Workshop: spatial data analysis

Roger Bivand

4-5 October 2018

Introduction

This workshop

- 4/10 10:00-13:00 Representing and visualizing spatial (and spatio-temporal) data - Pebesma (2018), Tennekes (2018); I'll talk about the current status, and maybe speculate about what may happen to raster and spacetime data objects.
- 4/10 14:30-17:30 Two data sets (could be more) to examine and represent: Katrina business re-openings (Lam et al., 2012; LeSage et al., 2011b,a); Hägerstrand adoption of agricultural subsidies (Bivand, 2015). I may want to take up the Boston housing data set too (Bivand, 2017).
- 5/10 10:00-13:00 A partial review of packages covered by recent articles in JSS and RJ and others. Does it help to provide code as supplementary material?
- 5/10 14:30-17:30 Maybe trying out and comparing some of these estimation methods based on the fresh articles and their attached code; open for participants' own suggestions for discussion and attempts to implement workflows

Software

- While we are obviously discussing R software, this is by no means homogeneous, and many alternatives exist, for example for spatio-temporal point processes
- The main focus will be on trends in R software, but notice also **reticulate** for cross-working with Python
- We'll also mention the open source projects that R-spatial crucially depends on: PROJ, GDAL and GEOS
- Things are changing fast; not only RStudio, also Microsoft (SQL Server and PowerBI with R inside)
- Should R+packages instances be fully-fitted or lean: MP challenges?

sf and revised representations

A short history of handling spatial data in R

- pre-2003: several people doing spatial statistics or map manipulation with S-Plus, and later R (e.g. spatial in MASS; spatstat, maptools, geoR, splancs, gstat, ...)
- 2003: workshop at DSC, concensus that a package with base classes should be useful; this ended up being a multiplicator
- 2003: start of r-sig-geo
- 2003: rgdal released on CRAN
- 2005: sp released on CRAN; sp support in rgdal
- 2008: Applied Spatial Data Analysis with R

A short history of handling spatial data in R

- 2011: rgeos released on CRAN
- 2013: second edition of Applied Spatial Data Analysis with R

Simple feature access in R: package sf

- Simple feature access is an ISO standard that is widely adopted. It is used in spatial databases, GIS, open source libraries, GeoJSON, GeoSPARQL, etc.
- 2016-7: simple features for R, R consortium support (considered pretty much “finished”)
- 2017-8: spatiotemporal tidy arrays for R, R consortium support (design phase)

Simple feature access in R: package `sf`

What is this about?

- *feature*: abstraction of real world phenomena (type, or instance)
- *simple feature*: feature with all geometric attributes described piecewise by straight line or planar interpolation between sets of points
- 7 + 10 types, 68 classes, of which 7 used in like 99% of the use cases
- text and binary serialisations (WKT, WKB)
- support for mixed type (`GEOMETRYCOLLECTION`), and type mix (`GEOMETRY`)
- support for empty geometry (empty set: somewhat like `NA`)

Simple feature access in R: package `sf`

```
> library(sf)
## Linking to GEOS 3.7.0, GDAL 2.4.0dev-358fa65bf3, PROJ 5.2.0
> sf_extSoftVersion()
##                    GEOS                    GDAL
##                "3.7.0" "2.4.0dev-358fa65bf3"
##                 proj.4         GDAL_with_GEOS
##                "5.2.0"                 "true"
```

Support for the PostGIS **`lwgeom`** library has been split out into the lwgeom package

```
> library(lwgeom)
## Linking to liblwgeom 2.5.0dev r16016, GEOS 3.7.0, proj.4 5.2.0
> lwgeom_extSoftVersion()
##             lwgeom               GEOS
## "2.5.0dev r16016"            "3.7.0"
##             proj.4
##            "5.2.0"
```

Simple feature access in R: package sf

access refers to standardised encodings, such as well-known text (WKT):

```
> (pt = st_point(c(2,4)))
## POINT (2 4)
```

and well-known binary, the binary form in which spatial databases put geometries in BLOBs, binary large objects, converted back by

```
> (pt_bin = st_as_binary(pt))
##  [1] 01 01 00 00 00 00 00 00 00 00 00 00 40 00 00
## [16] 00 00 00 00 10 40
> st_as_sfc(list(pt_bin))[[1]]
## POINT (2 4)
```

Simple feature access in R: package `sf`

Package `sf` uses simple R structures to store geometries:

```
> str(pt)
##  'XY' num [1:2] 2 4
> str(st_linestring(rbind(c(0,0), c(0,1), c(1,1))))
##  'XY' num [1:3, 1:2] 0 0 1 0 1 1
> str(st_polygon(list(rbind(c(0,0), c(0,1), c(1,1),
+   c(0,0)))))
## List of 1
##  $ : num [1:4, 1:2] 0 0 1 0 0 1 1 0
##  - attr(*, "class")= chr [1:3] "XY" "POLYGON" "sfg"
```

Tidyverse, list-columns

According to the “tidy data” paper (Wickham 2014), data is tidy when

1. Each variable forms a column.
2. Each observation forms a row.
3. Each type of observational unit forms a table.

It is not directly clear how this maps to geometries: should a coordinate dimension (e.g. latitude) form a column, should each coordinate (x,y pair) form a colum, or should a whole geometry (e.g. polygon, with holes), form a column?

Early attempts (and ggplot2 up to version 2.2.1) wanted *simple* columns, meaning each coordinate split over two columns. An approach called *fortify* would mold (tidy?) polygons into simple `data.frame`s. It is well known that this approach has its limitations when polygons have holes.

Since the UseR! 2016 keynote of Hadley, list-columns have been declared tidy. One of the arguments for this was exactly this: polygons with holes are hard to represent in simple `data.frame`s. Other cases are: nested `data.frame`s, or columns that contain, for each record, a model object e.g. obtained from `lm`.

The tidy data rule for simple feature means: we have a `data.frame` where each *feature* forms a row. A single column (a list-column) contains the geometry for each observation. This resembles spatial databases, such as PostGIS.

Package `sf` puts features in `sf` tables deriving from `data.frame` or `tbl_df`, which have geometries in a list-column of class `sfc`, where each list element is a single feature's geometry of class `sfg`. Feature geometries are represented in R by

- a numeric vector for a single point (`POINT`)
- a numeric matrix (each row a point) for a set of points (`MULTIPOINT` or `LINESTRING`)
- a list of matrices for a set of set of points (`MULTIINESTRING`, `POLYGON`)
- a list of lists of matrices (`MULTIPOLYGON`)
- a list of anything mentioned above (`GEOMETRYCOLLECTION`)

(all other classes also fall in one of these categories)

Other tidy aspects of `sf`:

- all functions/methods start with `st_` (press tab to search), use `_` and lower case
- all function have data as first argument, “pipe-friendly”
- `read_sf` is an alias for `st_read` with tidy defaults: silent, `stringAsFactors = FALSE`
- many tidy verbs implemented as methods for `sf` objects (see further down)

Reference systems

If one wants to know which position of the Earth we refer to, coordinates of geospatial data require a reference system:

- geodesic/geographic coordinates need an order (long/lat or lat/long?), a unit (rad, arc_degree?) and a datum (a reference ellipsoid: WGS84, ETRS89, NAD27?)
- cartesian/projected coordinates (e.g. UTM, web Mercator) need also measurement units, and some way of encoding how they relate to geodesic coordinates, in which datum (the Proj.4 string)

Reference systems

To handle coordinate reference systems, to convert coordinates (projections) and do datum transformations, Proj.4 is the code base most widely adopted, and actively maintained, by the open source geospatial community. Package `sf` has `crs` objects that register coordinate reference systems. `crs` objects are registered as an attribute of geometry collections.

```
> st_crs("+proj=longlat +datum=WGS84")  # "Proj.4 string"
## Coordinate Reference System:
##   EPSG: 4326
##   proj4string: "+proj=longlat +datum=WGS84 +no_defs"
> st_crs(3857)                          # EPSG code
## Coordinate Reference System:
##   EPSG: 3857
##  proj4string: "+proj=merc +a=6378137 +b=6378137 +lat_ts=0.0 +lon_0=0.0 +
> st_crs(3857)$units                    # reveal units
## [1] "m"
> st_crs(NA)                            # unknown
## Coordinate Reference System: NA
```

Array data: rasters, spatial time series

Although `sp` has some simple infrastructure for them in the `Spatial` class framework, raster data in R are best handled by the `raster` package. It is feature-rich, well integrated with the `Spatial` class framework, and can deal with massive rasters, as long as they fit on the local disk. It does not integrate particularly well with the tidyverse, or with simple features. Neither does it handle four- or higher-dimensional dimensional data (e.g. x,y,t,color), or time series data for features.

A follow-up project of the “simple features for R” project tries to address a number of these issues

Summary/outlook

In comparison to `sp`, package `sf`

- implements all types and classes of simple features
- has no support for gridded (raster) data
- uses `data.frame`s for features, and list columns for feature geometry
- uses S3 instead of S4
- is also built on top of GDAL, GEOS and Proj.4
- tries to provide a simpler and cleaner API (or user experience) conversions to/from `sp` makes it still easy to work “backwards”

Summary/outlook

What's really new, and better in `sf`, compared to `sp`?

- simple features is a widely adopted standard
- tidyverse compatibility
- ggplot2 support (`install_github`, under development)
- support for measurement units
- partial support for computations using geographic coordinates
- support by mapview, tmap, mapedit; 15 revdeps on CRAN
- binary geom ops: fast (indexed), low memory footprint; flexible `st_join`
- (c)lean Rcpp interface to external dependencies GDAL/GEOS/Proj.4
- fast WKB (de)serialization, in C++

Should we use sf for finding neighbours

- There is a vignette in **spdep** that describes this in painful detail
- For now, convert to the **sp** representation and continue as before
- Because **sf** uses GEOS for most predicates, it requires valid geometries, and operates on the geometry as such
- **spdep** simplifies by only using (snapped) boundary points, and is faster; beware of centroids of observations with multiple exterior rings

tmap and alternatives for visualization

Using tmap

- Martijn Tenneke's article on **tmap** is now published
- Using the article as a template, we'll see how JSS and RJ articles (and others) may be used for finding out where things are going
- There are further contributions in Lovelace, Nowosad and Muenchow: Geocomputation in R, especially chapter 8
- Extra code in the **code** directory
- Zev Ross' blog

Starting article code

The article uses built-in and downloaded data sets

```
> library("tmap") # required version 1.11-1 or later
> library("tmaptools") # required version 1.2-3 or later
> library(sf)
> data("World", "metro", package = "tmap")
> metro$growth <- (metro$pop2020 - metro$pop2010) / (metro$pop2010 * 10) * 100
```

Figure 1

```
> m1 <- tm_shape(World) +
+     tm_polygons("income_grp", palette = "-Blues",
+       title = "Income class", contrast = 0.7, border.col = "grey30", id = "name") +
+     tm_text("iso_a3", size = "AREA", col = "grey30", root = 3) +
+   tm_shape(metro) +
+     tm_bubbles("pop2010", col = "growth", border.col = "black",
+       border.alpha = 0.5,
+       breaks = c(-Inf, 0, 2, 4, 6, Inf) ,
+       palette = "-RdYlGn",
+       title.size = "Metro population (2010)",
+       title.col = "Annual growth rate (%)",
+       id = "name",
+       popup.vars = c("pop2010", "pop2020", "growth")) +
+   tm_format_World() + tm_style_gray(frame.lwd = 2)

## Warning in tm_format_World(): tm_format_World
## is deprecated as of tmap version 2.0. Please use
## tm_format("World", ...) instead

## Warning in tm_style_gray(frame.lwd = 2):
## tm_style_gray is deprecated as of tmap version
## 2.0. Please use tm_style("gray", ...) instead
```

Figure 1

```
> m1 <- tm_shape(World) +
+     tm_polygons("income_grp", palette = "-Blues",
+       title = "Income class", contrast = 0.7, border.col = "grey30", id = "name") +
+     tm_text("iso_a3", size = "AREA", col = "grey30", root = 3) +
+   tm_shape(metro) +
+     tm_bubbles("pop2010", col = "growth", border.col = "black",
+       border.alpha = 0.5,
+       breaks = c(-Inf, 0, 2, 4, 6, Inf) ,
+       palette = "-RdYlGn",
+       title.size = "Metro population (2010)",
+       title.col = "Annual growth rate (%)",
+       id = "name",
+       popup.vars = c("pop2010", "pop2020", "growth")) +
+   tm_format("World") + tm_style("gray", frame.lwd = 2)
```

Figure 1

```
## Note that tm_style("gray") resets all options set with tm_layout, tm_view, tm_format, or tm_legend. It is therefore recommended to place the tm_style

## Variable "growth" contains positive and negative values, so midpoint is set to 0. Set midpoint = NA to show the full spectrum of the color palette.
```

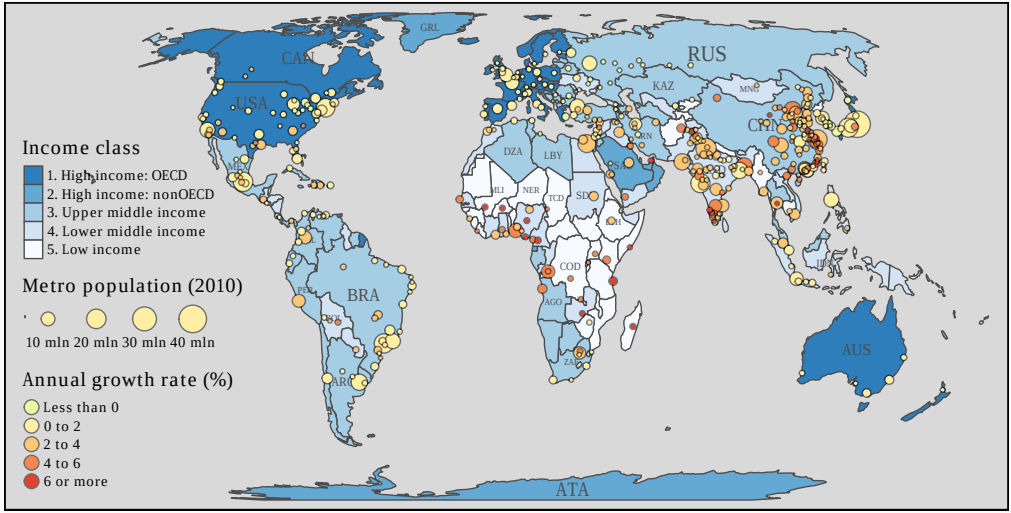


Figure 1

```
> m1 <- tm_shape(World) + tm_style("gray", frame.lwd = 2) +
+     tm_polygons("income_grp", palette = "-Blues",
+       title = "Income class", contrast = 0.7, border.col = "grey30", id = "name") +
+     tm_text("iso_a3", size = "AREA", col = "grey30", root = 3) +
+   tm_shape(metro) +
+     tm_bubbles("pop2010", col = "growth", border.col = "black",
+       border.alpha = 0.5,
+       breaks = c(-Inf, 0, 2, 4, 6, Inf) ,
+       palette = "-RdYlGn",
+       title.size = "Metro population (2010)",
+       title.col = "Annual growth rate (%)",
+       id = "name",
+       popup.vars = c("pop2010", "pop2020", "growth")) +
+   tm_format("World")
```

Figure 1

```
## Variable "growth" contains positive and negative values, so midpoint is set to 0. Set midpoint = NA to show the full spectrum of the color palette.
```

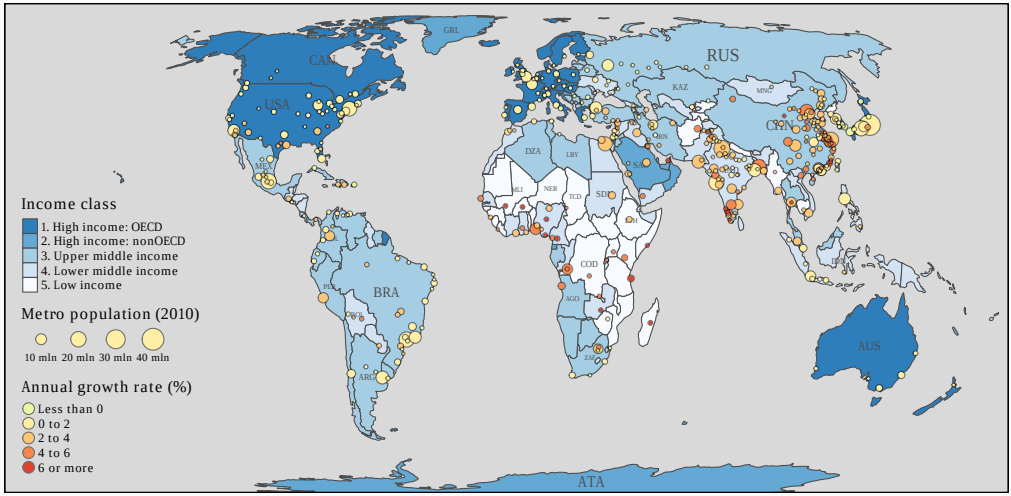


Figure 2

```
> m0 <- tm_shape(metro) +
+   tm_bubbles(size = "pop2030") +
+   tm_format_World() +
+   tm_style_cobalt()

## Warning in tm_format_World(): tm_format_World
## is deprecated as of tmap version 2.0. Please use
## tm_format("World", ...) instead

## Warning in tm_style_cobalt(): tm_style_white is
## deprecated as of tmap version 2.0. Please use
## tm_style("cobalt", ...) instead
```

Figure 2

```
> m0 <- tm_shape(metro) +
+   tm_bubbles(size = "pop2030") +
+   tm_format("World") +
+   tm_style("cobalt")
```

Figure 2

```
## Note that tm_style("cobalt") resets all options set with tm_layout, tm_view, tm_format, or tm_legend. It is therefore recommended to place the tm_sty
```

Figure 2

```
> m0 <- tm_shape(metro) +
+   tm_style("cobalt") +
+   tm_bubbles(size = "pop2030") +
+   tm_format("World")
```

Figure 2

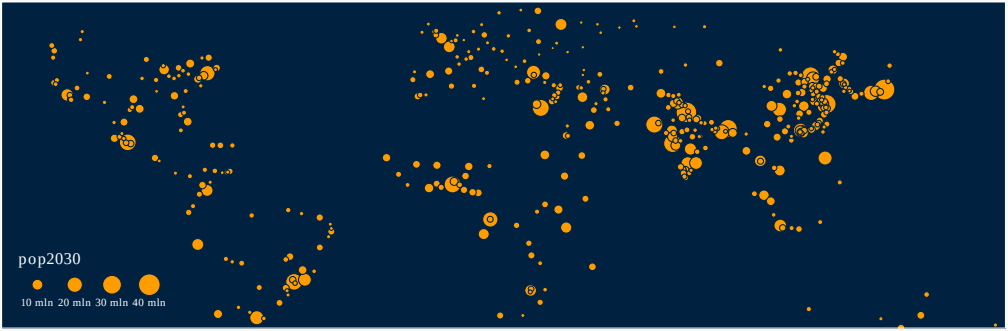


Figure 3

```
> m21 <- tm_shape(World) + tm_polygons(c("blue", "red")) + tm_layout(frame.lwd = 1.5)
```

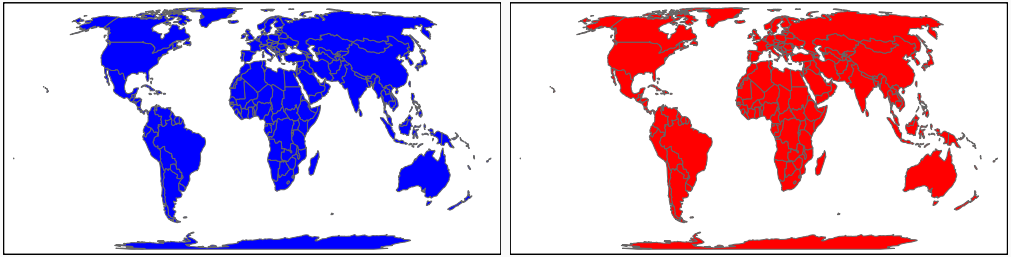


Figure 3

(transparent background)

```
> m21 <- m21 + tm_layout(bg.color="transparent")
```

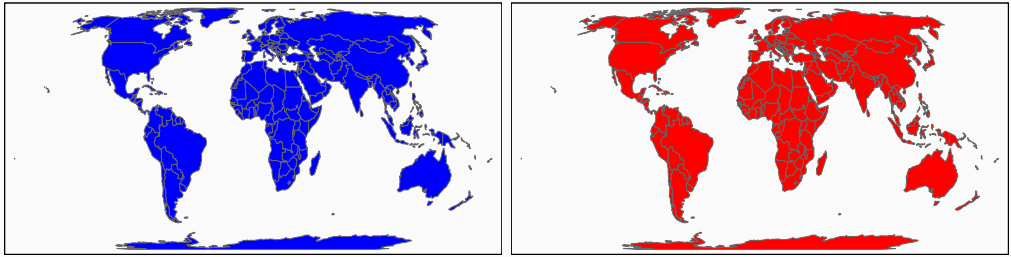
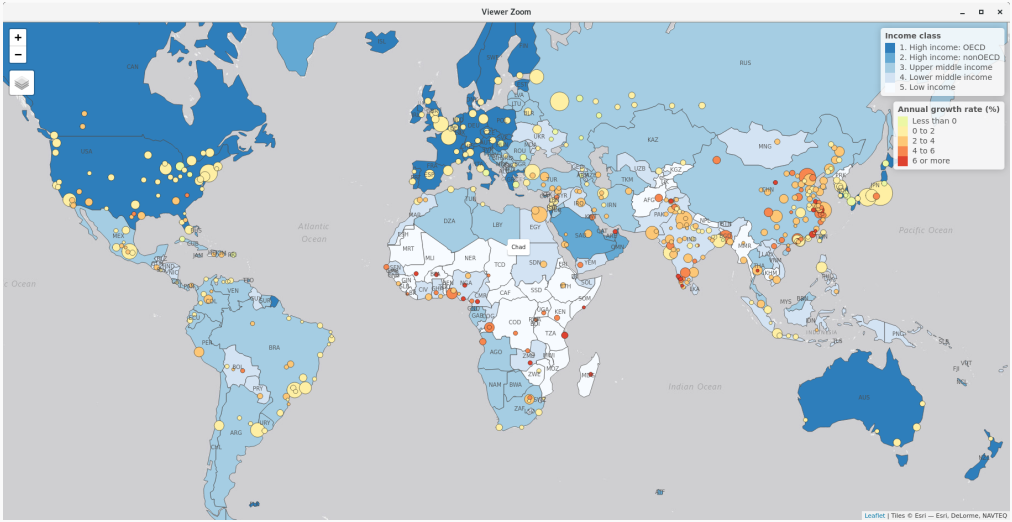


Figure 6

```
> tmap_mode("view")
> m1
```



Same approach to Geocomputation with R, chapter 8

Geocomputation with R, chapter 8

- A new book is approaching completion, online at: https://geocompr.robinlovelace.net/
- It is also available on github, with the book code: https://github.com/Robinlovelace/geocompr
- This means that we can reproduce the examples used in the book chapters
- Chapter 8 is an introduction to **tmap**; but see blogs for cartography and use of geom_sf() in **ggplot2**

Figure 8.1

(code in `code/08-tmshape.R`)

```
> library(tmap)
> library(spData)
> library(spDataLarge)
> library(magrittr)
> # Add fill layer to nz shape
> m1 = tm_shape(nz) + tm_fill() +
+   tm_layout(title = "tm_shape(nz) +\n  tm_fill()", title.size = 0.7) + tm_layout(bg.color="transparent")
> # Add border layer to nz shape
> m2 = tm_shape(nz) + tm_borders() +
+   tm_layout("tm_shape(nz) +\n  tm_borders()", title.size = 0.7) + tm_layout(bg.color="transparent")
> # Add fill and border layers to nz shape
> m3 = tm_shape(nz) + tm_fill() + tm_borders() +
+   tm_layout("tm_shape(nz) +\n  tm_fill() +\n  tm_borders()", title.size = 0.7) + tm_layout(bg.color="transparent")
> m123 <- tmap_arrange(m1, m2, m3, nrow = 1)
```

Figure 8.1

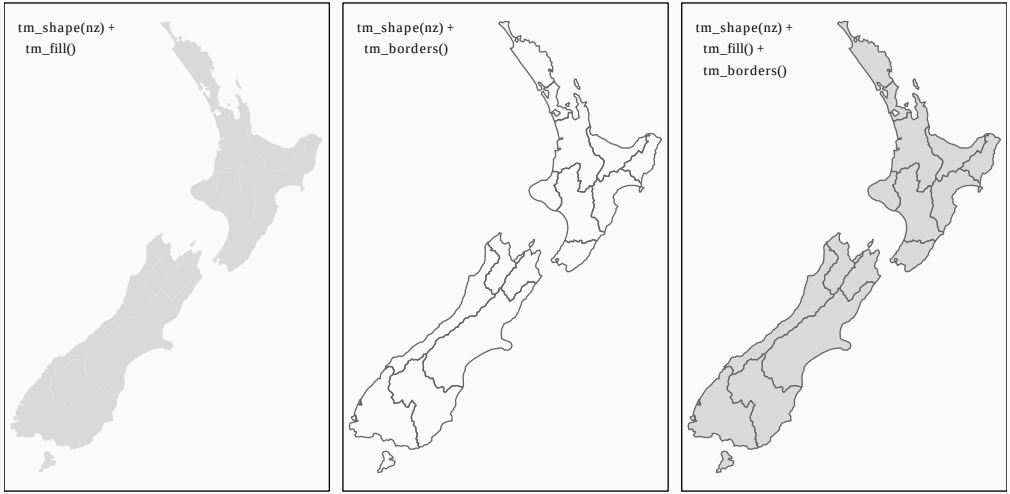

tm_shape(nz) +
tm_fill()
tm_shape(nz) +
tm_borders()
tm_shape(nz) +
tm_fill() +
tm_borders()

Figure 8.2

The `nz_elev` data file has been in spDataLarge from 0.2.6.1, mine was 0.2.6.0

```
> map_nz = tm_shape(nz) + tm_polygons() + tm_layout(bg.color="transparent")
> map_nz1 = map_nz +
+   tm_shape(nz_elev) + tm_raster(alpha = 0.7)
```

Figure 8.2

```
> library(elevatr)
> nz_elev_new <- get_elev_raster(as(nz, "Spatial"), z=5, src="aws")
> nz_water = st_union(nz) %>% st_buffer(22200) %>%
+   st_cast(to = "LINESTRING")
> is.na(nz_elev_new) <- nz_elev_new < 0
> nz_elev_new <- crop_shape(nz_elev_new, nz_water)
> map_nz1 = map_nz +
+   tm_shape(nz_elev_new) + tm_raster(alpha = 0.7)
```

Figure 8.2

```
> map_nz2 = map_nz1 +
+   tm_shape(nz_water) + tm_lines()
> map_nz3 = map_nz2 +
+   tm_shape(nz_height) + tm_dots()
> map_nz123 <- tmap_arrange(map_nz1, map_nz2, map_nz3)
```

Figure 8.2

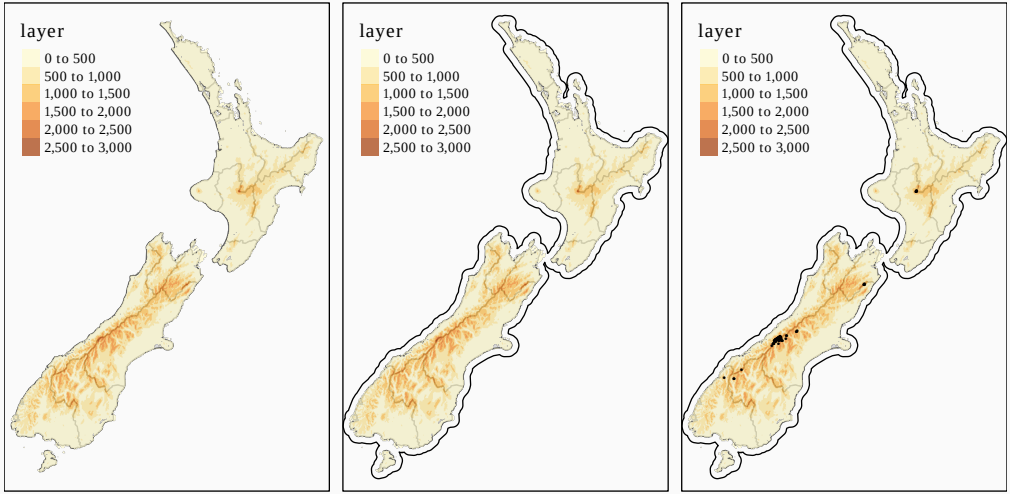

layer
0 to 500
500 to 1,000
1,000 to 1,500
1,500 to 2,000
2,000 to 2,500
2,500 to 3,000
layer
0 to 500
500 to 1,000
1,000 to 1,500
1,500 to 2,000
2,000 to 2,500
2,500 to 3,000
layer
0 to 500
500 to 1,000
1,000 to 1,500
1,500 to 2,000
2,000 to 2,500
2,500 to 3,000

Figure 8.14

```
> nz_region = st_bbox(c(xmin = 1340000, xmax = 1450000,
+                       ymin = 5130000, ymax = 5210000)) %>%
+   st_as_sfc() %>%
+   st_set_crs(st_crs(nz_height))
> nz_region

## Geometry set for 1 feature
## geometry type:  POLYGON
## dimension:      XY
## bbox:           xmin: 1340000 ymin: 5130000 xmax: 1450000 ymax: 5210000
## epsg (SRID):    2193
## proj4string:   +proj=tmerc +lat_0=0 +lon_0=173 +k=0.9996 +x_0=1600000 +y_0=10000000 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs

## POLYGON ((1340000 5130000, 1450000 5130000, 145...
```

Figure 8.14

```
> nz_height_map = tm_shape(nz_elev, bbox = tmaptools::bb(nz_region)) +
+   tm_raster(style = "cont", palette = "YlGn", legend.show = TRUE) +
+   tm_shape(nz_height) + tm_symbols(shape = 2, col = "red", size = 1) +
+   tm_scale_bar(position = c("left", "bottom"))
```

Figure 8.14

```
> nz_map = tm_shape(nz) + tm_polygons() +
+   tm_shape(nz_height) + tm_symbols(shape = 2, col = "red", size = 0.1) +
+   tm_shape(nz_region) + tm_borders(lwd = 3)
```

Figure 8.14

```
> library(grid)
> nz_height_map
> print(nz_map, vp = viewport(0.8, 0.27, width = 0.5, height = 0.5))
```

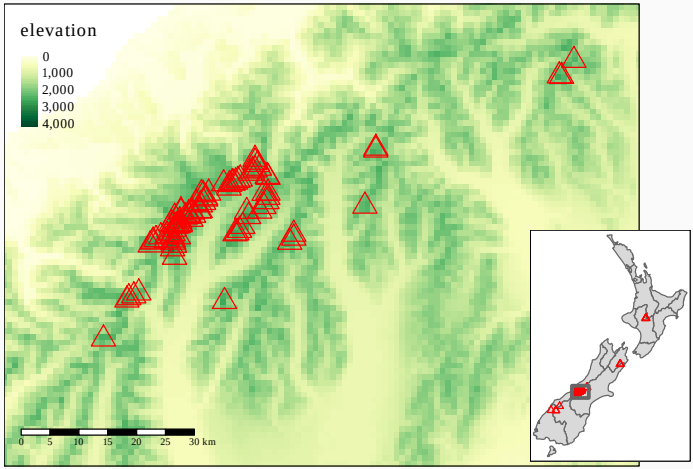
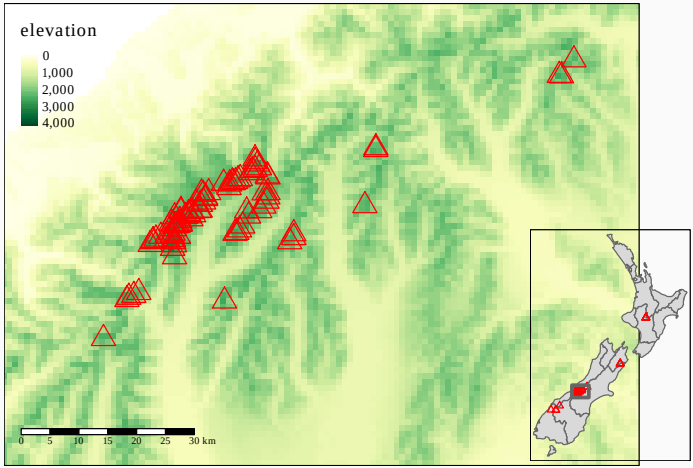


Figure 8.14

```
> nz_map = nz_map + tm_layout(bg.color="transparent")
```



Katrina and spatial probit

Katrina and spatial probit

LeSage et al. (2011b) provided data and code (link now stale) for their analysis, and this data is included in **spatialprobit** (see also Lam et al., 2012; LeSage et al., 2011a)

```
> library(spatialprobit)
> data(Katrina)
```

Create sf object

```
> library(sf)
> sf_katrina <- st_as_sf(Katrina.raw, coords=c("long", "lat"))
> st_crs(sf_katrina) <- "+proj=longlat +datum=WGS84"
```

Use Date class

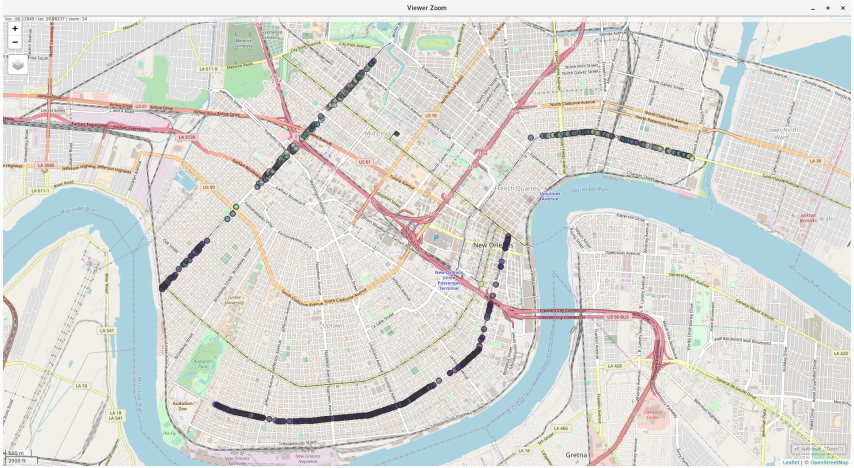
```
> is.na(sf_katrina$days) <- sf_katrina$days == 99999
> sf_katrina$Days <- as.Date("2005-08-29") + sf_katrina$days
> sf_katrina$y0_90 <- as.numeric(sf_katrina$days < 90)
> diff(as.Date(names(table(sf_katrina$Days))))

## Time differences in days
##  [1]  7  7  7  7  7  7  7  7  7  7  6  8 14  7  7
## [16]  7  7  7  7 14  7  7 14  7  7  7  7 14 14 14
## [31] 14 14 14 14 45 20 35
```

Recode factors

```
> library(car)

## Loading required package: carData

> sf_katrina$f_owntype <- recode(sf_katrina$owntype, "1='sole'; 2='local'; 3='national'", as.factor=TRUE)
> sf_katrina$f_sesstatus <- recode(sf_katrina$sesstatus, "c(1,2)='lower'; c(4,5)='upper'; else='av'", as.factor=TRUE)
> sf_katrina$f_sizeemp <- recode(sf_katrina$sizeemp, "1='small'; 2='av'; 3='large'", as.factor=TRUE)
> sf_katrina$f_street1 <- recode(sf_katrina$street1, "1='Magazine Street'; 2='Carrollton Avenue'; 3='St. Claude Avenue'", as.factor=TRUE)
```

Where are they?

```
> library(mapview)
> mapview(sf_katrina)
```



Katrina businesses staying shut: open questions

- The JRSS-A article looks at a spatial probit scenario, where the spatial process is in the decision to reopen spilling over between neighbours given covariates
- Is this the only feasible approach? Why at least is there no street random effect?
- Are the neighbours sensibly chosen (no attention paid to whether businesses compete or cooperate)?
- Could we think of this as a spatial survival model: **spatsurv** or **spBayesSurv**?

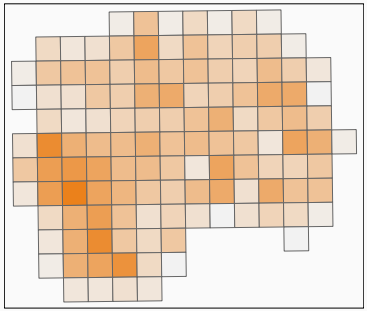
Hägerstrand

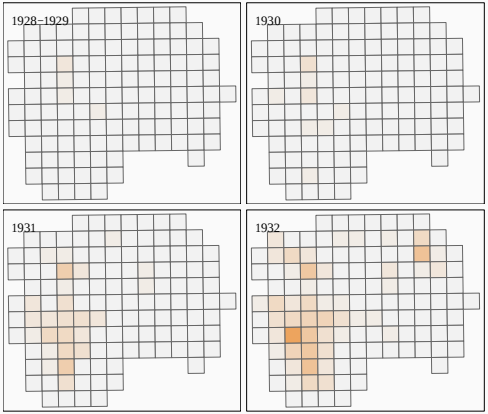
Innovationsforloppet ur Korologisk Synspunkt

- Torsten Hägerstrand played an important role in the promotion of mathematical geography, both through his pioneering research and through active recruitment of guest scholars, both bringing foreigners to Sweden, and sending his own students abroad
- This nicely mirrors his own work on the diffusion of innovations Hägerstrand (1953, 1967), in which he hypothesises that farmers are more likely to adopt innovations if they are in close proximity to earlier adopters
- Initial and subsequent adopters were recorded in 125 5km square grid cells around the settlement of Asby for 1929–1932, together with all potential adopters who could be entitled to receive a subsity for pasture improvement

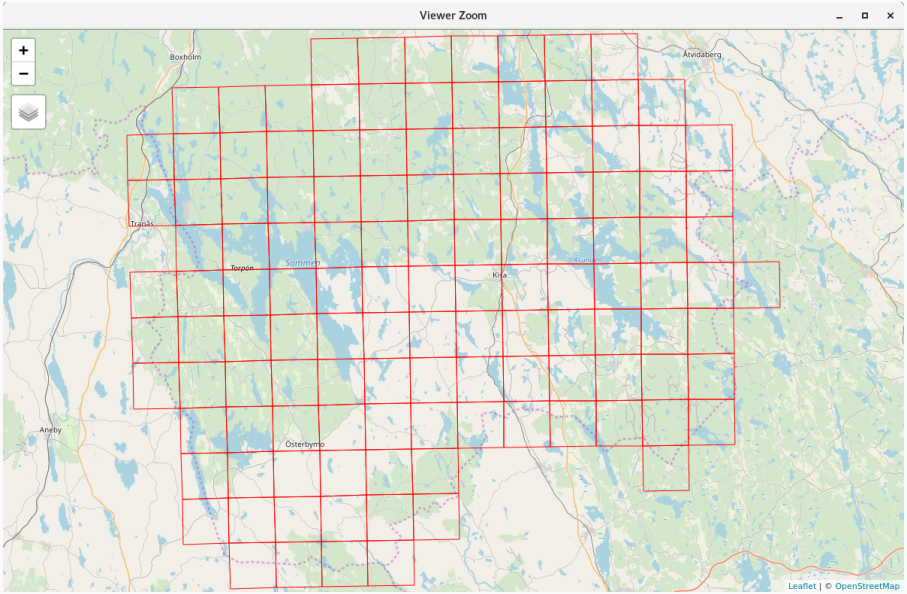
```
> library(sf)
> SPol2_RT38 <- st_read("SPol2_RT38.gpkg")
> #library(RColorBrewer)
> rds <- colorRampPalette(c("grey95", "#EB811B"))
> dive <- colorRampPalette(c("#0E302F", "grey95", "#EB811B"))
```

Pasture subsidies, Asby 1929–1932





Asby study area in GE



- The model of spatial interaction fitted in Hägerstrand (1953, p. 246) was calibrated from numbers of telephone calls and measured distances for logged telephone calls from each local exchange to destinations up to 50km:

$$\log F_I = 0.7966 - 1.585 \log d$$

where d is distance measured in km (Hägerstrand, 1967; Cliff, 1970)

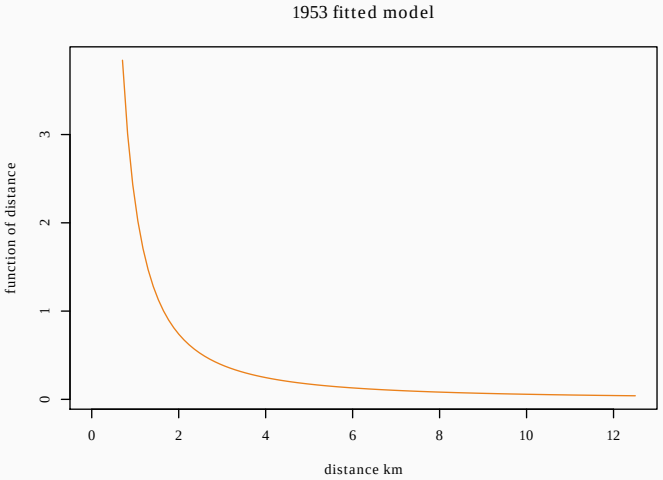
- Note that this relationship is isotropic
- Using the same coefficient estimates, we can also reconstruct the MIF

Mean information field — creation

We follow Hägerstrand (1953) by creating a 25×25 grid of one km squares to generate the predicted interactions, which were then summed to 5km squares, and an (unknown) arbitrary value entered in the central cell. From this we can create a MIF summing to unity.

```
> x <- y <- seq(-12.5, 12.5, 1)
> x5 <- y5 <- seq(-12.5, 12.5, 5)
> xy <- expand.grid(x=x, y=y)
> xy$z <- sqrt(xy$x^2 + xy$y^2)
> xy$f <- exp(0.7966 + -1.585*log(xy$z))
> m <- matrix(xy$f, length(x), length(y))
> m1 <- matrix(NA, 5, 5)
> s <- c(1,6,11,16,21,26)
> for (i in 1:5)
+   for (j in 1:5)
+     m1[i,j] <- sum(m[s[i]:s[i+1], s[j]:s[j+1]])
> m1[3,3] <- 57.4
> MIF <- as.matrix(read.csv("asbyMIF.csv", header=FALSE))
> MIFgrd <- expand.grid(x=1:5, y=1:5)
```

Mean information field — comparison



Original MIF

0.0096	0.0140	0.0168	0.0140	0.0096
0.0140	0.0301	0.0547	0.0301	0.0140
0.0168	0.0547	0.4431	0.0547	0.0168
0.0140	0.0301	0.0547	0.0301	0.0140
0.0096	0.0140	0.0168	0.0140	0.0096

Generated MIF

0.0094	0.0139	0.0166	0.0139	0.0094
0.0139	0.0305	0.0549	0.0305	0.0139
0.0166	0.0549	0.4433	0.0549	0.0166
0.0139	0.0305	0.0549	0.0305	0.0139
0.0094	0.0139	0.0166	0.0139	0.0094

Mean information field and diffusion

- The mean information field provides a view of the expected diffusion paths between grid cells
- Better, it has a clear behavioural motivation in the underlying relationship between contacts generating information spillovers and distance

-However, reviews including Cliff (1970) and Tinline (1971), extended in Cliff and Ord (1973), suggest that this micro-model is not fully successful when compared with the data

```
> library(sp)
> MIF <- as.matrix(read.csv("asbyMIF.csv", header=FALSE))
> GT <- GridTopology(c(-2*4972.927, -2*4972.927), c(4972.927, 4972.927), c(5, 5))
> adopts <- names(SPol2_RT38)[grep("adopt", names(SPol2_RT38))]
>
> pre_sim_lut <- function(GT, SPols) {
+   require(maptools)
+   SPix_MIF <- as(SpatialGrid(GT), "SpatialPixels")
+   crds <- coordinates(SPols)
+   lut <- matrix(as.integer(NA), ncol=length(SPix_MIF), nrow=length(SPols))
+   for (i in 1:length(SPols)) {
+     m0 <- elide(SPix_MIF, shift=crds[i,])
+     proj4string(m0) <- CRS(proj4string(SPols))
+     lut[i,] <- over(m0, as(SPols, "SpatialPolygons"))
+   }
```

Mean information field and diffusion

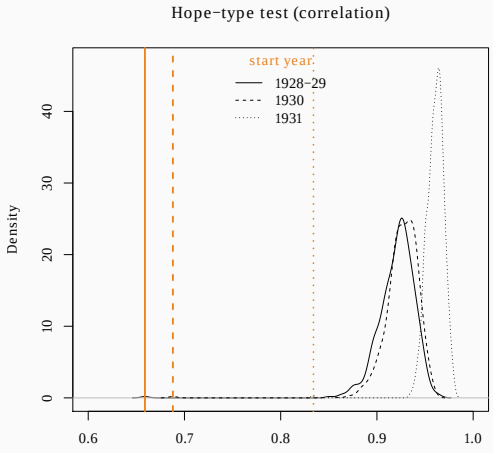
```
> do_sim <- function(SPols, MIF, lut, adopts, farms, t=1) {
+   cMIF <- cumsum(c(MIF))
+   if (cMIF[length(cMIF)] <  1) cMIF[length(cMIF)] <- 1
+   T <- length(adopts)
+   xx <- sapply(adopts, function(x) sum(SPols[[x]]))
+   y <- SPols[[adopts[t]]]
+   yy <- xx[t]
+   while (t < T) {
+     t_cands <- rep(x=1:length(SPols), times=SPols[[adopts[t]]])
+     while (yy < xx[(t+1)]) {
+       sender <- sample(t_cands, 1)
+       hits <- which(runif(1) > cMIF)
+       if (length(hits) == 0) hits <- 1
+       hit <- hits[length(hits)]
+       ihit <- lut[sender, hit]
+       if (!is.na(ihit)) {
+         if (y[ihit] < farms[ihit]) {
+           y[ihit] <- y[ihit] + 1
+           yy <- yy + 1
+         }
+       }
+     }
+ #    cat(t, yy, "\n")
+     t <- t + 1
+   }
+   y
+ }
```

Mean information field and diffusion

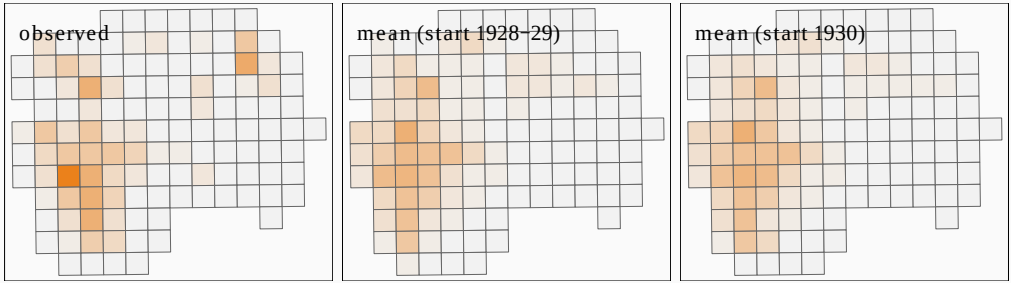
```
> lut <- pre_sim_lut(GT, as(SPol2_RT38, "Spatial"))
> nsim <- 500
> res <- matrix(0, nrow=length(as(SPol2_RT38, "Spatial")), ncol=nsim)
> set.seed(1)
> for (i in 1:nsim) res[, i] <- do_sim(SPols=as(SPol2_RT38, "Spatial"), MIF=MIF, lut=lut, adopts=adopts, farms=SPol2_RT38$farms, t=1)
> res1 <- cbind(res, SPol2_RT38$adopt32)
> mm1 <- apply(res1, 1, mean)
> hope1 <- apply(res1, 2, function(x) cor(x, mm1))
> for (i in 1:nsim) res[, i] <- do_sim(SPols=as(SPol2_RT38, "Spatial"), MIF=MIF, lut=lut, adopts=adopts, farms=SPol2_RT38$farms, t=2)
> res2 <- cbind(res, SPol2_RT38$adopt32)
> mm2 <- apply(res2, 1, mean)
> hope2 <- apply(res2, 2, function(x) cor(x, mm2))
> for (i in 1:nsim) res[, i] <- do_sim(SPols=as(SPol2_RT38, "Spatial"), MIF=MIF, lut=lut, adopts=adopts, farms=SPol2_RT38$farms, t=3)
> res3 <- cbind(res, SPol2_RT38$adopt32)
> mm3 <- apply(res3, 1, mean)
> hope3 <- apply(res3, 2, function(x) cor(x, mm3))
```

Hope-type test

We can conduct a Hope-type test as suggested in Cliff (1970) and Cliff and Ord (1973) with the Pearson correlation coefficient between the mean map of simulations and observed for Hägerstrand's MIF. We simulate up to actual annual adoption counts as proposed by Bivand (1980) and used in Kamiński (1988). It is very unlikely that the original (or equivalently the reconstructed) MIF could have generated the observed diffusion pattern, starting with 1928–29, 1930 or 1931.



Pasture subsidies adopted and predicted, Asby 1932



Boston

Overview

- In addition, the results I presented at useR! 2016 on this data set aggregated the data to air pollution model output zones (see also Bivand, 2017)
- Here, based on Bivand et al. (2017), spatially structured random effects are added by air pollution model output zones instead

Boston data set

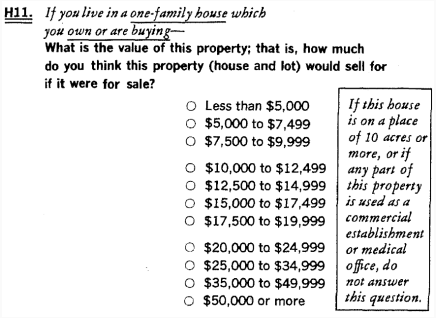
- Harrison and Rubinfeld (1978) used a hedonic model to find out how house values were affected by air pollution in Boston, when other variables were taken into consideration
- They chose to use 506 census tracts as units of observation, but air pollution values were available from model output for 122 zones, of which less than 100 fell within the study area
- By taking the 96 air pollution model output zones as the upper level in a hierarchical spatial model, we explore the consequences for the results

- The Harrison and Rubinfeld (1978) Boston housing data set has been widely used because of its availability from Belsley et al. (1980), Pace and Gilley (1997) and Gilley and Pace (1996)
- The underlying research question in the original article was the estimation of willingness to pay for clean air, using air pollution levels and house values in a hedonic regression
- As Pace and Gilley (1997, p. 337) showed clearly, the air pollution coefficient estimate in the model changed when residual spatial autocorrelation was taken into account (from -0.0060 to -0.0037), as did its standard error (from 0.0012 to 0.0016)

Motivation

- Is the strength of spatial autocorrelation observed in this data set a feature of the census tract observations themselves, or has it been introduced or strengthened by changes in the observational units used for the different variables?
- Our focus will be on the choices of observational units made in assembling the original data set, and on another relevant alternative
- Using an approximation to the model output zones from which the air pollution variable levels were taken, it will be shown that much of the puzzling spatial autocorrelation is removed

House value data



H11. *If you live in a one-family house which you own or are buying—*
What is the value of this property; that is, how much do you think this property (house and lot) would sell for if it were for sale?

- ○ Less than $5,000
- ○ $5,000 to $7,499
- ○ $7,500 to $9,999
- ○ $10,000 to $12,499
- ○ $12,500 to $14,999
- ○ $15,000 to $17,499
- ○ $17,500 to $19,999
- ○ $20,000 to $24,999
- ○ $25,000 to $34,999
- ○ $35,000 to $49,999
- ○ $50,000 or more

If this house is on a place of 10 acres or more, or if any part of this property is used as a commercial establishment or medical office, do not answer this question.

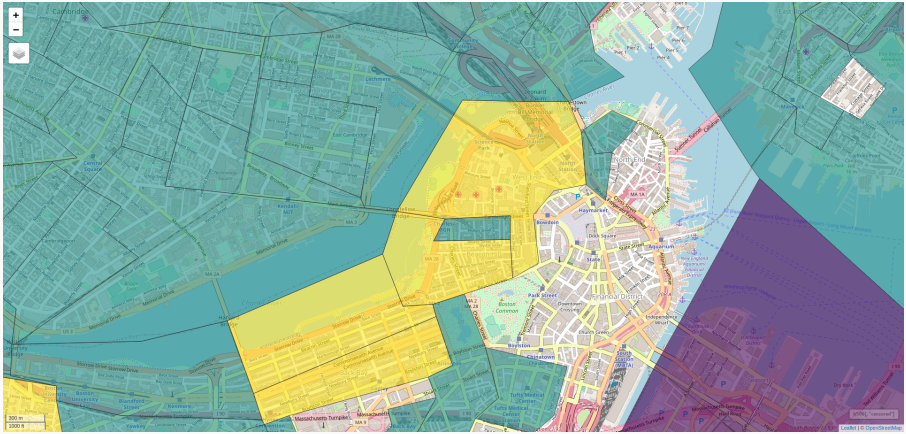
Harrison and Rubinfeld (1978) used median house values in 1970 USD for 506 census tracts in the Boston SMSA for owner-occupied one-family houses; census tracts with no reported owner-occupied one-family housing units were excluded from the data set. The relevant question is H11, which was answered by crossing off one grouped value alternative, ranging from under USD 5,000 to over USD 50,000

House value data

- The house value data have census tract support, and are median values calculated from group counts from the alternatives offered in H11; tracts with weighted median values in these upper and lower alternative value classes are censored
- The published census tract tabulations show the link between question H11 and the Statlib-based data (after correction)
- The median values tabulated in the census report can be reconstructed from the tallies shown in the same Census tables fairly accurately using the `weightedMedian` function in the **matrixStats** package in R, using linear interpolation

House value data

The effectiveness of the study was prejudiced by the fact that areas of central Boston with the highest levels of air pollution also lose house value data, either because of tract exclusion (no one-family housing units reported) or right or left censored tracts

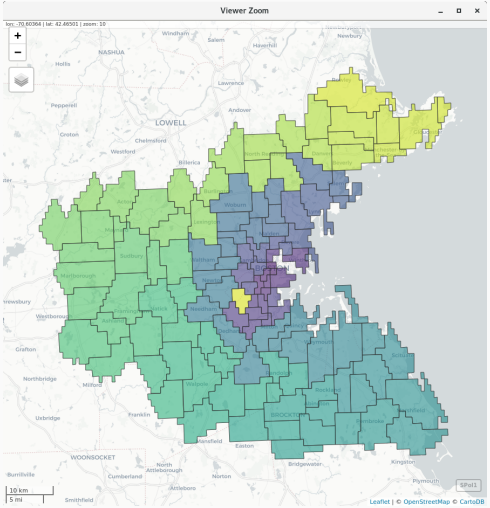


Air pollution data

- The data on air pollution concentrations were obtained from the Transportation and Air Shed SIMulation model (TASSIM) applied to the Boston air shed (Ingram and Fauth, 1974)
- The calibrated model results were obtained for 122 zones, and assigned proportionally to the 506 census tracts
- The NOX values in the published data sets are in units of 10 ppm (10 parts per million), and were then multiplied by 10 again in the regression models to yield parts per 100 million (pphm)
- Many of the smaller tracts belong to the same TASSIM zones; this is a clear case of change of support, with very different spatial statistical properties under the two different entitation schemes (Gotway and Young, 2002)

Air pollution data

```
> TASSIM <- st_read("TASSIM.gpkg")
> mapview(TASSIM)
```



A two-part report details the use of the TASSIM simulation model (Ingram and Fauth, 1974; Ingram et al., 1974). Both of these volumes include line-printer maps of the TASSIM zones, and the Fortran code in volume 2 (Ingram et al., 1974, pp. 183–185) shows the links between the 122 TASSIM zones and the line printer output. Western TASSIM zones appear to lie outside the Boston SMSA tracts included in the 506 census tract data set.

Using sf

- The Boston data set has 2D polygon and multipolygon geometries stored in a shapefile; shapefiles are pre-SF
- **sf** depends on GDAL for reading vector geometries and attribute data of features into rows of a data.frame
- The geometries are put in a special column in the data.frame
- The other columns contain the data for the census tracts included in the original data set, supplemented with others, such as the censoring status and model output zones

Reading the shapefile

Functions in **sf** are prefixed with `st_` meaning spatio-temporal (following PostGIS); `sf::st_read` uses GDAL vector drivers:

```
> library(sf)
> library(spData)
> b506 <- st_read(system.file("shapes/boston_tracts.shp", package="spData")[1])

## Reading layer 'boston_tracts' from data source '/home/rsb/lib/r_libs/spData/shapes/boston_tracts.shp' using driver 'ESRI Shapefile'
## Simple feature collection with 506 features and 36 fields
## geometry type:  POLYGON
## dimension:      XY
## bbox:           xmin: -71.52311 ymin: 42.00305 xmax: -70.63823 ymax: 42.67307
## epsg (SRID):    4267
## proj4string:    +proj=longlat +datum=NAD27 +no_defs
```

Aggregating geometries to model output zones

After dropping the censored census tracts, we need to derive the model output zones. The aggregate method calls `sf::st_union` on each unique grouping value, but the function called internally is a trick, putting only the first value of each variable in the output; for this reason we only retain the ids:

```
> b489 <- b506[b506$censored == "no",]
> t0 <- aggregate(b489, list(ids = b489$NOX_ID), head, n = 1)
> b94 <- t0[, c("ids", attr(t0, "sf_column"))]
```

Finding model output zones neighbours

We can find contiguous neighbours using `sf::st_relate`, but this does not yet scale to large numbers of geometries; we also find a no-neighbour model output zone:

```
> st_queen <- function(a, b = a) st_relate(a, b, pattern = "F***T****")
> qm1 <- st_queen(b94)

## although coordinates are longitude/latitude, st_relate_pattern assumes that they are planar

> any(sapply(qm1, length) == 0)

## [1] TRUE
```

Problem of no-neighbour zone

Both Bayesian multilevel spatial model fitting approaches fail on the no-neighbour case (the spatially structured random effect cannot be estimated), so we need to drop those census tracts and re-aggregate to model output zones with neighbours:

```
> NOX_ID_no_neighs <- b94$ids[which(sapply(qm1, length) == 0)]
> b487 <- b489[is.na(match(b489$NOX_ID, NOX_ID_no_neighs)),]
> t0 <- aggregate(b487, list(ids = b487$NOX_ID), head, n = 1)
> b93 <- t0[, c("ids", attr(t0, "sf_column"))]
```

Problem of no-neighbour zone

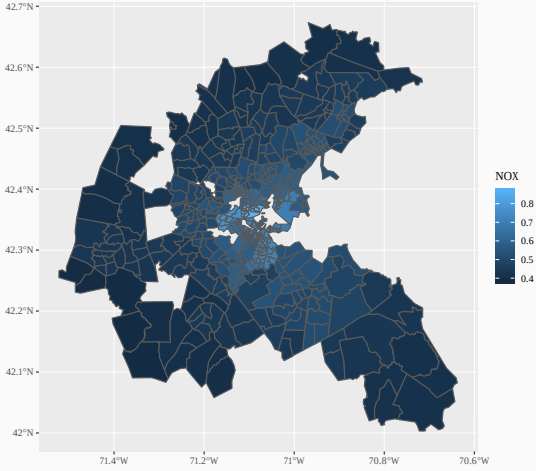
and finally create the same kind of **nb** object as used in **spdep**:

```
> qm_93 <- st_queen(b93)

## although coordinates are longitude/latitude, st_relate_pattern assumes that they are planar

> class(qm_93) <- "nb"
> attr(qm_93, "region.id") <- as.character(b93$ids)
```

NOX values (parts per 10 million)

The `geom_sf` function is in ggplot2:

```
> library(ggplot2)
> ggplot(b487) + geom_sf(aes(fill=NOX))
```



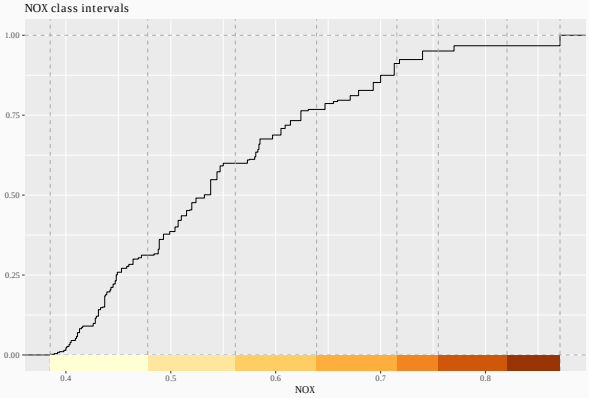
NOX values (parts per 10 million)

Let's try to use **classInt** to help present the NOX values by choosing natural breaks classes using `e1071::bclust`:

```
> library(classInt)
> set.seed(1)
> cI <- classIntervals(b487$NOX, n=7L, style="bclust", verbose=FALSE)
> cI

## style: bclust
##   one of 256,851,595 possible partitions of this variable into 7 classes
##  [0.385,0.478) [0.478,0.5615) [0.5615,0.639)
##            152             140              82
## [0.639,0.7155) [0.7155,0.755) [0.755,0.8205)
##             70              19               8
## [0.8205,0.871]
##             16
```

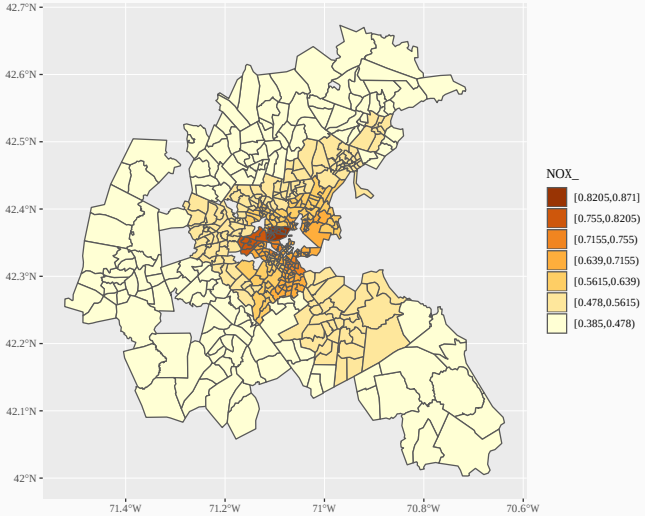
NOX values (parts per 10 million)

We can show the intervals on an empirical cumulative distribution function plot:



NOX values (parts per 10 million)

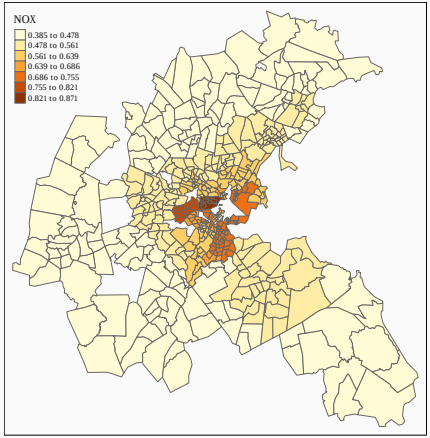
So it is possible to get to a ggplot rendering of an **sf** object:



NOX values (parts per 10 million)

Or use **tmap** which also supports **sf** objects:

```
> library(tmap)
> qtm(b487, fill="NOX", fill.n=7L, fill.style="bclust")
```



Air pollution data

- The figures show clearly that the study of the relationship between NOX and house value will be impacted by “copying out’’ NOX values to census tracts, as noted by Harrison and Rubinfeld (1978, p. 86, footnote 14)
- Even if we were to use more class intervals in these choropleth maps, the visual impression would be the same, because the underlying data have support approximated by the TASSIM zones, not by the census tracts

Other independent variables

- Besides NOX, the other census covariates included in the hedonic regression to account for median house values are the average number of rooms per house, the proportion of houses older than 1940, the proportion low-status inhabitants in each tract, and the Black proportion of population in the tract — originally expressed as a broken-stick relationship, but here taken as a percentage
- The crime rate is said to be taken from FBI data by town, but which is found on inspection to vary by tract
- The distance from tract to employment centres is derived from other sources, as is the dummy variable for tracts bordering Charles River

Other independent variables

- Other covariates are defined by town, with some also being fixed for all towns in Boston
- The town aggregates of census tracts are used in many of the census report tabulations, and of the 92 towns, 17 only contain one census tract, while one town contains thirty census tracts
- The variables are the proportion of residential lots zoned over 25000 sq. ft, the proportion of non-retail business acres, accessibility to radial highways, full-value property-tax rate per USD 10,000, and pupil-teacher ratio by town school district

Towns and TASSIM zones

- In the case of 80 approximate TASSIM zones aggregated from census tracts, the boundaries do coincide exactly with town boundaries
- For the remaining 12 towns and 16 TASSIM zones, there are overlaps between more than one town and TASSIM zone, mostly in Boston itself
- Using TASSIM zones for analysis should therefore also reduce the levels of autocorrelation induced by “copying out” town values to tracts within towns
- The exact match between town boundaries defined using census tracts, and approximated TASSIM zones also constructed using census tracts is not necessarily an indication that towns were used as TASSIM zones

Hedonic modelling of house values

- Pace and Gilley (1997), drawing on earlier work, felt that it should be worthwhile to check whether the original model was not spatially misspecified
- They considered that the use of spatial aggregate units as observations might involve spillovers of some kind, chiefly in the house values used
- Had the included explanatory variables accounted for the similarities between neighbours, there might not have been any reason to go further, but the residuals turn out to be spatially highly patterned
- So now we will turn to spatial econometrics methods to try to unravel the question of the "real" link between house values and NOX

The ZN, INDUS, NOX, RAD, TAX and PTRATIO variables show effectively no variability within the TASSIM zones, so in a multilevel model the random effect may absorb their influence. The model as a whole, before introducing random effects, is:

```
> form <- formula(log(median) ~ CRIM + ZN + INDUS + CHAS + I((NOX*10)^2) + I(RM^2) +
+   AGE + log(DIS) + log(RAD) + TAX + PTRATIO + I(BB/100) + log(I(LSTAT/100)))
```

R's sessionInfo()

```
> sessionInfo()

## R version 3.5.1 (2018-07-02)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Fedora 28 (Workstation Edition)
##
## Matrix products: default
## BLAS: /home/rsb/topics/R/R351-share/lib64/R/lib/libRblas.so
## LAPACK: /home/rsb/topics/R/R351-share/lib64/R/lib/libRlapack.so
##
## locale:
##  [1] LC_CTYPE=en_GB.UTF-8
##  [2] LC_NUMERIC=C
##  [3] LC_TIME=en_GB.UTF-8
##  [4] LC_COLLATE=en_GB.UTF-8
##  [5] LC_MONETARY=en_GB.UTF-8
##  [6] LC_MESSAGES=en_GB.UTF-8
##  [7] LC_PAPER=en_GB.UTF-8
##  [8] LC_NAME=C
##  [9] LC_ADDRESS=C
## [10] LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_GB.UTF-8
## [12] LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats4    grid      stats     graphics
## [5] grDevices utils     datasets  methods
## [9] base
```