release

[1]part of the **VR** bundle, a recommended package in R, and associated with Venables and Ripley (2002), chapter 15, written by Venables and Ripley, R port by Ripley, following earlier work by Gebhardt.

[2]Rowlingson and Diggle (1993), written by Rowlingson and Diggle, adapted and packaged for R by Bivand.

of **splancs** introduced a `"ribfit"` class in a very few places, and the porter introduced a `"khat"` class to distinguish two types of objects returned by a function, but neither package uses classes in any structured way.

Both of these packages use lists to hold point location data, **spatial** using a list with three named compoments, `x`, and `y` — the planar coordinate values, and `area`, which is a vector of four named numeric values, the lower and upper x and y values of the rectangular domain used for edge correction; this region is set in the underlying compiled code using `ppregion()`. In the case of **splancs**, the point coordinates are the `x` and `y` components of a list, but because happily data frames also permit access to the same named components, functions treat lists with named `x` and `y` components and data frames including column names `x` and `y` as equivalent — these are termed ”points” data sets, and are typically made using `as.points()`. In **splancs**, a further use of ”points” data sets is to express bounding polygons used for edge correction. In addition, **splancs** allows for the marking of ”points” data sets by separate vectors of times, with their own range windows, and by using two ”points” data sets to represent two types of points.

The **spatstat**[3] package is S3 class based, with fundamental classes `"ppp"` and `"owin"`. The `"owin"` class is included as component `"window"` — a rectangle, one or more polygons ordered as in `polygon()`in **base**, or a pixel/raster cell mask — in the `"ppp"` class. This class is a list with a window component of class `"owin"`, an `"n"` component for the number of points, `x` and `y` components for the point coordinates on the plane, and (optionally) a numeric or factor mark component attached to each point. Other classes are used for derived and fitted objects, such as the `"ppm"` — point process model class. A class for grey value pixel images, `"im"`, is also provided, although it does not seem to be fully elaborated in the current release. Methods are provided for coercing, printing, and plotting these classes, predicting from `"ppm"` objects, and package-specific generic functions for rotation and affine transformation, the package also uses formulae abstractions for modelling.

2.2 Geostatistics and similar packages

The **spatial**[4] package provides trend surface and kriging functions, based on arguments for location (`x`, `y`) and attribute value (`z`), or a data frame with columns `x`, `y`, and `z`. An `S3` class `"trls"` has been provided for the fitted object returned by the trend surface fitting function, `surf.ls()`, but otherwise there are no classes by design.

In contrast, **sgeostat**[5] is by design based on `S3` classes, starting from a `point()` function changing the names of the columns to be the coordinate position values to `x` and `y`, and adding class `"point"` to the existing class of the data frame. The remaining classes, such as `"pair"` and `"variogram"` are then derived after checking that the input data are of class `"point"`. Methods are provided for printing and plotting these objects.

The **fields**[6] package also uses `S3` classes, chiefly to hold fitted objects. These classes are furnished with appropriate summary, print, and plot methods. The model fitting functions do not in general use any classes their arguments may possess. In particular `rdist()` and `rdist.earth` for planar and latitude-longitude coordinates respectively often are used within `if() else` tests for function arguments, typically `lat.lon`. Had the coordinate

[3] `http://www.maths.uwa.edu.au/~adrian/spatstat.html`, written by Baddeley and Turner.

[4] see footnote 1.

[5] `S` original by Majure, `R` port and extensions by Gebhardt, `http://www.gis.iastate.edu/SGeoStat/homepage.html`

[6] `http://www.cgd.ucar.edu/stats/Software/Fields`, names of authors cited on homepage.

matrix, typically n by 2, but not restricted to 2D, been self-describing, the appropriate distance function could have been chosen by method dispatch.

The model-based **geoR**[7] and **geoRglm**[8] packages make similar extensive use of `S3` classes for model fits. Both also use an `S3` `"geodata"` class for input data, returned by the `as.geodata()` function, selecting two coordinate columns, one data column, and possible covariate columns from a matrix or data frame. The `"geodata"` objects are lists with at least `"coords"` and `"data"` components, others, including a `"borders"` component and various attributes may be present. The `"coords"` and `"borders"` components are matrices with two columns, `"borders"` ordered as in `polygon()` in **base**.

A different approach is used in **gstat**[9] to input data classes - the properties of `S3` data frames and formulae are used to specify locations by name from a data frame, with the same column names required for prediction from a newdata data frame. `S3` classes are also used comprehensively for derived objects. This provides considerable flexibility in the choice of styles of kriging. The workhorse class is `"gstat"`, which is provided with a powerful `predict.gstat()` method.

Finally, **RandomFields**[10] does not seem to use classes. The locations are used as a matrix with up to three columns, or as up to three numeric vectors. An aside not related to the use of classes is that, quite understandably, these packages tend to mask each others' functions, in particular `plot.variogram()` functions, which exist in **sgeostat**, **geoR**, and **gstat** because `"variogram"` is used as a `S3` class name in each of these packages. Used separately, this is not a serious problem, but does raise issues for package namespaces (and for the ownership of classes across packages).

2.3 Lattice/area data analysis

The lattice/area style of spatial data analysis is only supported directly by one package, **spdep**[11]. Use is made of `S3` classes, including their use in relation to **tripack**, on which this package depends for the construction of some types of spatial neighbours weighting schemes. However, input location coordinates are used, as in the other packages reviewed above, as pairs of numeric vectors, or as a two-column matrix, with no other constraints imposed. Since the spatial neighbours objects of class `"nb"` exist apart from the spatial observation attribute data, a simple test can now be imposed to check that the identities of the objects in the `"nb"` neighbours list correspond to those of the attribute data. While some of the examples include boundary polygons for spatial observations, there is, as yet, no proper class mechanism for accessing point location or polygon boundary data, although some first trials are present in the code.

2.4 Others, including mapping functions and GIS interfaces

We will start with three packages for reading (and possibly writing) image and raster cell data, going on to packages for vector data, and rounding off with a few other packages selected among many. The first package of interest also sets the scene for this discussion, because many of the packages examined here are based on underlying libraries written

[7] `http://www.est.ufpr.br/geoR`, written by Ribeiro and Diggle.

[8] `http://www.maths.lancs.ac.uk/~christen/geoRglm`, written by Christensen and Ribeiro.

[9] `http://www.gstat.org/s.html`, Pebesma and Wesseling (1998), `S` port by the author of Pebesma's Gstat program.

[10] `http://www.geo.uni-bayreuth.de/~martin/R/RandomFields`, written by Schlather.

[11] written by Bivand with contributions from Lewin-Koh, Tiefelsdorf, and Ono.

by Frank Warmerdam and used widely not only in Free Software projects. The **GDAL**[12] package contains bindings in R to the Geospatial Data Abstraction Library (GDAL). The package as written against Rversion 1.5.0, and the corresponding **methods** package, used `S4` classes to initiate and handle the drivers and files used in GDAL, chiefly to contain external pointers returned by GDAL. In general, data is only read on demand, and can be subsetted. Unfortunately, changes in **methods** at 1.6.0 and later broke **GDAL**, so I have backported it to `S3` classes. In both cases the classes are used as containers for handles — which may be seen as proxies for the data, not for the data objects as such. The **pixmap** package uses `S4` classes the other way round; the interpreted code using connections to read PNM images does not use classes, but `S4` classes are used for the data once it has been read, or before it is written.

While **GDAL** can be used to access many kinds of data in a loose-coupled fashion, the **GRASS**[13] package is designed to permit the tight-coupled interfacing of the GRASS geographical information system with R for raster and sites (point location) data. Use is made of an `S3` class — `"grassmeta"` — to pass metadata between systems, and to provide wrapper functions for surface interpolation with the current region and raster parameter values in use in GRASS. The issue raised here is that it may perhaps not be important that location metadata, including current window, resolution, projection, etc., are lost when data are moved out of a GIS, but it does matter if the data are to be moved back. It is not immediately obvious how such metadata could be made “viral”, so that, say, `predict.gam()` would return an object with the matadata attributes of its argument `newdata`, without using wrapper functions (another package facing this issue is **grasper**).

No use of classes is evident in **maps**[14], in which coordinates are expected to be list components `x` and `y`. Augmenting the map database structure with user data is not for the fainthearted. A much more recent package based on a relatively common and topology-based vector GIS file format is **RArcInfo**[15], it does not however use classes as such, using data frames to contain data taken directly from the original ArcInfo structures. Usefully, it contains utilities to convert between `E00` files and binary files.

The unpublished **maptools** package by Nicholas Lewin-Koh uses `S3` classes extensively to interface with Frank Warmerdam’s Shapelib[16] for accessing ArcView `shp` files and their associated `dbf` files. A `"Map"` class contains a `"ShapeList"` of `"Shape"`s and a data frame; the classes contain data.

In the **Rmap**[17] package, the `S3` classes used are more like proxies pointing at data in external files, also permitting subsetting, as in **GDAL**. In fact **Rmap** provides, though OGR, a generalization of **RArcInfo** and **maptools**, because both ArcInfo binary coverages and ArcView `shp` files, as well as many other formats, can be read. Not all the formats as yet plot correctly, for instance ArcInfo binary coverages. The OGR library also supports the creation of georeferenced output in `shp` and Mapinfo format files, and in

[12] `http://keittlab.biosci.utexas.edu/R/GDAL/`, written by Timothy Keitt, based on Frank Warmerdam’s `http://www.remotesensing.org/gdal/`. GDAL can read many image and raster cell formats, including georeferencing for many, and can write a subset of these.

[13] `http://grass.itc.it/statsgrass/index.html`, Bivand (2000), `http://www.ci.tuwien.ac.at/Conferences/DSC-2001/Proceedings/Bivand.pdf`.

[14] `ftp://ftp.mcs.vuw.ac.nz/pub/statistics/map/`, original `S` code by Becker and Wilks, `R` version by Brownrigg and Murdoch

[15] `http://matheron.uv.es/~virgil/Rpackages/RArcInfo`, written by Gmez-Rubio, and based on the Avcelib library written by Daniel Morissette.

[16] `http://gdal.velocet.ca/projects/shapelib/`.

[17] `http://www.maths.lancs.ac.uk/Software/Rmap/`, written by Barry Rowlingson, based on Shapelib (footnote 16), Frank Warmerdam’s OGR Simple Features Library: `http://gdal.velocet.ca/projects/opengis/index.html` — which is part of his GDAL software, and Warmerdam’s projection library: `http://www.remotesensing.org/proj`.

PostGIS/PostgreSQL and Oracle Spatial databases.

Finally, none of the **akima** interpolation package, or the **tripack** and **deldir** triangulation packages, use classes in respect of the input coordinates, although they use some `S3` classes for derived objects.

3 Overlap between packages

There are several salient areas of overlap in structural, rather than functional, terms between these packages, and indeed other packages that have not been surveyed here. Firstly, classes, where used, are most often employed for derived or fitted objects, not as requirements on input data, except in the most general terms (data frames, numeric). In addition, it appears that the `S3` classes used have grown incrementally rather than been designed, and often serve to simplify the most common methods: `print`, `plot`, and `summary` by method dispatch. Secondly, many of the packages implement skeleton map display functions, duplicating effort that, if concentrated, could yield a choice of generic display functions. Only **gstat** appears to use grid/lattice graphics. Finally, the packages use often similar informal ways of treating input geographical coordinate data, such as lists with `x` and `y` components.

4 Proposals to provide data class foundations for analysis packages

Although it is obvious that many of the functions available for statistical data analysis, modelling, and visualization in these packages can be used for spatial data not specifically within the geographical domain, some way of supporting data with geographical metadata, perhaps by analogy with the handling of date/time data, would be valuable. This applies particularly for visualization, for merging data sets with georeferencing, and for export to other software that could use such referencing. It is also associated with the establishing some mechanisms for integrity checking, to ensure that the positional and attribute data of a (geographical) object remain linked to one another.

It can also be argued that this is a more general issue, affecting other software development communities, and that some of the pieces needed are already in place. The abstractions used in GDAL and the OGR library, as well as the proj library, are also used in other projects, such as GRASS and Mapserver. Re-use of external components of this kind, which are commonly used by others and thus benefit from community review and updating, including revisions to access possible future formats, seems preferable when compared to the alternative of writing code from often unclear specifications. On the other hand, it is not as easy to handle the distribution of Rcontributed packages with compiled code when linkage against external libraries is required. Both **GDAL** and **Rmap** rely on `Makevars` files; while for Unix and Linux systems, configuration, compilation, and installation of the necessary static and shared libraries, and header files is readily accomplished with care, distribution of Windows package binary builds depending on external DLLs may be trying, unless suitable models are thought through first.

Figure 1 provides a view of how goals for map visualization might be approached, by modularizing rather than maintaining the model implicit in the **maps** package. As far as spatial statistics packages are concerned, this would mean that they would depend on a package providing `GIS data connections` for their input data, and for outputting

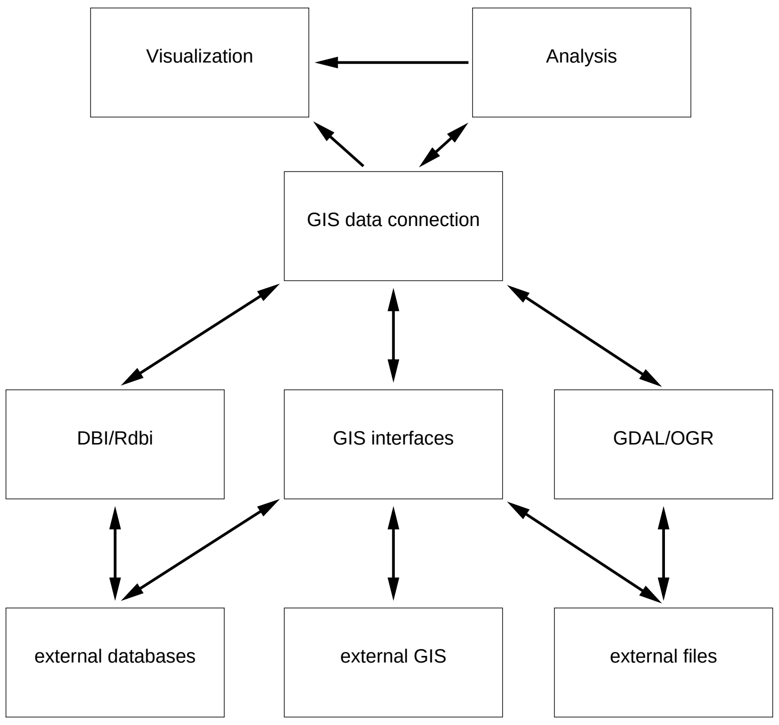

Figure 1: Data flow through a GIS data connection component

spatially expressed reults, such as predicted surfaces, and on a package providing cartographic visualization functions if they wish to display maps. Quite naturally, these facilities should be available on a general basis as packages, and the classes defined in the `GIS data connections` package should have rich `as.` methods, permitting existing point. line or polygon data to be provided with appropriate geographical metadata (such as `NULL`, `NA`, or "I still prefer my own view of my surroundings").

A fruitful model seems to be that of the **DBI** package extended by the various R/DBMS implementations — where at least some of the implementations have been abstracted into GDAL and OGR, although differeing data structures may be returned. This is attractive for another reason, since GIS as geographical databases have been moving towards more general database representations for some time. Within the open source/free software domain, two projects using this model for point, line, polygon, and cell data are the GRASS 5.1 development version[18], and TerraLib[19]. Indeed, given a `"DBIConnection"`, it may be possible to read some GIS data directly. Contrasted with the legacy **maps** package, using a GIS or geodata abstraction offers the advantage of relative ease in introducing and employing user position data. On the other hand, **DBI** is firmly in the realm of `S4` classes, so that relying on incremental development from a classless or initial `S3` style of coding may not be adequate.

An intermediate step may however be to write and merge code from packages accessing GIS data using "`S4`-ready" `S3` classes, not a pretty concept, but maybe a way of combining the relatively greater ease of debugging `S3` class code with the conceptual discipline of

[18]`http://grass.itc.it/grass51/index.html`
[19]`http://www.terralib.org/index.html`

thinking before coding implicit in the S4 model. This is to some extent an area that has been looked at before, to judge from discussions on R-sig-DB — the **Rdbi** package is an S3 implementation of the generic API for database interfaces. An important difference for GIS data is that Frank Warmerdam's work provides a layer of abstraction between the files in the filesystem and the functions using GDAL or OGR, perhaps a little like ODBC. The size and liveliness of the GDAL/OGR user communities also means that access to future formats, and improvements in access to current formats, is available to software using the libraries without the need to track the formats independently.

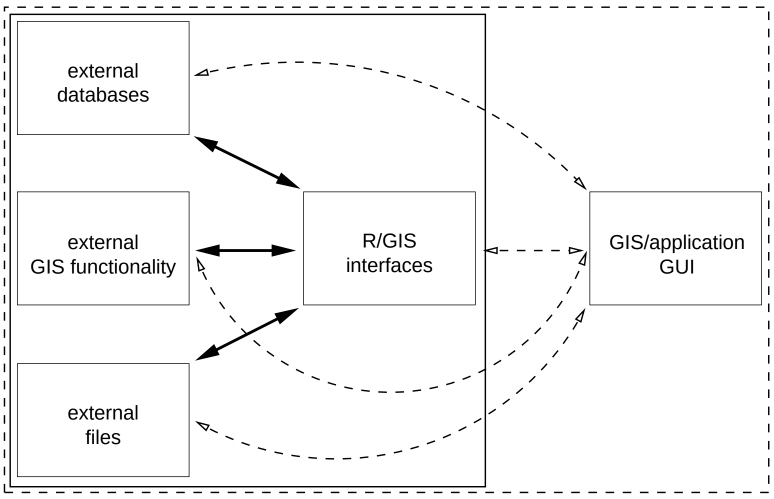

Figure 2: GIS — R links: alternatives

Beyond these requirements, it is still relevant to retain close coupling to GIS among the design criteria, in addition to loose coupling through external files and databases. There are several ways of stacking up the blocks, as shown in Figure 2. For many purposes, putting the R user interface at the top is sufficient, and the required coding is in R and compiled code for dynamically loaded shared libraries. For GRASS, the fact that it is simply an augmented command shell means that R runs among other programs within GRASS, just like other applications executed within that shell environment, and subject to the simple GRASS locking mechanism. However, the most recent release of the **GRASS** interface package permits the essential environment variables to be set from within R, allowing R and the **GRASS** package stand-alone access to the GRASS database, just like Frank Warmerdam's libgrass project (and partly based on his code). As far as TerraLib is concerned, since it is a library, other applications use it, and an R package could be one of these applications. There is also a sample TerraView application, and here the stippled connections on Figure 2 would come into play, as well as a link between the TerraView GUI and R. Finally, very initial contact has been established between ArcGIS and R under Windows using the StatConnector mechanism. The TerraView and ArcGIS cases suggest that classes in the GIS data connection should be able not only to handle "backstore" links, but also "frontside" ones, when R is being used as a server for some client.

In conclusion, the classes appear to map naturally into the data objects offered by GDAL and OGR, because these already abstract from varying GIS realizations. The fundamental types are point, line, and polygon, with extensions to cell — rasters in a data frame format with no NAs — and raster/pixmaps. These could be enlarged from 2D points to 3D points, 2D polygons to 3D objects, points with a time attribute, or perhaps lines as paths, although most of these are hard to represent in GIS. There also remain plenty of attribute issues, such as icons or symbols for points, colour tables, topology for line networks — see the Bioconductor **graph** package, line generalization, and topology for polygons (ArcInfo topology-aware polygons plot in **RArcInfo**, not yet in **Rmap**).

Given this start, a path of lesser resistance will be to explore the formats the interacting loosely or tightly coupled software use by preference, and then whether those representations can be by proxy, loading, reprojecting, and line simplifying on the fly for visualization, or whether they can helpfully be represented in R object space. In order to write to connections, an R object format will also be needed, consequently objects which are otherwise the same in their essence and effects, say for cartographic visualization, may have differing storage status.

Perhaps happily, there is now more standardisation in the ways in which spatial representations are recorded, so that attribute capsules can be designed to accompany location position classes. There is still quite a lot to establish, and it appears clear that complete class specifications will not be available for some time. Even when they are, they (or their metadata attributes) may also need links or pointers to external metadata repositories, for instance recording the specific operational parameters of satellite sensors, or the details of datum specifications. But this is not dissimilar to other data analysis tasks, so that watching and following other interesting solutions may be reasonable.

References

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.

Chambers, J. M. (1998) *Programming with Data*. Springer, New York.

Chambers, J. M. and Hastie, T., editors. (1993) *Statistical Models in S*. Chapman & Hall, New York.

Pebesma, E. J. and Wesseling, C. G. (1998) Gstat: a program for geostatistical modelling, prediction and simulation. *Computers and Geosciences* 24, 17–31.

Rowlingson, B. and Diggle, P. (1993) Splancs: spatial point pattern analysis code in S-PLUS. *Computers and Geosciences* 19, 627–655.

Venables, W. N. and Ripley, B. D. (2000) *S Programming*. Springer, New York.

Venables, W. N. and Ripley, B. D. (2002) *Modern Applied Statistics with S*. Springer, New York.