Journal of Geographical Systems (2021) 23:515–546
https://doi.org/10.1007/s10109-020-00336-0

ORIGINAL ARTICLE

Progress in the R ecosystem for representing and handling spatial data

Roger S. Bivand[1]

Received: 9 October 2019 / Accepted: 8 September 2020 / Published online: 16 October 2020

Abstract
Twenty years have passed since Bivand and Gebhardt (J Geogr Syst 2(3):307–317, 2000. https://doi.org/10.1007/PL00011460) indicated that there was a good match between the then nascent open-source R programming language and environment and the needs of researchers analysing spatial data. Recalling the development of classes for spatial data presented in book form in Bivand et al. (Applied spatial data analysis with R. Springer, New York, 2008, Applied spatial data analysis with R, 2nd edn. Springer, New York, 2013), it is important to present the progress now occurring in representation of spatial data, and possible consequences for spatial data handling and the statistical analysis of spatial data. Beyond this, it is imperative to discuss the relationships between R-spatial software and the larger open-source geospatial software community on whose work R packages crucially depend.

Keywords Spatial data analysis · Open-source software · R programming language

JEL Classification C00 · C88 · R15

1 Introduction

While Bivand and Gebhardt (2000) did provide an introduction to R as a statistical programming language and to why one might choose to use a scripted language like R (or Python), this article is both retrospective and prospective. It is possible that those approaching the choice of tools for spatial analysis and for handling spatial data will find the following less than inviting; in that case, perusal of early chapters of Lovelace et al. (2019) will provide useful context. Two further pointers include the fact that R and most R add-on packages are open-source software and so without licence fees or other such restrictions. The second pointer is that all scripting

✉ Roger S. Bivand
roger.bivand@nhh.no

[1] Department of Economics, Norwegian School of Economics, Helleveien 30, 5045 Bergen, Norway

languages provide the structures needed for reproducible research, and open-source software gives the interested researcher the means to run the scripts needed to replicate results without cost, given access to adequate hardware. Hence, an overview of the development of the use of R for handling spatial data can cast light on how and why steps fashioning today's software were taken. Of course, an overview of the use of R for analysing spatial data would also be tempting, but, with about 900 R packages using spatial data handling classes and objects, would far exceed the bounds of a single article.

The R statistical programming language and environment has been used for handling and analysing spatial data since its inception, partly building on its heritage from S and S-Plus. When the conceptualization of spatial data was introduced in the **sp** package (Pebesma and Bivand 2005, 2020, on the Comprehensive R Archive Network (CRAN) since 2005), it was expected that some packages would adopt its classes. Some years later, adoption rates had picked up strongly, as had use of the **sp**-based packages **rgdal** (Bivand et al. 2020, on CRAN since 2003) for input/output and **rgeos** (Bivand and Rundel 2020, on CRAN since 2011) for geometric manipulation of vector data.

Packages depending on **sp** classes continue to require support as more modern data representations have been introduced in the **sf** (Pebesma 2018, 2020a, on CRAN since 2016) and **stars** (Pebesma 2020c, on CRAN since 2018) packages. The **sf** package provides the data reading and writing, and geometry manipulation functionalities found in **rgdal** and **rgeos**, and an alternative class representation for vector data based on the Simple Features standard (Herring 2011; ISO 2004). The **stars** package adds facilities for handling spatial and spatio-temporal raster and vector data, building in part on work with the **spacetime** package (Pebesma 2012, 2020b, on CRAN since 2010) and on raster handling in the **raster** package (Hijmans 2020a, on CRAN since 2010). The **raster** package will not be discussed in this review, mostly because information in the Github "rspatial" organization (https://github.com/rspatial) repositories suggests that development is in flux and that **raster** is becoming a new package called **terra** (Hijmans 2020b). Discussion here will concentrate on packages developed and maintained by the Github "r-spatial" organization (https://github.com/r-spatial) of which I am a member.

Newer visualization packages, such as **tmap** (Tennekes 2018, 2020, on CRAN since 2014), **mapview** (Appelhans et al. 2020, on CRAN since 2015) and **cartography** (Giraud and Lambert 2016, 2017, 2020, on CRAN since 2015), give broader scope for data exploration and communication. Modelling and analysis packages demonstrate the considerable range of implementations now available and are often supported with additional code provided as supplementary material to journal articles, for example in the Journal of Statistical Software spatial statistics special issue (Pebesma et al. 2015). The availability of software and scripts provides a helpful mechanism supporting reproducible research and hands-on reviewing in which readers can read the code and scripts used in calculating the results presented in published work (see, for example, Sawicka et al. 2018; Lovelace and Ellison 2018; Evangelista and Beskow 2018, in one issue of the R Journal).

Some packages are used by others in turn forming dependency trees; because of these dependencies, we can speak of an ecosystem. Class representations of data

are central, with the data frame conceptualization shaping much of the whole R ecosystem. For the modelling infrastructure to perform correctly, the relationships between objects containing data and the formula representations of models are crucial. Because both **sp** and **sf** provide similar interfaces for the `data=` arguments for model fitting functions by behaving as `data.frame` objects, transition from **sp** to **sf** representations is convenient by design. The **spdep** package (Bivand 2020b, on CRAN since 2002) for exploratory spatial data analysis and the new **spatialreg** package (Bivand and Piras 2019, on CRAN since 2019) for spatial regression (split out of **spdep**) have been adapted to accommodate geometries held in **sf** classes, so both approaches are viable. Other packages, such as **mapview**, **tmap** or **stplanr** (Lovelace et al. 2020) have been revised to permit use with both **sp** and **sf** objects. In order to retain backward compatibility, other central packages may choose to handle the coexistence of **sp** and **sf** classes in the same way.

Developers of new packages should choose to use **sf** and **stars** rather than **sp**, **rgdal** and **rgeos** (and **terra** rather than **raster**), but existing packages are free to adapt, or indeed to stay with **sp**, **rgdal** and **rgeos**, hoping that they may continue to be maintained. Naturally, contributions to maintenance and development from those using software on which one's work depends are among the "prices" to be "paid" for open-source software, so "hope" may involve commitment to take responsibility for maintenance. If a maintainer is unable to continue in service, software like R add-on packages is termed "orphaned", but may be adopted. This occurred very recently with an R linear programming package **lpSolve**, which has been adopted by Gábor Csárdi to widespread relief and gratitude. Because there is no corporation tasked with maintaining most R add-on packages, their future use has to depend on the user community.

One key reason why **sf** is much easier to maintain is that it was written using the **Rcpp** package (Eddelbuettel et al. 2011, 2020; Eddelbuettel 2013; Eddelbuettel and Balamuta 2018, on CRAN since 2008) to interface C++ code in GDAL and now PROJ (both C++11), and GEOS code through the C API, whereas **rgdal** and **rgeos** have more fragile handcrafted interfaces originally written for C99 and C++98. Maintaining **Rcpp**/C++11 interfaces is very much easier than older adaptations not as well understood by younger developers. However, the choice of C++ interfaces is not necessarily robust (Kalibera 2019).

Since there is a viable alternative to **rgdal** and **rgeos**, a fallback in the future for **sp**-dependent packages will be to use **sf** for reading and writing data, and geometry manipulation, and to coerce to **sp** classes before passing to existing modelling code if **sp** classes are needed. Maintainers of actively developed packages using **sp** vector classes intensively and that are also impacted by changes in coordinate reference systems (Sect. 4) are advised to consider transitioning to **sf**, as substantial revisions will be needed anyway.

In this review and prospect, the progress seen over the last 20 years will be described, together with the unexpected consequences it engendered. The emergence of new technologies and standards has also led to the desirability of re-implementation of data representations and analysis functionality, some of which is now in place and which will be described. Changes also impact the open-source libraries on which core spatial functionality is built, leading to challenges in ensuring

backward compatibility. This will be shown using the example of coordinate reference systems. In terms of positionality, much of what follows will be presented from the point of view of the author, documented as far as possible from email exchanges and similar sources. It is more than likely that other participants in the development of the R-spatial community may recall things differently, and of course, I acknowledge that this description is bound to be partial.

2 Spatial data classes in the sp package

In the early and mid-1990s, those of us who were teaching courses in spatial analysis beyond the direct application of geographical information systems (GIS) found the paucity of software limiting. In institutions with funding for site licences for GIS, it was possible to write or share scripts for Arc/Info (in AML), ArcView (in Avenue) or later in Visual Basic for ArcGIS. If site licences and associated dongles[1] used in the field were a problem (including problems for students involved in fieldwork in research projects), there were few alternatives, but opportunities were discussed on mailing lists. One of these was the AI-Geostats listserve/mailing list started by Gregoire Dubois in 1995; AI meant Arc/Info. Another community gathered around GRASS GIS and its transition to open-source development; it hosted, among other things, `src.contrib` and `src.garden` directory trees with analysis tools (see code stored in the https://github.com/OSGeo/grass-legacy repository).

From late 1996, the R programming language and environment began to be seen as an alternative for teaching and research involving spatial analysis by a few people including the author and Albrecht Gebhardt. R uses much of the syntax of S, then available commercially as S-Plus, but was and remains free to install, use and extend under the GNU General Public License (GPL). In addition, it could be installed portably across multiple operating systems, including Windows and Apple Mac OS. At about the same time, the S-Plus SpatialStats module was published (Kaluzny et al. 1998), and a meeting occurred in Leicester in which many of those looking for solutions took part. (My contribution was published in the meeting special issue Bivand 1998.)

Much of the porting of S code to R for spatial statistics was begun by Albrecht Gebhardt as soon as the R package mechanism matured. The `library()` function was upgraded in R 0.60 published in December 1997, and the Comprehensive R Archive Network was operating in January 1998. An exchange between Albrecht Gebhardt and Thomas Lumley on the R-beta mailing list (https://stat.ethz.ch/pipermail/r-help/1997-November/001882.html) shows that the package mechanism was not yet working predictably before 0.60 for contributed packages. Since teachers moving courses from S to R needed access to the S libraries previously used, porting was a crucial step. CRAN listings show **tripack** (Renka and Gebhardt 2020) and **akima** (Akima and Gebhardt 2020)—both with non-open-source

[1] Typically plastic boxes containing static software licences attached to a computer port, often the parallel port otherwise used for printers in the pre-USB era.

licences—available from August 1998 ported by Albrecht Gebhardt; **ash** and **sgeostat** (Majure and Gebhardt 2016) followed in April 1999. The **spatial** package was available as part of **MASS** (the software supporting the four editions of Venables and Ripley 2002), also ported in part by Albrecht Gebhardt. In the earliest period, CRAN administrators helped practically with porting and publication. Albrecht and I presented an overview of possibilities of using R for research and teaching in spatial analysis and statistics in August 1998, subsequently published in this journal as Bivand and Gebhardt (2000).

Rowlingson and Diggle (1993) describe the S-PLUS version of **splancs** (Rowlingson and Diggle 2017) for point pattern analysis. I had contacted Barry Rowlingson in 1997, but only moved forward with porting as R's package mechanism matured. In September 1998, I wrote to him: "It wasn't at all difficult to get things running, which I think is a result of your coding, thank you!" However, I added this speculation: "An issue I have thought about a little is whether at some stage Albrecht and I wouldn't integrate or harmonize the points and pairs objects in **splancs**, **spatial** and **sgeostat**—they aren't the same, but for users maybe they ought to appear to be so". This concern with class representations for geographical data turned out to be fruitful.

A further step was to link GRASS and R, described in Bivand (2000), and followed up at several meetings and working closely with Markus Neteler. The interface has evolved, and its current status is presented by Lovelace et al. (2019, chapter 9). A consequence of this work was that the CRAN team suggested that I attend a meeting in Vienna in early 2001 to talk about the GRASS GIS interface. The meeting gave unique insights into the dynamics of R development, and very valuable contacts. Later the same year Luc Anselin and Serge Rey asked me to take part in a workshop in Santa Barbara, which again led to many fruitful new contacts; my talk eventually appeared as Bivand (2006), but the contacts made at the workshop were very useful. Further progress during the intervening 4 years in the use of R in spatial econometrics was reported in Bivand (2002), building on Bivand and Gebhardt (2000), but preceding the release of the **spdep** package.

During the second half of 2002, it seemed relevant to propose a spatial statistics paper session at the next Vienna meeting to be held in March 2003 (known as Distributed Statistical Computing (DSC) and led to useR! meetings), together with a workshop to discuss classes for spatial data. I had reached out to Edzer Pebesma as an author of the stand-alone open-source program `gstat` (Pebesma and Wesseling 1998); it turned out that he had just been approached to wrap the program for S-Plus. He saw the potential of the workshop immediately, and in November 2002 wrote in an email: "I wonder whether I should start writing S classes. I'm afraid I should". Virgilio Gómez-Rubio had been developing two spatial packages, **RArcInfo** (Gómez-Rubio and López-Quílez 2005; Gómez-Rubio 2011) and **DCluster** (Gómez-Rubio et al. 2005, 2015), and was committed to participating. Although he could not get to the workshop, Nicholas Lewin-Koh wrote in March 2003 that: "I was looking over all the DSC material, especially the spatial stuff. I did notice, after looking through peoples' packages that there is a lot of duplication of effort. My suggestion is that we set up a repository for spatial packages similar to the Bioconductor mode, where we have a base spatial package that has S-4-based methods and

classes that are efficient and general." Straight after the workshop, a collaborative repository for the development of software using SourceForge was established, and the R-sig-geo mailing list (still with over 3600 subscribers) was created to facilitate interaction.

So the mandate for the development of the **sp** package emerged in discussions between interested contributors before, during and especially following the 2003 Vienna workshop; most of us met at Pörtschach am Wörthersee in October 2003 at a meeting organized by Albrecht Gebhardt. Coding meetings were organized by Barry Rowlingson in Lancaster in November 2004 and by Virgilio Gómez-Rubio in Valencia in May 2005, at both of which the class definitions and implementations were stress-tested and often changed radically; the package was first published on CRAN in April 2005. The underlying model adopted was for S4 (new style) classes to be used, for `"Spatial"` objects, whether raster or vector, to behave like `"data.frame"` objects, and for visualization methods to make it easy to show the objects. Package developers could choose whether they would use **sp** classes and methods directly, or rather use those classes for functionality that they did not provide themselves. The **spatstat** package (Baddeley and Turner 2005; Baddeley et al. 2015, 2020) was an early example of such object conversion (known as coercion in S and R) to and from **sp** classes and classes, with the coercion methods published in the **maptools** package (Bivand and Lewin-Koh 2020).

Reading and writing ESRI Shapefiles had been possible using the **maptools** package (Bivand and Lewin-Koh 2020) available from CRAN since August 2003, but **rgdal**, on CRAN from November 2003, interfacing the external GDAL library (Warmerdam 2008) and first written by Tim Keitt, initially only supported accessing and reading raster data. Further code contributions by Barry Rowlingson for handling projections using the external PROJ.4 library and the vector drivers in the then OGR part of GDAL were folded into **rgdal**, permitting reading vector and raster data into **sp**-objects and writing from **sp**-objects. For vector data, it became possible to project coordinates, and in addition to transform them where datum specifications were available. Until 2019, the interfaces to GDAL and PROJ had been relatively stable, and upstream changes had not led to breaking changes for users of packages using **sp** classes or **rgdal** functionalities, although they have involved significant maintenance effort. The final part of the framework for spatial vector data handling was the addition of the **rgeos** package interfacing the external GEOS library in 2011, thanks to Colin Rundell's 2010 Google Summer of Coding project. The **rgeos** package provided vector topological predicates and operations typically found in GIS such as intersection; note that by this time, both GDAL and GEOS used the Simple Features vector representation internally.

By the publication of Bivand et al. (2008), a few packages not written or maintained by the book authors and their nearest collaborators had begun to use **sp** classes. By the publication of the second edition (Bivand et al. 2013), we had seen that the number of packages depending on **sp**, importing from and suggesting it (in CRAN terminology for levels of dependency) had grown strongly. In late 2014, de Vries (2014) looked at CRAN package clusters from a page rank graph and found a clear spatial cluster that we had not expected. Figure 1 shows word clouds with character sizes proportional to pagerank scores for two clusters found in August 2020

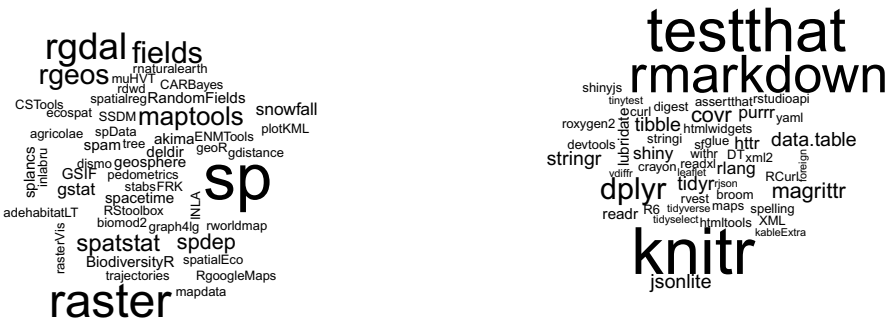

Fig. 1 Wordclouds of CRAN and BioConductor package dependencies, August 2020, left panel: cluster 6 (pagerank range 0.002196–0.000070), right panel: cluster 2 (pagerank range 0.022419–0.000390, **sf** = 0.000972)

among the cumulated packages held on CRAN and those published by the Bioconductor project. The left panel shows cluster 6, the spatial cluster with **sp** having a pagerank of 0.002196, while the right panel shows cluster 2, which is dominated by packages developed by RStudio, a commercial company. The **sf** package is in cluster 2, with a pagerank score of 0.000972, most likely in that cluster because it itself uses many of the functionalities of RStudio packages. The two word clouds are scaled by the largest pagerank of included packages, so the scales differ by almost an order of magnitude.

3 Spatial data classes in the sf and stars packages

The **raster** package complemented **sp** for handling raster objects and their interactions with vector objects. It added to input/output using GDAL through **rgdal**, and better access to NetCDF files. It may be mentioned in passing that thanks to help from CRAN administrators and especially Brian Ripley, CRAN binary builds of **rgdal** for Windows and Apple Mac OSX became available from 2006, but with a limited set of vector and raster drivers. Support from CRAN adminstrators and the maintainers of the Github rwinlib/gdal2 and rwinlib/gdal3 repositories[2] remains central to making packages available to users who are not able to install R source packages themselves, particularly linking to external libraries. Initially, **raster** was written in R using functionalities in **sp** and **rgdal** with **rgeos** coming later. It used a feature of GDAL raster drivers permitting the successive reading of subsets of rasters by row and column, allowing the processing of much larger objects than could be held in memory. In addition, the concepts of bricks and stacks of rasters were introduced, diverging somewhat from the **sp** treatment of raster bands as stacked column vectors in a data frame.

[2] https://github.com/rwinlib/gdal2, https://github.com/rwinlib/gdal3.

As **raster** evolved, two other packages emerged raising issues with the ways in which spatial objects had been conceptualized in **sp**. The **rgeos** package used the C application programming interface (API) to the C++ GEOS library, which is itself a translation of the Java Topology Suite (JTS). While the GDAL vector drivers did use the standard Simple Features representation of vector geometries, it was not strongly enforced. This laxity now seems most closely associated with the use of ESRI Shape files as a de facto file standard for representation, in which some Simple Features are not consistently representable.[3] Both JTS and GEOS required a Simple Features compliant representation and led to the need for curious and fragile adaptations. For example, these affected the representation of **sp** `"Polygons"` objects, which were originally conceptualized after the Shapefile specification: ring direction determined whether a ring was exterior or interior (a hole), but no guidance was given to show which exterior ring holes might belong to. As R provides a way to add a character string comment to any object, such comments were added to each `"Polygons"` object encoding the necessary information. In this way, GEOS functionality could be used, but the fragility of vector representation in **sp** was made obvious.

Another package affecting thinking about representation was **spacetime**, stacking columns for regular spatio-temporal objects with space varying faster than time. So a single earth observation band observed repeatedly would be stored in a single column in a data frame, rather than in the arguably more robust form of a four-dimensional array, with the band taking one position on the final dimension. The second edition of Bivand et al. (2013) took up all of these issues in one way or another, but after completing a spatial statistics special issue of the Journal of Statistical Software (Pebesma et al. 2015), it was time to begin fresh implementations of classes for spatial data.

3.1 Simple Features in R

It was clear that vector representations needed urgent attention, so the **sf** package was begun, aiming to implement the most frequently used parts of the specification (ISO 2004; Kralidis 2008; Herring 2011). Development was supported by a grant from the then newly started R Consortium, which brings together R developers and industry members. A key breakthrough came at the useR! 2016 conference, following an earlier decision to re-base vector objects on data frames, rather than as in **sp** to embed a data frame inside a spatial object. Although data frame objects in S and R have always been able to take list columns as valid columns, such list columns were not seen as "tidy" (Wickham 2014):

[3] For recent examples see https://github.com/OSGeo/gdal/issues/1787 and https://github.com/r-spatial/sf/issues/1121.

```
> df <- data.frame(a=letters[1:3], b=1:3)
> df$c <- list(d=1, e="1", f=TRUE)
> str(df)
'data.frame': 3 obs. of  3 variables:
 $ a: chr  "a" "b" "c"
 $ b: int  1 2 3
 $ c:List of 3
  ..$ d: num 1
  ..$ e: chr "1"
  ..$ f: logi TRUE
```

At useR! in 2016, list columns were declared "tidy", using examples including the difficulty of encoding polygon interior rings in non-list columns. The decision to accommodate "tidy" workflows as well as base-R workflows had already been made, as at least some users only know how to use "tidy" workflows. Pebesma (2018) showed the status of the **sf** towards the end of 2017, with a geometry list column containing R wrappers around objects adhering to Simple Features specification definitions. Note also that from R 4.0.0, `data.frame()` does not convert character columns to `factor` as it did previously (Hornik 2020); column `"a"` is character in R 4 or later, and factor before R 4.

```
> library(sf)
```

The feature geometries are stored in numeric vectors, matrices or lists of matrices and may also be subject to arithmetic operations. Features are held in the `"XY"` class if two-dimensional, or `"XYZ"`, `"XYM"` or `"XYZM"` if such coordinates are available (`"Z"` is usually treated as height and `"M"` as some measure, perhaps accuracy; both need to have specified units); all single features are `"sfg"` (Simple Features geometry) objects, with arithmetic and other operators:

```
> pt1 <- st_point(c(1,3))
> pt2 <- pt1 + 1
> pt3 <- pt2 + 1
> str(pt3)
 'XY' num [1:2] 3 5
```

Geometries may be represented as "Well-Known Text" (WKT):

```
> st_as_text(pt3)
[1] "POINT (3 5)"
```

or as "Well-Known Binary" (WKB) as in database "binary large objects" (BLOBs), resolving the problem of representation when working with GDAL vector drivers and functions, and with GEOS predicates and topological operations:

```
> st_as_binary(pt3)
 [1] 01 01 00 00 00 00 00 00 00 00 00 08 40 00 00 00 00 00 00 14 40
```

A column of Simple Features geometries ("sfc") is constructed as a list of "sfg" objects, which do not have to belong to the same Simple Features category; here, we assign the Web Mercator CRS to indicate that the points are projected to the plane, with position measured in metres:

```
> pt_sfc <- st_as_sfc(list(pt1, pt2, pt3), crs=3857)
> str(pt_sfc)
sfc_POINT of length 3; first list element:  'XY' num [1:2] 1 3
```

When **sf** was written, the **units** package was available (Pebesma et al. 2016, 2020) and could utilize the metric of the coordinates given in the declared CRS, so here inter-point distances are measured in metres:

```
> st_distance(pt_sfc)
Units: [m]
         [,1]     [,2]     [,3]
[1,] 0.000000 1.414214 2.828427
[2,] 1.414214 0.000000 1.414214
[3,] 2.828427 1.414214 0.000000
```

If we re-specify the points as geographical coordinates in decimal degrees on the WGS84 ellipsoid, the distances will be given as metres, but measured over the ellipsoid:

```
> pt_sfc1 <- st_as_sfc(list(pt1, pt2, pt3), crs=4326)
> st_distance(pt_sfc1)
Units: [m]
         [,1]     [,2]     [,3]
[1,]      0.0 156759.1 313424.7
[2,] 156759.1      0.0 156665.6
[3,] 313424.7 156665.6      0.0
```

The most recent R Consortium grant covers the extension of analysis and data handling to global data representation; Pebesma and Dunnington (2020) present a roadmap for the use of the s2 library for operations on geographical coordinates and its integration in **sf**.

```
> mat <- matrix(0, 3, 3)
> mat[1,2] <- mat[2,1] <- s2::s2_distance(st_as_text(pt1), st_as_text(pt2))
> mat[1,3] <- mat[3,1] <- s2::s2_distance(st_as_text(pt1), st_as_text(pt3))
> mat[2,3] <- mat[3,2] <- s2::s2_distance(st_as_text(pt2), st_as_text(pt3))
> mat
         [,1]     [,2]     [,3]
[1,]      0.0 157106.0 314116.3
[2,] 157106.0      0.0 157010.4
[3,] 314116.3 157010.4      0.0
```

Finally, an "sfc" object, a geometry column, can be added to a data.frame object using st_geometry(), which sets a number of attributes on the object and defines it as also being an "sf" object (the "agr" attribute if populated shows how observations on non-geometry columns should be understood):

```
> st_geometry(df) <- pt_sfc
> str(df)
Classes ‘sf’ and ’data.frame’: 3 obs. of  4 variables:
 $ a       : chr  "a" "b" "c"
 $ b       : int  1 2 3
 $ c       :List of 3
  ..$ d: num 1
  ..$ e: chr "1"
  ..$ f: logi TRUE
 $ geometry:sfc_POINT of length 3; first list element:  ’XY’ num  1 3
 - attr(*, "sf_column")= chr "geometry"
 - attr(*, "agr")= Factor w/ 3 levels "constant","aggregate",..: NA NA NA
  ..- attr(*, "names")= chr [1:3] "a" "b" "c"
```

The **sf** package implements all of the Simple Features geometry categories, but some geometries need be converted to be used inside R, with, for example, the gdal_utils() function to convert curve geometries in an input file to linear geometries.[4] Many of the functions in the **sf** package begin with st_ as a reference to the same usage in PostGIS, where the letters were intended to symbolize "spatial type".

The vector file creation and reading functionality found in **rgdal** is also available in **sf**, with substantial improvements with regard to creating spatial database tables and reading from databases. Writing a GeoPackage[5] is as easy or easier than in **rgdal**, as the layer= and driver= arguments can be inferred from the given file name. List columns cannot be written, because R list columns are not bound to be consistently of the same type.

```
> tf <- tempfile(fileext=".gpkg")
> st_write(df, dsn=tf, quiet=TRUE)
Warning message:
In clean_columns(as.data.frame(obj), factorsAsCharacter) :
   Dropping column(s) c of class(es) list
```

The standard reading method is st_read, providing a similar functionality to that in **rgdal**, with a number of differences related to character string encoding that will cease to matter when users migrate to modern formats such as GeoPackage.

[4] util="ogr2ogr", options="-nlt CONVERT_TO_LINEAR".

[5] https://www.ogc.org/standards/geopackage.

```
> df1 <- st_read(dsn=tf, quiet=TRUE)
> df1
Simple feature collection with 3 features and 2 fields
geometry type:  POINT
dimension:      XY
bbox:           xmin: 1 ymin: 3 xmax: 3 ymax: 5
projected CRS:  WGS 84 / Pseudo-Mercator
  a b        geom

1 a 1 POINT (1 3)
2 b 2 POINT (2 4)
3 c 3 POINT (3 5)
```

In a "tidy" workflow, `read_sf()` can be used, returning a `"tibble-sf"` inheriting from a `"tbl_df"` object rather than from a data frame, and not converting character string columns into `"factor"` categorical variables by default (before R 4.0.0):

```
> read_sf(dsn=tf) %>% dplyr::filter(a == "c")
Simple feature collection with 1 feature and 2 fields
geometry type:  POINT
dimension:      XY
bbox:           xmin: 3 ymin: 5 xmax: 3 ymax: 5
projected CRS:  WGS 84 / Pseudo-Mercator
# A tibble: 1 x 3
  a         b        geom
* <chr> <int> <POINT [m]>
1 c         3       (3 5)
```

sf also integrates GEOS topological predicates and operations into the same framework, replacing the **rgeos** package for access to GEOS functionality. The precision and scale defaults differ between **sf** and **rgeos** slightly; both remain fragile with respect to invalid geometries, of which there are many in circulation. From GEOS 3.8, both **rgeos** and **sf** offer functions to create valid geometries from invalid ones; prior to GEOS 3.8, a function in **lwgeom** was used. Most recently, measurements and operations on non-planar geometries have been moved from **lwgeom** to **s2** (Dunnington et al. 2020; Pebesma and Dunnington 2020). Native support for units in **sf** objects carries through to measurements on outputs of topological operations:

```
> buf_df1 <- st_buffer(df1, dist=0.3)
> st_area(buf_df1)
Units: [m^2]
[1] 0.2826142 0.2826142 0.2826142

> plot(st_geometry(buf_df1))
> plot(st_geometry(df1), add=TRUE, pch=4)
```

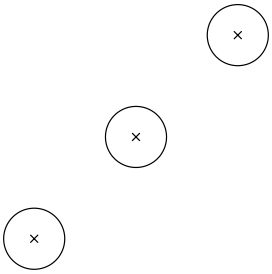

Fig. 2 Three points and three buffers plotted with geometry-specific plot methods for `"sfc"` objects

The **sf** package provides simple base graphics plotting methods. Those for just the `"sfc"` column retrieved by `st_geometry()` do not take over the layout of the graphics device, but for the `"sf"` object, they do, showing multiple non-geometry columns in separate displays. Figure 2 shows a simple plot of the three buffer polygons overplotted with the three points.

3.2 Raster representations

Like **sf**, the **stars** package for scalable, spatio-temporal tidy arrays was supported by an R Consortium grant. Spatio-temporal arrays were seen as an alternative way of representing multivariate spatio-temporal data from the choices made in the **spacetime** package, where a two-dimensional data frame contained stacked observation positions in space within stacked time points or intervals. The proposed arrays might collapse to a raster layer if only one variable was chosen for one time point or interval. More important, the development of the package was extended to accommodate a backend for earth data processing in which the data are retrieved and re-sampled as needed from servers, most often cloud-based servers. In most cases, these would be raster geometries, but the array representation also handles irregular geometries through time. The R Consortium support was chiefly used to let contributors meet to make progress on concepts and implementation.

This example only covers a multiband raster taken from a Landsat 7 view of a small part of the Brazilian coast. In the first part, a GeoTIFF file is read into memory, using three array dimensions, two in planar space, the third across six bands:

```
> library(stars)
> fn <- system.file("tif/L7_ETMs.tif", package = "stars")
> L7 <- read_stars(fn)
> L7
stars object with 3 dimensions and 1 attribute
attribute(s):
  L7_ETMs.tif
 Min.   :  1.00
 1st Qu.: 54.00
 Median : 69.00
 Mean   : 68.91
 3rd Qu.: 86.00
 Max.   :255.00
dimension(s):
     from  to  offset delta                       refsys point values
x       1 349  288776  28.5 UTM Zone 25, Southern Hem... FALSE   NULL [x]
y       1 352 9120761 -28.5 UTM Zone 25, Southern Hem... FALSE   NULL [y]
band    1   6      NA    NA                           NA    NA   NULL
```

The bands can be operated on arithmetically, for example to generate a new object containing values of the normalized difference vegetation index through a function applied across the *x* and *y* spatial dimensions, using the `st_apply` abstraction:

```
> ndvi <- function(x) (x[4] - x[3])/(x[4] + x[3])
> (s2.ndvi <- st_apply(L7, c("x", "y"), ndvi))
stars object with 2 dimensions and 1 attribute
attribute(s):
     ndvi
 Min.   :-0.75342
 1st Qu.:-0.20301
 Median :-0.06870
 Mean   :-0.06432
 3rd Qu.: 0.18667
 Max.   : 0.58667
dimension(s):
  from  to  offset delta                       refsys point values
x    1 349  288776  28.5 UTM Zone 25, Southern Hem... FALSE   NULL [x]
y    1 352 9120761 -28.5 UTM Zone 25, Southern Hem... FALSE   NULL [y]
```

The same file can also be accessed using the proxy mechanism, which creates a link to the external entity, here a file:

```
> L7p <- read_stars(fn, proxy=TRUE)
> L7p
stars_proxy object with 1 attribute in file:
$L7_ETMs.tif
[1] "[...]/L7_ETMs.tif"

dimension(s):
     from  to  offset delta                       refsys point values
x       1 349  288776  28.5 UTM Zone 25, Southern Hem... FALSE   NULL [x]
y       1 352 9120761 -28.5 UTM Zone 25, Southern Hem... FALSE   NULL [y]
band    1   6      NA    NA                           NA    NA   NULL
```

The same function can also be applied across the same two spatial dimensions of the array, but no calculation is carried out until the data are needed and the output resolution known, with the command needed to create the output stored in the object:

```
> (L7p.ndvi = st_apply(L7p, c("x", "y"), ndvi))
stars_proxy object with 1 attribute in file:
$L7_ETMs.tif
[1] "[...]/L7_ETMs.tif"

dimension(s):
     from  to  offset delta                       refsys point values
x       1 349  288776  28.5 UTM Zone 25, Southern Hem... FALSE   NULL [x]
y       1 352 9120761 -28.5 UTM Zone 25, Southern Hem... FALSE   NULL [y]
band    1   6      NA    NA                           NA    NA   NULL
call list:
[[1]]
st_apply(X = X, MARGIN = c("x", "y"), FUN = ndvi)
```

The array object can also be split, here on the band dimension, to yield a representation as six rasters in list form:

```
> (x6 <- split(L7, "band"))
stars object with 2 dimensions and 6 attributes
attribute(s):
      X1              X2              X3              X4
 Min.   : 47.00   Min.   : 32.00   Min.   : 21.00   Min.   :  9.00
 1st Qu.: 67.00   1st Qu.: 55.00   1st Qu.: 49.00   1st Qu.: 52.00
 Median : 78.00   Median : 66.00   Median : 63.00   Median : 63.00
 Mean   : 79.15   Mean   : 67.57   Mean   : 64.36   Mean   : 59.24
 3rd Qu.: 89.00   3rd Qu.: 79.00   3rd Qu.: 77.00   3rd Qu.: 75.00
 Max.   :255.00   Max.   :255.00   Max.   :255.00   Max.   :255.00
      X5              X6
 Min.   :  1.00   Min.   :  1.00
 1st Qu.: 63.00   1st Qu.: 32.00
 Median : 89.00   Median : 60.00
 Mean   : 83.18   Mean   : 59.98
 3rd Qu.:112.00   3rd Qu.: 88.00
 Max.   :255.00   Max.   :255.00
dimension(s):
  from  to  offset delta                       refsys point values
x    1 349  288776  28.5 UTM Zone 25, Southern Hem... FALSE   NULL [x]
y    1 352 9120761 -28.5 UTM Zone 25, Southern Hem... FALSE   NULL [y]
```

These rasters may also be subjected to arithmetical operations, and as may be seen, explicit arithmetic on the six rasters has the same outcome as applying the same calculation to the three-dimensional array:

```
> x6$mean <- (x6[[1]] + x6[[2]] + x6[[3]] + x6[[4]] + x6[[5]] + x6[[6]])/6
> xm <- st_apply(L7, c("x", "y"), mean)
> all.equal(xm[[1]], x6$mean)
[1] TRUE
```

The extension to a gridded temporal dimension or to irregular spatial and temporal entities is not particularly difficult, but it remains to document the possibilities of the package more fully.

3.3 Visualization

The **classInt** package (Bivand 2020a) for finding thematic mapping class intervals is used directly in plot methods in **sf** and **stars**, and in the **tmap** (Tennekes 2018, 2020) and **cartography** (Giraud and Lambert 2016, 2017, 2020) packages. Lapa et al. (2001) (Leprosy surveillance in Olinda, Brazil, using spatial analysis techniques) made available the underlying data set of Olinda census tracts (setor) in the Corrego Alegre 1970-72 / UTM zone 25S projection (EPSG:22525); we will use the data set and the deprivation variable to point to visualization alternatives now available.

```
> olinda <- st_read("olinda.gpkg", quiet=TRUE)
```

The `style=` argument gives the choice of method used for finding the number of classes specified, with `"fisher"` being a natural breaks method, and `"bclust"` bagged clustering from the **e1071** package (Meyer et al. 2019):

```
> library(classInt)
> cI_fisher <- classIntervals(olinda$"DEPRIV", n=7, style="fisher")
> set.seed(1)
> cI_bclust <- classIntervals(olinda$"DEPRIV", n=7, style="bclust")
Committee Member: 1(1) 2(1) 3(1) 4(1) 5(1) 6(1) 7(1) 8(1) 9(1) 10(1)
Computing Hierarchical Clustering
```

For a long time, the **RColorBrewer** package (Neuwirth 2014) colour palettes were the obvious choice for thematic cartography, but more recently other packages, such as **rcartocolor** (Nowosad 2019), have become available, often as supersets of the **RColorBrewer** palettes:

```
> pal <- rcartocolor::carto_pal(7, "SunsetDark")

> plot(cI_fisher, pal, xlab="DEPRIV", ylab="")
> plot(cI_bclust, pal, xlab="DEPRIV", ylab="")
```

Figure 3 shows plots of the two class interval schemes, with obvious differences between the two styles. Both styles are attempting to balance low within-class variance and high between-class variance.

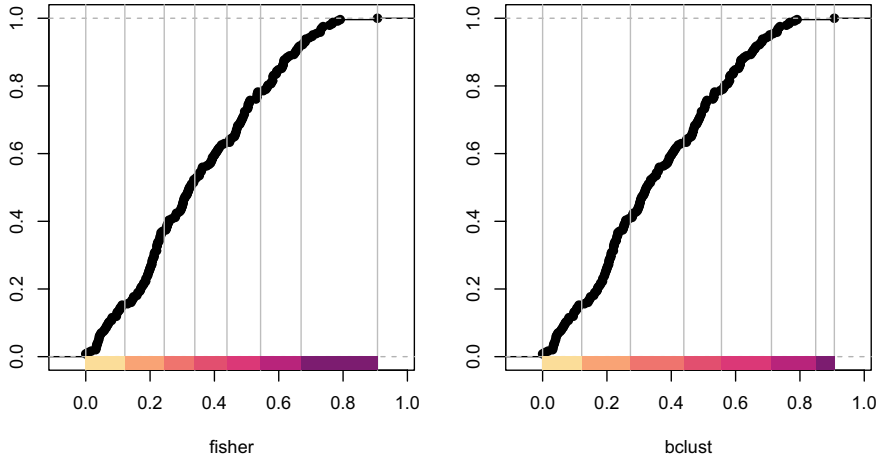

Fig. 3 Deprivation by census tract in Olinda, Brazil; empirical cumulative distribution function and class intervals for two class intervals: left panel: natural breaks; right panel: bagged clustering

Figure 4 shows maps of the same variable with the same class intervals chosen internally using **classInt** functionality, with the same palette. The syntax is a little different, and here, the **sf** `plot` method default key has been turned off to permit side-by-side display. This method is really best at providing glimpses of the data, rather than at creating complete figures, in contrast to the richer functions in **cartography**; both use base graphics.

```
> plot(olinda[,"DEPRIV"], nbreaks=7, breaks="fisher", pal=pal, key.pos=NULL,
+      main="")
> library(cartography)
> choroLayer(olinda, var="DEPRIV", method="fisher-jenks", nclass=7, col=pal,
+            legend.values.rnd=3)
```

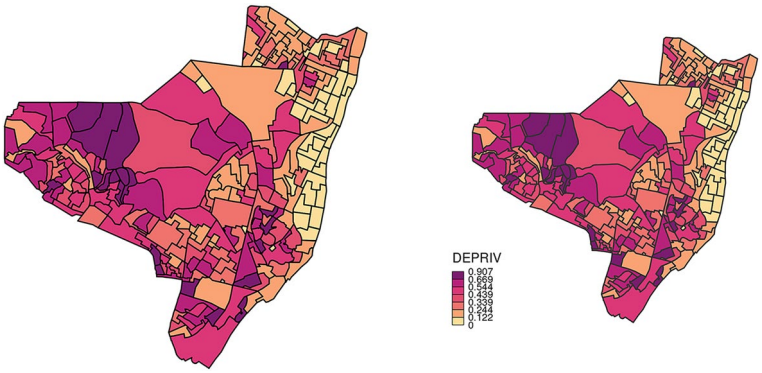

Fig. 4 Deprivation by census tract in Olinda, Brazil, natural breaks class intervals; left panel: **sf** `plot` method; right panel: **cartography** choropleth map

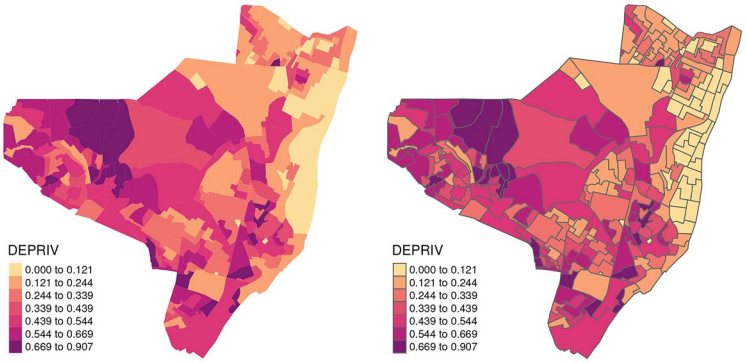

Fig. 5 **tmap** output, natural breaks class intervals; left panel: object `o1` without boundaries; right panel: object `o2` with boundaries

Figure 5 shows how **tmap** functions use grid graphics to permit graphics objects to be updated and then plotted using `tmap_arrange()`. Once again, the same class intervals are chosen internally using **classInt**, with the same palette. The left panel is updated on the right to add thin boundaries between census tracts. **tmap** also offers small multiples of facets, for example thematic maps of the same variable observed at successive time periods using the same class intervals. An extensive discussion of the use of **tmap** is provided by Lovelace et al. (2019).

```
> library(tmap)
> o1 <- tm_shape(olinda) + tm_fill("DEPRIV", style="fisher", n=7, palette=pal)
> o2 <- o1 + tm_borders(lwd=0.8)
```

The **ggplot2** package (Wickham et al. 2020) provides the `geom_sf()` facility for mapping **sf** objects:

```
> library(ggplot2)
> g1 <- ggplot(olinda) + geom_sf(aes(fill=DEPRIV))
> g2 <- g1 + theme_void() + scale_fill_gradientn(colours=pal,
+   breaks=round(cI_fisher$brks, 3))
```

This approach also builds on grid graphics. It is possible to set a theme that drops the arguably unnecessary graticule, but there is a lot of intervention required to get a simple map. To get proper class intervals involves even more work, because the package takes specific, not general, positions on how graphics are observed. ColorBrewer, for example, eschews continuous colour scales based on cognitive research, but **ggplot2** enforces them by default for continuous variables. Figure 6 shows the default choice of palette, updated to remove the unnecessary graticule and to use the user-specified palette and class intervals:

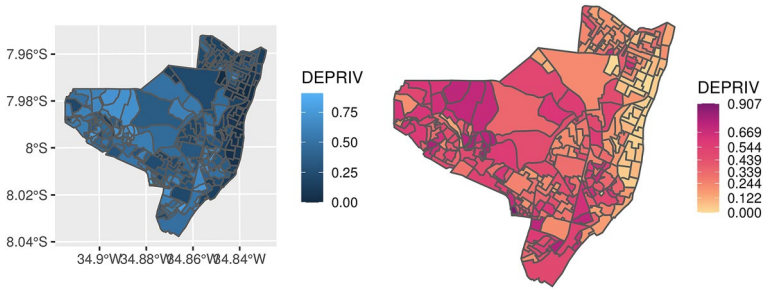

Fig. 6 Thematic maps with **ggplot2**: left panel: default map of a continuous variable; right panel: graticule removed and palette modified

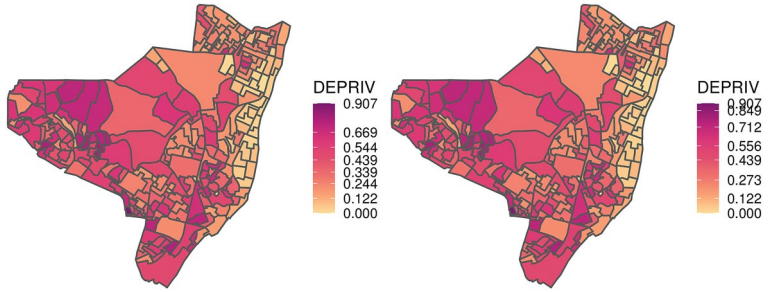

Fig. 7 Thematic maps with **ggplot2**, natural breaks class intervals: left panel: natural breaks class intervals; right panel: bagged clustering class intervals

We can also display the bagged clustering class intervals beside the natural breaks map, again using **ggplot2**; Figure 7 shows the maps, but because a continuous scale is still enforced, all that changes is the position of the breaks on the key.

```
> g3 <- g1 + theme_void() + scale_fill_gradientn(colours=pal,
+   breaks=round(cI_bclust$brks, 3))
```

3.4 Reverse dependencies of the sp and sf packages

R packages can possess forward or reverse dependencies. Forward or upstream dependencies are typically on R itself, a small number of packages whose functionalities are used in the package in question (by loading and attaching the package (dependencies) or just loading the namespace of the package (imports)), and possibly external software libraries. Reverse or downstream dependencies are packages that themselves use the package in question by loading and attaching it, only loading its namespace or using it on demand (suggests). **sp** and **sf** were written carefully to minimize forward dependencies, with **sp** only depending on and importing packages included in every R distribution by default and **sf** adding CRAN contributed packages **Rcpp** and **units** required to build the package and

classInt for class intervals, **DBI** for interfacing spatial databases and **magrittr** for piped operations, where none of these extra forward dependencies draws in many other packages.

In **sp**, the compiled code (written in C) is self-contained and is made available to other packages, chiefly **rgdal** and **rgeos**, to link to their compiled code. **rgdal** links to **sp** and to the external libraries PROJ and GDAL. GDAL itself links to PROJ and can link to GEOS and many other libraries needed for specific drivers. The external software versions used may be reported using `*_extSoftVersion`, here using the `::` operator to avoid attaching the packages being queried:

```
> rgdal::rgdal_extSoftVersion()
          GDAL GDAL_with_GEOS            PROJ              sp
       "3.1.2"          "TRUE"        "7.1.0"        "1.4-4"
```

The versions vary between platforms and by the installation method used; as package maintainer, I often run with pre-release or latest versions of external software to attempt to detect and mitigate changes before they impact users' workflows. For **rgdal**, the versions of GDAL, PROJ and **sp** are reported, together with a test showing whether GDAL was built linking to GEOS, something that affects the behaviour of some drivers. The report for **rgeos** is simpler, only listing the versions of GEOS itself and **sp**.

```
> rgeos::rgeos_extSoftVersion()
   GEOS      sp
"3.8.1" "1.4-4"
```

In the case of **sf**, and because it brings together access to the GDAL and GEOS external libraries through Simple Features representation for vector objects, the external software versions supported are the union of those seen above, omitting linkage to a separate package defining classes for objects. In addition, it reports which API is used for PROJ, either `proj.h` or not (the earlier `proj_api.h`).

```
> sf_extSoftVersion()
          GEOS           GDAL         proj.4 GDAL_with_GEOS     USE_PROJ_H
        "3.8.1"        "3.1.2"        "7.1.0"         "true"         "true"
```

Table 1 shows the structure of reverse dependency counts for **sp** and **sf**. Recursive dependencies traverse through the whole CRAN dependency tree; the first column of the table shows counts of "depends" and "imports" dependencies counted across the whole tree. These split into 1285 only involving **sp**, 232 involving both packages and 65 only involving **sf**. If we additionally include "suggests" dependencies, both packages may be used at least indirectly by all CRAN packages. The two right columns show the same counts, but only for packages' first-order dependencies on **sp**, **sf** or both. We can note that of these for "depends" and "imports" dependencies, 459 only involve **sp**, 63 involve both packages and 121 only involve **sf**. In the first

Table 1 Reverse dependency counts for **sp** and **sf**, August 2020, for recursive and non-recursive reverse dependencies taken as "Depends" and "Imports" only, and with "Suggests"

	Recursive	Recursive w/suggests	Not recursive	Not recursive w/suggests
Sum **sp**	1517	16,619	522	629
Sum **sf**	297	16,619	184	277
Only **sp**	1285	0	459	513
Only **sf**	65	0	121	161
Both	232	16,619	63	116

column, the number of packages only depending on **sf** is less than when we ignore recursive dependencies in the third column, which is packages using **sf** that also use **sp**. The number of packages only using **sf** is encouraging, given that it first entered CRAN in October 2016.

It is also encouraging that a fair number of these packages use both **sp** and **sf**, showing existing packages are preserving legacy workflows, but also opening up for more modern object representations. It takes time and effort to communicate the desirability of migrating from **sp** representations to **sf** and probably **stars**. Although keeping the R code running is feasible, including compiled code not using external software, migration to **sf** is advisable.

4 Upstream software dependencies of the R-spatial ecosystem

When changes occur in upstream external software, R packages using these libraries often need to adapt, but package maintainers try very hard to shield users from any negative consequences, so that legacy workflows continue to provide the same or at least similar results from the same data. The code shown in Bivand et al. (2008, (2013) is almost all run nightly on a platform with updated R packages and external software. This does not necessarily trap all differences (figures are not compared), but is helpful in detecting impacts of changes in packages or external software. It is also very helpful that CRAN servers using the released and development versions of R, and with different versions of external software, also run nightly checks. Again, sometimes changes are only noticed by users, but quite often checks run by maintainers and by CRAN alert us to impending challenges. Tracking the development mailing lists of the external software communities, all open source, can also show how thinking is evolving. However, sometimes code tidying in external software can have unexpected consequences, breaking not **sf** or **sp** with **rgdal** or **rgeos**, but a package further downstream. Bivand (2014) discusses open-source geospatial software stacks more generally, but here we will consider ongoing changes in PROJ and linked changes in GDAL.

We will use the example of the location of the Broad Street pump in Soho, London, related to the 1854 Cholera epidemic and Dr John Snow's intervention Brody et al.

(2000), distributed with **sf** (from version 0.8–1). Although it was known that changes in upstream software would impact workflows, the extent of those impacts became clear using a standard example following upgrading to PROJ 6 and GDAL 3 in 2019:

```
> bp_file <- system.file("gpkg/b_pump.gpkg", package="sf")
> b_pump_sf <- st_read(bp_file, quiet=TRUE)
```

Before R packages upgraded the way coordinate reference systems were represented in early 2020, our Proj4 string representation suffered degradation. Taking the Proj4 string defined in PROJ 5 for the British National Grid, we can see a `+datum=OSGB36` key–value pair. But when processing this input with PROJ 6 and GDAL 3, this key is removed. Checking, we can see that reading the input string appears to work, but the output for the Proj4 string drops the `+datum=OSGB36` key–value pair, introducing instead the ellipse implied by that datum:

```
> proj5 <- paste0("+proj=tmerc +lat_0=49 +lon_0=-2 +k=0.9996012717",
+  " +x_0=400000 +y_0=-100000 +datum=OSGB36 +units=m +no_defs")
> legacy <- st_crs(proj5)
> proj6 <- legacy$proj4string
> proj5_parts <- unlist(strsplit(proj5, " "))
> proj6_parts <- unlist(strsplit(proj6, " "))
> proj5_parts[!is.element(proj5_parts, proj6_parts)]
[1] "+datum=OSGB36"
> proj6_parts[!is.element(proj6_parts, proj5_parts)]
[1] "+ellps=airy"
```

We can emulate the problem seen following the release in May 2019 of GDAL 3.0.0 using PROJ 6, by inserting the degraded Proj4 string into the Broad Street pump object. The coordinate reference system representation is now ignorant of the proper datum specification:

```
> b_pump_sf1 <- b_pump_sf
> st_crs(b_pump_sf1) <- st_crs(st_crs(b_pump_sf1)$proj4string)
```

Why does this matter? For visualization on a web map, for example using the **mapview** package, the projected geometries are transformed to the same WGS84 ellipse and datum (EPSG:4326) that were used in PROJ 4 as a transformation hub. In **leaflet**, these are projected to Web Mercator (EPSG:3857). In `mapview()`, the `sf::st_transform()` function is used, so we will emulate this step before handing on the geometries for display. The projected British National Grid point location is unchanged:

```
> all.equal(st_coordinates(st_geometry(b_pump_sf)),
+           st_coordinates(st_geometry(b_pump_sf1)))
[1] TRUE
```

However, because the one of the objects now has a degraded Proj4 string representation of its coordinate reference system, the output points, apparently transformed identically to WGS84, are now some distance apart, as is also shown in Figure 8:

```
> b_pump_sf_ll <- st_transform(b_pump_sf, 4326)
> b_pump_sf1_ll <- st_transform(b_pump_sf1, 4326)
> st_distance(b_pump_sf_ll, b_pump_sf1_ll)
Units: [m]
         [,1]
[1,] 125.0578
```

Once PROJ 6 and GDAL 3 had stabilized in the summer of 2019, we identified the underlying threat as lying in the advertised degradation of GDAL's `exportToProj4()` function. When reading raster and vector files, the coordinate reference system representation using Proj4 strings would often be degraded, so that further transformation within R (also using GDAL/PROJ functionality) would be at risk of much greater inaccuracy than with PROJ 5 and GDAL 2. Since then, **sf**, **sp** with **rgdal**

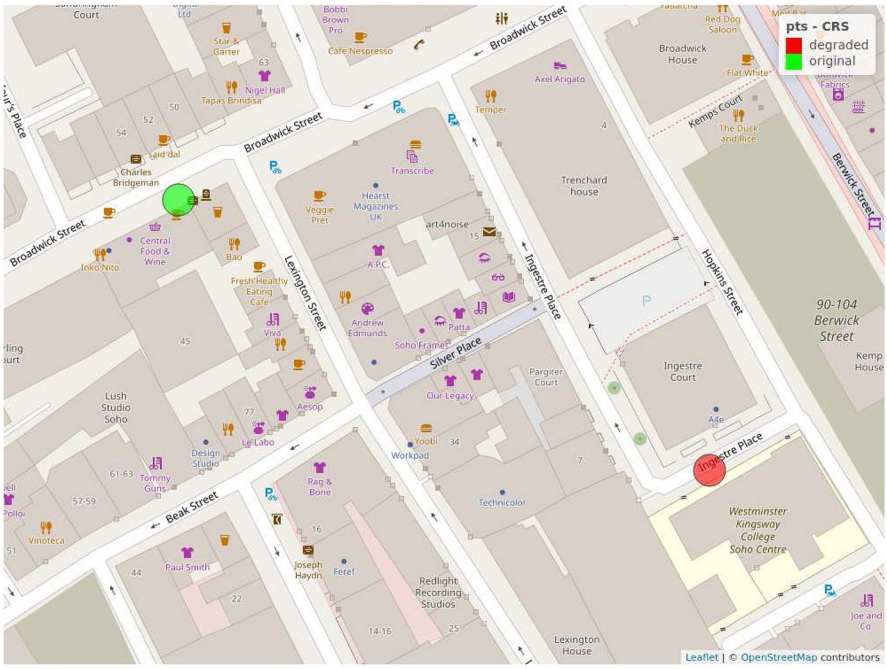

Fig. 8 Displays made using **mapview** displays of the Broad Street pump, with the green point within 2 m of the pump location, and the red point in Ingestre Place because of the loss of the datum specification

and **raster** have adopted the 2019 version of the "Well-Known Text" coordinate reference system representation WKT2-2019 (ISO 2019) instead of Proj4 strings to contain coordinate reference system definitions.[6] Accommodations have also been provided so that the S3 class `"crs"` objects used in objects defined in **sf**, and the formal S4 class `"CRS"` objects used objects defined in **sp** and **raster**, can continue to attempt to support Proj4 strings in addition, while other package maintainers and workflow users catch up.[7]

Following an extended campaign of checking about 900 reverse dependencies (packages depending on **sp**, **rgdal** and others) and dozens of github issues, most of the consequences of the switch to WKT2 among packages have now been addressed. Most recently (late August 2020), 115 packages have been offered rebuilt stored objects that had included `"CRS"` objects without WKT2 definitions.

This approach has ensured that spatial objects, whether created within R, read in from external data sources or read as stored objects, all have WKT2 string representations of their coordinate reference systems, and for backward compatibility can represent these in addition as Proj4 strings. Operations on objects should carry forward the new representations, which should be written out to external data formats correctly. There is a minor divergence between **sf** and **sp** (and thus **rgdal**): In **sf**, the axis order of the CRS is preserved as instantiated, but objects do not have their axes swapped to accord with authorities unless `sf::st_axis_order()` is set `TRUE`. This can appear odd, because although the representation records a northings–eastings axis order, data are treated as eastings–northings in plotting, variogram construction and so on:

```
> st_crs("EPSG:4326")
Coordinate Reference System:
  User input: EPSG:4326
  wkt:
GEOGCRS["WGS 84",
    DATUM["World Geodetic System 1984",
        ELLIPSOID["WGS 84",6378137,298.257223563,
            LENGTHUNIT["metre",1]]],
    PRIMEM["Greenwich",0,
        ANGLEUNIT["degree",0.0174532925199433]],
    CS[ellipsoidal,2],
        AXIS["geodetic latitude (Lat)",north,
            ORDER[1],
            ANGLEUNIT["degree",0.0174532925199433]],
        AXIS["geodetic longitude (Lon)",east,
            ORDER[2],
            ANGLEUNIT["degree",0.0174532925199433]],
    USAGE[
        SCOPE["unknown"],
        AREA["World"],
        BBOX[-90,-180,90,180]],
    ID["EPSG",4326]]
```

In **sp/rgdal**, attempts are made to ensure that axis order is in the form termed GIS, traditional or visualization that is always eastings–northings:

[6] https://www.r-spatial.org/r/2020/03/17/wkt.html.

[7] https://cran.r-project.org/web/packages/rgdal/vignettes/CRS_projections_transformations.html.

```
> library(sp)
> cat(wkt(CRS("EPSG:4326")))
GEOGCRS["WGS 84",
    DATUM["World Geodetic System 1984",
        ELLIPSOID["WGS 84",6378137,298.257223563,
            LENGTHUNIT["metre",1]],
        ID["EPSG",6326]],
    PRIMEM["Greenwich",0,
        ANGLEUNIT["degree",0.0174532925199433],
        ID["EPSG",8901]],
    CS[ellipsoidal,2],
        AXIS["longitude",east,
            ORDER[1],
            ANGLEUNIT["degree",0.0174532925199433,
                ID["EPSG",9122]]],
        AXIS["latitude",north,
            ORDER[2],
            ANGLEUNIT["degree",0.0174532925199433,
                ID["EPSG",9122]]],
    USAGE[
        SCOPE["unknown"],
        AREA["World"],
        BBOX[-90,-180,90,180]]]
```

The probability of confusion increases when coercing from **sf** to **sp** and vice versa, with the representations most often remaining unchanged.[8]

```
> sf_from_sp <- st_crs(CRS("EPSG:4326"))
> o <- strsplit(sf_from_sp$wkt, nl)[[1]]
> cat(paste(o[grep("CS|AXIS|ORDER", o)], collapse=nl))
    CS[ellipsoidal,2],
        AXIS["longitude",east,
            ORDER[1],
        AXIS["latitude",north,
            ORDER[2],
```

```
> sp_from_sf <- as(st_crs("EPSG:4326"), "CRS")
> o <- strsplit(wkt(sp_from_sf), nl)[[1]]
> cat(paste(o[grep("CS|AXIS|ORDER", o)], collapse=nl))
    CS[ellipsoidal,2],
        AXIS["geodetic latitude (Lat)",north,
            ORDER[1],
        AXIS["geodetic longitude (Lon)",east,
            ORDER[2],
```

Both of these coercions are using the same underlying PROJ and GDAL versions, and the same PROJ metadata. Once work in progress is completed, coercions should respect the setting of `sf::st_axis_order()`.

[8] State of https://github.com/rsbivand/sp of 18 August 2020 or later.

It may useful for users to know of other differences between **sf** and **sp/rgdal**. Transformation in **sf** uses code in GDAL, which in turn uses functions in PROJ; in **sp/rgdal**, PROJ is used directly for transformation. In order to demonstrate more of what is happening, let us coerce these **sf** objects to **sp** (they are both planar with an *x*–*y* axis order):

```
> b_pump_sp <- as(b_pump_sf, "Spatial")
> b_pump_sp1 <- as(b_pump_sf1, "Spatial")
```

We will also set up a temporary directory for use with the on-demand grid download functionality in PROJ 7; this must be done before **rgdal** is loaded:

```
> td <- tempfile()
> dir.create(td)
> Sys.setenv("PROJ_USER_WRITABLE_DIRECTORY"=td)
> library(rgdal)
```

In **sf**, areas of interest need to be given by the users, while in transformation and projection in **rgdal**, these are calculated from the object being projected or transformed. The provision of areas of interest is intended to reduce the number of candidate coordinate operations found by PROJ.

```
> WKT <- wkt(b_pump_sp)
> o <- list_coordOps(WKT, "EPSG:4326")
> c(nrow(o), nrow(o[o$instantiable,]), sum(o$number_grids))
[1] 8 7 1
> aoi0 <- project(t(unclass(bbox(b_pump_sp))), WKT, inv=TRUE)
> aoi <- c(t(aoi0 + c(-0.1, +0.1)))
> o_aoi <- list_coordOps(WKT, "EPSG:4326", area_of_interest=aoi)
> c(nrow(o_aoi), nrow(o_aoi[o_aoi$instantiable,]), sum(o_aoi$number_grids))
[1] 5 4 1
```

`rgdal::list_coordOps()` accesses the PROJ metadata database to search through candidate coordinate operations, ranking them by accuracy, returning a data frame of operations. When an area of interest is provided, candidates falling outside it are dropped. Coordinate operations that cannot be instantiated because of missing grids are also listed. We can see here that when an area of interest is not given, 8 candidate operations are found when the WKT string contains datum information. Of these, 7 may be instantiated, with 1 needing a grid (here, the operation that cannot be instantiated). Three operations cease to be candidates if we use an area of interest.

In **sp/rgdal**, the coordinate operation last used is returned and can be retrieved using `rgdal::get_last_coordOp()`; coordinate operations are represented as pipelines (Knudsen and Evers 2017; Evers and Knudsen 2017), introduced in PROJ 5 and using the PROJ key–value pair notation:[9]

[9] `nl` is used to represent the verbatim new line character in the processed article.

```
> b_pump_sp_ll <- spTransform(b_pump_sp, "EPSG:4326")
> cat(strwrap(get_last_coordOp()), sep=nl)
+proj=pipeline +step +inv +proj=tmerc +lat_0=49 +lon_0=-2 +k=0.999601
+x_0=400000 +y_0=-100000 +ellps=airy +step +proj=push +v_3 +step +proj=cart
+ellps=airy +step +proj=helmert +x=446.448 +y=-125.157 +z=542.06 +rx=0.15
+ry=0.247 +rz=0.842 +s=-20.489 +convention=position_vector +step +inv
+proj=cart +ellps=WGS84 +step +proj=pop +v_3 +step +proj=unitconvert
+xy_in=rad +xy_out=deg
```

Here, we can see that an inverse projection from the specified Transverse Mercator projection is made to geographical coordinates, followed by a seven-parameter Helmert transformation to WGS84 ellipsoid and datum. The parameters are contained in the best instantiable coordinate operation retrieved from the PROJ database.

```
> o <- list_coordOps(wkt(b_pump_sp1), "EPSG:4326", area_of_interest=aoi)
> cat(nrow(o), o$ballpark, nl)
1 TRUE
> b_pump_sp1_ll <- spTransform(b_pump_sp1, "EPSG:4326")
> cat(strwrap(get_last_coordOp()), sep=nl)
+proj=pipeline +step +inv +proj=tmerc +lat_0=49 +lon_0=-2 +k=0.999601
+x_0=400000 +y_0=-100000 +ellps=airy +step +proj=unitconvert +xy_in=rad
+xy_out=deg
```

Going on to the case of the degraded representation, only 1 operation is found, with only ballpark accuracy. With our emulation of the dropping of `+datum=` support in GDAL's `exportToProj4()`, we see that the coordinate operation pipeline only contains the inverse projection step, accounting for the observed shift of the Broad Street pump to Ingestre Place.

Finally, **sp/rgdal** may use the provision of on-demand downloading of transformation grids to provide more accuracy (CDN, from PROJ 7).[10] Before finding and choosing to use a coordinate operation using an on-demand downloaded grid, the designated directory is empty:

```
> enable_proj_CDN()
[1] "Using: /tmp/Rtmp5keMtw/file1c4592b3b37c2"
> list.files(td)
character(0)
```

Using the CDN, all the candidate operations are instantiable, and the pipeline now shows a horizontal grid transformation rather than a Helmert transformation.

[10] https://cdn.proj.org.

```
> o <- list_coordOps(WKT, "EPSG:4326", area_of_interest=aoi)
> c(nrow(o), nrow(o[o$instantiable,]), sum(o$number_grids))
[1] 5 5 1
> b_pump_sp_llg <- spTransform(b_pump_sp, "EPSG:4326")
> cat(strwrap(get_last_coordOp()), sep=nl)
+proj=pipeline +step +inv +proj=tmerc +lat_0=49 +lon_0=-2 +k=0.999601
+x_0=400000 +y_0=-100000 +ellps=airy +step +proj=hgridshift
+grids=uk_os_OSTN15_NTv2_OSGBtoETRS.tif +step +proj=unitconvert +xy_in=rad
+xy_out=deg
```

Now the downloaded grid is cached in the database in the designated CDN directory and may be used for other transformations using the same operation.

```
> list.files(td)
[1] "cache.db"
> file.size(file.path(td, list.files(td)[1]))
[1] 319488
> disable_proj_CDN()
```

Once again, the distance between the point transformed from the **sf** object as read from file and the point with the degraded coordinate reference system emulating the effect of the change in behaviour of GDAL's `exportToProj4()` in GDAL 6 and later is about 125 m. Using the CDN shifts the output point by 1.7 m. For confirmation, the output transformed coordinates for the **sp** and **sf** objects using the Helmert transformation are the same.

```
> c(spDists(b_pump_sp1_ll, b_pump_sp_ll),
+   spDists(b_pump_sp_llg, b_pump_sp_ll))*1000
[1] 125.057683   1.751474
> all.equal(unname(coordinates(b_pump_sp_ll)),
+           unname(st_coordinates(st_geometry(b_pump_sf_ll))))
[1] TRUE
```

Although it appears that most of the consequences of the change in representation of coordinate reference systems from Proj4 to WKT2 strings have now been addressed, we still see signs on the mailing list and on Twitter that users, naturally directing their attention to their analytical or visualization work, may still be confused. The extent of the spatial cluster of R packages is so great that it will undoubtedly take time before the dust settles. However, we trust that the operation of upgrading representations is now largely complete. Multiple warnings issued in **sp** workflows, now noisily drawing attention to possible degradations in workflows, will by default be muted when **sp** 1.5 and **rgdal** 1.6 are released.

5 Outlook

As has already been mentioned, either most **sp**-based workflows continue to function using **sf** objects or changes have been made in packages like **spdep** and its modelling counterpart **spatialreg** to permit scripts and packages using these packages to continue to function. Some workflows require more attention than others, but

the transition to **sf** from **sp**, **rgdal** and **rgeos** should be unproblematic. Shifting to new visualization packages like **tmap**, **cartography** and **mapview** should also be relatively easy, and use of **sf** and the new visualization packages should certainly become standard for new research and teaching. It may take a little longer for **stars** to find its place, but work here is continuing, and will stabilize before long. Similar remarks apply to transition from **raster** to **terra**.

The key challenges for handling spatial data using R concern the upstream software libraries in the open-source geospatial software stack, not just PROJ as described above or changes in validity requirements in GEOS and other libraries, but also the opportunities opened up by access to cloud-based earth observation data streams. Because incoming data may take forms as yet not provided for, even where GDAL drivers become available, changes in object representations may become necessary. In particular, this relates to spatio-temporal data, where trajectory data are especially demanding. These data representation challenges are actually opportunities for users and developers to continue cooperating to contribute to making the R ecosystem for handling and analysing spatial and spatio-temporal data even more capable and performant. Finally, please note that a CODECHECK certificate for this paper is available at https://doi.org/10.5281/zenodo.4003848. CODECHECK is an open-science initiative to facilitate sharing of computer programs and results presented in scientific publications (see https://codecheck.org.uk/)

Funding Open Access funding provided by Norwegian School of Economics.

References

Akima H, Gebhardt A (2020) akima: interpolation of irregularly and regularly spaced data. https://CRAN.R-project.org/package=akima, R package version 0.6-2.1

Appelhans T, Detsch F, Reudenbach C, Woellauer S (2020) mapview: interactive viewing of spatial data in R. https://CRAN.R-project.org/package=mapview, R package version 2.9.0

Baddeley A, Turner R (2005) spatstat: an R package for analyzing spatial point patterns. J Stat Softw 12(6):1–42

Baddeley A, Rubak E, Turner R (2015) Spatial point patterns: methodology and applications with R. Chapman and Hall, London

Baddeley A, Turner R, Rubak E (2020) spatstat: spatial point pattern analysis, model-fitting, simulation, tests. https://CRAN.R-project.org/package=spatstat, R package version 1.64-1

Bivand R (1998) Software and software design issues in the exploration of local dependence. The Statistician 47:499–508

Bivand R (2000) Using the R statistical data analysis language on GRASS 5.0 GIS database files. Comput Geosci 26(9):1043–1052
Bivand R (2002) Spatial econometrics functions in R: classes and methods. J Geogr Syst 4:405–421
Bivand R (2006) Implementing spatial data analysis software tools in R. Geogr Anal 38:23–40
Bivand R (2014) Geocomputation and open source software: components and software stacks. In: Abrahart RJ, See LM (eds) Geocomputation. CRC Press, Boca Raton, pp 329–355 (chap 14)
Bivand R (2020a) classInt: choose univariate class intervals. https://CRAN.R-project.org/package=classInt, R package version 0.4-3
Bivand R (2020b) spdep: spatial dependence: weighting schemes, statistics. https://CRAN.R-project.org/package=spdep, R package version 1.1-5
Bivand R, Gebhardt A (2000) Implementing functions for spatial statistical analysis using the R language. J Geogr Syst 2(3):307–317
Bivand R, Lewin-Koh N (2020) maptools: tools for handling spatial objects. https://CRAN.R-project.org/package=maptools, R package version 1.0-1
Bivand R, Piras G (2019) spatialreg: spatial regression analysis. https://CRAN.R-project.org/package=spatialreg, R package version 1.1-5
Bivand R, Rundel C (2020) rgeos: interface to geometry engine—open source ('GEOS'). https://CRAN.R-project.org/package=rgeos, R package version 0.5-3
Bivand R, Pebesma E, Gomez-Rubio V (2008) Applied spatial data analysis with R. Springer, New York
Bivand R, Pebesma E, Gomez-Rubio V (2013) Applied spatial data analysis with R, 2nd edn. Springer, New York
Bivand R, Keitt T, Rowlingson B (2020) rgdal: bindings for the 'geospatial' data abstraction library. https://CRAN.R-project.org/package=rgdal, R package version 1.5-16
Brody H, Rip MR, Vinten-Johansen P, Paneth N, Rachman S (2000) Map-making and myth-making in Broad Street: the London cholera epidemic, 1854. Lancet 356:64–68
de Vries A (2014) Finding clusters of CRAN packages using igraph. https://blog.revolutionanalytics.com/2014/12/finding-clusters-of-cran-packages-using-igraph.html, Revolutions blog. Accessed 25 Aug 2020
Dunnington D, Pebesma E, Rubak E (2020) s2: spherical geometry operators using the S2 geometry library. https://CRAN.R-project.org/package=s2, r package version 1.0.2
Eddelbuettel D (2013) Seamless R and C++ integration with Rcpp. Springer, New York
Eddelbuettel D, Balamuta JJ (2018) Extending R with C++: a brief introduction to Rcpp. Am Stat 72(1):28–36. https://doi.org/10.1080/00031305.2017.1375990
Eddelbuettel D, François R, Allaire J, Ushey K, Kou Q, Russel N, Chambers J, Bates D (2011) Rcpp: seamless R and C++ integration. J Stat Softw 40(8):1–18
Eddelbuettel D, Francois R, Allaire J, Ushey K, Kou Q, Russell N, Bates D, Chambers J (2020) Rcpp: seamless R and C++ integration. https://CRAN.R-project.org/package=Rcpp, R package version 1.0.5
Evangelista PF, Beskow D (2018) Geospatial point density. R J 10(2):347–356. https://doi.org/10.32614/RJ-2018-061
Evers K, Knudsen T (2017) Transformation pipelines for PROJ.4. https://www.fig.net/resources/proceedings/fig_proceedings/fig2017/papers/iss6b/ISS6B_evers_knudsen_9156.pdf, fIG working week 2017 proceedings. Accessed 25 Aug 2020
Giraud T, Lambert N (2016) cartography: create and integrate maps in your R workflow. J Open Source Softw. https://doi.org/10.21105/joss.00054
Giraud T, Lambert N (2017) Reproducible cartography. In: Peterson M (ed) Advances in cartography and GIScience. ICACI 2017. Lecture notes in geoinformation and cartography. Springer, Cham, pp 173–183. https://doi.org/10.1007/978-3-319-57336-6_13
Giraud T, Lambert N (2020) cartography: thematic cartography. https://CRAN.R-project.org/package=cartography, R package version 2.4.1
Gómez-Rubio V (2011) RArcInfo: functions to import data from Arc/Info V7.x binary coverages. https://CRAN.R-project.org/package=RArcInfo, R package version 0.4-12
Gómez-Rubio V, López-Quílez A (2005) RArcInfo: using GIS data with R. Comput Geosci 31(8):1000–1006
Gómez-Rubio V, Ferrándiz-Ferragud J, Lopez-Quólez A (2005) Detecting clusters of disease with R. J Geogr Syst 7(2):189–206

Gómez-Rubio V, Ferrándiz-Ferragud J, López-Quílez A (2015) DCluster: functions for the detection of spatial clusters of diseases. https://CRAN.R-project.org/package=DCluster, R package version 0.2-7
Herring JR (2011) Opengis implementation standard for geographic information-simple feature access-part 1: common architecture. Open Geospatial Consortium Inc, Wayland, p 111
Hijmans RJ (2020a) terra: geographic data analysis and modeling. https://CRAN.R-project.org/package=raster, R package version 3.3-13
Hijmans RJ (2020b) terra: spatial data analysis. https://CRAN.R-project.org/package=terra, R package version 0.8-6
Hornik K (2020) stringsAsFactors. https://developer.r-project.org/Blog/public/2020/02/16/stringsasfactors/index.html. Accessed 25 Aug 2020
ISO (2004) ISO 19125-1:2004 geographic information—simple feature access—part 1: common architecture. https://www.iso.org/standard/40114.html, iSO 19125-1:2004. Accessed 25 Aug 2020
ISO (2019) ISO 19111:2019 Geographic information—referencing by coordinates. https://www.iso.org/standard/74039.html, ISO 19111:2019. Accessed 25 Aug 2020
Kalibera T (2019) Use of C++ in packages. https://developer.r-project.org/Blog/public/2019/03/28/use-of-c-in-packages/index.html, R blog. Accessed 25 Aug 2020
Kaluzny S, Vega S, Cardoso T, Shelly A (1998) S+SpatialStats. Springer, New York
Knudsen T, Evers K (2017) Transformation pipelines for PROJ.4. Geophysical research abstracts, vol 19, EGU2017-8050. https://meetingorganizer.copernicus.org/EGU2017/EGU2017-8050.pdf. Accessed 25 Aug 2020
Kralidis AT (2008) Geospatial open source and open standards convergences. In: Hall GB, Leahy M (eds) Open source approaches in spatial data handling. Springer, Berlin, pp 1–20
Lapa T, Ximenes R, Silva NN, Souza W, Albuquerque MdFM, Campozana G (2001) Vigilância da hanseníase em Olinda, Brasil, utilizando técnicas de análise espacial. Cadernos de Saúde Pública 17:1153–1162. https://doi.org/10.1590/S0102-311X2001000500016
Lovelace R, Ellison R (2018) stplanr: a package for transport planning. R J 10(2):7–23. https://doi.org/10.32614/RJ-2018-053
Lovelace R, Nowosad J, Muenchow J (2019) Geocomputation with R. CRC, Boca Raton
Lovelace R, Ellison R, Morgan M (2020) stplanr: sustainable transport planning. https://CRAN.R-project.org/package=stplanr, r package version 0.6.2
Majure JJ, Gebhardt A (2016) sgeostat: an object-oriented framework for geostatistical modeling in S+. https://CRAN.R-project.org/package=sgeostat, R package version 1.0-27
Meyer D, Dimitriadou E, Hornik K, Weingessel A, Leisch F (2019) e1071: misc functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien. https://CRAN.R-project.org/package=e1071, r package version 1.7-3
Neuwirth E (2014) RColorBrewer: ColorBrewer palettes. https://CRAN.R-project.org/package=RColorBrewer, R package version 1.1-2
Nowosad J (2019) 'CARTOColors' palettes. https://nowosad.github.io/rcartocolor, R package version 2.0.0
Pebesma E (2012) spacetime: spatio-temporal data in R. J Stat Softw 51(7):1–30
Pebesma E (2018) Simple features for R: standardized support for spatial vector data. R J 10(1):439–446
Pebesma E (2020a) sf: simple features for R. https://CRAN.R-project.org/package=sf, R package version 0.9-5
Pebesma E (2020b) spacetime: classes and methods for spatio-temporal data. https://CRAN.R-project.org/package=spacetime, R package version 1.2-3
Pebesma E (2020c) stars: spatiotemporal arrays, raster and vector data cubes. https://CRAN.R-project.org/package=stars, R package version 0.4-3
Pebesma E, Bivand R (2005) Classes and methods for spatial data in R. R News 5(2):9–13
Pebesma E, Bivand R (2020) sp: classes and methods for spatial data. https://CRAN.R-project.org/package=sp, R package version 1.4-2
Pebesma E, Dunnington D (2020) In r-spatial, the Earth is no longer flat. https://www.r-spatial.org/r/2020/06/17/s2.html. Accessed 25 Aug 2020
Pebesma EJ, Wesseling CG (1998) Gstat, a program for geostatistical modelling, prediction and simulation. Comput Geosci 24:17–31
Pebesma E, Bivand R, Ribeiro P (2015) Software for spatial statistics. J Stat Softw 63(1):1–8. https://doi.org/10.18637/jss.v063.i01

Pebesma E, Mailund T, Hiebert J (2016) Measurement Units in R. R J 8(2):486–494. https://doi.org/10.32614/RJ-2016-061
Pebesma E, Mailund T, Kalinowski T (2020) units: measurement units for R vectors. https://CRAN.R-project.org/package=units, r package version 0.6-7
Renka RJ, Gebhardt A (2020) tripack: triangulation of irregularly spaced data. https://CRAN.R-project.org/package=tripack, R package version 1.3-9.1
Rowlingson B, Diggle PJ (1993) Splancs: spatial point pattern analysis code in S-Plus. Comput Geosci 19:627–655
Rowlingson B, Diggle P (2017) splancs: spatial and space-time point pattern analysis. https://CRAN.R-project.org/package=splancs, R package version 2.01-40
Sawicka K, Heuvelink GB, Walvoort DJ (2018) Spatial uncertainty propagation analysis with the spup R package. R J 10(2):180–199. https://doi.org/10.32614/RJ-2018-047
Tennekes M (2018) tmap: thematic maps in R. J Stat Softw 84(6):1–39
Tennekes M (2020) tmap: thematic maps. https://CRAN.R-project.org/package=tmap, R package version 3.1
Venables WN, Ripley BD (2002) Modern applied statistics with S, 4th edn. Springer, New York
Warmerdam F (2008) The geospatial data abstraction library. In: Hall GB, Leahy M (eds) Open source approaches in spatial data handling. Springer, Berlin, pp 87–104
Wickham H (2014) Tidy data. J Stat Softw 59(10):1–23. https://doi.org/10.18637/jss.v059.i10
Wickham H, Chang W, Henry L, Pedersen TL, Takahashi K, Wilke C, Woo K, Yutani H (2020) ggplot2: create elegant data visualisations using the grammar of graphics. https://CRAN.R-project.org/package=ggplot2, R package version 3.3.2