Applied Spatial Data Analysis with R: retrospect and prospect

Roger Bivand

27 September 2020, 11:00-12:00 (with discussion)

Slides and script

The slides and script for parts of the code used are at: https://github.com/rsbivand/whyR20_files

Outline

- Twenty years ago, most labs and universities had little access to tools for spatial data analysis.
- The access they had was mostly closed source and costly.
- We do not know specific adoption rates of open source software for spatial data analysis, nor the shares of R packages.
- However, the proliferation of reverse dependencies on core packages in the Spatial Task View and pagerank analyses do suggest that needs have been met.
- This gives strong incentives both to maintain backwards compatibility, and to adapt to emerging data sources and methods of analysis.

Spatial data

Spatial data typically combine position data in 2D (or 3D), attribute data and metadata related to the position data. Much spatial data could be called map data or GIS data. We collect and handle much more position data since global navigation satellite systems (GNSS) like GPS came on stream 20 years ago, earth observation satellites have been providing data for longer. Here we use **osmdata** (Padgham et al. 2017, 2020) , **mapview** (Appelhans et al. 2020) and **sf** (Pebesma 2018; E. Pebesma 2020a):

```
> suppressPackageStartupMessages(library(osmdata))
> library(sf)
## Linking to GEOS 3.8.1, GDAL 3.1.3, PROJ 7.1.1
> bbox <- opq(bbox = 'bergen norway')
> byb0 <- osmdata_sf(add_osm_feature(bbox, key = 'railway',
+   value = 'light_rail'))$osm_lines
> tram <- osmdata_sf(add_osm_feature(bbox, key = 'railway',
+   value = 'tram'))$osm_lines
> byb1 <- tram[!is.na(tram$name),]
> o <- intersect(names(byb0), names(byb1))
> byb <- rbind(byb0[,o], byb1[,o])
> library(mapview)
> mapview(byb)
```

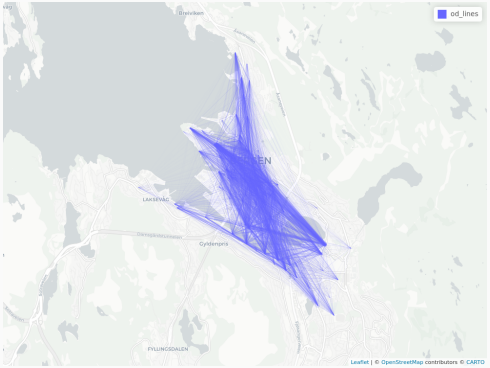
Data handling

We can download monthly CSV files of city bike use, and manipulate the input to let us use the **stplanr** package (Lovelace and Ellison 2018; Lovelace, Ellison, and Morgan 2020) to aggregate origin-destination data. One destination is in Oslo, some are round trips, but otherwise things are OK. We can use CycleStreets to route the volumes onto OSM cycle paths, via an API and API key. We'd still need to aggregate the bike traffic by cycle path segment for completeness.

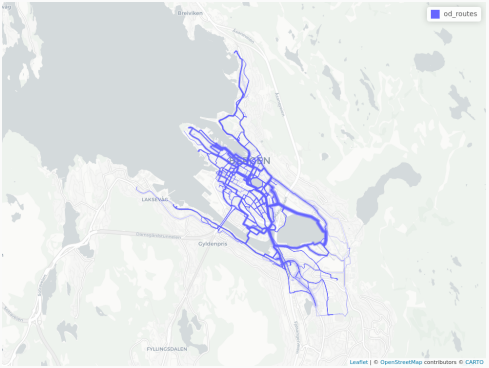
```
> bike_fls <- list.files("bbs")
> trips0 <- NULL
> for (fl in bike_fls) trips0 <- rbind(trips0,
+   read.csv(file.path("bbs", fl), header=TRUE))
> trips0 <- trips0[trips0[, 8] < 6 & trips0[, 13] < 6,]
> trips <- cbind(trips0[,c(1, 4, 2, 9)], data.frame(count=1))
> from <- unique(trips0[,c(4,5,7,8)])
> names(from) <- substring(names(from), 7)
> to <- unique(trips0[,c(9,10,12,13)])
> names(to) <- substring(names(to), 5)
> stations0 <- st_as_sf(merge(from, to, all=TRUE),
+   coords=c("station_longitude", "station_latitude"))
> stations <- aggregate(stations0, list(stations0$station_id),
+   head, n=1)
> suppressWarnings(stations <- st_cast(stations, "POINT"))
> st_crs(stations) <- 4326
> od <- aggregate(trips[,-(1:4)], list(trips$start_station_id,
+   trips$end_station_id), sum)
> od <- od[-(which(od[,1] == od[,2])),]
> library(stplanr)
> od_lines <- od2line(flow=od, zones=stations, zone_code="Group.1",
+   origin_code="Group.1", dest_code="Group.2")
> Sys.setenv(CYCLESTREET="XxXxXxXxXxXxXxXx")
> od_routes <- line2route(od_lines, plan = "fastest")
```

Data handling

Origin-destination lines



Routed lines along cycle routes



Late 1990s spatial data analysis

Teaching and research 20 years ago

- In the early and mid 1990s, those of us who were teaching courses in spatial data analysis beyond the direct application of geographical information systems (GIS) found the paucity of software limiting.
- In institutions with funding for site licenses 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 licenses and associated dongles used in the field were a problem (including students involved in fieldwork in research projects), there were few alternatives, but opportunities were discussed on mailing lists.

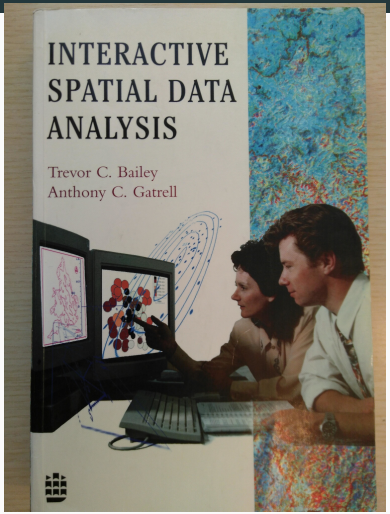
Software for teaching and research

- From late 1996, the R programmimg language and environment began to be seen as an alternative for teaching and research involving spatial analysis.
- 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 MACOS.
- At about the same time, the S-Plus SpatialStats module was published (Kaluzny et al. 1998), and a meeting occurred in Leicester to which many of those looking for solutions took part (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. Since teachers moving courses from S to R needed access to the S libraries previously used, porting was a crucial step.

Packages

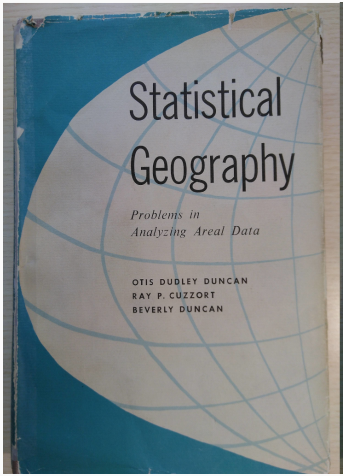
- CRAN listings show **tripack** (Renka and Gebhardt 2020) and **akima** (Akima and Gebhardt 2020) — both with non-open source licenses — available from August 1998 ported by Albrecht Gebhardt; **ash** and **sgeostat** (Majure and Gebhardt 2016) followed in April 1999.
- The **spatial** package was available as part of **MASS** (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 usin R for research and teaching in spatial analysis and statistics in August 1998 (Bivand and Gebhardt 2000).

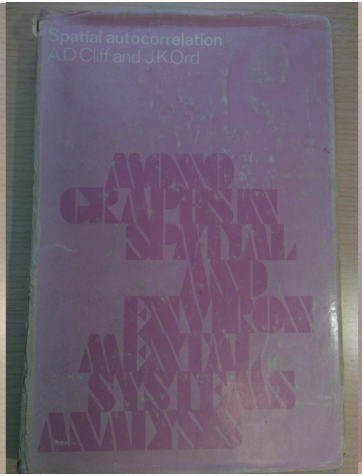
Using R in the computer lab

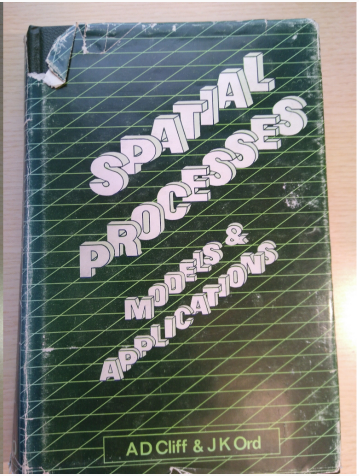


(Bailey and Gatrell 1995)

Books 1961-1981

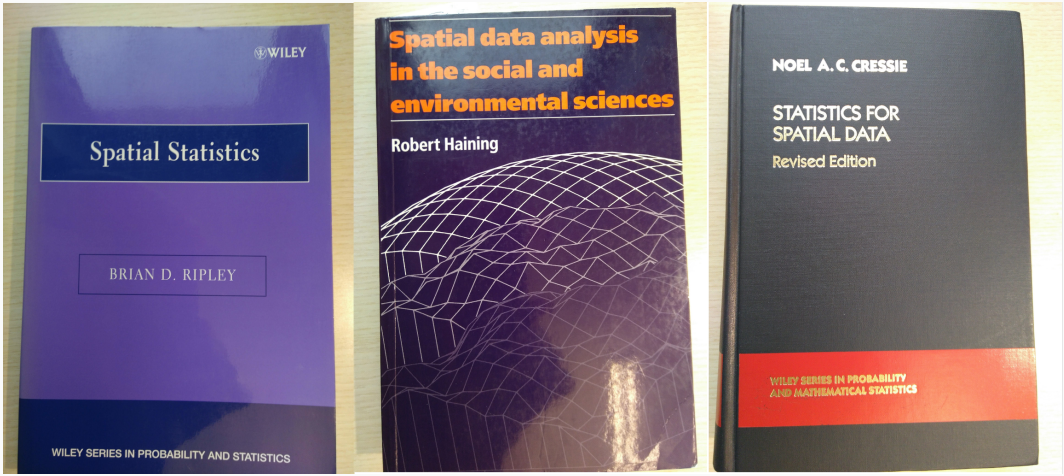






(Duncan, Cuzzort, and Duncan 1961; Cliff and Ord 1973, 1981)

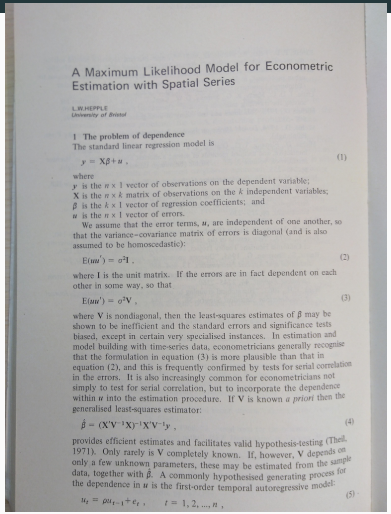
Books 1981-1991



(Ripley 1981; Haining 1990; Cressie 1993)

Intended directions

The need was to cover the sections on spatial autocorrelation and spatial regression in Bailey and Gatrell (1995) for areal data. As with time series, spatial “series” cannot just assume that proximate observations are independent of each other. I’d written Fortan code, and code to create an open source SYSTAT module for testing for spatial autocorrelation and fitting spatial regression models, and some AWK code for for testing for spatial autocorrelation (Bivand 1996, 1997).



A Maximum Likelihood Model for Econometric Estimation with Spatial Series

L.W.HEPPLE
University of Bristol

1 The problem of dependence

The standard linear regression model is

$$y = X\beta + u \,, \tag{1}$$

where
y is the $n \times 1$ vector of observations on the dependent variable;
$\mathbf{X}$ is the $n \times k$ matrix of observations on the k independent variables;
β is the $k \times 1$ vector of regression coefficients; and
u is the $n \times 1$ vector of errors.

We assume that the error terms, u_i, are independent of one another, so that the variance–covariance matrix of errors is diagonal (and is also assumed to be homoscedastic):

$$\mathrm{E}(uu') = \sigma^2\mathbf{I} \,, \tag{2}$$

where $\mathbf{I}$ is the unit matrix. If the errors are in fact dependent on each other in some way, so that

$$\mathrm{E}(uu') = \sigma^2\mathbf{V} \,, \tag{3}$$

where $\mathbf{V}$ is nondiagonal, then the least-squares estimates of β may be shown to be inefficient and the standard errors and significance tests biased, except in certain very specialised instances. In estimation and model building with time-series data, econometricians generally recognise that the formulation in equation (3) is more plausible than that in equation (2), and this is frequently confirmed by tests for serial correlation in the errors. It is also increasingly common for econometricians not simply to test for serial correlation, but to incorporate the dependence within u into the estimation procedure. If $\mathbf{V}$ is known *a priori* then the generalised least-squares estimator:

$$\hat{\beta} = (\mathbf{X}'\mathbf{V}^{-1}\mathbf{X})^{-1}\mathbf{X}'\mathbf{V}^{-1}y \,, \tag{4}$$

provides efficient estimates and facilitates valid hypothesis-testing (Theil, 1971). Only rarely is $\mathbf{V}$ completely known. If, however, $\mathbf{V}$ depends on only a few unknown parameters, these may be estimated from the sample data, together with $\hat{\beta}$. A commonly hypothesised generating process for the dependence in u is the first-order temporal autoregressive model:

$$u_t = \rho u_{t-1} + \epsilon_t \,, \quad t = 1, 2, \ldots, n \,, \tag{5}$$

(Hepple 1976)

Writing software for handling spatial autocorrelation

- There was R software to teach the analysis of spatial point patterns and geostatistical interpolation, but none for spatial autocorrelation or spatial econometrics-style spatial regression
- The need for some standardised way to create graphs of proximate neighbours led to the need for standardising the representation of the spatial observations
- Consequently, early implementations involved learning S3 classes and methods, leading to the `"nb"` and `"listw"` classes for neighbour objects; had they been written later, sparse matrices (**spam** or **Matrix**) could well have been a better choice
- We'll see later how **sp** classes were created

What followed for this subarea

- From my perspective, learning S3 classes for objects needed for testing for spatial autocorrelation and for fitted regression models led to the question of how to write `predict()` methods (Bivand 2002; Goulard, Laurent, and Thomas-Agnan 2017); this led to understanding impacts better.
- **sphet** provided implementations of GMM estimators for cross-sectional models (Piras 2010, 2020), and **splm** for spatial panel models (Millo and Piras 2012, 2020)
- I worked with colleagues comparing measures of spatial autocorrelation (Bivand and Wong 2018), and model fitting approaches (Bivand, Hauke, and Kossowski 2013; Bivand and Piras 2015; Bivand et al. 2017)
- The neighbour objects and measures of spatial autocorrelation have been used across a wide range of packages, in ecology, environmental data analysis, epidemiology and phytogenetics among others

What’s special about spatial data?

Is spatial autocorrelation just poor data collection design and/or model mis-specification?

- Spatial, time series, and spatio-temporal data break the fundamental rule of independence of observations (social networks do too)
- Often, we are not sampling from a known population to get observations, so caution is needed for inference
- Often again, the observation units or entities are not chosen by us, but by others
- Designing spatial samples has become a less important part of the field than previously (Ripley 1981; Müller 2007)

Spatial modelling: why?

- If we want to detect and classify patterns (ML), infer about covariates, or interpolate from a fitted model, we need models that take account of the possible interdependency of spatial, time series and spatio-temporal observations
- Here we are focussing on the spatial case; time differs in having a direction of flow, and spatio-temporal processes are necessarily more complex, especially when non-separable
- We will not look at machine learning issues, although the partition into training and test sets raises important spatial questions for tuning (Schratz et al. 2019)

Spatial modelling: which kinds?

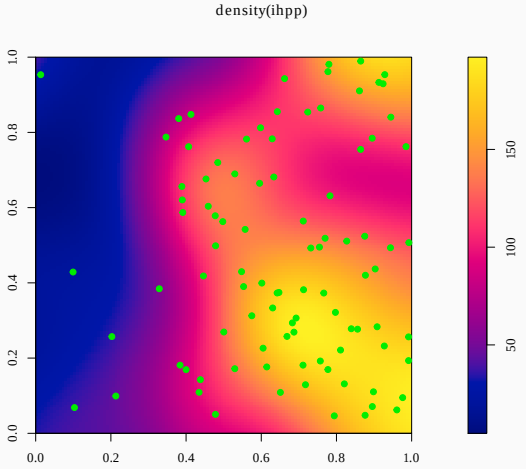
- Spatial point processes are most closely tied to the relative positions of the observations, but may accommodate inhomogeneities
- Geostatistics is concerned with interpolating values of a variable of interest at unobserved locations, and the relative positions of the observations contribute through the modelling a function of differences in observed values of that variable between observations at increasing distances from each other
- Disease mapping, spatial econometrics/regression and the application of generalized linear mixed models (GLMM, GAMM) in for example ecology are more interested in inferences about the spatial processes in play and the included covariates; some approaches may use distance based spatial correlation functions, others use graph or lattice neighbourhoods

When is a “spatial” process actually induced by the analyst

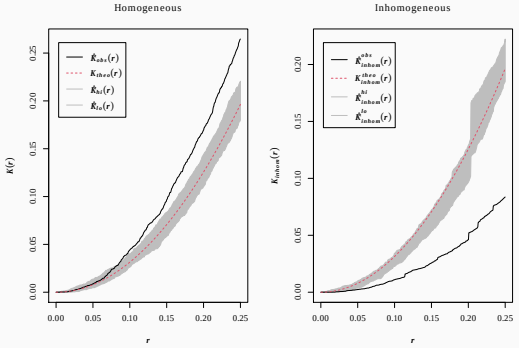
- Sometimes when we use spatial data, we jump to spatial statistical methods because of the widely cited first law of geography, that nearby observations are more likely to be similar than those further away
- But maybe what we see as clustering, patchiness, pattern, that looks spatial is actually mis-specification, such as a missing covariate, and/or inappropriate functional form, and/or including variables acting at different scales, and/or consequences of suboptimal bounding of tesselated observations ...
- Here, we’ll first look at the consequences of treating inhomogeneous points as homogeneous (the intensity trends upwards with `x`)
- Then we’ll use those points to see what happens in adding a similar trend to a random variable; empirical variograms are constructed ignoring and including the trend, and tests for global and local spatial autocorrelation are calculated

We can start by using **spatstat** (Baddeley, Rubak, and Turner 2015) to generate completely spatially random (CSR) points with intensity increasing with **x** to introduce trend inhomogeneity in a unit square:

```
> suppressPackageStartupMessages(library(spatstat))
> intenfun <- function(x, y) 200 * x
> set.seed(1)
> (ihpp <- rpoispp(intenfun, lmax = 200))
## Planar point pattern: 95 points
## window: rectangle = [0, 1] x [0, 1] units
```



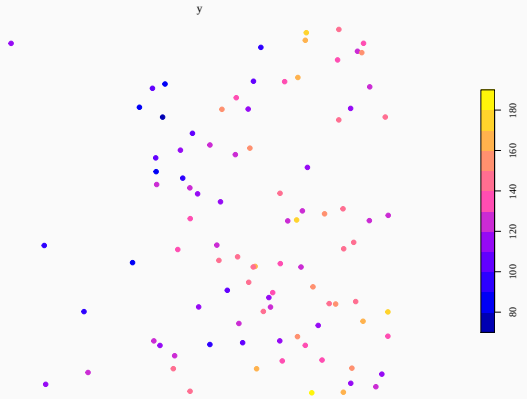
We can use $\hat{K}$ tests ignoring and including inhomogeneity to adapt the test to the underlying data generation process. The homogeneous test examines the divergence from a theoretical CSR at distance bins, the inhomogeneous tries to guess the kind of patterning in the relative positions of the point observations. If we ignore the inhomogeneity, we find significant clustering at most distances, with the opposite finding when using the test attempting to accommodate inhomogeneity:



Adding a y trend variable

Coercing the **spatstat** `"ppp"` object to an **sf** `"sf"` object is trivial, but first adds the window of the point process, which we need to drop by retaining only those labelled `"point"`. Then we take the **x** coordinate and use it to create **y**, which trends with **x**; the DGP is not very noisy.

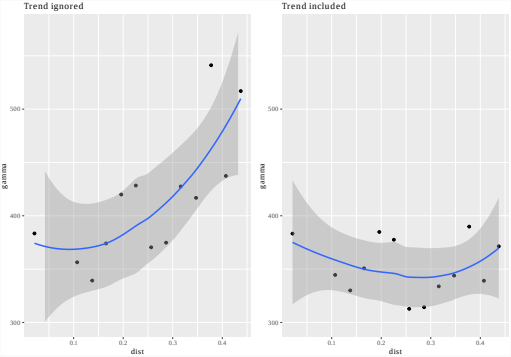
```
> library(sf)
> st_as_sf(ihpp) %>% dplyr::filter(label == "point") -> sf_ihpp
> crds <- st_coordinates(sf_ihpp)
> sf_ihpp$x <- crds[,1]
> sf_ihpp$y <- 100 + 50 * sf_ihpp$x + 20 * rnorm(nrow(sf_ihpp))
```



Variogram model

Variograms for models ignoring (`y ~ 1`) and including (`y ~ x`) the trend; if we ignore the trend, we find spurious relationships, shown by `loess()` fits:

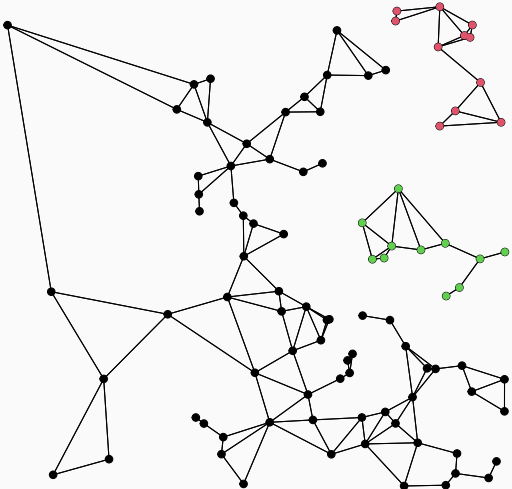
```
> suppressPackageStartupMessages(library(gstat))
> vg0 <- variogram(y ~ 1, sf_ihpp)
> vg1 <- variogram(y ~ x, sf_ihpp)
```



Going on from a continuous to a discrete treatment of space, we can use functions in **spdep** to define neighbours and then test for global and local spatial autocorrelation. Making first a triangulated neighbour objects, we can thin out improbable neighbours lying too far from one another using a sphere of influence graph to make a symmetric neighbour object:

```
> suppressPackageStartupMessages(library(spdep))
> nb_tri <- tri2nb(crds)
> nb_soi <- graph2nb(soi.graph(nb_tri, crds), sym=TRUE)
> comps <- n.comp.nb(nb_soi)
> sf_ihpp$comps <- comps$comp.id
> comps$nc
## [1] 3
```

The sphere of influence graph generates three subgraphs; a bit unfortunate, but we can live with that (for now).



Using binary weights, looking at the variable but ignoring the trend, we find strong global spatial autocorrelation using global Moran's I (tests return `"htest"` objects)

```
> lwB <- nb2listw(nb_soi, style="B")
> moran.test(sf_ihpp$y, listw=lwB, randomisation=FALSE, alternative="two.sided")

##
##  Moran I test under normality
##
## data:  sf_ihpp$y
## weights: lwB
##
## Moran I statistic standard deviate = 4.4358, p-value = 9.174e-06
## alternative hypothesis: two.sided
## sample estimates:
## Moran I statistic       Expectation          Variance
##       0.342200833      -0.010638298       0.006327254
```

Spatial autocorrelation

If, however, we take the residuals after including the trend using a linear model, the apparent spatial autocorrelation evaporates:

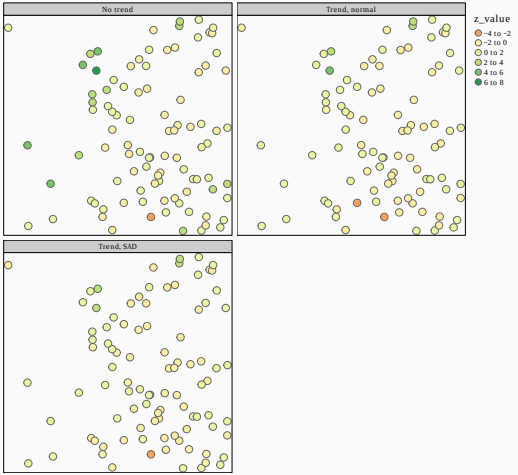
```
> lm_obj <- lm(y ~ x, data=sf_ihpp)
> lm.morantest(lm_obj, listw=lwB, alternative="two.sided")

##
##  Global Moran I for regression residuals
##
## data:
## model: lm(formula = y ~ x, data = sf_ihpp)
## weights: lwB
##
## Moran I statistic standard deviate = 0.98593, p-value = 0.3242
## alternative hypothesis: two.sided
## sample estimates:
## Observed Moran I      Expectation         Variance
##      0.057922986     -0.019881383      0.006227577
```

Spatial autocorrelation

The same happens if we calculate local Moran's I ignoring the trend (randomisation assumption), and including the trend (normality assumption and saddlepoint approximation); had global spatial autocorrelation also been present, we could use exact or saddlepoint approximation to explore local autocorrelation after accounting for global autocorrelation.

```
> lmor0 <- localmoran(sf_ihpp$y, listw=lwB, alternative="two.sided")
> lmor1 <- as.data.frame(localmoran.sad(lm_obj, nb=nb_soi, style="B",
+   alternative="two.sided"))
> sf_ihpp$z_value <- lmor0[,4]
> sf_ihpp$z_lmor1_N <- lmor1[,2]
> sf_ihpp$z_lmor1_SAD <- lmor1[,4]
```



Packages for spatial data analysis

Learning from S-PLUS

- The S-PLUS version of **splancs** provided point pattern analysis (Rowlingson and Diggle 1993, 2017).
- I had contacted Barry Rowlingson in 1997 but only moved forward with porting as R's ability to load shared objects advanced.
- 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 (Bivand 2000), and followed up at several meetings and working closely with Markus Neteler.
- 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 (Bivand 2006).
- Further progress was made in spatial econometrics (Bivand 2002).

2003 Vienna workshop

- 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, 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, Ferrándiz-Ferragud, and Lopez-Quílez 2005; Gómez-Rubio, Ferrándiz-Ferragud, and López-Quílez 2015), and was committed to participating.

2003 Vienna workshop

- 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 3,600 subscribers) was created to facilitate interaction.

Beginnings of sp

- 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.
- Coding meetings were organized by Barry Rowlingson in Lancaster in November 2004 (and kindly hosted by Peter Diggle) 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.

Relationships with other packages

- From an early point in time, object conversion (known as coercion in S and R) to and from **sp** classes and classes in for example the **spatstat** package (Baddeley and Turner 2005; Baddeley, Rubak, and Turner 2015; Baddeley, Turner, and Rubak 2020).
- Packages could choose whether they would use **sp** classes and methods directly, or rather use those classes for functionality that they did not provide themselves through coercion.
- 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** (Bivand, Keitt, and Rowlingson 2020), on CRAN from November 2003, initially only supported raster data read and written using the external GDAL library (Warmerdam 2008).
- 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 into **sp**-objects and writing from **sp**-objects of vector and raster data.

Completing the sp-verse

- For vector data it became possible to project coordinates, and in addition to transform them where datum specifications were available.
- Until recently, the interfaces to external libraries GDAL and PROJ have been relatively stable, and upstream changes have 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 (Bivand and Rundel 2020).
- 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 ASDAR (Bivand, Pebesma, and Gomez-Rubio 2008), a few packages not written or maintained by the authors and their nearest collaborators had begun to use sp classes. By the publication of the second edition (Bivand, Pebesma, and Gomez-Rubio 2013), we had seen that the number of packages depending on sp had grown strongly. In late 2014, a clear spatial cluster was found from a page rank graph of CRAN packages (de Vries 2014). This cluster is from mid-September 2020:

The raster package

- The **raster** package (Hijmans 2020) was written to provide wrappers for **sp** `"SpatialGrid"` objects, for file access through **rgdal** and other packages, and **rgeos** predicates and operations
- Among other benefits, large rasters could be handled by reading and writing chunks using facilities in **rgdal** and GDAL
- In ASDAR 2, we wrote "We have not attempted to cover the **raster** package in the detail that it deserves, hoping that a **raster** book will appear before long"
- Perhaps **raster** has continued to evolve so fast that writing a book has not been seen as possible; Google Scholar notes over 3,000 citations of the package
- The **terra** package to replace **raster** has been on CRAN since March 2020, with new S4 classes, including `"SpatRaster"` and `"SpatVector"`, which are external pointers to C++ objects

Ways forward

The sf package

- The **sf** package provides the input/output and geometry manipulation functionalities found in **rgdal** and **rgeos**, and an alternative class representation for vector data based on the Simple Features standard and the **units** package (Pebesma, Mailund, and Hiebert 2016; Pebesma 2018; Ucar, Pebesma, and Azcorra 2018)
- The **stars** package (E. Pebesma 2020c) adds some facilities for handling spatio-temporal raster and vector data, building in part on work with the **spacetime** package (E. Pebesma 2020b).
- The **stars** package uses **sf** to interface GDAL for reading and writing rasters, and can handle data by proxy, with the work potentially executed on a back-end, or read from a back-end on demand and in a resolution suited to a chosen output device
- By extension **gdalcubes** builds on the **stars** proxy approach and extends itfor building earth observation workflows (Appel and Pebesma 2019; Appel 2020)

Other packages; reproducible research

- Newer visualization packages, such as **tmap** (Tennekes 2018, 2020), **mapview** (Appelhans et al. 2020) and **cartography** (Giraud and Lambert 2016, 2020), give broader scope for data exploration and communication.
- An overview of modelling and analysis packages shows the considerable range of approaches now available in contributed packages and other R code present in supplementary material to published papers (Pebesma, Bivand, and Ribeiro 2015).
- This 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.

Package dependencies

- Because contributed packages form an ecosystem, some packages are used by others in turn in dependency trees.
- Class representations of data are central, with the data frame conceptualisation shaping much of the whole R ecosystem.
- For the modelling infrastructure to perform correctly, the relationships between objects containing data and formula interfaces constructing model responses and matrices are crucial.
- Because both **sp** and **sf** provide similar interfaces, transition from **sp** to **sf** representations is convenient by design.

Reverse dependencies of the sp and sf packages

- 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 minimise forward dependencies, with **sp** only depending on and importing packages included in every R distribution by default.
- **sf** adds 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.

Reverse dependencies of the sp and sf packages

The table shows the structure of reverse dependency counts for **sp** and **sf** in mid-September 2020. 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 1299 only involving **sp**, 234 involving both packages, and 67 only involving **sf**. If we additionally include “suggests” dependencies, both packages may be used at least indirectly by all CRAN packages.

```
##         Recursive Recursive w/Suggests Not recursive Not recursive w/Suggests
## Sum sp       1533                16725           523                       630
## Sum sf        301                16725           184                       278
## Only sp      1299                    0           461                       514
## Only sf        67                    0           122                       162
## Both          234                16725            62                       116
```

Reverse dependencies of the sp and sf packages

- The two right columns show the same counts but only for packages first-order dependencies on **sp**, **sf** or both.
- 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 use both packages, showing existing packages 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**.

Upstream software dependencies of the R-spatial ecosystem

Upstream dependencies of sp workflows

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 versions vary between platforms and by the installation method used; 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, while the report for **rgeos** is simpler, only listing the versions of GEOS itself and **sp**.

```
> rgdal::rgdal_extSoftVersion()

##           GDAL GDAL_with_GEOS           PROJ             sp
##        "3.1.3"         "TRUE"        "7.1.1"        "1.4-4"

> rgeos::rgeos_extSoftVersion()

##    GEOS      sp
## "3.8.1" "1.4-4"
```

Upstream dependencies of sf workflows

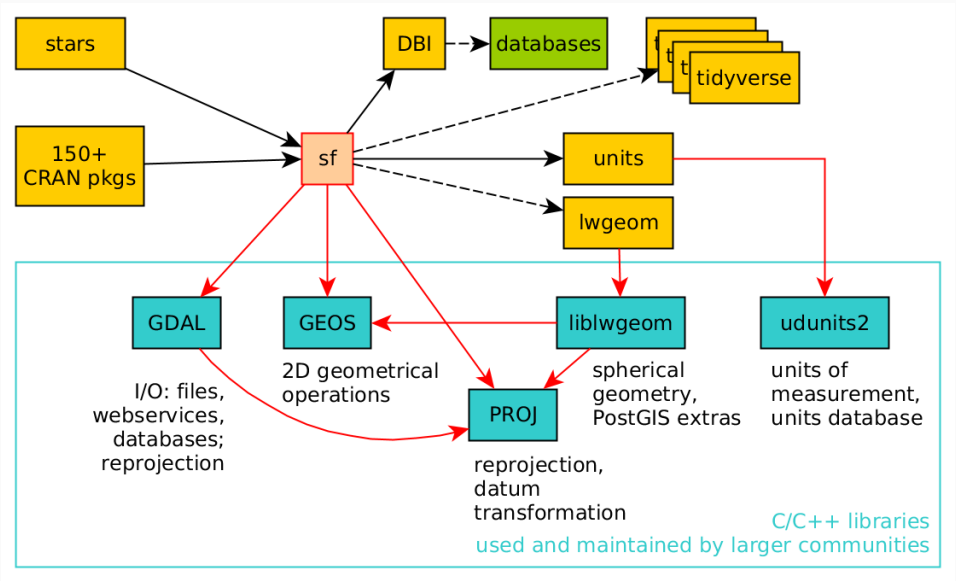
In the case of **sf**, and because it brings together access to the GDAL and GEOS external libraries through Simple Features representation for vector objects, the external software versions supported are the union of those seen above, omitting linkage to a separate package defining classes for objects.

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"         "3.1.3"         "7.1.1"          "true"          "true"
```

Upstream dependencies of sf workflows



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 consequences, so that legacy workflows continue to provide the same or at least similar results from the same data.
- ASDAR code (Bivand, Pebesma, and Gomez-Rubio 2008, 2013) is almost all run nightly on a platform with updated R packages and external software; obviously, this does not test **sf** workflows.
- 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 levels of external software also run nightly checks, often on updated versions of external software.

- Again, sometimes changes are first 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, although sometimes code tidying in external software can have unexpected consequences, breaking not **sf** or **sp** with **rgdal** or **rgeos**, but a package further downstream.
- I discuss open source geospatial software stacks more generally elsewhere (Bivand 2014), but here we will consider ongoing changes in PROJ in relation to coordinate reference system (CRS) representation.

The **mapproj** package provided coordinate reference system and projection support for the **maps** package. From `mapproj/src/map.h`, line 20, we can see that the eccentricity of the Earth is defined as `0.08227185422`, corrresponding to the Clark 1866 ellipsoid (Iliffe and Lott 2008):

```
> ellps <- sf_proj_info("ellps")
> (clrk66 <- unlist(ellps[ellps$name=="clrk66",]))
##          name         major           ell   description
##      "clrk66" "a=6378206.4" "b=6356583.8" "Clarke 1866"
```

With a very few exceptions, projections included in `mapproj::mapproject()` use the Clarke 1866 ellipsoid, with the remainder using a sphere with the Clarke 1866 major axis radius. The function returns coordinates for visualization in an unknown metric; no inverse projections are available.

```
> eval(parse(text=clrk66["major"]))
> eval(parse(text=clrk66["ell"]))
> print(sqrt((a^2-b^2)/a^2), digits=10)
## [1] 0.08227185422
```

Approaching and implemented changes in PROJ

- PROJ developers not only point out how the world has changed since a World Geodetic System of 1984 (WGS84) was adopted as a hub for coordinate transformation in PROJ, but also introduced transformation pipelines (Knudsen and Evers 2017; Evers and Knudsen 2017).
- In using a transformation hub, PROJ had worked adequately when the errors introduced by transforming first to WGS84 and then from WGS84 to the target coordinate reference system were acceptable, but with years passing from 1984, the world has undergone sufficient tectonic shifts for errors to increase, just as needs for accuracy sharpen.
- In addition, the need for accuracy has risen in agriculture and engineering.
- So PROJ, as it was, risked ceasing to be fit for purpose as a fundamental component of the geospatial open source software stack.

Approaching and implemented changes in PROJ

PROJ will also become more tightly linked to authorities responsible for the specification components. While the original well-known text (WKT1) descriptions also contained authorities, WKT2 (2019) is substantially more stringent. PROJ continues to use the European Petroleum Survey Group (EPSG) database, the local copy PROJ uses is now an SQLite database, with a large number of tables:

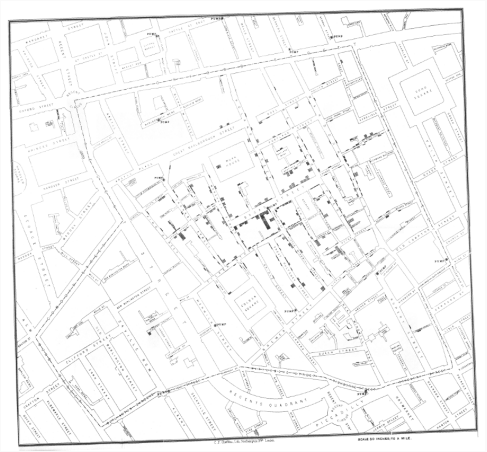
```
> library(RSQLite)
> DB0 <- strsplit(sf:::CPL_get_data_dir(), .Platform$path.sep)[[1]]
> DB <- file.path(DB0[length(DB0)], "proj.db")
> db <- dbConnect(SQLite(), dbname=DB)
> cat(strwrap(paste(dbListTables(db), collapse=", ")), sep="\n")
## alias_name, area, authority_list,
## authority_to_authority_preference, axis,
## celestial_body, compound_crs, concatenated_operation,
## concatenated_operation_step, conversion,
## conversion_method, conversion_param,
## conversion_table, coordinate_operation_method,
## coordinate_operation_view,
## coordinate_operation_with_conversion_view,
## coordinate_system, crs_view, deprecation, ellipsoid,
## geodetic_crs, geodetic_datum, geoid_model,
## grid_alternatives, grid_packages,
## grid_transformation, helmert_transformation,
## helmert_transformation_table, metadata, object_view,
## other_transformation, prime_meridian, projected_crs,
## supersession, unit_of_measure, vertical_crs,
## vertical_datum
> dbDisconnect(db)
```

- The initial use of coordinate reference systems for objects defined in **sp** was based on the PROJ string representation, which built on a simplified key=value form.
- Keys began with plus (`+`), and the value format depended on the key.
- If essential keys were missing, some might be added by default from a file that has now been eliminated as misleading; if `+ellps=` was missing and not added internally from other keys, `+ellps=WGS84` would be added silently to refer to the World Geodetic System 1984 ellipsoid definition.

- Accurate coordinate transformation has always been needed for the integration of data from different sources, but has become much more pressing as web mapping has become available in R, through the **leaflet** package (Cheng, Karambelkar, and Xie 2019), on which **mapview** and the `"view"` mode of **tmap** build.
- As web mapping provides zooming and panning, possible infelicities that were too small to detect as mismatches in transformation jump into prominence.
- The web mapping workflow transforms input objects to `OGC:CRS84` (geographical CRS WGS 84, World area of relevance, WGS84 datum, visualization order) as expected by **leaflet**, then on to `EPSG:3857` (WGS 84 / Pseudo-Mercator) for display on web map backgrounds (this is carried out internally in **leaflet**).
- Objects shown in **mapview** and **tmap** are now coerced to **sf** classes, then `st_transform()` transforms to `OGC:CRS84` if necessary (or until we are sure `EPSG:4326` never swaps axes).

Broad Street Cholera Data

John Snow did not use maps to *find* the Broad Street pump, the polluted water source behind the 1854 cholera epidemic in Soho in central London, because he associated cholera with water contaminated with sewage, based on earlier experience (Brody et al. 2000). The basic data to be used here were made available by Jim Detwiler, who had collated them for David O'Sullivan for use on the cover of O'Sullivan and Unwin (2003), based on earlier work by Waldo Tobler and others.



Where is the Broad Street pump?

We'll use the example of the location of the Broad Street pump in Soho, London, to be distributed with **sf**; the object has planar coordinates in the OSGB British National Grid projected CRS with the OSGB datum:

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

## Reading layer `b_pump' from data source `/home/rsb/lib/r_libs/sf/gpkg/b_pump.gpkg' using driver `GPKG'
## Simple feature collection with 1 feature and 1 field
## geometry type:  POINT
## dimension:      XY
## bbox:           xmin: 529393.5 ymin: 181020.6 xmax: 529393.5 ymax: 181020.6
## projected CRS:  Transverse_Mercator
```

Proj4 string degradation

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, there is 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"
```

Proj4 string degradation

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. The apparent `"proj4string"` component of the `sf` `"crs"` is used to permit packages to adapt, even though its contents are degraded.

```
> b_pump_sf1 <- b_pump_sf
> st_crs(b_pump_sf1) <- st_crs(st_crs(b_pump_sf1)$proj4string)
## Warning: st_crs<- : replacing crs does not reproject data;
## use st_transform for that
```

Proj4 string degradation

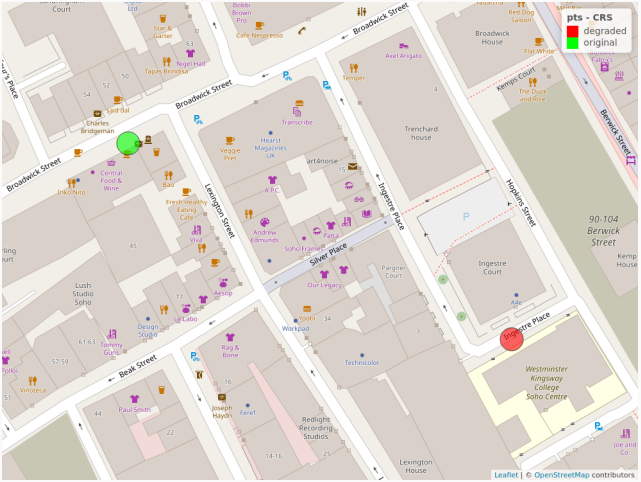
Why does this matter? For visualization on a web map, for example using the **mapview** package, the projected geometries are transformed to the same WGS84 ellipse and datum (`OGC:CRS84`) that were used in PROJ 4 as a transformation hub. `OGC:CRS84` is the visualization axis order equivalent of `EPSG:4326`. In **leaflet**, these are projected to Web Mercator (`EPSG:3856`). Inside `mapview()`, the `sf::st_transform()` function is used, so we will emulate this coordinate operation before handing on the geometries for display.

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:

```
> b_pump_sf_ll <- st_transform(b_pump_sf, "OGC:CRS84")
> b_pump_sf1_ll <- st_transform(b_pump_sf1, "OGC:CRS84")
> st_distance(b_pump_sf_ll, b_pump_sf1_ll)
## Units: [m]
##          [,1]
## [1,] 125.0578
```

Proj4 string degradation

The Broad Street pump is within 2m of the green point (relative accuracy preserved) but the red point is now in Ingestre Place, because of the loss of the datum specification.



Implemented resolutions: WKT2 2019

- 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 before.
- Since then, **sf**, **sp** with **rgdal** 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.
- 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.

Implemented resolutions: WKT2 2019

- Following an extended campaign of repeated 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.
- More recently (late August 2020), 115 packages are being 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 also be written out to external data formats correctly.

Specified axis order

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]]
```

Specified axis order

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. From very recent **rgdal** commits (from rev. 1060), PROJ rather than GDAL is used for instantiating CRS:

```
> library(sp)
> cat(wkt(CRS("EPSG:4326")), "\n")
## GEOGCRS["WGS 84 (with axis order normalized for visualization)",
##     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["geodetic longitude (Lon)",east,
##             ORDER[1],
##             ANGLEUNIT["degree",0.0174532925199433,
##                 ID["EPSG",9122]]],
##         AXIS["geodetic latitude (Lat)",north,
##             ORDER[2],
##             ANGLEUNIT["degree",0.0174532925199433,
##                 ID["EPSG",9122]]]]
```

Specified axis order

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

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

Both of these coercions are using the same underlying PROJ and GDAL versions, and the same PROJ metadata. Work on this question has not yet stabilized; we perhaps prefer all data to be GIS-order, but to be able to read/write from and to authority order.

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

Specified axis order

Current thinking is to avoid the `EPSG:4326` axis order issue by recommending use of `OGC:CRS84`, which is the representation used in GeoJSON, and also known as `urn:ogc:def:crs:OGC::CRS84`. This specification is always eastings-northings:

```
> cat(st_crs("OGC:CRS:84")$wkt, "\n")
## 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]]]]
```

```
> cat(wkt(CRS("OGC:CRS84")), "\n")
## GEOGCRS["WGS 84 (CRS84)",
##     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 longitude (Lon)",east,
##             ORDER[1],
##             ANGLEUNIT["degree",0.0174532925199433]],
##         AXIS["geodetic latitude (Lat)",north,
##             ORDER[2],
##             ANGLEUNIT["degree",0.0174532925199433]],
##     USAGE[
##         SCOPE["unknown"],
##         AREA["World"],
##         BBOX[-90,-180,90,180]],
##     ID["OGC","CRS84"]]
```

PROJ or GDAL SRS?

As just mentioned, from **rgdal** rev. 1060, PROJ is preferred for instantiating **"CRS"** objects, but GDAL may still be chosen; **sf** uses GDAL. On the previous slide, the **sp** and **sf** WKT2 renderings differ for this reason, but can be made equal. The two are also see as strictly equivalent by PROJ:

```
> rgdal::set_prefer_proj(FALSE)
> GDAL_SRS <- wkt(CRS("OGC:CRS84"))
> all.equal(st_crs("OGC:CRS:84")$wkt, GDAL_SRS)
## [1] TRUE
> rgdal::set_prefer_proj(TRUE)
```

```
> cat(GDAL_SRS, "\n")
## 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]]]]
> rgdal::compare_CRS(CRS("OGC:CRS84"), as(st_crs("OGC:CRS:84"), "CRS"))
##                       strict                   equivalent
##                        FALSE                         TRUE
## equivalent_except_axis_order
##                         TRUE
```

Coordinate operations

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)

## rgdal: version: 1.5-17, (SVN revision 1066)
## Geospatial Data Abstraction Library extensions to R successfully loaded
## Loaded GDAL runtime: GDAL 3.1.3, released 2020/09/01
## Path to GDAL shared files: /usr/local/share/gdal
## GDAL binary built with GEOS: TRUE
## Loaded PROJ runtime: Rel. 7.1.1, September 1st, 2020, [PJ_VERSION: 711]
## Path to PROJ shared files: /tmp/Rtmpkehjf9/file9b564103e2a3:/usr/local/share/proj:/usr/local/share/proj
## PROJ CDN enabled: FALSE
## Linking to sp version:1.4-4
## To mute warnings of possible GDAL/OSR exportToProj4() degradation,
## use options("rgdal_show_exportToProj4_warnings"="none") before loading rgdal.
```

Areas of interest

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")
> 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)
```

`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 not intersecting it are dropped. Coordinate operations that cannot be instantiated because of missing grids are also listed. Here without an area-of-interest: 8 candidate operations are found when the WKT string contains datum information. Of these, 7 may be instantiated, with 1 needing a grid. 3 operations cease to be candidates if we use an area-of-interest.

Coordinate operations

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:

```
> b_pump_sp_ll <- spTransform(b_pump_sp, "OGC:CRS84")
> cat(strwrap(get_last_coordOp()), sep="\n")
## +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. The `+push +v_3` and `+pop +v_3` operations are used when only horizontal components are needed in the Helmert transformation.

Coordinate operations

```
> o <- list_coordOps(wkt(b_pump_sp1), "OGC:CRS84",
+   area_of_interest=aoi)
> b_pump_sp1_ll <- spTransform(b_pump_sp1, "OGC:CRS84")
> cat(strwrap(get_last_coordOp()), sep="\n")
## +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.

Using the content download network to access grids

Finally, **sp**/**rgdal** may use the provision of on-demand downloading of transformation grids to provide more accuracy (CDN, from PROJ 7, https://cdn.proj.org). 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/Rtmpkehjf9/file9b564103e2a3"
> 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.

```
> b_pump_sp_llg <- spTransform(b_pump_sp, "OGC:CRS84")
> cat(strwrap(get_last_coordOp()), sep="\n")
## +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.

```
> (fls <- list.files(td))
## [1] "cache.db"
> file.size(file.path(td, fls[1]))
## [1] 319488
> disable_proj_CDN()
> ll <- st_as_sf(b_pump_sp_ll)
> ll1 <- st_as_sf(b_pump_sp1_ll)
> llg <- st_as_sf(b_pump_sp_llg)
> sf_use_s2(FALSE)
> c(st_distance(ll, ll1), st_distance(ll, llg))
## Units: [m]
## [1] 125.057812   1.751479
```

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 125m. Using the CDN shifts the output point by 1.7m, here using both the new **s2** and legacy interfaces for measurements in **sf** using geographical coordinates (Dunnington, Pebesma, and Rubak 2020):

```
> sf_use_s2(TRUE)
> c(st_distance(ll, ll1), st_distance(ll, llg))
## Units: [m]
## [1] 124.728570   1.745973
> sf_use_s2(FALSE)
```

CRS representation: status

- 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.

Conclusions

Conclusions i

- Progress with the **sf** and **stars** packages, including regular spatio-temporal data structures, continues; proxy access to data via APIs or cloud repositories and raster and vector tiles are of growing importance
- Work on **sf** and its use in **stars** is linked to using **Rcpp** to interface external C and C++ libraries; should this be extended to other spatial analysis packages?
- Both **gdalcubes** and **terra** link to GDAL and PROJ using **Rcpp**; **terra**, like **sf**, also links to **GEOS**
- While **terra** appears to be a re-implementation of **raster** using C++ for increased speed, **stars** and **gdalcubes** both use array data structures accommodating multiband space-time data without extra steps

Conclusions ii

- **stars** and **gdalcubes** may handle proxy data sets on remote nodes; **openeo** is a language-neutral API for handling earth observation data no remote nodes (https://openeo.org/)
- Changes in handling coordinate reference systems through PROJ are continuing, and have impacted most work; web mapping depends on knowing the correct CRS of the data
- Users of Proj4 strings should understand the consequences of the transition to WKT2-2019
- Changes are also coming to the EPSG registry, which from version 10 will support CRS epoch/temporal scope definitions among other modifications; we hope that PROJ will shield us from possible consequences

Conclusions iii

- The legacy shapefile format for vector data should be discontinued as soon as possible, with binary SQLite-based GeoPackage (GPKG) a more robust choice than text-based GeoJSON or GML for non-database storage
- `sf::st_read()` can take a `query=` argument to permit the use of SQL to subset a dataset on reading without reading the whole object into memory first
- Visualization does not need to be web mapping; the **tmap** and **cartography** packages provide modern thematic mapping functionality; **ggplot2** also supports `"sf"` objects through `geom_sf()` and `geom_stars()`
- Support for Bayesian approaches, including **INLA**, **R2BayesX**, **CARBayes**, **spBayes** and very many others remains important, going back to **geoR** twenty years ago

Conclusions iv

- Analytical use of spatial infrastructure packages in other GLMM contexts is also very diversified and is important in applied studies in ecology and epidemiology
- An important reason for increasing attention on prediction is that it is fundamental for machine learning approaches, in which prediction for validation and test data sets drives model specification choice (rather than (spatio-)temporal prediction)
- Using spatial data requires care in splitting training data from other data; see **mlr3** (Schratz et al. 2019; Lang et al. 2020), (https://mlr3spatiotempcv.mlr-org.com/), and **blockCV** (Valavi et al. 2019, 2020)
- This is a topical area seeing a lot of activity at the moment, see for important insights (Meyer and Pebesma 2020) and (Ploton et al. 2020)

Aftermatter

R's sessionInfo() i

```
> sessionInfo()

## R version 4.0.2 (2020-06-22)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Fedora 32 (Workstation Edition)
##
## Matrix products: default
## BLAS:   /home/rsb/topics/R/R402-share/lib64/R/lib/libRblas.so
## LAPACK: /home/rsb/topics/R/R402-share/lib64/R/lib/libRlapack.so
##
## locale:
##  [1] LC_CTYPE=en_GB.UTF-8       LC_NUMERIC=C               LC_TIME=en_GB.UTF-8        LC_COLLATE=en_GB.UTF-8
##  [5] LC_MONETARY=en_GB.UTF-8    LC_MESSAGES=en_GB.UTF-8    LC_PAPER=en_GB.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C             LC_MEASUREMENT=en_GB.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] wordcloud_2.6       RColorBrewer_1.1-2  tmap_3.1            ggplot2_3.3.2       rgdal_1.5-17        RSQLite_2.2.0
##  [7] spdep_1.1-5         spData_0.3.8        sp_1.4-4            gstat_2.0-6         spatstat_1.64-1     rpart_4.1-15
## [13] nlme_3.1-149        spatstat.data_1.4-3 sf_0.9-6            osmdata_0.1.3       extrafont_0.17
##
```

R's sessionInfo() ii

```
## loaded via a namespace (and not attached):
##   [1] leafem_0.1.3          colorspace_1.4-1      deldir_0.1-28         ellipsis_0.3.1        class_7.3-17
##   [6] leaflet_2.0.3         base64enc_0.1-3       dichromat_2.0-0       bit64_4.0.5           fansi_0.4.1
##  [11] lubridate_1.7.9       xml2_1.3.2            codetools_0.2-16      splines_4.0.2         knitr_1.29
##  [16] polyclip_1.10-0       jsonlite_1.7.0        tmaptools_3.1         broom_0.7.0           Rttf2pt1_1.3.8
##  [21] png_0.1-7             rgeos_0.5-5           compiler_4.0.2        httr_1.4.2            backports_1.1.9
##  [26] assertthat_0.2.1      Matrix_1.2-18         cli_2.0.2             s2_1.0.2              htmltools_0.5.0
##  [31] tools_4.0.2           coda_0.19-3           gtable_0.3.0          glue_1.4.2            RANN_2.6.1
##  [36] dplyr_1.0.2           wk_0.3.2              gmodels_2.18.1        Rcpp_1.0.5            raster_3.3-15
##  [41] vctrs_0.3.4           gdata_2.18.0          extrafontdb_1.0       leafsync_0.1.0        crosstalk_1.1.0.1
##  [46] lwgeom_0.2-5          xfun_0.16             stringr_1.4.0         rvest_0.3.6           lifecycle_0.2.0
##  [51] gtools_3.8.2          XML_3.99-0.5          goftest_1.2-2         LearnBayes_2.15.1     MASS_7.3-52
##  [56] zoo_1.8-8             scales_1.1.1          spatstat.utils_1.17-0 parallel_4.0.2        expm_0.999-5
##  [61] yaml_2.2.1            curl_4.3              memoise_1.1.0         stringi_1.4.6         highr_0.8
##  [66] spDataLarge_0.4.1     e1071_1.7-3           boot_1.3-25           intervals_0.15.2      rlang_0.4.7
##  [71] pkgconfig_2.0.3       evaluate_0.14         lattice_0.20-41       purrr_0.3.4           tensor_1.5
##  [76] htmlwidgets_1.5.1     bit_4.0.4             tidyselect_1.1.0      magrittr_1.5          R6_2.4.1
##  [81] generics_0.0.2        DBI_1.1.0             pillar_1.4.6          withr_2.2.0           mgcv_1.8-33
##  [86] units_0.6-7           stars_0.4-4           xts_0.12-0            abind_1.4-5           tibble_3.0.3
##  [91] spacetime_1.2-3       crayon_1.3.4          KernSmooth_2.23-17    utf8_1.1.4            rmarkdown_2.3
##  [96] grid_4.0.2            blob_1.2.1            FNN_1.1.3             digest_0.6.25         classInt_0.4-3
## [101] tidyr_1.1.2           munsell_0.5.0         viridisLite_0.3.0
```

Akima, H., and A. Gebhardt. 2020. *akima: Interpolation of irregularly and regularly spaced data*. https://CRAN.R-project.org/package=akima.

Appel, M. 2020. *Gdalcubes: Earth observation data cubes from satellite image collections*. https://CRAN.R-project.org/package=gdalcubes.

Appel, M., and E. Pebesma. 2019. On-demand processing of data cubes from satellite image collections with the gdalcubes library. *Data* 4 (3). https://www.mdpi.com/2306-5729/4/3/92.

Appelhans, T., F. Detsch, C. Reudenbach, and S. Woellauer. 2020. *mapview: Interactive viewing of spatial data in R*. https://CRAN.R-project.org/package=mapview.

Baddeley, A., E. Rubak, and R. Turner. 2015. *Spatial point patterns: Methodology and applications with R*. Boca Raton, FL: Chapman and Hall/CRC.

Baddeley, A., and R. Turner. 2005. spatstat: An R package for analyzing spatial point patterns. *Journal of Statistical Software* 12 (6):1–42. http://www.jstatsoft.org/v12/i06/.

Baddeley, A., R. Turner, and E. Rubak. 2020. *Spatstat: Spatial point pattern analysis, model-fitting, simulation, tests*. https://CRAN.R-project.org/package=spatstat.

Bailey, T. C., and A. C. Gatrell. 1995. *Interactive spatial data analysis*. Harlow: Longman.

Bivand, R. 1996. Scripting and toolbox approaches to spatial analysis in a GIS context. In *Spatial analytical perspectives on GIS*, eds. M. M. Fischer, H. J. Scholten, and D. Unwin, 39–52. London: Taylor & Francis.

———. 1997. Scripting and tool integration in spatial analysis: Prototyping local indicators and distance statistics. In *Innovations in GIS* 4, ed. Z. Kemp, 127–138. London: Taylor & Francis.

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

———. 2000. Using the R statistical data analysis language on GRASS 5.0 GIS database files. *Computers & Geosciences* 26 (9):1043–1052.

———. 2002. Spatial econometrics functions in R: Classes and methods. *Journal of Geographical Systems* 4:405–421.

———. 2006. Implementing spatial data analysis software tools in R. *Geographical Analysis* 38:23–40.

———. 2014. Geocomputation and open source software: Components and software stacks. In *Geocomputation*, eds. R. J. Abrahart and L. M. See, 329–355. Boca Raton: CRC Press http://hdl.handle.net/11250/163358.

Bivand, R., and A. Gebhardt. 2000. Implementing functions for spatial statistical analysis using the R language. *Journal of Geographical Systems* 2 (3):307–317. https://doi.org/10.1007/PL00011460.

Bivand, R., J. Hauke, and T. Kossowski. 2013. Computing the Jacobian in Gaussian spatial autoregressive models: An illustrated comparison of available methods. *Geographical Analysis* 45 (2):150–179. https://doi.org/10.1111/gean.12008.

Bivand, R., T. Keitt, and B. Rowlingson. 2020. *rgdal: Bindings for the 'geospatial' data abstraction library*. https://CRAN.R-project.org/package=rgdal.

Bivand, R., and N. Lewin-Koh. 2020. *Maptools: Tools for handling spatial objects*. https://CRAN.R-project.org/package=maptools.

Bivand, R., E. Pebesma, and V. Gomez-Rubio. 2008. *Applied spatial data analysis with R*. Springer, NY. https://asdar-book.org/.

———. 2013. *Applied spatial data analysis with R, second edition*. Springer, NY. https://asdar-book.org/.

Bivand, R., and G. Piras. 2015. Comparing implementations of estimation methods for spatial econometrics. *Journal of Statistical Software* 63 (1):1–36.

Bivand, R., and C. Rundel. 2020. *rgeos: Interface to geometry engine - open source ('geos')*. https://CRAN.R-project.org/package=rgeos.

Bivand, R., Z. Sha, L. Osland, and I. S. Thorsen. 2017. A comparison of estimation methods for multilevel models of spatially structured data. *Spatial Statistics*. https://doi.org/10.1016/j.spasta.2017.01.002.

Bivand, R., and D. W. S. Wong. 2018. Comparing implementations of global and local indicators of spatial association. *TEST* 27 (3):716–748. https://doi.org/10.1007/s11749-018-0599-x.

Brody, H., M. R. Rip, P. Vinten-Johansen, N. Paneth, and S. Rachman. 2000. Map-making and myth-making in Broad Street: The London cholera epidemic, 1854. *Lancet* 356:64–68.

Cheng, J., B. Karambelkar, and Y. Xie. 2019. *Leaflet: Create interactive web maps with the javascript 'leaflet' library*. https://CRAN.R-project.org/package=leaflet.

Cliff, A. D., and J. K. Ord. 1973. *Spatial autocorrelation*. London: Pion.

———. 1981. *Spatial processes*. London: Pion.

Cressie, N. A. C. 1993. *Statistics for spatial data*. New York:Wiley.

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.

Duncan, O. D., R. P. Cuzzort, and B. Duncan. 1961. *Statistical geography: Problems in analyzing areal data*. Glencoe, IL: Free Press.

Dunnington, D., E. Pebesma, and E. Rubak. 2020. *S2: Spherical geometry operators using the s2 geometry library*. https://CRAN.R-project.org/package=s2.

Evers, K., and T. Knudsen. 2017. *Transformation pipelines for proj.4*. https://www.fig.net/resources/proceedings/fig/_proceedings/fig2017/papers/iss6b/ISS6B/_evers/_knudsen/_9156.pdf.

Giraud, T., and N. Lambert. 2016. cartography: Create and integrate maps in your R workflow. *Journal of Open Source Software* 1 (4). https://doi.org/10.21105/joss.00054.

———. 2020. *cartography: Thematic cartography*. https://CRAN.R-project.org/package=cartography.

Goulard, M., T. Laurent, and C. Thomas-Agnan. 2017. About predictions in spatial autoregressive models: Optimal and almost optimal strategies. *Spatial Economic Analysis* 12 (2-3):304–325. https://doi.org/10.1080/17421772.2017.1300679.

Gómez-Rubio, V. 2011. *RArcInfo: Functions to import data from arc/info v7.x binary coverages*. https://github.com/becarioprecario/RArcInfo.

Gómez-Rubio, V., J. Ferrándiz-Ferragud, and A. Lopez-Quílez. 2005. Detecting clusters of disease with R. *Journal of Geographical Systems* 7 (2):189–206.

Gómez-Rubio, V., J. Ferrándiz-Ferragud, and A. López-Quílez. 2015. *DCluster: Functions for the detection of spatial clusters of diseases*. https://CRAN.R-project.org/package=DCluster.

Gómez-Rubio, V., and A. López-Quílez. 2005. RArcInfo: Using GIS data with R. *Computers & Geosciences* 31 (8):1000–1006.

Haining, R. P. 1990. *Spatial data analysis in the social and environmental sciences*. Cambridge: Cambridge University Press.

Hepple, L. W. 1976. A maximum likelihood model for econometric estimation with spatial series. In *Theory and practice in regional science*, London papers in regional science., ed. I. Masser, 90–104. London: Pion.

Hijmans, R. J. 2020. *Raster: Geographic data analysis and modeling*. https://CRAN.R-project.org/package=raster.

Iliffe, J., and R. Lott. 2008. *Datums and map projections: For remote sensing, GIS and surveying*. Boca Raton: CRC.

ISO. 2019. *ISO 19111:2019 geographic information – referencing by coordinates*. https://www.iso.org/standard/74039.html.

Kaluzny, S. P., S. C. Vega, T. P. Cardoso, and A. A. Shelly. 1998. *S+SpatialStats*. New York, NY: Springer.

Knudsen, T., and K. Evers. 2017. *Transformation pipelines for proj.4*. https://meetingorganizer.copernicus.org/EGU2017/EGU2017-8050.pdf.

Lang, M., B. Bischl, J. Richter, P. Schratz, and M. Binder. 2020. *Mlr3: Machine learning in R - next generation*. https://CRAN.R-project.org/package=mlr3.

Lovelace, R., and R. Ellison. 2018. stplanr: A Package for Transport Planning. *The R Journal* 10 (2):7–23. https://doi.org/10.32614/RJ-2018-053.

Lovelace, R., R. Ellison, and M. Morgan. 2020. *Stplanr: Sustainable transport planning*. https://CRAN.R-project.org/package=stplanr.

Majure, J. J., and A. Gebhardt. 2016. *sgeostat: An object-oriented framework for geostatistical modeling in S+*. https://CRAN.R-project.org/package=sgeostat.

Meyer, H., and E. Pebesma. 2020. Predicting into unknown space? Estimating the area of applicability of spatial prediction models. http://arxiv.org/abs/2005.07939.

Millo, G., and G. Piras. 2012. Splm: Spatial panel data models in R. *Journal of Statistical Software, Articles* 47 (1):1–38. https://www.jstatsoft.org/v047/i01.

———. 2020. *Splm: Econometric models for spatial panel data*. https://CRAN.R-project.org/package=splm.

Müller, W. G. 2007. *Collecting spatial data: Optimum design of experiments for random fields*. Berlin: Springer-Verlag.

O'Sullivan, D., and D. J. Unwin. 2003. *Geographical information analysis*. Hoboken, NJ: Wiley.

Padgham, M., B. Rudis, R. Lovelace, and M. Salmon. 2017. Osmdata. *The Journal of Open Source Software* 2 (14). https://doi.org/10.21105/joss.00305.

———. 2020. *Osmdata: Import 'openstreetmap' data as simple features or spatial objects*. https://CRAN.R-project.org/package=osmdata.

Pebesma, E. 2018. Simple Features for R: Standardized Support for Spatial Vector Data. *The R Journal* 10 (1):439–446. https://doi.org/10.32614/RJ-2018-009.

———. 2020a. *sf: Simple features for R*. https://CRAN.R-project.org/package=sf.

———. 2020b. *spacetime: Classes and methods for spatio-temporal data*. https://CRAN.R-project.org/package=spacetime.

———. 2020c. *stars: Spatiotemporal arrays, raster and vector data cubes*. https://CRAN.R-project.org/package=stars.

Pebesma, E., R. Bivand, and P. Ribeiro. 2015. Software for spatial statistics. *Journal of Statistical Software, Articles* 63 (1):1–8. https://www.jstatsoft.org/v063/i01.

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

Pebesma, E., T. Mailund, and J. Hiebert. 2016. Measurement units in R. *The R Journal* 8 (2):486–494. https://doi.org/10.32614/RJ-2016-061.

Piras, G. 2010. Sphet: Spatial models with heteroskedastic innovations in R. *Journal of Statistical Software, Articles* 35 (1):1–21. https://www.jstatsoft.org/v035/i01.

———. 2020. *Sphet: Estimation of spatial autoregressive models with and without heteroscedasticity*. https://CRAN.R-project.org/package=sphet.

Ploton, P., F. Mortier, M. Réjou-Méchain, N. Barbier, N. Picard, V. Rossi, C. Dormann, G. Cornu, G. Viennois, N. Bayol, A. Lyapustin, S. Gourlet-Fleury, and R. Pélissier. 2020. Spatial validation reveals poor predictive performance of large-scale ecological mapping models. *Nature Communications* 11 (4540). https://doi.org/10.1038/s41467-020-18321-y.

Renka, R. J., and A. Gebhardt. 2020. *tripack: Triangulation of irregularly spaced data*. https://CRAN.R-project.org/package=tripack.

Ripley, B. D. 1981. *Spatial statistics*. New York: Wiley.

Rowlingson, B., and P. Diggle. 2017. *Splancs: Spatial and space-time point pattern analysis*. https://CRAN.R-project.org/package=splancs.

Rowlingson, B., and P. J. Diggle. 1993. Splancs: Spatial point pattern analysis code in S-Plus. *Computers and Geosciences* 19:627–655.

Schratz, P., J. Muenchow, E. Iturritxa, J. Richter, and A. Brenning. 2019. Hyperparameter tuning and performance assessment of statistical and machine-learning algorithms using spatial data. *Ecological Modelling* 406:109–120.

Tennekes, M. 2018. tmap: Thematic maps in R. *Journal of Statistical Software* 84 (6):1–39. https://www.jstatsoft.org/v084/i06.

———. 2020. *tmap: Thematic maps*. https://CRAN.R-project.org/package=tmap.

Ucar, I., E. Pebesma, and A. Azcorra. 2018. Measurement Errors in R. *The R Journal* 10 (2):549–557. https://doi.org/10.32614/RJ-2018-075.

Valavi, R., J. Elith, J. Lahoz-Monfort, and G. Guillera-Arroita. 2020. *BlockCV: Spatial and environmental blocking for k-fold cross-validation*. https://CRAN.R-project.org/package=blockCV.

Valavi, R., J. Elith, J. J. Lahoz-Monfort, and G. Guillera-Arroita. 2019. BlockCV: An R package for generating spatially or environmentally separated folds for k-fold cross-validation of species distribution models. *Methods in Ecology and Evolution* 10 (2):225–232. https://doi.org/10.1111/2041-210X.13107.

Venables, W. N., and B. D. Ripley. 2002. *Modern applied statistics with S* Fourth edition. New York: Springer. http://www.stats.ox.ac.uk/pub/MASS4.

Warmerdam, F. 2008. The Geospatial Data Abstraction Library. In *Open source approaches in spatial data handling*, eds. G. B. Hall and M. Leahy, 87–104. Berlin: Springer-Verlag.