Exploring and analysing rates of Hansen's Disease in Olinda, Brazil

Marilia Sa Carvalho Roger Bivand

August 30, 2003

1 Introduction

This set of exercises will show how you can take a shapefile containing the boundaries of urban tracts (sectors), and values of variables, read them into R, clean the data, and proceed to explore and analyse them. The sample analyses made here are not necessarily successful in demonstrating any particular relationships in the data as such: it is up to you to try to vary the approaches made, and to examine the results actively, to see what more you can find out.

2 Reading and cleaning the shapefile

The data examined here were presented in the following article: LAPA, TM; XIMENES, RAA; SILVA, NN; SOUZA, WV; ALBUQUERQUE, MFM; GOUVEIA, GC. Leprosy surveillance in Olinda, Brazil, using spatial analysis techniques. *Cadernos de Saúde Pública*, 17(5): 1153-1162, 2001, and have been made available by kind permission from the authors. The city of Olinda is located on Brazil's coast immediately north of the larger city of Recife, at approximately 35°W, 8°S. In the following, we will be terming Leprosy Hansen's Disease, because one of us is located in the city where Dr Armauer Hansen studied the disease, Bergen in Norway.

In order to read the data into R, we load the **maptools** package into R. This is one of three contributed packages needed here over and above the R base distribution, and should be obtained from CRAN and installed before work begins. The shapefile format presupposes that you have three files with extensions `*.shp`, `*.shx`, and `*.dbf`, where the first contains the geometry data, the second the spatial index, and the third the attribute data. We are distributing a zip archive file with this document called `olinda.zip` in the `olinda` directory below the directory in which this document is placed. Once you have found the zip file, unpack the three shapefile components to R's temporary directory, and read them from there using `read.shapefile()`. By default, this function reads in the data in all three files, although it is only given the name of the file with the geometry.

Since there is no projection information in this minimal set of shapefiles, we need to establish from external sources that the map data are projected, with the following parameters: UTM zone 25 South, measured in metres (for PROJ.4, use: +proj=utm +zone=25 +south); most probably WGS84 datum/ellipse. Knowing the projection metadata of the geometric data is of importance if, for example, further data is to be added at a later data, for example for point addresses of health stations to check

whether more cases are reported near a station, or for lines such as roads or drainage channels.

```
> library(maptools)
> zip.file.extract("setor1.shp", "olinda/olinda.zip")
[1] "/tmp/Rtmp16337/setor1.shp"
> zip.file.extract("setor1.shx", "olinda/olinda.zip")
[1] "/tmp/Rtmp16337/setor1.shx"
> zip.file.extract("setor1.dbf", "olinda/olinda.zip")
[1] "/tmp/Rtmp16337/setor1.dbf"
> setor.shp <- read.shape(paste(tempdir(), "setor1.shp", sep = "/"))
Shapefile Type: Polygon   # of Shapes: 243

DBF field with "_" changed to SETOR.
DBF field with "_" changed to SETOR.ID
DBF field with "_" changed to DENS.DEMO
> class(setor.shp)
[1] "Map"
> names(setor.shp)
[1] "Shapes"   "att.data"
```

The imported object in R has class `Map`, and is a list with two components, `"Shapes"`, which is a list of shapes, and `"att.data"`, which is a data frame with tabular data, one row for each shape in `"Shapes"`. Let us move the attribute data to a separate data frame for convenience:

```
> setor <- setor.shp$att.data
> names(setor)
 [1] "AREA"      "PERIMETER" "SETOR."    "SETOR.ID"  "VAR5"      "DENS.DEMO"
 [7] "SET"       "CASES"     "POP"       "DEPRIV"
> setor.polys <- Map2poly(setor.shp, region.id = setor$SET)
> setor.cents <- get.Pcent(setor.shp)
> cols = c("red", "grey")
> plot(setor.shp, fg = cols[(setor$SET > 0) + 1], xlab = "", ylab = "")
```

We can examine the names of the columns of the data frame to see what it contains. For our purposes, the variables of interest are "CASES" — case counts of the disease, "POP" — the population at risk in this period, and "DEPRIV" — a measure of social deprivation. We will similarly convert the geometry format of the `Map` object to that of a `polylist` object, which will be easier to handle. Finally, we retreive the centroids of the tract polygons to use as label points.

One problem that we need to handle straight away is that we have geometry for 243 shapes, but we are missing data for 2 tracts with no population - they have zero values of variable `SET` in our data frame. In Figure 1, these are marked by being coloured red; the function used is for plotting `Map` objects.

```
> complete <- setor$SET > 0
> subsetor <- subset(setor, complete)
> subsetor.polys <- subset(setor.polys, complete)
> attributes(subsetor.polys) <- attributes(setor.polys)
> attr(subsetor.polys, "region.id") <- attr(subsetor.polys, "region.id")[complete]
> subsetor.cents <- setor.cents[complete, ]
> plotpolys(subsetor.polys)
```

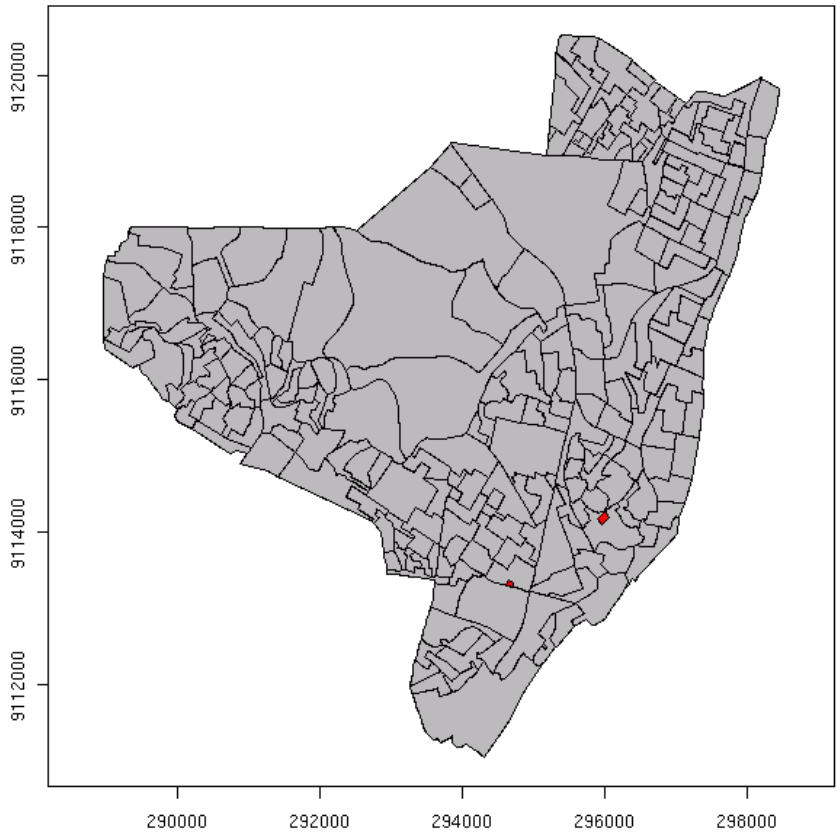

Figure 1: Initial plot of Olinda setor1.shp shapefile data showing two tracts with missing data.

Since the `polylist` class does not yet have a `subset` function, we need to update the attributes of our subset, as well as to generate it using the general function for subsetting lists. Subsetting the data frame and matrix of polygon centroids do not need extra steps. The outline map shown in Figure 2 has no polygons for the omitted tracts, but this cannot be seen, because they are entirely surrounded by other polygons. In colour-filled maps below, they will be left unfilled.

Figure 2: Plot of subset of Olinda polygons.

It may be of interest to look at the structure of a polygon list member. This is made up of a two-column matrix with polygon coordinates. In general, each sub-polygon will have equal first and last coordinates to ensure closure, but this is not absolutely required. Rows in the coordinate matrix set to `NA` represent breaks between sub-polygons, and are respected by the underlying R graphics functions. The attributes contain further information about the polygon: `pstart` is a list with `from` and `to` components, which are vectors of first and last rows in the matrix for each sub-polygon in the object — there are `nParts` elements in both `from` and `to`. `RingDir` and `ringDir` should be the same, and are computed in two different ways to determine whether each of the `nParts` sub-polygons runs clockwise or counter-clockwise. Counter-clockwise sub-polygons are "holes" in the surrounding sub-polygon. Finally, `bbox` contains the bounding box of this object. Its appearance is shown in Figure 3.

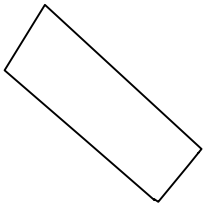

Figure 3: Plot of polygon 6 from the list of polygons.

```
> subsetor.polys[[6]]
          [,1]    [,2]
 [1,] 296693.5 9119921
 [2,] 296674.0 9119899
 [3,] 296612.9 9119832
 [4,] 296604.4 9119837
 [5,] 296605.2 9119835
 [6,] 296465.2 9119945
 [7,] 296303.8 9120071
 [8,] 296384.9 9120190
 [9,] 296699.9 9119928
[10,] 296693.5 9119921
attr(,"pstart")
attr(,"pstart")$from
[1] 1

attr(,"pstart")$to
[1] 10

attr(,"bbox")
[1]  296303.8 9119832.0  296699.9 9120190.0
attr(,"RingDir")
[1] 1
attr(,"nParts")
[1] 1
attr(,"ringDir")
[1] 1
> subsetor.cents[6, ]
[1]  296498.2 9120007.1
> plot(subsetor.polys[[6]], type = "l", axes = FALSE, xlab = "",
+     ylab = "")
```

Now that we have made sure that our data and the polygons that they cover are cleaned, we can draw a choropleth map showing levels of the social deprivation variable DEPRIV — see Figure 4. Something that requires thought and care is the choice of class intervals; mapping numeric variables almost always requires binning into classes, and different numbers of bins and bin boundaries can yield maps telling different "stories".

```
> brks <- fivenum(subsetor$DEPRIV)
> cols <- rev(heat.colors(4))
```

```
> plotpolys(subsetor.polys, col = cols[findInterval2(subsetor$DEPRIV,
+     brks)])
> legend(c(289000, 291000), c(9112000, 9114000), legend = leglabs(brks,
+     "<", ">="), fill = cols, bty = "n", cex = 0.9, y.intersp = 0.9)
```

Here we use the Tukey five-number summary to choose class intervals at the minimum and maximum values observed, at their median and lower and upper hinges; five boundaries cutting the distribution into four parts with (approximately) equal numbers of observations. The choice of colours is also rather arbitrary, but we reverse them to make the "hotter" light colors correspond to less deprivation, the "cooler" red colours to more deprivation.

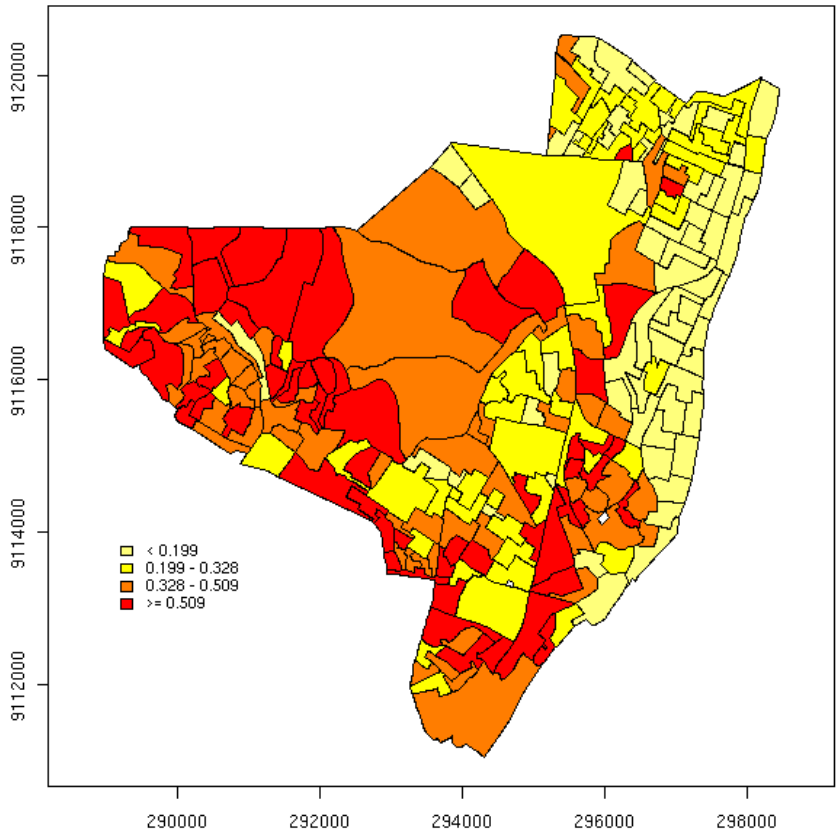

Figure 4: Deprivation index values for Olinda, divided by into four classes of equal numbers of tracts.

We will now turn to our main variable of interest, the counts of cases of Hansen's Disease by urban tract in Olinda. The histogram shown in Figure 5 is typical for counts — there are many tracts with no or few cases, and very few tracts with many cases. Almost 200 of the 241 tracts have ten cases or less each, and almost 150 tracts have five cases or less each.

```
> table(subsetor$CASES)
```

```
 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 18 19 24 25 26 33 37
31 27 21 26 26 14 19  9  9  7  3 13  3  4  3  6  6  6  3  1  1  1  1  1
> histres <- hist(subsetor$CASES, main = "", ylab = "", xlab = "")
> histres$counts
[1] 145  47  29  15   2   1   1   1
> brks <- histres$breaks
> cols <- rev(gray(1:length(histres$counts)/length(histres$breaks)))
> plotpolys(subsetor.polys, col = cols[findInterval2(subsetor$CASES,
+     brks)])
> legend(c(289000, 291000), c(9112000, 9114000), legend = leglabs(brks,
+     "<", ">="), fill = cols, bty = "n", cex = 0.9, y.intersp = 0.9)
```

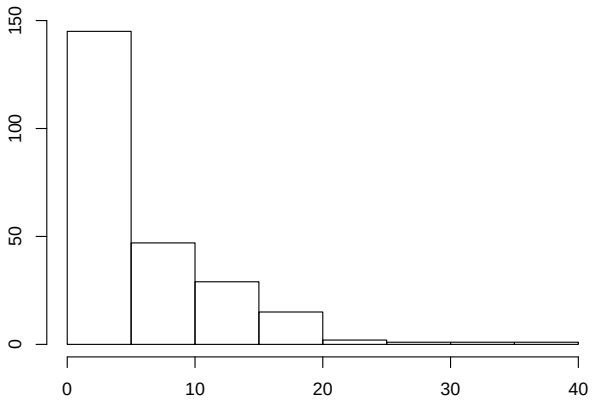

Figure 5: Distribution of the number of cases of Hansen's Disease.

The corresponding map (Figure 6) uses the bins returned by the histogram function; it appears that there are only a very few tracts with high counts of cases, perhaps concentrated in one part of the city.

We could construct a raw rate of Hansen's Disease counts by dividing the counts of cases for each tract by the recorded population (for the same period - if counts are collected over several years, the population figures also need to be adjusted to reflect the total person-years of risk). Note that population figures for small tracts may also vary greatly in time, requiring care in drawing conclusions from apparently high rates (where we have confidence in the case counts) but where the population at risk has actually grown considerably since our baseline population count.

Here we choose to start using the **spdep** package: spdep, version 0.2-2, 2003-08-28, which provides a `probmap()` function providing the raw rate among other measures, to which we will return below. We will for now be using the `raw` rates returned by the function, and map them using rounded quantiles. Since `quantile()` by default returns the minimum, maximum, and 25%, 50%, and 75% percentiles, this may correspond to `fivenum()` above. To avoid unnecessarily long legend items, we convert the raw rates to rates per thousand, and round the result to two digits after the decimal point. The resulting map is shown in Figure 7.

```
> library(spdep)
> pm <- probmap(subsetor$CASES, subsetor$POP)
> names(pm)
```

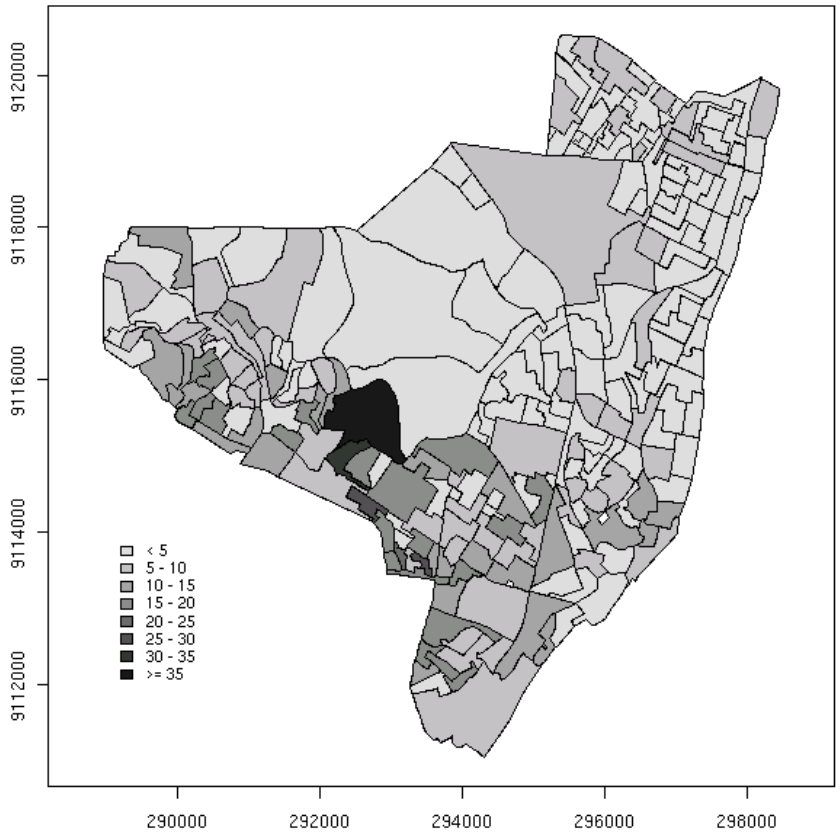

Figure 6: Map of the raw counts of occurrences of Hansen's Disease by tract.

```
[1] "raw"      "expCount" "relRisk"  "pmap"
```

```
> brks <- round(quantile(1000 * pm$raw), digits = 2)
> cols <- rev(gray(2:5/6))
```

```
> plotpolys(subsetor.polys, col = cols[findInterval2(pm$raw * 1000,
+     brks)])
> legend(c(289000, 291000), c(9112000, 9114000), fill = cols, bty = "n",
+     legend = leglabs(brks, "<", ">="), cex = 0.9, y.intersp = 0.9)
```

Unlike Figure 6, in which the numbers of tracts in each bin differ greatly (exactly as in the histogram), Figure 7 has equal numbers of tracts in each bin — this is a choice that has to be made by the analyst; although software may make it “look” easy, finding bins — class intervals — for maps that are suited to the needs both of the analyst and to people to whom the analyst addresses her results is very difficult. If you try alternative breakpoints, make sure that you have a vector of colour fills that is at least one less in length than the length of the breakpoint vector. Also try to remember that we don’t want to use "white", which is the default background colour used for our dropped empty tracts.

Figure 7: Map of raw rates of occurrence per 1000 of Hansen’s Disease by tract.

3 Spatial dependence and neighbour relations

One of the features of data collected for spatially located objects or phenomena is that observations made close to each other are typically more similar to each other than observations made at greater distances from each other. This nearness is often related to spatial dependence in the observed data, because it is unusual in many domains of knowledge for very steep gradients to be present between neighbouring points of observation — for instance, air temperature is often similar between points near one another on a flat surface (change in elevation can have a much greater impact than change across a surface). In order to see whether our data may display spatial dependence, we can choose between a number of different approaches. One of these is to assume that we can represent the structure of the dependency by some graph showing which of our tracts are "neighbours" of which other tracts. This graph does not have to be symmetric, but the first example is symmetric.

```
> viz <- poly2nb(subsetor.polys, row.names = subsetor$SET)
> viz
```

```
Neighbour list object:
Number of regions: 241
Number of nonzero links: 1324
Percentage nonzero weights: 2.279575
Average number of links: 5.493776
```

```
> attr(viz, "region.id")[viz[[match("222", attr(viz, "region.id"))]]]
```

```
[1] "223" "225"
```

```
> plotpolys(subsetor.polys, border = "grey")
> plot(viz, subsetor.cents, add = TRUE, col = "blue")
```

The `poly2nb()` function checks the boundaries of each of the polygons against those of those nearby to see if they share boundary points. If they do, they touch each other, and are contiguous neighbours. If only a single point is shared between two tracts, they are "queen"-style neighbours, if two or more consecutive points, they are "rook"-style neighbours (using the chess analogy). Because quite a lot of checking is involved, this function is slow, but in general is only used once for each list of polygons. Recall that we have dropped two empty polygons — they will not be included in the list of neighbours returned by the function.

Printing the new object shows that it is a neighbour list object, with a very sparse structure — if displayed as a matrix, only 2.3% of cells would be filled. Objects of class `nb` contain a list as long as the number of tracts; each component of the list is a vector with the index numbers of the neighbours of the tract in question, so that the neighbours of the tract with `region.id` of "222" can be retreived by matching against the indices. More information can be obtained by using `summary()` on an `nb` object. Finally, we associate a vector of names with the neighbour list, through the `row.names` argument. The names should be unique, as with data frame row names.

```
> olinda.nb2 <- knn2nb(knearneigh(subsetor.cents, 2), row.names = subsetor$SET)
> olinda.nb2
```

```
Neighbour list object:
Number of regions: 241
Number of nonzero links: 482
Percentage nonzero weights: 0.8298755
Average number of links: 2
Non-symmetric neighbours list
```

```
> n.comps <- n.comp.nb(olinda.nb2)
> cols <- rainbow(n.comps$nc)
```

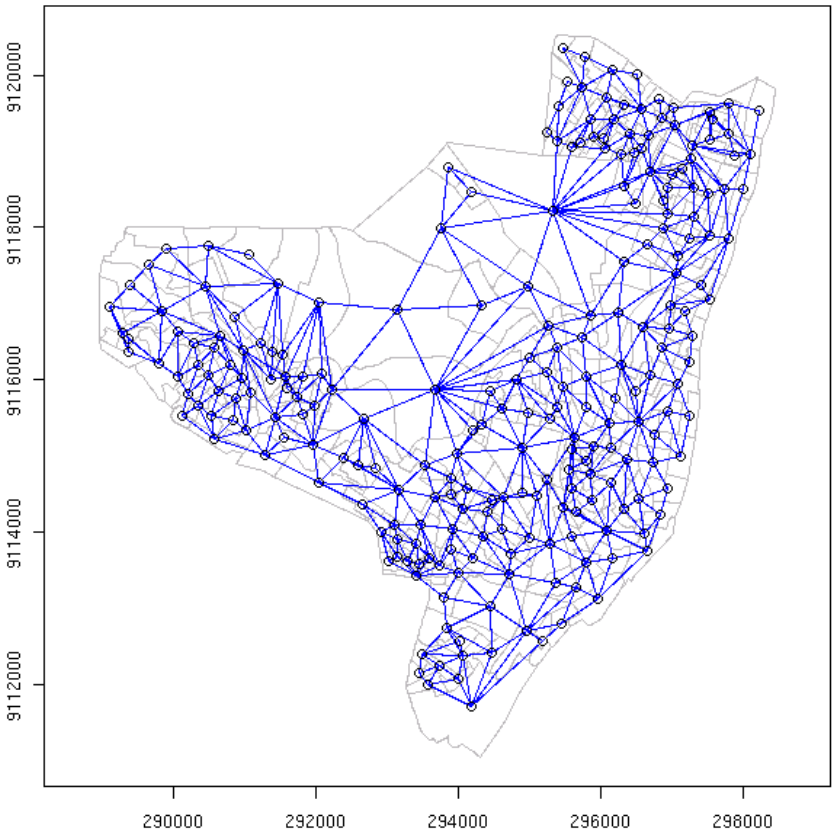

Figure 8: Contiguity-based neighbour relations for tracts in Olinda.

```
> plotpolys(subsetor.polys, border = "grey")
> plot(olinda.nb2, subsetor.cents, add = TRUE, col = cols[n.comps$comp.id])
```

The k-nearest neighbour definition of neighbourhood is quite often met, because it makes sure that all tracts have the same number of neighbours. It is sensitive to the choice of point used to represent the unit of observation, and polygon centroids may not be the best choice, especially if the distribution of the phenomena of interest within the polygons is not uniform. For smaller k, all the tracts will have k neighbours, but there may be more than one components in the graph of relationships. This can be checked with the `n.comp.nb()` function.

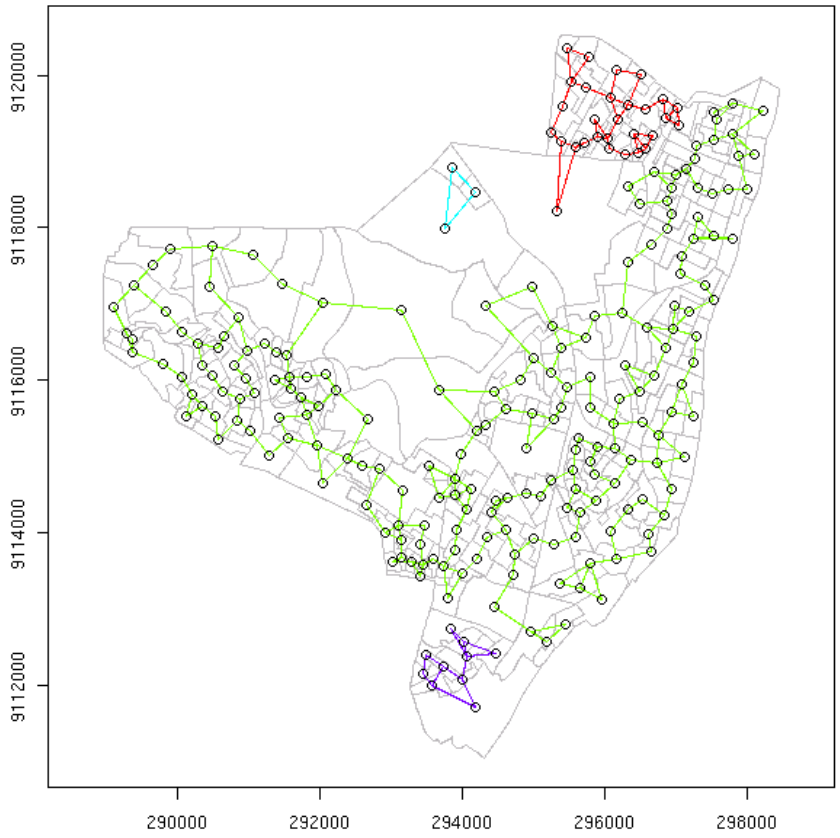

Figure 9: Two-nearest neighbours coloured by graph components.

```
> olindatri.nb <- tri2nb(subsetor.cents, row.names = subsetor$SET)
> olindasoi.nb <- graph2nb(soi.graph(olindatri.nb, subsetor.cents),
+     sym = TRUE, row.names = subsetor$SET)
> olindasoi.nb

Neighbour list object:
Number of regions: 241
Number of nonzero links: 1112
Percentage nonzero weights: 1.914568
Average number of links: 4.614108

> n.comp.nb(olindasoi.nb)$nc
```

```
[1] 1
```

Graph neighbour relations are also provided, for a full triangulation using `tri2nb()`, which has the disadvantage of including "convex hull" neighbours, that is points that can "see" each other along the edges of the study area, but which may be distant from one another. The sphere-of-influence graph removes many such spurious links, but may be too severe, dividing the graph into several components.

All of these methods should make sure that each tract has at least one defined neighbour, unless using `poly2nb()` there are islands with no shared boundaries with other polygons. However, it seems all too easy to generate neighbour lists with no-neighbour tracts using `dnearneigh()`, even though distance between points representing objects might be taken as the most natural way of expressing their mutual positions.

4 Spatial autocorrelation

Before we get to estimating Moran's I for the raw disease rate, let us stop to consider how to get from the representation of neighbour relations that we have constructed as neighbour list objects to the spatial weights needed to estimate the statistic. The function used to do this is `nb2listw()`, which has to take at least one argument, a neighbour list object, and returns a `listw` object.

```
> args(nb2listw)

function (neighbours, glist = NULL, style = "W", zero.policy = FALSE)
NULL

> lw.viz <- nb2listw(viz)
> lw.viz

Characteristics of weights list object:
Neighbour list object:
Number of regions: 241
Number of nonzero links: 1324
Percentage nonzero weights: 2.279575
Average number of links: 5.493776

Weights style: W
Weights constants summary:
    n  n1  n2  n3    nn  S0       S1       S2
W 241 240 239 238 58081 241 93.57845 1007.857
```

Objects of class `listw` have a `neighbours` component, which is the same as the neighbour list argument, and a `weights` component, which follows the same pattern as the `neighbours` component, but contains the actual values of the non-zero weights. An argument that matters sometimes both here and in functions using `listw` objects is the `zero.policy` argument. When it is `FALSE` (default), the function exits with an error message if no-neighbour tracts are found. If it is `TRUE`, zero "rows" are permitted in the `weights` component, meaning that zero will be assigned for that row to the product of any vector and the `listw` object. Further, different styles are defined for `listw` objects, where `"W"` is the default row-standardised form (with rows of weights summing to unity), `"B"` is used for binary 0/1 weights, and `"S"` is for the variance-stabilizing coding scheme proposed by Tiefelsdorf et al. (1999) — for further details on these schemes see the help page for `nb2listw()`. Finally, there is a `glist` argument for passing general weights values into the function; we will return to this later.

```
> moran.test(pm$raw, nb2listw(viz, style = "W"))
```

```
        Moran's I test under randomisation

data:  pm$raw
weights: nb2listw(viz, style = "W")

Moran I statistic standard deviate = 10.0853, p-value = < 2.2e-16
alternative hypothesis: greater
sample estimates:
Moran I statistic       Expectation          Variance
      0.393147809      -0.004166667       0.001551989
```

```
> moran.test(pm$raw, nb2listw(viz, style = "B"))
```

```
        Moran's I test under randomisation

data:  pm$raw
weights: nb2listw(viz, style = "B")

Moran I statistic standard deviate = 10.0156, p-value = < 2.2e-16
alternative hypothesis: greater
sample estimates:
Moran I statistic       Expectation          Variance
      0.376768961      -0.004166667       0.001446604
```

```
> moran.test(pm$raw, nb2listw(viz, style = "S"))
```

```
        Moran's I test under randomisation

data:  pm$raw
weights: nb2listw(viz, style = "S")

Moran I statistic standard deviate = 10.1664, p-value = < 2.2e-16
alternative hypothesis: greater
sample estimates:
Moran I statistic       Expectation          Variance
      0.386051749      -0.004166667       0.001473261
```

The results of testing the estimates of Moran's *I* statistic under randomisation with different weighting styles (among which there are few differences) show that spatial dependence is present in the data. It is very unlikely that the patterns we can see could have arisen at random. But we do not know why neighbouring tracts seem to be similar to each other in raw rates.

When contagion is a direct cause, it is reasonable to interpret the spatial dependence as resulting from proximity. But it is also very possible that the boundaries of the tracts are arbitrarily imposed on more complex social patterns, so that a district of the city which otherwise appears as a well-connected and functioning whole, is represented by multiple neighbouring tracts in the data - hence leading us to think that the data are spatially autocorrelated when they are perhaps not much more than arbitrarily aggregated. Only knowledge of the study area can help here. When, however, the tracts are for instance health station districts, and the variables of interest can be related to the actions of the "subject" of the district — the staff of the health station — such as a study of the impact of preventive measures, autocorrelation may indicate flows of information (once background variables have been taken into account).

```
> dists <- nbdists(viz, subsetor.cents)
> invdists <- lapply(dists, function(x) 1/x)
> moran.test(pm$raw, nb2listw(viz, glist = invdists, style = "W"))
```

```
        Moran's I test under randomisation

data:  pm$raw
weights: nb2listw(viz, glist = invdists, style = "W")
```

```
Moran I statistic standard deviate = 9.6142, p-value = < 2.2e-16
alternative hypothesis: greater
sample estimates:
Moran I statistic       Expectation          Variance
      0.402103625      -0.004166667       0.001785673
```

Another variation is to use general weights, in this case inverse distance weights. Recall that the "W" style will row-standardise the weights anyway, but this approach will give nearer neighbours a larger share of the weights in each case. The conclusion remains unchanged.

```
> moran.test(pm$raw, nb2listw(olinda.nb2, style = "W"))

        Moran's I test under randomisation

data:  pm$raw
weights: nb2listw(olinda.nb2, style = "W")

Moran I statistic standard deviate = 7.1662, p-value = 3.855e-13
alternative hypothesis: greater
sample estimates:
Moran I statistic       Expectation          Variance
      0.403312954      -0.004166667       0.003233183

> moran.test(pm$raw, nb2listw(olindasoi.nb, style = "W"))

        Moran's I test under randomisation

data:  pm$raw
weights: nb2listw(olindasoi.nb, style = "W")

Moran I statistic standard deviate = 9.2476, p-value = < 2.2e-16
alternative hypothesis: greater
sample estimates:
Moran I statistic       Expectation          Variance
      0.396213892      -0.004166667       0.001874497
```

It is still unchanged after trying two more neighbour lists. The raw rates are obviously very similar across tract boundaries. We can go on to see what happens when we step out across the neighbours list, to look at second — neighbours of neighbours — and higher order relationships in a correlogram.

```
> cg.I <- sp.correlogram(viz, pm$raw, order = 8, method = "I")
> print(cg.I)

Spatial correlogram for pm$raw
method: Moran's I
     estimate  expectation     variance
1  0.39314781 -0.004166667 0.0015519894
2  0.31917859 -0.004166667 0.0006809378
3  0.26431254 -0.004166667 0.0004267140
4  0.20545206 -0.004166667 0.0003200714
5  0.11205115 -0.004166667 0.0002761596
6  0.03672393 -0.004166667 0.0002555164
7 -0.05056479 -0.004166667 0.0002483865
8 -0.10716380 -0.004166667 0.0002996402

> plot(cg.I, main = "")
```

The tabular results of still substantial but declining dependence are plotted in Figure 10. Unlike time series, the numbers of higher-order neighbours do increase until the study area is exhausted, which will necessarily bring higher-order estimates down — there is not much reason to see the highest order negative autocorrelation as more than

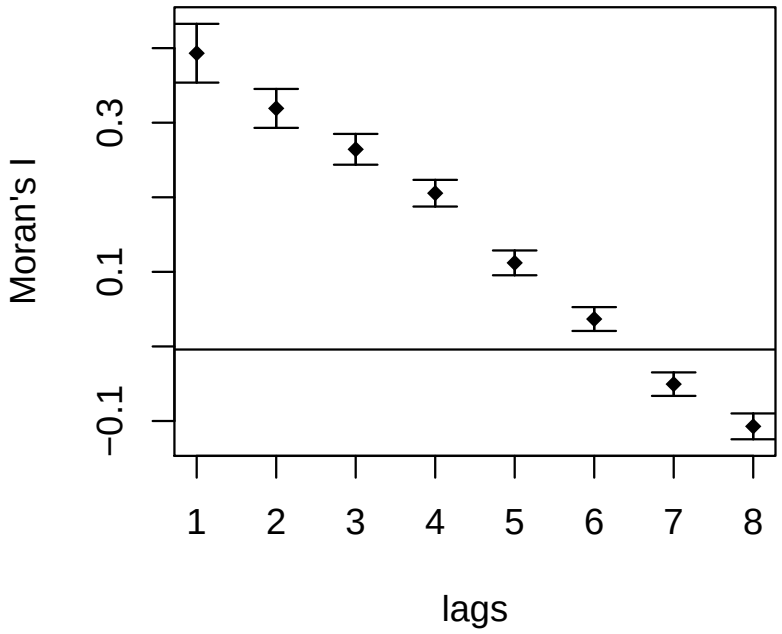

Figure 10: Correlogram of Moran's *I* values for raw rates up to 8 contiguity lags.

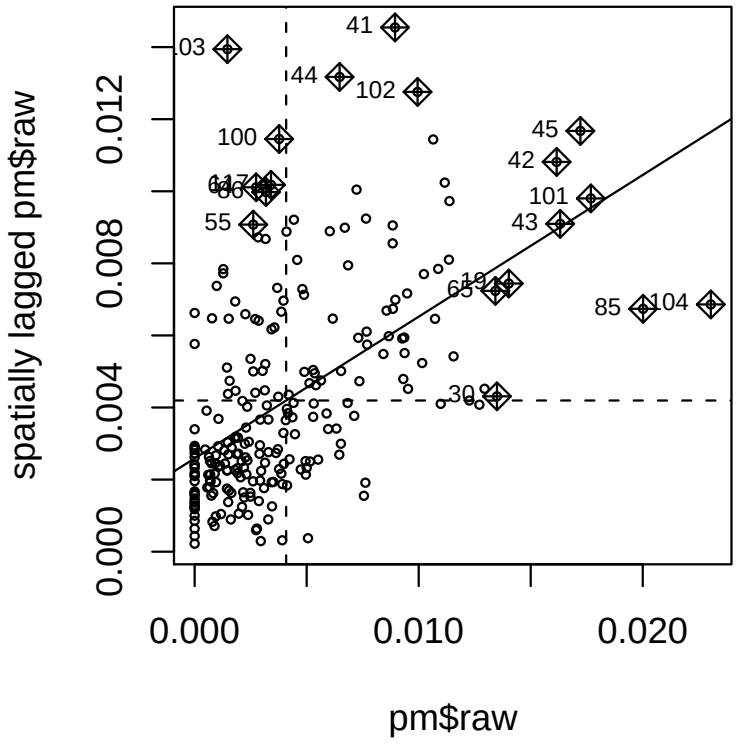

Figure 11: Moran scatterplot of raw rate values.

a reflection of most higher raw rates being in the south and west of the city, and most lower rates being along the coast in the east and in the north.

It is time to look a little more closely at the way in which the global Moran's I test works. It is actually a measure of correlation between the variable of interest, here the raw disease rate, and the "spatially lagged" values. In the case of row-standardised spatial weights, the "spatially lagged" values are the weighted average of the values of the tracts surrounding each tract, and can be plotted as a scatterplot, shown in Figure 11. In addition, Moran's I is a linear measure, with all the benefits and drawbacks this involves.

```
> infl <- moran.plot(pm$raw, nb2listw(viz, style = "W"), cex = 0.5,
+     quiet = TRUE)
```

```
> infl.region <- as.integer(apply(infl$is.inf, 1, any)) + 1
> summary(card(viz))
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  2.000   4.000   5.000   5.494   6.000  16.000
```

```
> by(card(viz), infl.region, summary)
```

```
INDICES: 1
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  2.000   4.000   5.000   5.511   6.500  16.000
------------------------------------------------------------
INDICES: 2
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  3.000   4.000   5.000   5.278   6.000   9.000
```

```
> cols <- c("azure", "powderblue")
```

```
> plotpolys(subsetor.polys, border = "grey", col = cols[infl.region])
> text(subsetor.cents[infl.region == 2, ], labels = subsetor$SET[infl.region ==
+     2], cex = 0.8)
```

One of the drawbacks is that some observations may influence the outcome very much more than others. Fitting a linear model to the scatterplot (by convention displayed in this way), we can see from Figure 11 that some points have been distinguished. The `moran.plot()` function calls the `influence.measures()` function on the `lm` model object represented on the plot by the fitted line. Several measures of influence are used, and points are marked if any of the thresholds set by default in `influence.measures()` are exceeded. By storing the output of `moran.plot()`, we can determine which tracts are exerting more than proportional influence on Moran's I, and map them as shown in Figure 12. In addition, we can check whether there is any relationship between the numbers of neighbours that tracts possess (the `card()` function returns the cardinality of the neighbour set), and whether they exert more than proportional influence — typically, tracts on the edge of the study area will have fewer neighbours, because a part of their boundaries are study area boundaries. If there are large differences in the distributions of numbers of neighbours between the two influence classes, that would suggest that geometry is affecting the result.

```
> args(localmoran)
```

```
function (x, listw, zero.policy = FALSE, spChk = NULL)
NULL
```

```
> resI <- localmoran(pm$raw, nb2listw(viz, style = "W"))
> summary(resI$Z.Ii)
```

```
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-2.81000  0.01287  0.45620  0.91860  1.05800 14.62000
```

```
> round(quantile(resI$Z.Ii, prob = seq(0, 1, 1/10)), digits = 3)
```

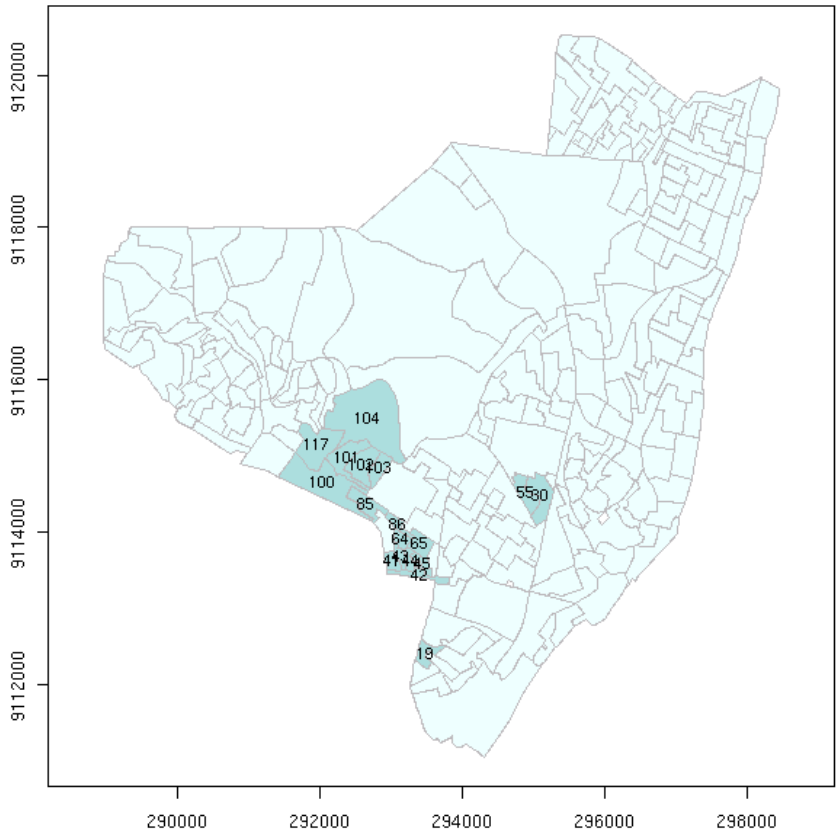

Figure 12: Tracts with significant influence in Moran scatterplot.

```
    0%    10%    20%    30%    40%    50%    60%    70%    80%    90%   100%
-2.810 -0.315 -0.043  0.080  0.264  0.456  0.715  0.942  1.360  2.011 14.624
```

```
> by(card(viz), abs(resI$Z.Ii) > 2, summary)
```

```
INDICES: FALSE
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  2.000   4.000   5.000   5.519   6.750  16.000
------------------------------------------------------------
INDICES: TRUE
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  3.000   4.000   5.000   5.296   6.000  10.000
```

There are two functions in **spdep** for calculating local Moran's I as local indicators of spatial association. The first — `localmoran()` is similar in its form to the global measure, and provides estimates of the standard deviate of the measure under randomisation. No probability values are provided, but they can be obtained using the usual `pnorm()` function. It should however be remembered that here we are making multiple tests using the same data — we are calculating a test statistic for each tract, using the values from neighbouring tracts multiple times. It is possible to use the `p.adjust()` function to alter the probability levels if desired.

```
> args(localmoran.sad)
```

```
function (model, select, nb, glist = NULL, style = "W", zero.policy = FALSE,
    alternative = "greater", spChk = NULL, save.Vi = FALSE)
NULL
```

```
> resI <- localmoran.sad(lm(pm$raw ~ 1), 1:length(viz), viz, style = "W")
> reslI <- as.data.frame(resI, row.names = subsetor$SET)
> summary(reslI$Saddlepoint)
```

```
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-2.21600  0.01286  0.68520  0.73800  1.25300  5.49300
```

```
> round(quantile(reslI$Saddlepoint, prob = seq(0, 1, 1/10)), digits = 3)
```

```
    0%    10%    20%    30%    40%    50%    60%    70%    80%    90%   100%
-2.216 -0.510 -0.085  0.126  0.423  0.685  0.963  1.163  1.466  1.846  5.493
```

```
> by(card(viz), abs(reslI$Saddlepoint) > 2, summary)
```

```
INDICES: FALSE
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  2.000   4.000   5.000   5.511   6.500  16.000
------------------------------------------------------------
INDICES: TRUE
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  3.000   4.000   5.000   5.318   6.000  10.000
```

One difficulty in local indicators is that each test is being made on a neighbourhood with very few observations, in which case the distribution of local Moran's I may diverge a good deal from the Normal. One possible solution to this is to use a Saddlepoint estimate of the standard deviate of the statistic, as implemented in `localmoran.sad()`. This function takes an `lm` object as its first argument — typically a model with just the intercept term —, a vector of spatial unit indices for which to calculate the statistic — here all of them, a neighbour list object, and the weighting style to use. Internally, it generates a symmetric"star" weights matrix for each selected spatial unit, and so needs access just to the neighbours list.

```
> brks <- rev(round(qnorm(c(1, 0.999, 0.99, 0.975, 0.025, 0.01,
+     0.001, 0)), 3))
> cols <- cm.colors(7)
```

```
> plotpolys(subsetor.polys, col = cols[findInterval2(reslI$Saddlepoint,
+     brks)])
> legend(c(289000, 291000), c(9112000, 9114000), fill = cols, bty = "n",
+     legend = leglabs(brks, "<", ">="), cex = 0.9, y.intersp = 0.9)
```

The use of the unadjusted Normal quantiles in the class intervals and legend of Figure 13 is informal, and simply provided an indication of which neighbour relations seem to be driving the localised dependency pattern. Untangling the global pattern — assumed to apply across the whole study area, and used both in global tests and in modelling below — from local spatial association is difficult, and may require good field knowledge of background variables as well.

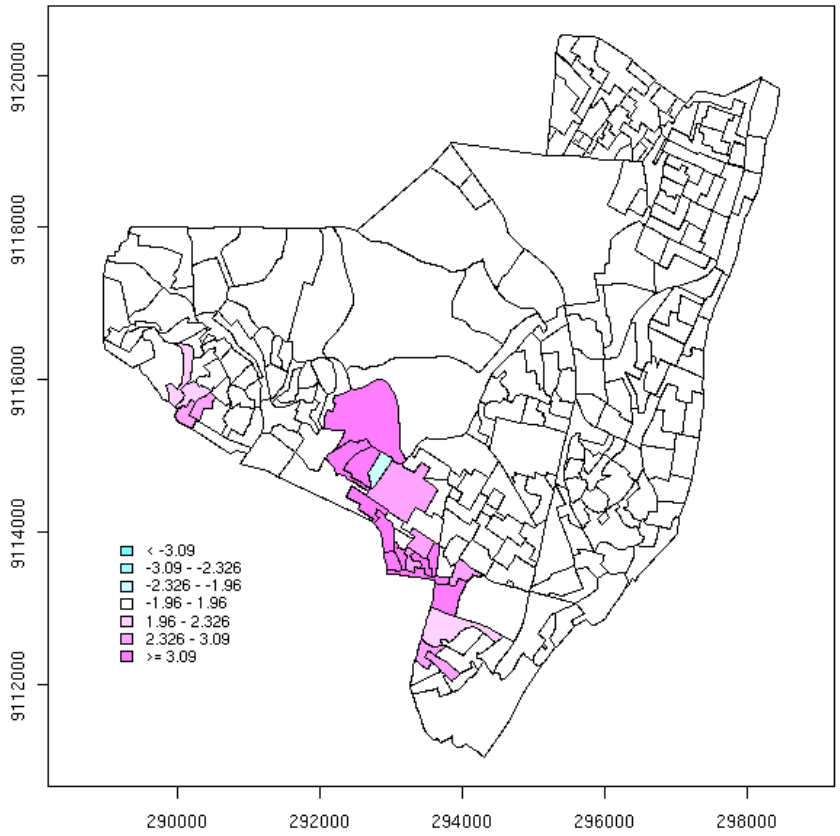

Figure 13: Local Moran's *I* Saddlepoint estimates

5 Assumptions about rates variables

So far, we have been modelling a raw rate variable, constructed from two separate variables, the incidence counts and the numbers of population at risk. It is however clear that rates may be unusually extreme in tracts with low populations at risk.

```
> names(pm)
```

```
[1] "raw"      "expCount" "relRisk"  "pmap"
```

```
> brks <- rev(c(1, 0.999, 0.99, 0.975, 0.025, 0.01, 0.001, 0))
> cols <- cm.colors(7)
```

```
> plotpolys(subsetor.polys, col = cols[findInterval2(pm$pmap, brks)])
> legend(c(289000, 291000), c(9112000, 9114000), fill = cols, bty = "n",
+     legend = leglabs(brks, "<", ">="), cex = 0.9, y.intersp = 0.9)
```

One approach, called probability mapping, is to use the `ppois()` function to compare the observed counts with the expected count expressed as the population in the tract multiplied by the raw rate for the study area as a whole. This measure is implemented in `probmap`, which also returns the raw rate for each tract, the expected count, the relative risk, and the probability map values. The latter are reported in Figure 14. These differ from the first definitions of probability maps, because the sign is reflected in the tail of the legend — in the original representation, the tails were folded together, which was perhaps more meaningful in numerical terms in relation to the probabilities, but more difficult to visualise.

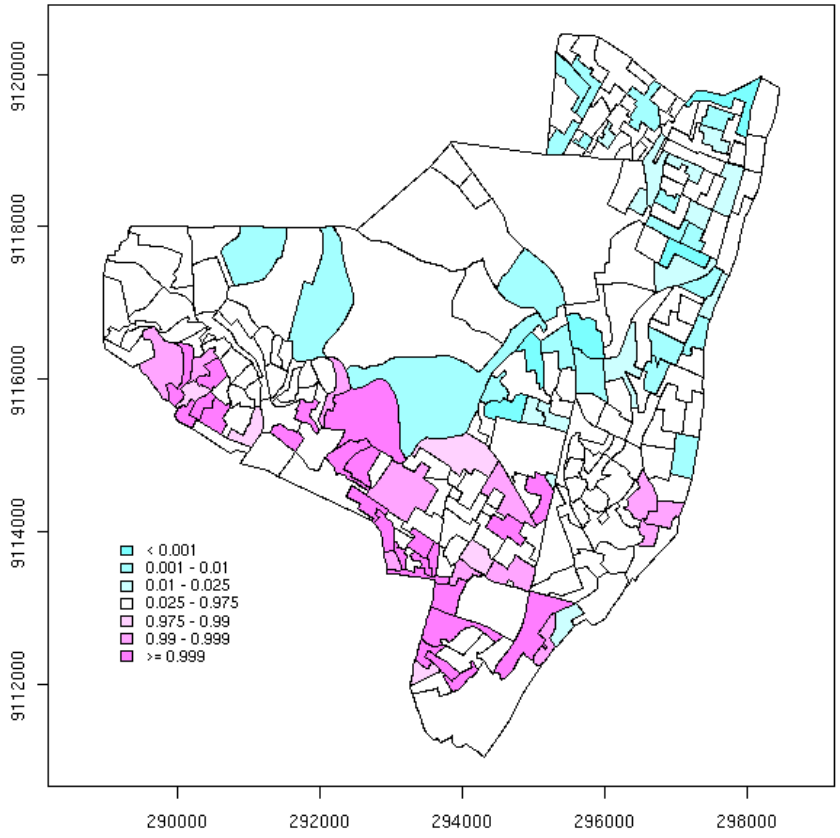

Figure 14: Probability map of Hansen's Disease rates in Olinda.

```
> args(EBlocal)
function (ri, ni, nb, zero.policy = FALSE, spChk = NULL, geoda = FALSE)
NULL
```

```
> rate.bayes <- EBlocal(subsetor$CASES, subsetor$POP, viz)
> brks <- round(quantile(1000 * pm$raw), digits = 2)
> cols <- rev(gray(2:5/6))
> plotpolys(subsetor.polys, col = cols[findInterval2(rate.bayes$est *
+     1000, brks)])
> legend(c(289000, 291000), c(9112000, 9114000), fill = cols, bty = "n",
+     legend = leglabs(brks, "<", ">="), cex = 0.9, y.intersp = 0.9)
```

It is also possible to use local and global Empirical Bayes estimates of the rates, "shrinking" the estimated rate towards the local or global values when the population at risk is small (and so our estimate of the actual rate is much less certain than in tracts with larger populations). Local Empirical Bayes rates estimates are calculated using `EBlocal()`, which, like `localmoran.sad()` above, takes a neighbour list, not a weights list argument. Weighting styles are not available for this function, which depends on being able to count sums of cases and populations at risk for local sets of neighbours. The results of estimating local Empirical Bayes rates are shown in Figure 15, which can be compared with the map of the raw rates (Figure 7)

Figure 15: Local Empirical Bayes estimates of rates.

Finally, we can calculate a version of Moran's *I* for rates, based on Empirical Bayes correction of the rate; the result is assessed by permuting the Empirical Bayes estimates

about the map to see how likely it is that the observed pattern could have arisen at random.

```
> EBImoran.mc(subsetor$CASES, subsetor$POP, nb2listw(viz, style = "B"),
+     nsim = 999)

        Monte-Carlo simulation of Empirical Bayes Index

data:  cases: subsetor$CASES, risk population: subsetor$POP
weights: nb2listw(viz, style = "B")
number of simulations + 1: 1000

statistic = 0.3781, observed rank = 1000, p-value = 0.001
alternative hypothesis: greater
```

6 Modelling

It seems possible that there is a relationship between the levels of social deprivation aggregated to the tract level, and the rates of disease, again aggregated to the tract level. A suitable tool for approaching this modelling situation is to use a Generalised Linear Model, in this case with a quasi-Poisson family argument value (it seems that overdispersion is present, so the Poisson family is not as appropriate).

```
> CASES.glm <- glm(CASES ~ DEPRIV + offset(log(POP)), family = "quasipoisson",
+     data = subsetor)
> anova(CASES.glm, test = "F")

Analysis of Deviance Table

Model: quasipoisson, link: log

Response: CASES

Terms added sequentially (first to last)


        Df Deviance Resid. Df Resid. Dev      F    Pr(>F)
NULL                      240    1212.48
DEPRIV   1   115.87       239    1096.61 23.069 2.759e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
> infl <- influence.measures(CASES.glm)
> infl.region <- as.integer(apply(infl$is.inf, 1, any)) + 1
> cols <- c("azure", "powderblue")
```

```
> plotpolys(subsetor.polys, border = "grey", col = cols[infl.region])
> text(subsetor.cents[infl.region == 2, ], labels = subsetor$SET[infl.region ==
+     2], cex = 0.8)
```

While it seems that the deprivation variable has reduced residual deviation somewhat, we can still note from Figure 16 that some tracts are exerting more influence on the model fit than others, and that at least some of these were noticably present in earlier figures.

```
> pres <- influence(CASES.glm)$pear.res
> brks <- rev(round(qt(c(1, 0.999, 0.99, 0.975, 0.025, 0.01, 0.001,
+     0), df = df.residual(CASES.glm)), 3))
> cols <- cm.colors(7)
```

```
> plotpolys(subsetor.polys, col = cols[findInterval2(pres, brks)])
> legend(c(289000, 291000), c(9112000, 9114000), fill = cols, bty = "n",
+     legend = leglabs(brks, "<", ">="), cex = 0.9, y.intersp = 0.9)
```

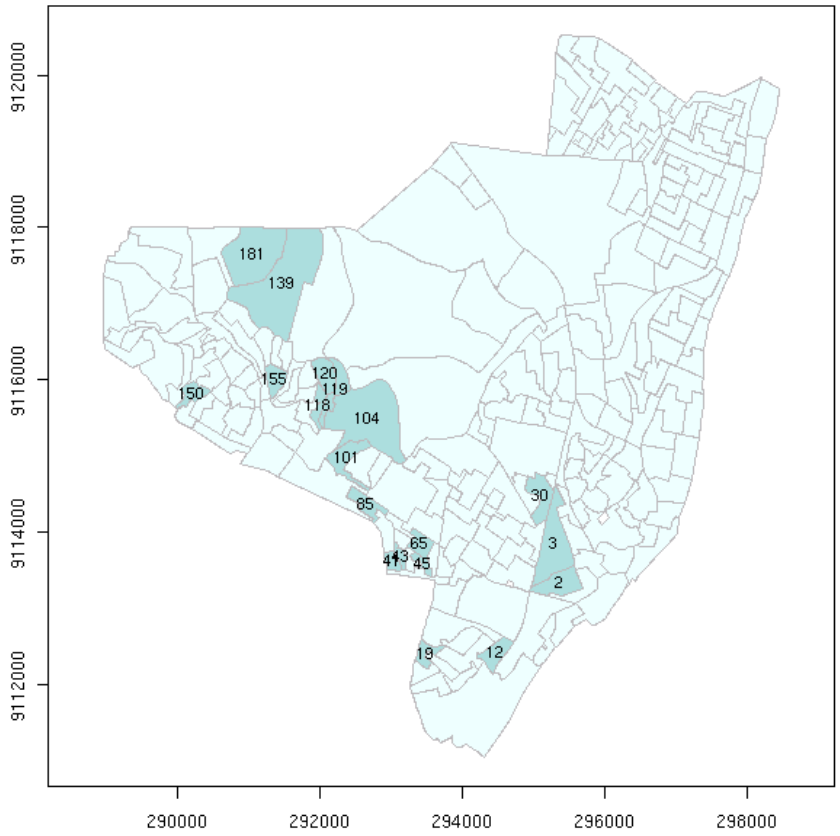

Figure 16: Tracts with significant influence in Quasi-Poisson model fit.

When we plot the Pearson residuals of the quasi-Poisson fit (Figure 17), we can also see that essentially the same spatial pattern is present here as was in the raw rates and the local Empirical Bayes estimates. If social deprivation had had a strong impact, we would not expect to see the same pattern repeat itself. As yet, there are no tests available for spatial dependence in the residuals of GLM model fits, so we proceed by transforming the dependent variable to a log rate, and by estimating a linear model.

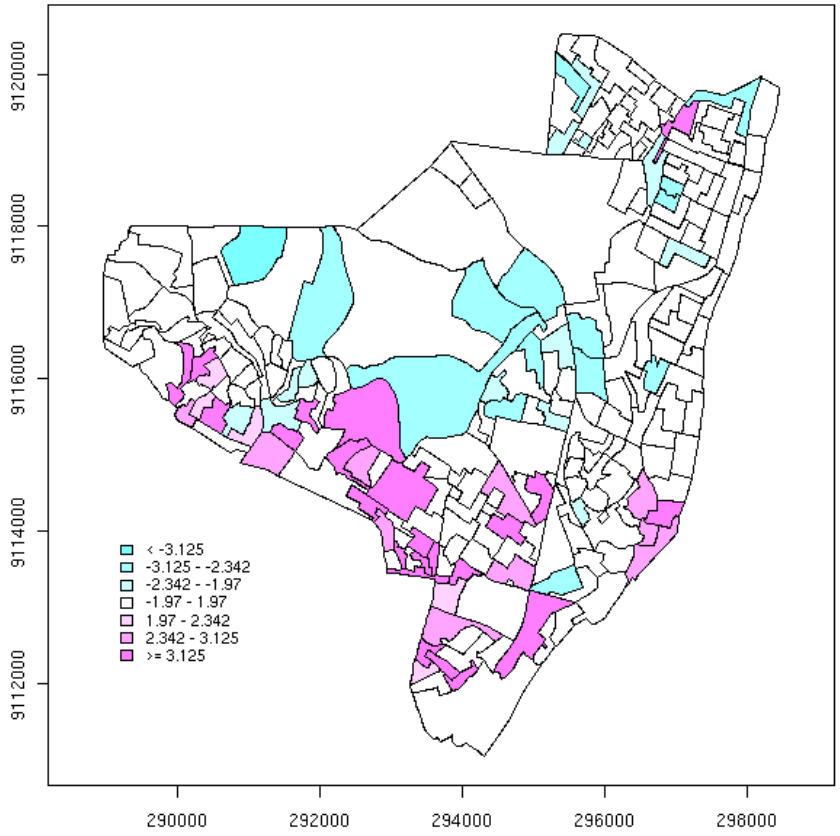

Figure 17: Map of Pearson residuals from the Quasi-Poisson model fit.

```
> logtx <- log((subsetor$CASES + 1)/subsetor$POP)
> logtx.lm <- lm(logtx ~ DEPRIV, data = subsetor)
> summary(logtx.lm)

Call:
lm(formula = logtx ~ DEPRIV, data = subsetor)

Residuals:
     Min       1Q   Median       3Q      Max
-2.22611 -0.53020  0.06585  0.56223  1.72346

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -6.1487     0.1027 -59.844  < 2e-16 ***
DEPRIV        1.3771     0.2482   5.547 7.67e-08 ***
```

```
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.7977 on 239 degrees of freedom
Multiple R-Squared: 0.1141,        Adjusted R-squared: 0.1104
F-statistic: 30.77 on 1 and 239 DF,  p-value: 7.671e-08
```

```
> anova(logtx.lm)
```

```
Analysis of Variance Table

Response: logtx
           Df  Sum Sq Mean Sq F value    Pr(>F)
DEPRIV      1  19.579  19.579  30.773 7.671e-08 ***
Residuals 239 152.063   0.636
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
> pres <- residuals(logtx.lm)
> brks <- round(quantile(pres, prob = seq(0, 1, 1/6)), digits = 2)
> cols <- rev(gray(2:7/8))
```

```
> plotpolys(subsetor.polys, col = cols[findInterval2(pres, brks)])
> legend(c(289000, 291000), c(9112000, 9114000), fill = cols, bty = "n",
+     legend = leglabs(brks, "<", ">="), cex = 0.9, y.intersp = 0.9)
```

Once again, it appears that deprivation is important in the estimated model, but the map of model residuals shown in Figure 18 again appears very familiar — the residuals seem to have a clear spatial pattern.

When we apply a version of Moran's *I* for linear model residuals, we find that they are clearly spatially autocorrelated, and that the estimates of the coefficient standard errors are therefore too small. But Moran's *I* is a very general test for mis-specification, and also detects non-stationarity.

```
> lm.morantest(logtx.lm, nb2listw(viz, style = "W"))
```

```
        Global Moran’s I for regression residuals

data:
model: lm(formula = logtx ~ DEPRIV, data = subsetor)
weights: nb2listw(viz, style = "W")

Moran I statistic standard deviate = 6.7822, p-value = 5.918e-12
alternative hypothesis: greater
sample estimates:
Observed Moran’s I        Expectation           Variance
       0.261308746       -0.006260090        0.001556432
```

So we also use two robust tests that accommodate alternative nuisance parameters for the spatial error and spatial lag models. The robust error test tests for the possible presence of spatial autocorrelation in the error term in the presence of a spatial lag term; the robust spatial lag test does the reverse.

```
> lm.LMtests(logtx.lm, nb2listw(viz, style = "W"), test = c("RLMerr",
+     "RLMlag"))
```

```
        Lagrange multiplier diagnostics for spatial dependence

data:
model: lm(formula = logtx ~ DEPRIV, data = subsetor)
weights: nb2listw(viz, style = "W")

RLMerr = 4.5921, df = 1, p-value = 0.03212
```

Figure 18: Map of residuals from the linear model fit of transformed dependent variable.

```
        Lagrange multiplier diagnostics for spatial dependence

data:
model: lm(formula = logtx ~ DEPRIV, data = subsetor)
weights: nb2listw(viz, style = "W")

RLMlag = 17.7724, df = 1, p-value = 2.490e-05
```

As we can see, the robust spatial lag test statistic (distributed as χ^2 with 1 degree of freedom) has a much higher value than the robust spatial error test, suggesting that the spatially lagged value of the dependent variable should be included in the model. We estimate three versions of the spatial lag model (using simultaneous autoregression — SAR), one with just the intercept term, one including the social deprivation variable, and finally a spatial Durbin model also including the spatially lagged social deprivation variable.

```
> ltx0.lsar <- lagsarlm(logtx ~ 1, listw = nb2listw(viz), type = "lag",
+     method = "eigen")


> ltx0.lsar

Call:
lagsarlm(formula = logtx ~ 1, listw = nb2listw(viz), type = "lag",
    method = "eigen")
Type: lag

Coefficients:
(Intercept)         rho
 -2.4539643   0.5685345

Log likelihood: -270.163
> ltx1.lsar <- lagsarlm(logtx ~ DEPRIV, data = subsetor, type = "lag",
+     listw = nb2listw(viz), method = "eigen")


> ltx1.lsar

Call:
lagsarlm(formula = logtx ~ DEPRIV, data = subsetor, listw = nb2listw(viz),
    type = "lag", method = "eigen")
Type: lag

Coefficients:
(Intercept)      DEPRIV         rho
  -3.118961    0.724765    0.496563

Log likelihood: -265.5303
> ltx2.lsar <- lagsarlm(logtx ~ DEPRIV, data = subsetor, type = "mixed",
+     listw = nb2listw(viz), method = "eigen")


> ltx2.lsar

Call:
lagsarlm(formula = logtx ~ DEPRIV, data = subsetor, listw = nb2listw(viz),
    type = "mixed", method = "eigen")
Type: mixed

Coefficients:
(Intercept)      DEPRIV  lag.DEPRIV         rho
 -3.6673151   0.2089467   1.1846246   0.4417988

Log likelihood: -261.829
```

Model output includes the fitted parameter values, including the value of the spatial lag parameter ρ, and the log likelihood value of the fitted model.

```
> anova(ltx0.lsar, ltx1.lsar, ltx2.lsar)
          Model df      AIC    logLik   Test  L.Ratio     p-value
ltx0.lsar     1  3 546.3260 -270.1630             NA          NA
ltx1.lsar     2  4 539.0605 -265.5303 1 vs 2 9.265476 0.002335138
ltx2.lsar     3  5 533.6580 -261.8290 2 vs 3 7.402510 0.006513294
> anova(ltx2.lsar, logtx.lm)
          Model df      AIC    logLik   Test  L.Ratio      p-value
ltx2.lsar     1  5 533.6580 -261.8290             NA           NA
logtx.lm      2  3 578.9471 -286.4735 1 vs 2 49.28905 1.981615e-11
```

These models are compared using an `anova()` function, first between the three spatial lag models — showing that including both DEPRIV and the spatial lag of DEPRIV improve the fit of the model, as measured by the likelihood ratio test. The spatial Durbin model also has the best Akaike Information Criterion (AIC) value. There is a big difference between the linear model and the spatial Durbin model, but note that the spatial lag model with just the intercept term outperforms the linear model including the social deprivation variable, both in terms of AIC and comparing log likelihood values.

```
> ltx2.esar <- errorsarlm(logtx ~ DEPRIV, data = subsetor, listw = nb2listw(viz),
+     method = "eigen")


> ltx2.esar

Call:
errorsarlm(formula = logtx ~ DEPRIV, data = subsetor, listw = nb2listw(viz),
    method = "eigen")
Type: error

Coefficients:
(Intercept)      DEPRIV      lambda
 -5.8944328   0.5963067   0.5116437

Log likelihood: -268.3350

> AIC(ltx2.esar)

[1] 544.6701

> LR.sarlm(ltx2.lsar, ltx2.esar)

        Likelihood ratio for spatial linear models

data:
Likelihood ratio = 13.012, df = 1, p-value = 0.0003095
sample estimates:
Log likelihood of ltx2.lsar Log likelihood of ltx2.esar
                  -261.8290                   -268.3350
```

Just to check, we estimate the spatial error model including the DEPRIV variable (lag and error with just the intercept are equivalent) too, and find that on log-likelihood comparison and AIC values, the spatial lag models seem to give a better fit than the spatial error models. The likelihood ratio test reported is used informally, because the models are not nested as such, although here the spatial error model is equal to the spatial Durbin model when constraints on the parameter estimates are met — here they are not.

```
> summary(ltx2.lsar)
```

```
Call:
lagsarlm(formula = logtx ~ DEPRIV, data = subsetor, listw = nb2listw(viz),
    type = "mixed", method = "eigen")

Residuals:
      Min        1Q    Median        3Q       Max
-2.108455 -0.466208  0.021365  0.554197  1.590469

Type: mixed
Coefficients: (asymptotic standard errors)
            Estimate Std. Error z value  Pr(>|z|)
(Intercept) -3.66732    0.51831 -7.0755 1.489e-12
DEPRIV       0.20895    0.30166  0.6927  0.488522
lag.DEPRIV   1.18462    0.43654  2.7137  0.006654

Rho: 0.4418 LR test value: 30.827 p-value: 2.8205e-08
Asymptotic standard error: 0.078559 z-value: 5.6238 p-value: 1.8679e-08

Log likelihood: -261.829 for mixed model
ML residual variance (sigma squared): 0.49372, (sigma: 0.70265)
Number of observations: 241
Number of parameters estimated: 5
AIC: 533.66, (AIC for lm: 562.49)
LM test for residual autocorrelation
test value: 11.046 p-value: 0.00088899
```

```
> pres <- residuals(logtx.lm)
> brks <- round(quantile(pres, prob = seq(0, 1, 1/6)), digits = 2)
```

```
> plotpolys(subsetor.polys, col = cols[findInterval2(pres, brks)])
> legend(c(289000, 291000), c(9112000, 9114000), fill = cols, bty = "n",
+     legend = leglabs(brks, "<", ">="), cex = 0.9, y.intersp = 0.9)
```

Finally, we can examine the output of `summary()` of the spatial Durbin model, and see that the social deprivation variable is no longer significant — while its spatial lag is, in terms of asymptotic standard errors. The likelihood ratio test on ρ compares the log likelihoods of the fitted spatial Durbin model and the equivalent linear model with no spatially lagged dependent variable. A similar comparison is available using the AIC values of this fit, and the linear model fit holding $\rho = 0$. At the foot of the output, the results of a Lagrange Multiplier test for residual autocorrelation for this model are presented — there is still a lot of spatial dependence present here, as we can also see from the map of the spatial Durbin model residuals in Figure 19.

At least we have been able to find that any relationship between the social deprivation variable and the log rate variable is not as simple as perhaps might have been thought. It is at least possible that there are different kinds of spatial dependence in different parts of the city — known as spatial regimes, or that there are more general mis-specification problems, such as non-stationarity. Modelling is in general much more demanding than visualisation, and in this case it may be that the social deprivation index is too compacted, containing perhaps data that in essence is multivariate rather than univariate. It is of course also possible that the population numbers reflect heterogeneity in levels of risk, even given levels of social deprivation, and finally, some or many of the tracts may be poorly bounded with respect to the underlying social and health-related phenomena being observed.

Figure 19: Map of residuals from the simultaneous autoregressive "Durbin" model fit of the transformed dependent variable.