9

Areal Data and Spatial Autocorrelation

9.1 Introduction

Spatial data are often observed on polygon entities with defined boundaries. The polygon boundaries are defined by the researcher in some fields of study, may be arbitrary in others and may be administrative boundaries created for very different purposes in others again. The observed data are frequently aggregations within the boundaries, such as population counts. The areal entities may themselves constitute the units of observation, for example when studying local government behaviour where decisions are taken at the level of the entity, for example setting local tax rates. By and large, though, areal entities are aggregates, bins, used to tally measurements, like voting results at polling stations. Very often, the areal entities are an exhaustive tessellation of the study area, leaving no part of the total area unassigned to an entity. Of course, areal entities may be made up of multiple geometrical entities, such as islands belonging to the same county; they may also surround other areal entities completely, and may contain holes, like lakes.

The boundaries of areal entities may be defined for some other purpose than their use in data analysis. Postal code areas can be useful for analysis, but were created to facilitate postal deliveries. It is only recently that national census organisations have accepted that frequent, apparently justified, changes to boundaries are a major problem for longitudinal analyses. In Sect. 5.1, we discussed the concept of spatial support, which here takes the particular form of the *modifiable areal unit problem* (Waller and Gotway, 2004, pp. 104–108). Arbitrary areal unit boundaries are a problem if their modification could lead to different results, with the case of political gerrymandering being a sobering reminder of how changes in aggregation may change outcomes.[1] They may also

[1] The CRAN **BARD** package for automated redistricting and heuristic exploration of redistricter revealed preference is an example of the use of R for studying this problem.

get in the way of the analysis if the spatial scale or footprint of an underlying data generating process is not matched by the chosen boundaries.

If data collection can be designed to match the areal entities to the data, the influence of the choice of aggregates will be reduced. An example could be the matching of labour market data to local labour markets, perhaps defined by journeys to work. On the other hand, if we are obliged to use arbitrary boundaries, often because there are no other feasible sources of secondary data, we should be aware of potential difficulties. Such mismatches are among the reasons for finding spatial autocorrelation in analysing areal aggregates; other reasons include substantive spatial processes in which entities influence each other by contagion, such as the adoption of similar policies by neighbours, and model misspecification leaving spatially patterned information in the model residuals. These causes of observed spatial autocorrelation can occur in combination, making the correct identification of the actual spatial processes an interesting undertaking.

A wide range of scientific disciplines have encountered spatial autocorrelation among areal entities, with the term ‘Galton’s problem’ used in several. The problem is to establish how many effectively independent observations are present, when arbitrary boundaries have been used to divide up a study area. In his exchange with Tyler in 1889, Galton questioned whether observations of marriage laws across areal entities constituted independent observations, since they could just reflect a general pattern from which they had all descended. So positive spatial dependence tends to reduce the amount of information contained in the observations, because proximate observations can in part be used to predict each other.

In Chap. 8, we have seen how distances on a continuous surface can be used to structure spatial autocorrelation, for example with the variogram. Here we will be concerned with areal entities that are defined as neighbours, for chosen definitions of neighbours. On a continuous surface, all points are neighbours of each other, though some may carry very little weight, because they are very distant. On a tessellated surface, we can choose neighbour definitions that partition the set of all entities (excluding observation i) into members or non-members of the neighbour set of observation i. We can also decide to give each neighbour relationship an equal weight, or vary the weights on the arcs of the directed graph describing the spatial dependence.

The next two sections will cover the construction of neighbours and of weights that can be applied to neighbourhoods. Once this important and often demanding prerequisite is in place, we go on to look at ways of measuring spatial autocorrelation, bearing in mind that the spatial patterning we find may only indicate that our current model of the data is not appropriate. This applies to areal units not fitting the data generation process, to missing variables including variables with the wrong functional form, and differences between our assumptions about the data and their actual distributions, often shown as over-dispersion in count data. The modelling of areal data will be dealt with in the next chapter, with extensions in Chap. 11.

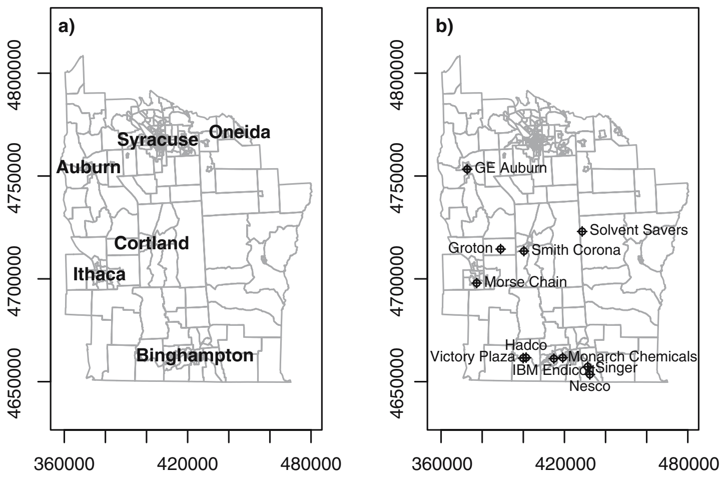

Fig. 9.1. (**a**) Major cities in the eight-county upper New York State study area; (**b**) locations of 11 inactive hazardous waste sites in the study area

While the tests build on models of spatial processes, we look at tests first, and only subsequently move on to modelling. We will also be interested to show how spatial autocorrelation can be introduced into independent data, so that simulations can be undertaken. The 281 census tract data set for eight central New York State counties featured prominently in Waller and Gotway (2004) will be used in many of the examples,[2] supplemented with tract boundaries derived from TIGER 1992 and distributed by SEDAC/CIESIN. This file is not identical with the boundaries used in the original source, but is very close and may be re-distributed, unlike the version used in the book. The area has an extent of about 160 km north–south and 120 km east–west; Fig. 9.1 shows the major cities in the study area and the location of 11 hazardous waste sites. The figures in Waller and Gotway (2004) include water bodies, which are not present in this version of the tract boundaries, in which tract boundaries follow the centre lines of lakes, rather than their shores.

9.2 Spatial Neighbours

Creating spatial weights is a necessary step in using areal data, perhaps just to check that there is no remaining spatial patterning in residuals. The first step is to define which relationships between observations are to be given a non-zero weight, that is to choose the neighbour criterion to be used; the second is to assign weights to the identified neighbour links. Trying to detect pattern in maps of residuals visually is not an acceptable choice, although one sometimes

[2] The boundaries have been projected from geographical coordinates to UTM zone 18.

hears comments explaining the lack of formal analysis such as 'they looked random', or alternatively 'I can see the clusters'. Making the neighbours and weights is, however, not easy to do, and so a number of functions are included in the **spdep** package to help. Further functions are found in some ecology packages, such as the **ade4** package – this package also provides `nb2neig` and `neig2nb` converters for inter-operability. The construction of spatial weights is touched on by Cressie (1993, pp. 384–385), Schabenberger and Gotway (2005, p. 18), Waller and Gotway (2004, pp. 223–225), Fortin and Dale (2005, pp. 113–118), O'Sullivan and Unwin (2003, pp. 193–194) and Banerjee et al. (2004, pp. 70–71). The paucity of treatments in the literature contrasts with the strength of the prior information being introduced by the analyst at this stage, and is why we have chosen to devote a more than proportionally large part of the book to this topic, since analysing areal data is crucially dependent on the choices made in constructing the spatial weights.

9.2.1 Neighbour Objects

In the **spdep** package, neighbour relationships between n observations are represented by an object of class `nb`; the class is an old-style class as presented on p. 24. It is a list of length n with the index numbers of neighbours of each component recorded as an integer vector. If any observation has no neighbours, the component contains an integer zero. It also contains attributes, typically a vector of character region identifiers, and a logical value indicating whether the relationships are symmetric. The region identifiers can be used to check for integrity between the data themselves and the neighbour object. The helper function `card` returns the cardinality of the neighbour set for each object, that is, the number of neighbours; it differs from the application of `length` to the list components because no-neighbour entries are coded as a single element integer vector with the value of zero.

```
> library(spdep)

> library(rgdal)
> NY8 <- readOGR(".", "NY8_utm18")
> NY_nb <- read.gal("NY_nb.gal", region.id = row.names(as(NY8,
+     "data.frame")))

> summary(NY_nb)

Neighbour list object:
Number of regions: 281
Number of nonzero links: 1522
Percentage nonzero weights: 1.927534
Average number of links: 5.41637
Link number distribution:

 1  2  3  4  5  6  7  8  9 10 11
 6 11 28 45 59 49 45 23 10  3  2
```

```
6 least connected regions:
55 97 100 101 244 245 with 1 link
2 most connected regions:
34 82 with 11 links
```

```
> isTRUE(all.equal(attr(NY_nb, "region.id"), row.names(as(NY8,
+     "data.frame"))))
```

```
[1] TRUE
```

```
> plot(NY8, border = "grey60")
> plot(NY_nb, coordinates(NY8), pch = 19, cex = 0.6, add = TRUE)
```

Starting from the census tract contiguities used in Waller and Gotway (2004) and provided as a DBF file on their website, a GAL format file has been created and read into R– we return to the import and export of neighbours on p. 255. Since we now have an `nb` object to examine, we can present the standard methods for these objects. There are `print`, `summary`, `plot`, and other methods; the `summary` method presents a table of the link number distribution, and both `print` and `summary` methods report asymmetry and the presence of no-neighbour observations; asymmetry is present when i is a neighbour of j but j is not a neighbour of i. Figure 9.2 shows the complete neighbour graph for the eight-county study area. For the sake of simplicity in showing how to create neighbour objects, we work on a subset of the map consisting of the census tracts within Syracuse, although the same principles apply to the full data set. We retrieve the part of the neighbour list in Syracuse using the `subset` method.

```
> Syracuse <- NY8[NY8$AREANAME == "Syracuse city", ]
> Sy0_nb <- subset(NY_nb, NY8$AREANAME == "Syracuse city")
> isTRUE(all.equal(attr(Sy0_nb, "region.id"), row.names(as(Syracuse,
+     "data.frame"))))
```

```
[1] TRUE
```

```
> summary(Sy0_nb)
```

```
Neighbour list object:
Number of regions: 63
Number of nonzero links: 346
Percentage nonzero weights: 8.717561
Average number of links: 5.492063
Link number distribution:

 1  2  3  4  5  6  7  8  9
 1  1  5  9 14 17  9  6  1
1 least connected region:
164 with 1 link
1 most connected region:
136 with 9 links
```

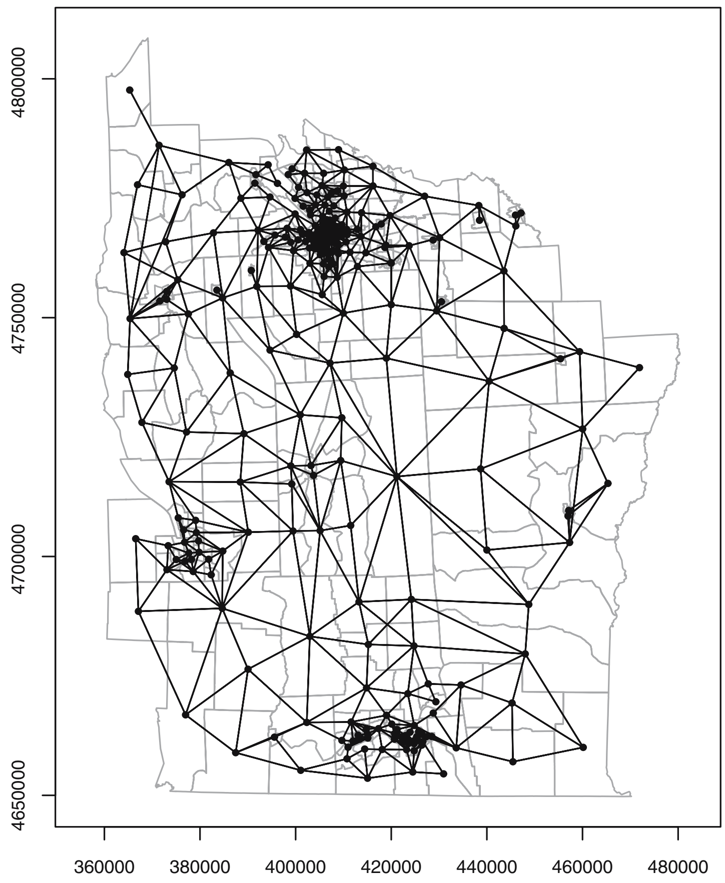

Fig. 9.2. Census tract contiguities, New York eight-county census tracts

9.2.2 Creating Contiguity Neighbours

We can create a copy of the same neighbours object for polygon contiguities using the `poly2nb` function in **spdep**. It takes an object extending the `SpatialPolygons` class as its first argument, and using heuristics identifies polygons sharing boundary points as neighbours. It also has a `snap` argument, to allow the shared boundary points to be a short distance from one another.

```
> class(Syracuse)

[1] "SpatialPolygonsDataFrame"
attr(,"package")
[1] "sp"

> Sy1_nb <- poly2nb(Syracuse)
> isTRUE(all.equal(Sy0_nb, Sy1_nb, check.attributes = FALSE))

[1] TRUE
```

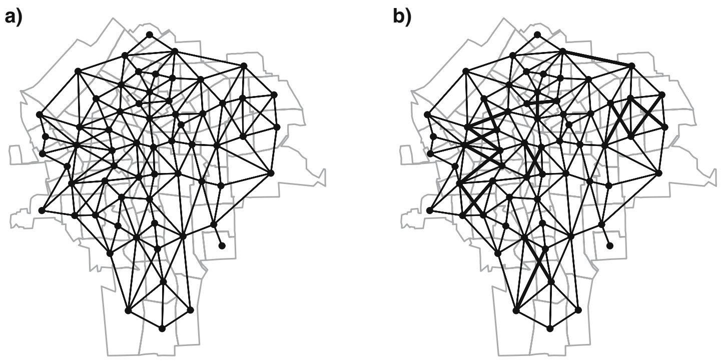

Fig. 9.3. (**a**) Queen-style census tract contiguities, Syracuse; (**b**) Rook-style contiguity differences shown as thicker lines

As we can see, creating the contiguity neighbours from the `Syracuse` object reproduces the neighbours from Waller and Gotway (2004). Careful examination of Fig. 9.2 shows, however, that the graph of neighbours is not planar, since some neighbour links cross each other. By default, the contiguity condition is met when at least one point on the boundary of one polygon is within the snap distance of at least one point of its neighbour. This relationship is given by the argument `queen=TRUE` by analogy with movements on a chessboard. So when three or more polygons meet at a single point, they all meet the contiguity condition, giving rise to crossed links. If `queen=FALSE`, at least two boundary points must be within the snap distance of each other, with the conventional name of a 'rook' relationship. Figure 9.3 shows the crossed line differences that arise when polygons touch only at a single point, compared to the stricter rook criterion.

```
> Sy2_nb <- poly2nb(Syracuse, queen = FALSE)
> isTRUE(all.equal(Sy0_nb, Sy2_nb, check.attributes = FALSE))

[1] FALSE
```

If we have access to a GIS such as GRASS or ArcGIS™, we can export the `SpatialPolygonsDataFrame` object and use the topology engine in the GIS to find contiguities in the graph of polygon edges – a shared edge will yield the same output as the rook relationship. Integration with GRASS was discussed in Sect. 4.4, and functions in **RArcInfo** and the equivalent `readOGR` function in **rgdal** for reading ArcGIS™ coverages in Sects. 4.2.2 and 4.2.1[3] can also be used for retrieving rook neighbours.

This procedure does, however, depend on the topology of the set of polygons being clean, which holds for this subset, but not for the full eight-county data set. Not infrequently, there are small artefacts, such as slivers where boundary lines intersect or diverge by distances that cannot be seen on plots,

[3] A script to access ArcGIS™ coverages using Python and R(D)COM using `readOGR` is on the book website.

but which require intervention to keep the geometries and data correctly associated. When these geometrical artefacts are present, the topology is not clean, because unambiguous shared polygon boundaries cannot be found in all cases; artefacts typically arise when data collected for one purpose are combined with other data or used for another purpose. Topologies are usually cleaned in a GIS by 'snapping' vertices closer than a threshold distance together, removing artefacts – for example, snapping across a river channel where the correct boundary is the median line but the input polygons stop at the channel banks on each side. The `poly2nb` function does have a `snap` argument, which may also be used when input data possess geometrical artefacts.

```
> library(spgrass6)
> writeVECT6(Syracuse, "SY0")
> contig <- vect2neigh("SY0")

> Sy3_nb <- sn2listw(contig)$neighbours
> isTRUE(all.equal(Sy3_nb, Sy2_nb, check.attributes = FALSE))

[1] TRUE
```

Similar approaches may also be used to read ArcGIS™ coverage data by tallying the left neighbour and right neighbour arc indices with the polygons in the data set, using either **RArcInfo** or **rgdal**.

In our Syracuse case, there are no exclaves or 'islands' belonging to the data set, but not sharing boundary points within the snap distance. If the number of polygons is moderate, the missing neighbour links may be added interactively using the `edit` method for `nb` objects, and displaying the polygon background. The same method may be used for removing links which, although contiguity exists, may be considered void, such as across a mountain range.

9.2.3 Creating Graph-Based Neighbours

Continuing with irregularly located areal entities, it is possible to choose a point to represent the polygon-support entities. This is often the polygon centroid, which is not the average of the coordinates in each dimension, but takes proper care to weight the component triangles of the polygon by area. It is also possible to use other points, or if data are available, construct, for example population-weighted centroids. Once representative points are available, the criteria for neighbourhood can be extended from just contiguity to include graph measures, distance thresholds, and k-nearest neighbours.

The most direct graph representation of neighbours is to make a Delaunay triangulation of the points, shown in the first panel in Fig. 9.4. The neighbour relationships are defined by the triangulation, which extends outwards to the convex hull of the points and which is planar. Note that graph-based representations construct the interpoint relationships based on Euclidean distance, with no option to use Great Circle distances for geographical coordinates. Because it joins distant points around the convex hull, it may be worthwhile

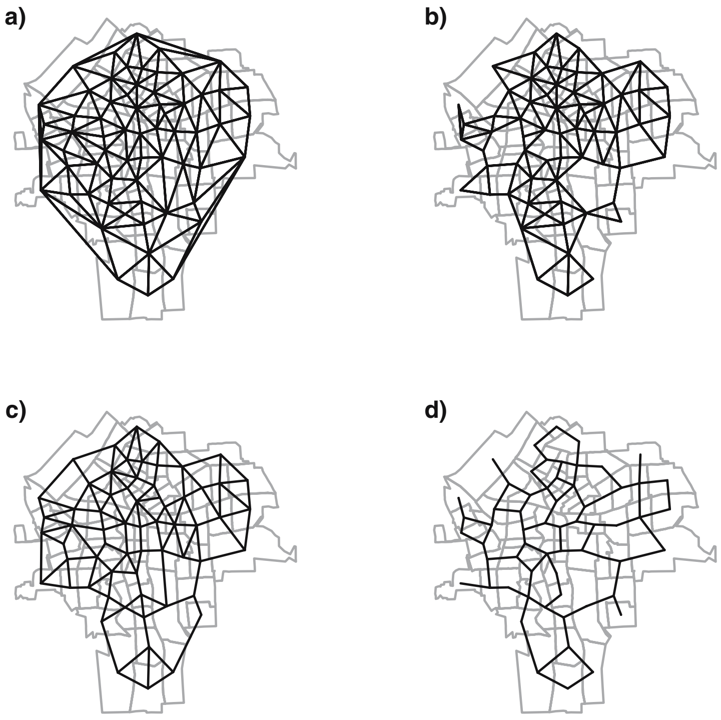

Fig. 9.4. (**a**) Delauney triangulation neighbours; (**b**) Sphere of influence neighbours; (**c**) Gabriel graph neighbours; (**d**) Relative graph neighbours

to thin the triangulation as a Sphere of Influence (SOI) graph, removing links that are relatively long. Points are SOI neighbours if circles centred on the points, of radius equal to the points' nearest neighbour distances, intersect in two places (Avis and Horton, 1985).[4]

```
> coords <- coordinates(Syracuse)
> IDs <- row.names(as(Syracuse, "data.frame"))
> library(tripack)
> Sy4_nb <- tri2nb(coords, row.names = IDs)
> Sy5_nb <- graph2nb(soi.graph(Sy4_nb, coords), row.names = IDs)
> Sy6_nb <- graph2nb(gabrielneigh(coords), row.names = IDs)
> Sy7_nb <- graph2nb(relativeneigh(coords), row.names = IDs)
```

Delaunay triangulation neighbours and SOI neighbours are symmetric by design – if i is a neighbour of j, then j is a neighbour of i. The Gabriel graph is also a subgraph of the Delaunay triangulation, retaining a different set of neighbours (Matula and Sokal, 1980). It does not, however, guarantee symmetry; the same applies to Relative graph neighbours (Toussaint, 1980). The `graph2nb` function takes a `sym` argument to insert links to restore symmetry,

[4] Functions for graph-based neighbours were kindly contributed by Nicholas Lewin-Koh.

but the graphs then no longer exactly fulfil their neighbour criteria. All the graph-based neighbour schemes always ensure that all the points will have at least one neighbour. Subgraphs of the full triangulation may also have more than one graph after trimming. The functions `is.symmetric.nb` can be used to check for symmetry, with argument `force=TRUE` if the symmetry attribute is to be overridden, and `n.comp.nb` reports the number of graph components and the components to which points belong (after enforcing symmetry, because the algorithm assumes that the graph is not directed). When there are more than one graph component, the matrix representation of the spatial weights can become block-diagonal if observations are appropriately sorted.

```
> nb_l <- list(Triangulation = Sy4_nb, SOI = Sy5_nb, Gabriel = Sy6_nb,
+     Relative = Sy7_nb)
> sapply(nb_l, function(x) is.symmetric.nb(x, verbose = FALSE,
+     force = TRUE))
```

```
Triangulation           SOI       Gabriel      Relative
         TRUE          TRUE         FALSE         FALSE
```

```
> sapply(nb_l, function(x) n.comp.nb(x)$nc)
```

```
Triangulation           SOI       Gabriel      Relative
            1             1             1             1
```

9.2.4 Distance-Based Neighbours

An alternative method is to choose the k nearest points as neighbours – this adapts across the study area, taking account of differences in the densities of areal entities. Naturally, in the overwhelming majority of cases, it leads to asymmetric neighbours, but will ensure that all areas have k neighbours. The `knearneigh` returns an intermediate form converted to an `nb` object by `knn2nb`; `knearneigh` can also take a `longlat` argument to handle geographical coordinates.

```
> Sy8_nb <- knn2nb(knearneigh(coords, k = 1), row.names = IDs)
> Sy9_nb <- knn2nb(knearneigh(coords, k = 2), row.names = IDs)
> Sy10_nb <- knn2nb(knearneigh(coords, k = 4), row.names = IDs)
> nb_l <- list(k1 = Sy8_nb, k2 = Sy9_nb, k4 = Sy10_nb)
> sapply(nb_l, function(x) is.symmetric.nb(x, verbose = FALSE,
+     force = TRUE))
```

```
   k1    k2    k4
FALSE FALSE FALSE
```

```
> sapply(nb_l, function(x) n.comp.nb(x)$nc)
```

```
k1 k2 k4
15  1  1
```

Figure 9.5 shows the neighbour relationships for $k = 1, 2, 4$, with many components for $k = 1$. If need be, k-nearest neighbour objects can be made symmetrical using the `make.sym.nb` function. The $k = 1$ object is also useful in

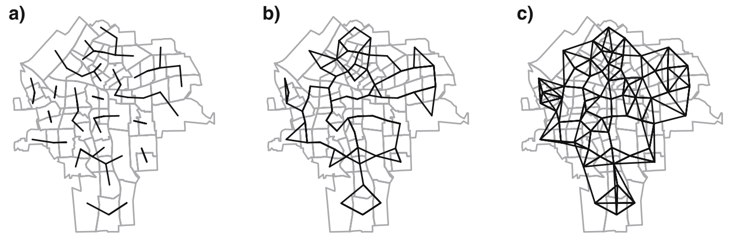

Fig. 9.5. (**a**) $k = 1$ neighbours; (**b**) $k = 2$ neighbours; (**c**) $k = 4$ neighbours

finding the minimum distance at which all areas have a distance-based neighbour. Using the `nbdists` function, we can calculate a list of vectors of distances corresponding to the neighbour object, here for first nearest neighbours. The greatest value will be the minimum distance needed to make sure that all the areas are linked to at least one neighbour. The `dnearneigh` function is used to find neighbours with an interpoint distance, with arguments `d1` and `d2` setting the lower and upper distance bounds; it can also take a `longlat` argument to handle geographical coordinates.

```
> dsts <- unlist(nbdists(Sy8_nb, coords))
> summary(dsts)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  395.7   587.3   700.1   760.4   906.1  1545.0

> max_1nn <- max(dsts)
> max_1nn

[1] 1544.615

> Sy11_nb <- dnearneigh(coords, d1 = 0, d2 = 0.75 * max_1nn,
+     row.names = IDs)
> Sy12_nb <- dnearneigh(coords, d1 = 0, d2 = 1 * max_1nn,
+     row.names = IDs)
> Sy13_nb <- dnearneigh(coords, d1 = 0, d2 = 1.5 * max_1nn,
+     row.names = IDs)
> nb_l <- list(d1 = Sy11_nb, d2 = Sy12_nb, d3 = Sy13_nb)
> sapply(nb_l, function(x) is.symmetric.nb(x, verbose = FALSE,
+     force = TRUE))

  d1   d2   d3
TRUE TRUE TRUE

> sapply(nb_l, function(x) n.comp.nb(x)$nc)

d1 d2 d3
 4  1  1
```

Figure 9.6 shows how the numbers of distance-based neighbours increase with moderate increases in distance. Moving from 0.75 times the minimum

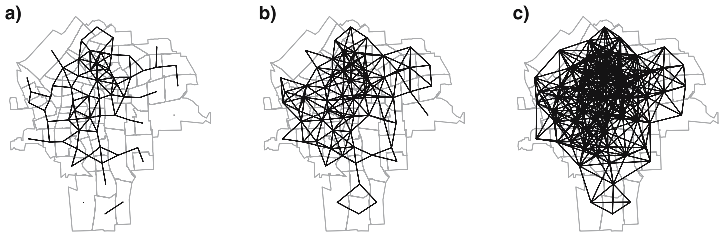

Fig. 9.6. (**a**) Neighbours within 1,158 m; (**b**) neighbours within 1,545 m; (**c**) neighbours within 2,317 m

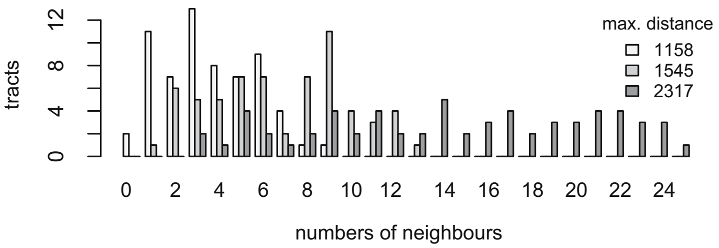

Fig. 9.7. Distance-based neighbours: frequencies of numbers of neighbours by census tract

all-included distance, to the all-included distance, and 1.5 times the minimum all-included distance, the numbers of links grow rapidly. This is a major problem when some of the first nearest neighbour distances in a study area are much larger than others, since to avoid no-neighbour areal entities, the distance criterion will need to be set such that many areas have many neighbours. Figure 9.7 shows the counts of sizes of sets of neighbours for the three different distance limits. In Syracuse, the census tracts are of similar areas, but were we to try to use the distance-based neighbour criterion on the eight-county study area, the smallest distance securing at least one neighbour for every areal entity is over 38 km.

```
> dsts0 <- unlist(nbdists(NY_nb, coordinates(NY8)))
> summary(dsts0)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   82.7  1505.0  3379.0  5866.0  8954.0 38440.0
```

If the areal entities are approximately regularly spaced, using distance-based neighbours is not necessarily a problem. Provided that care is taken to handle the side effects of 'weighting' areas out of the analysis, using lists of neighbours with no-neighbour areas is not necessarily a problem either, but certainly ought to raise questions. Different disciplines handle the definition of neighbours in their own ways by convention; in particular, it seems that ecologists frequently use distance bands. If many distance bands are used, they approach the variogram, although the underlying understanding of spatial autocorrelation seems to be by contagion rather than continuous.

9.2.5 Higher-Order Neighbours

Distance bands can be generated by using a sequence of `d1` and `d2` argument values for the `dnearneigh` function if needed to construct a spatial autocorrelogram as understood in ecology. In other conventions, correlograms are constructed by taking an input list of neighbours as the first-order sets, and stepping out across the graph to second-, third-, and higher-order neighbours based on the number of links traversed, but not permitting cycles, which could risk making i a neighbour of i itself (O'Sullivan and Unwin, 2003, p. 203). The `nblag` function takes an existing neighbour list and returns a list of lists, from first to `maxlag` order neighbours.

```
> Sy0_nb_lags <- nblag(Sy0_nb, maxlag = 9)
```

Table 9.1 shows how the wave of connectedness in the graph spreads to the third order, receding to the eighth order, and dying away at the ninth

Table 9.1. Higher-order contiguities: frequencies of numbers of neighbours by order of neighbour list

	First	Second	Third	Fourth	Fifth	Sixth	Seventh	Eighth	Ninth
0	0	0	0	0	0	6	21	49	63
1	1	0	0	0	0	3	7	6	0
2	1	0	0	0	0	0	4	5	0
3	5	0	0	0	1	2	5	2	0
4	9	2	0	0	1	8	9	1	0
5	14	2	0	0	3	2	7	0	0
6	17	0	0	0	1	5	3	0	0
7	9	6	1	0	1	5	5	0	0
8	6	6	3	1	3	4	1	0	0
9	1	11	5	3	7	8	0	0	0
10	0	11	5	5	13	9	0	0	0
11	0	4	7	7	12	5	0	0	0
12	0	3	14	16	8	5	1	0	0
13	0	7	6	16	9	1	0	0	0
14	0	4	8	5	3	0	0	0	0
15	0	6	3	3	1	0	0	0	0
16	0	1	3	3	0	0	0	0	0
17	0	0	0	2	0	0	0	0	0
18	0	0	1	0	0	0	0	0	0
19	0	0	1	1	0	0	0	0	0
20	0	0	1	1	0	0	0	0	0
21	0	0	3	0	0	0	0	0	0
22	0	0	1	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0
24	0	0	1	0	0	0	0	0	0

order – there are no tracts nine steps from each other in this graph. Both the distance bands and the graph step order approaches to spreading neighbourhoods can be used to examine the shape of relationship intensities in space, like the variogram, and can be used in attempting to look at the effects of scale.

9.2.6 Grid Neighbours

When the data are known to be arranged in a regular, rectangular grid, the `cell2nb` function can be used to construct neighbour lists, including those on a torus. These are useful for simulations, because, since all areal entities have equal numbers of neighbours, and there are no edges, the structure of the graph is as neutral as can be achieved. Neighbours can either be of type rook or queen.

```
> cell2nb(7, 7, type = "rook", torus = TRUE)
```

```
Neighbour list object:
Number of regions: 49
Number of nonzero links: 196
Percentage nonzero weights: 8.163265
Average number of links: 4
```

```
> cell2nb(7, 7, type = "rook", torus = FALSE)
```

```
Neighbour list object:
Number of regions: 49
Number of nonzero links: 168
Percentage nonzero weights: 6.997085
Average number of links: 3.428571
```

When a regular, rectangular grid is not complete, then we can use knowledge of the cell size stored in the grid topology to create an appropriate list of neighbours, using a tightly bounded distance criterion. Neighbour lists of this kind are commonly found in ecological assays, such as studies of species richness at a national or continental scale. It is also in these settings, with moderately large n, here $n = 3{,}103$, that the use of a sparse, list based representation shows its strength. Handling a 281×281 matrix for the eight-county census tracts is feasible, easy for a 63×63 matrix for Syracuse census tracts, but demanding for a $3{,}103 \times 3{,}103$ matrix.

```
> data(meuse.grid)
> coordinates(meuse.grid) <- c("x", "y")
> gridded(meuse.grid) <- TRUE
> dst <- max(slot(slot(meuse.grid, "grid"), "cellsize"))
> mg_nb <- dnearneigh(coordinates(meuse.grid), 0, dst)
> mg_nb
```

```
Neighbour list object:
Number of regions: 3103
Number of nonzero links: 12022
Percentage nonzero weights: 0.1248571
Average number of links: 3.874315
```

```
> table(card(mg_nb))

   1    2    3    4
   1  133  121 2848
```

9.3 Spatial Weights

The literature on spatial weights is surprisingly small, given their importance in measuring and modelling spatial dependence in areal data. Griffith (1995) provides sound practical advice, while Bavaud (1998) seeks to insert conceptual foundations under ad hoc spatial weights. Spatial weights can be seen as a list of weights indexed by a list of neighbours, where the weight of the link between i and j is the kth element of the ith weights list component, and k tells us which of the ith neighbour list component values is equal to j. If j is not present in the ith neighbour list component, j is not a neighbour of i. Consequently, some weights w_{ij} in the $\mathbf{W}$ weights matrix representation will set to zero, where j is not a neighbour of i. Here, we follow Tiefelsdorf et al. (1999) in our treatment, using their abstraction of spatial weights styles.

9.3.1 Spatial Weights Styles

Once the list of sets of neighbours for our study area is established, we proceed to assign spatial weights to each relationship. If we know little about the assumed spatial process, we try to avoid moving far from the binary representation of a weight of unity for neighbours (Bavaud, 1998), and zero otherwise. In this section, we review the ways that weights objects – `listw` objects – are constructed; the class is an old-style class as described on p. 24. Next, the conversion of these objects into dense and sparse matrix representations will be shown, concluding with functions for importing and exporting neighbour and weights objects.

The `nb2listw` function takes a neighbours list object and converts it into a weights object. The default conversion style is `W`, where the weights for each areal entity are standardised to sum to unity; this is also often called row standardisation. The `print` method for `listw` objects shows the characteristics of the underlying neighbours, the style of the spatial weights, and the spatial weights constants used in calculating tests of spatial autocorrelation. The `neighbours` component of the object is the underlying `nb` object, which gives the indexing of the `weights` component.

```
> Sy0_lw_W <- nb2listw(Sy0_nb)
> Sy0_lw_W

Characteristics of weights list object:
Neighbour list object:
Number of regions: 63
Number of nonzero links: 346
Percentage nonzero weights: 8.717561
Average number of links: 5.492063

Weights style: W
Weights constants summary:
   n   nn S0       S1      S2
W 63 3969 63 24.78291 258.564
```

```
> names(Sy0_lw_W)
```

```
[1] "style"      "neighbours" "weights"
```

```
> names(attributes(Sy0_lw_W))
```

```
[1] "names"     "class"     "region.id" "call"
```

For `style="W"`, the weights vary between unity divided by the largest and smallest numbers of neighbours, and the sums of weights for each areal entity are unity. This spatial weights style can be interpreted as allowing the calculation of average values across neighbours. The weights for links originating at areas with few neighbours are larger than those originating at areas with many neighbours, perhaps boosting areal entities on the edge of the study area unintentionally. This representation is no longer symmetric, but is similar to symmetric – this matters as we see below in Sect. 10.2.1.

```
> 1/rev(range(card(Sy0_lw_W$neighbours)))
```

```
[1] 0.1111111 1.0000000
```

```
> summary(unlist(Sy0_lw_W$weights))
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.1111  0.1429  0.1667  0.1821  0.2000  1.0000
```

```
> summary(sapply(Sy0_lw_W$weights, sum))
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      1       1       1       1       1       1
```

Setting `style="B"` – ‘binary’ – retains a weight of unity for each neighbour relationship, but in this case, the sums of weights for areas differ according to the numbers of neighbour areas have.

```
> Sy0_lw_B <- nb2listw(Sy0_nb, style = "B")
> summary(unlist(Sy0_lw_B$weights))
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      1       1       1       1       1       1
```

```
> summary(sapply(Sy0_lw_B$weights, sum))
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   4.500   6.000   5.492   6.500   9.000
```

Two further styles with equal weights for all links are available: `C` and `U`, where the complete set of `C` weights sums to the number of areas, and `U` weights sum to unity.

```
> Sy0_lw_C <- nb2listw(Sy0_nb, style = "C")
> length(Sy0_lw_C$neighbours)/length(unlist(Sy0_lw_C$neighbours))

[1] 0.1820809

> summary(unlist(Sy0_lw_C$weights))

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.1821  0.1821  0.1821  0.1821  0.1821  0.1821

> summary(sapply(Sy0_lw_C$weights, sum))

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.1821  0.8194  1.0920  1.0000  1.1840  1.6390
```

Finally, the use of a variance-stabilising coding scheme has been proposed by Tiefelsdorf et al. (1999) and is provided as `style="S"`. The weights vary, less than for `style="W"`, but the row sums of weights by area vary more than for `style="W"` (where they are alway unity) and less than for styles `B`, `C`, or `U`. This style also makes asymmetric weights, but as with `style="W"`, they may be similar to symmetric if the neighbours list was itself symmetric. In the same way that the choice of the criteria to define neighbours may affect the results in testing or modelling of the use of weights constructed from those neighbours, results may also be changed by the choice of weights style. As indicated above, links coming from areal entities with many neighbours may be either weighted up or down, depending on the choice of style. The variance-stabilising coding scheme seeks to moderate these conflicting impacts.

```
> Sy0_lw_S <- nb2listw(Sy0_nb, style = "S")
> summary(unlist(Sy0_lw_S$weights))

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.1440  0.1633  0.1764  0.1821  0.1932  0.4321

> summary(sapply(Sy0_lw_S$weights, sum))

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.4321  0.9152  1.0580  1.0000  1.1010  1.2960
```

9.3.2 General Spatial Weights

The `glist` argument can be used to pass a list of vectors of general weights corresponding to the neighbour relationships to `nb2listw`. Say that we believe that the strength of neighbour relationships attenuates with distance, one of the cases considered by Cliff and Ord (1981, pp. 17–18); O'Sullivan and Unwin (2003, pp. 201–202) provide a similar discussion. We could set the weights to be proportional to the inverse distance between points representing the areas, using `nbdists` to calculate the distances for the given `nb` object. Using `lapply` to invert the distances, we can obtain a different structure of spatial weights from those above. If we have no reason to assume any more knowledge about neighbour relations than their existence or absence, this step is potentially misleading. If we do know, on the other hand, that migration or commuting flows describe the spatial weights' structure better than the binary alternative, it may be worth using them as general weights; there may, however, be symmetry problems, because such flows – unlike inverse distances – are only rarely symmetric.

```
> dsts <- nbdists(Sy0_nb, coordinates(Syracuse))
> idw <- lapply(dsts, function(x) 1/(x/1000))
> Sy0_lw_idwB <- nb2listw(Sy0_nb, glist = idw, style = "B")
> summary(unlist(Sy0_lw_idwB$weights))
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.3886  0.7374  0.9259  0.9963  1.1910  2.5270
```

```
> summary(sapply(Sy0_lw_idwB$weights, sum))
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.304   3.986   5.869   5.471   6.737   9.435
```

Figure 9.8 shows three representations of spatial weights for Syracuse displayed as matrices. The `style="W"` image on the left is evidently asymmetric, with darker greys showing larger weights for areas with few neighbours. The other two panels are symmetric, but express different assumptions about the strengths of neighbour relationships.

The final argument to `nb2listw` allows us to handle neighbour lists with no-neighbour areas. It is not obvious that the weight representation of the empty set is zero – perhaps it should be `NA`, which would lead to problems later.

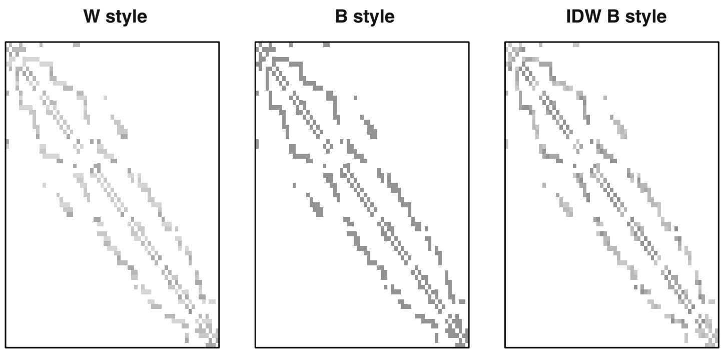

Fig. 9.8. Three spatial weights representations for Syracuse

For this reason, the default value of the argument is `zero.policy=FALSE`, leading to an error when given an `nb` argument with areas with no neighbours. Setting the argument to `TRUE` permits the creation of the spatial weights object, with zero weights. The `zero.policy` argument will subsequently need to be used in each function called, mainly to keep reminding the user that having areal entities with no neighbours is seen as unfortunate. The contrast between the set-based understanding of neighbours and conversion to a matrix representation is discussed by Bivand and Portnov (2004), and boils down to whether the product of a no-neighbour area's weights and an arbitrary n-vector should be a missing value or numeric zero. As we see later (p. 262), keeping the no-neighbour areal entities raises questions about the relevant size of n when testing for autocorrelation, among other issues.

```
> Sy0_lw_D1 <- nb2listw(Sy11_nb, style = "B")

Error in nb2listw(Sy11_nb, style = "B") : Empty neighbour sets found

> Sy0_lw_D1 <- nb2listw(Sy11_nb, style = "B", zero.policy = TRUE)
> print(Sy0_lw_D1, zero.policy = TRUE)

Characteristics of weights list object:
Neighbour list object:
Number of regions: 63
Number of nonzero links: 230
Percentage nonzero weights: 5.79491
Average number of links: 3.650794
2 regions with no links:
154 168

Weights style: B
Weights constants summary:
   n   nn  S0  S1   S2
B 61 3721 230 460 4496
```

The parallel problem of data sets with missing values in variables but with with fully specified spatial weights is approached through the `subset.listw` method, which re-generates the weights for the given subset of areas, for example given by `complete.cases`. Knowing which observations are incomplete, the underlying neighbours and weights can be subsetted in some cases, with the aim of avoiding the propagation of `NA` values when calculating spatially lagged values. Many tests and model fitting functions can carry this out internally if the appropriate argument flag is set, although the careful analyst will prefer to subset the input data and the weights before testing or modelling.

9.3.3 Importing, Converting, and Exporting Spatial Neighbours and Weights

Neighbour and weights objects produced in other software can be imported into R without difficulty, and such objects can be exported to other software too. As examples, some files have been generated in GeoDa[5] from the Syracuse census tracts written out as a shapefile, with the centroid used here stored in the data frame. The first two are for contiguity neighbours, using the queen and rook criteria, respectively. These so-called GAL-format files contain only neighbour information, and are described in detail in the help file accompanying the function `read.gal`.

```
> Sy14_nb <- read.gal("Sy_GeoDa1.GAL")
> isTRUE(all.equal(Sy0_nb, Sy14_nb, check.attributes = FALSE))

[1] TRUE

> Sy15_nb <- read.gal("Sy_GeoDa2.GAL")
> isTRUE(all.equal(Sy2_nb, Sy15_nb, check.attributes = FALSE))

[1] TRUE
```

The `write.nb.gal` function is used to write GAL-format files from `nb` objects. GeoDa also makes GWT-format files, described in the GeoDa documentation and the help file, which also contain distance information for the link between the areas, and are stored in a three-column sparse representation. They can be read using `read.gwt2nb`, here for a four-nearest-neighbour scheme, and only using the neighbour links. In general, **spdep** and GeoDa neighbours and weights are easy to exchange, not least because of generous contributions of code to **spdep** and time for testing by Luc Anselin, who created and administers GeoDa.

```
> Sy16_nb <- read.gwt2nb("Sy_GeoDa4.GWT")
> isTRUE(all.equal(Sy10_nb, Sy16_nb, check.attributes = FALSE))

[1] TRUE
```

A similar set of functions is available for exchanging spatial weights with the Spatial Econometrics Library[6] created by James LeSage. The sparse representation of weights is similar to the GWT-format and can be imported using `read.dat2listw`. Export to three different formats goes through the `listw2sn` function, which converts a spatial weights object to a three-column sparse representation, similar to the 'spatial.neighbor' class in the `S-PLUS`™ SpatialStats module. The output data frame can be written with `write.table` to a file to be read into `S-PLUS`™, written out as a GWT-format file with `write.sn2gwt` or as a text representation of a sparse matrix for Matlab™ with `write.sn2dat`. There is a function called `listw2WB` for creating a list of spatial weights for WinBUGS, to be written to file using `dput`.

In addition, `listw2mat` can be used to export spatial weights to, among others, Stata for use with the contributed `spatwmat` command there. This is done by writing the matrix out as a Stata™ data file, here for the binary contiguity matrix for Syracuse:

[5] `http://www.geoda.uiuc.edu/`, Anselin et al. (2006).
[6] `http://www.spatial-econometrics.com`.

```
> library(foreign)
> df <- as.data.frame(listw2mat(Sy0_lw_B))
> write.dta(df, file = "Sy0_lw_B.dta", version = 7)
```

The `mat2listw` can be used to reverse the process, when a dense weights matrix has been read into R, and needs to be made into a neighbour and weights list object. Unfortunately, this function does not set the `style` of the `listw` object to a known value, using `M` to signal this lack of knowledge. It is then usual to rebuild the `listw` object, treating the `neighbours` component as an `nb` object, the `weights` component as a list of general weights and setting the style in the `nb2listw` function directly. It was used for the initial import of the eight-county contiguities, as shown in detail on the `NY_data` help page provided with **spdep**.

Finally, there is a function `nb2lines` to convert neighbour lists into SpatialLinesDataFrame objects, given point coordinates representing the areas. This allows neighbour objects to be plotted in an alternative way, and if need be, to be exported as shapefiles.

9.3.4 Using Weights to Simulate Spatial Autocorrelation

In Fig. 9.8, use was made of `listw2mat` to turn a spatial weights object into a dense matrix for display. The same function is used for constructing a dense representation of the $(\mathbf{I} - \rho\mathbf{W})$ matrix to simulate spatial autocorrelation within the `invIrW` function, where $\mathbf{W}$ is a weights matrix, ρ is a spatial autocorrelation coefficient, and $\mathbf{I}$ is the identity matrix. This approach was introduced by Cliff and Ord (1973, pp. 146–147), and does not impose strict conditions on the matrix to be inverted (only that it be non-singular), and only applies to simulations from a simultaneous autoregressive process. The underlying framework for the covariance representation used here – simultaneous autoregression – will be presented in Sect. 10.2.1.

Starting with a vector of random numbers corresponding to the number of census tracts in Syracuse, we use the row-standardised contiguity weights to introduce autocorrelation.

```
> set.seed(987654)
> n <- length(Sy0_nb)
> uncorr_x <- rnorm(n)
> rho <- 0.5
> autocorr_x <- invIrW(Sy0_lw_W, rho) %*% uncorr_x
```

The outcome is shown in Fig. 9.9, where the spatial lag plot of the original, uncorrelated variable contrasts with that of the autocorrelated variable, which now has a strong positive relationship between tract values and the spatial lag – here the average of values of neighbouring tracts.

The `lag` method for `listw` objects creates ‘spatial lag’ values: $\text{lag}(y_i) = \sum_{j \in N_i} w_{ij} y_j$ for observed values y_i; N_i is the set of neighbours of i. If the

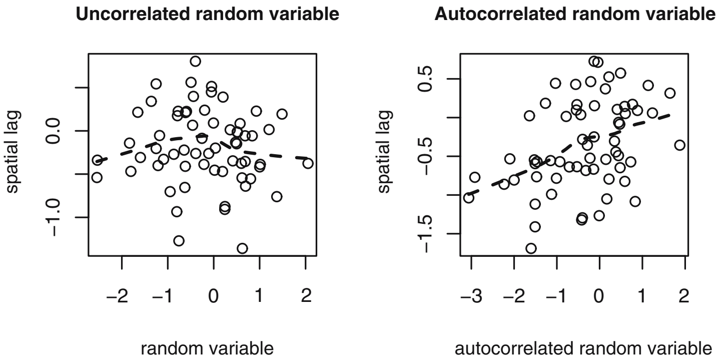

Fig. 9.9. Simulating spatial autocorrelation: spatial lag plots, showing a locally weighted smoother line

weights object style is row-standardisation, the $\text{lag}(y_i)$ values will be averages over the sets of neighbours for each i, rather like a moving window defined by N_i and including values weighted by w_{ij}.

9.3.5 Manipulating Spatial Weights

There are three contributed packages providing support for sparse matrices, **SparseM**, **Matrix**, and **spam**. The **spdep** package began by using compiled code shipped with the package for sparse matrix handling, but changed first to **SparseM**, next adding **Matrix** wrappers, and more recently introducing the use of **spam** and deprecating the interface to **SparseM**. The `as.spam.listw` wrapper to the **spam** package `spam` class is used internally in spatial regression functions among others. The `as_dgRMatrix_listw` wrapper provides the same conversion to the **Matrix** `dgRMatrix` class.

A function that is used a good deal within testing and model fitting functions is `listw2U`, which returns a symmetric `listw` object representing the $\frac{1}{2}(W + W^{\mathrm{T}})$ spatial weights matrix.

Analysing areal data is crucially dependent on the construction of the spatial weights, which is why it has taken some time to describe the breadth of choices facing the researcher. We can now go on to test for spatial autocorrelation, and to model using assumptions about underlying spatial processes.

9.4 Spatial Autocorrelation: Tests

Now that we have a range of ways of constructing spatial weights, we can start using them to test for the presence of spatial autocorrelation. Before doing anything serious, it would be very helpful to review the assumptions being made in the tests; we will be using Moran's I as an example, but the consequences apply to other tests too. As Schabenberger and Gotway (2005, pp. 19–23) explain clearly, tests assume that the mean model of the data removes systematic spatial patterning from the data. If we are examining ecological data, but neglect environmental drivers such as temperature, precipitation, or elevation, we should not be surprised if the data seem to display spatial autocorrelation (for a discussion, see Bivand, 2008, pp. 9–15). Such misspecification of the mean model is not at all uncommon, and may be unavoidable where observations on variables needed to specify it correctly are not available. In fact, Cressie (1993, p. 442) only discusses the testing of residual autocorrelation, and then very briefly, preferring to approach autocorrelation through modelling.

Another issue that can arise is that the spatial weights we use for testing are not those that generated the autocorrelation – our chosen weights may, for example not suit the actual scales of interaction between areal entities. This is a reflection of misspecification of the model of the variance of the residuals from the mean model, which can also include making distributional assumptions that are not appropriate for the data, for example assuming homoskedasticity or regular shape parameters (for example, skewness and kurtosis). Some of these can be addressed by transforming the data and by using weighted estimation, but in any case, care is needed in interpreting apparent spatial autocorrelation that may actually stem from misspecification.

The use of global tests for spatial autocorrelation is covered in much more detail that the construction of spatial weights in the spatial data analysis texts that we are tracking. Waller and Gotway (2004, pp. 223–236) follow up the problem of mistaking the misspecification of the mean model for spatial autocorrelation. This is less evident in Fortin and Dale (2005, pp. 122–132) and O'Sullivan and Unwin (2003, pp. 180–203), but they devote more space to join count statistics for categorical data. Banerjee et al. (2004, pp. 71–73) are, like Cressie (1993), more concerned with modelling than testing.

We begin with the simulated variable for the Syracuse census tracts (see Sect. 9.3.4). Since the input variable is known to be drawn at random from the Normal distribution, we can manipulate it to see what happens to test results under different conditions. The test to be used in this introductory discussion is Moran's I, which is calculated as a ratio of the product of the variable of interest and its spatial lag, with the cross-product of the variable of interest, and adjusted for the spatial weights used:

$$I = \frac{n}{\sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij}} \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij}(y_i - \bar{y})(y_j - \bar{y})}{\sum_{i=1}^{n} (y_i - \bar{y})^2},$$

where y_i is the ith observation, $\bar{y}$ is the mean of the variable of interest, and w_{ij} is the spatial weight of the link between i and j. Centring on the mean is equivalent to asserting that the correct model has a constant mean, and that any remaining patterning after centring is caused by the spatial relationships encoded in the spatial weights.

Table 9.2. Moran's I test results for five different data generating processes

	I	$E(I)$	$\text{var}(I)$	St. deviate	p-value
uncorr_x	−0.03329	−0.01613	0.00571	−0.227	0.59
autocorr_x	0.2182	−0.0161	0.0057	3.1	0.00096
autocorr_x k=1	0.1921	−0.0161	0.0125	1.86	0.031
trend_x	0.23747	−0.01613	0.00575	3.34	0.00041
lm(trend_x ∼ et)	−0.0538	−0.0309	0.0054	−0.312	0.62

The results for Moran's I are collated in Table 9.2 for five settings. The first column contains the observed value of I, the second is the expectation, which is $-1/(n-1)$ for the mean-centred cases, the third the variance of the statistic under randomisation, next the standard deviate $(I - E(I))/\sqrt{\text{var}(I)}$, and finally the p-value of the test for the alternative that $I > E(I)$. The test results are for the uncorrelated case first (`uncorr_x`) – there is no trace of spatial dependence with these weights. Even though a random drawing could show spatial autocorrelation, we would be unfortunate to find a pattern corresponding to our spatial weights by chance for just one draw. When the spatially autocorrelated variable is tested (`autocorr_x`), it shows, as one would expect, a significant result for these spatial weights. If we use spatial weights that differ from those used to generate the spatial autocorrelation (`autocorr_x k=1`), the value of I falls, and although it is marginally significant, it is worth remembering that, had the generating process been less strong, we might have come to the wrong conclusion based on the choice of spatial weights not matching the actual generating process.

```
> moran_u <- moran.test(uncorr_x, listw = Sy0_lw_W)
> moran_a <- moran.test(autocorr_x, listw = Sy0_lw_W)
> moran_a1 <- moran.test(autocorr_x, listw = nb2listw(Sy9_nb,
+     style = "W"))
```

The final two rows of Table 9.2 show what can happen when our assumption of a constant mean is erroneous (Schabenberger and Gotway, 2005, pp. 22–23). Introducing a gentle trend rising from west to east into the uncorrelated random variable, we have a situation in which there is no underlying spatial autocorrelation, just a simple linear trend. If we assume a constant mean, we reach the wrong conclusion shown in the fourth row of the table (`trend_x`). The final row shows how we get back to the uncorrelated residuals by including the trend in the mean, and again have uncorrelated residuals (`lm(trend_x ∼ et)`).

```
> et <- coords[, 1] - min(coords[, 1])
> trend_x <- uncorr_x + 0.00025 * et
> moran_t <- moran.test(trend_x, listw = Sy0_lw_W)
> moran_t1 <- lm.morantest(lm(trend_x ~ et), listw = Sy0_lw_W)
```

This shows how important it can be to understand that tests for spatial autocorrelation can also react to a misspecified model of the mean, and that the omission of a spatially patterned variable from the mean function will 'look like' spatial autocorrelation to the tests.

9.4.1 Global Tests

Moran's I – `moran.test` – is perhaps the most common global test, and for this reason we continue to use it here. Other global tests implemented in the **spdep** package include Geary's C (`geary.test()`), the global Getis-Ord G (`globalG.test()`), and the spatial general cross product Mantel test, which includes Moran's I, Geary's C, and the Sokal variant of Geary's C as alternative forms (`sp.mantel.mc()`). All these are for continuous variables, with `moran.test()` having an argument to use an adjustment for a ranked continuous variable, that is where the metric of the variable is by the ranks of its values rather than the values themselves. There are also join count tests for categorical variables, with the variable of interest represented as a factor (`joincount.test()` for same-colour joins, `joincount.multi()` for same-colour and different colour joins).

The values of these statistics may be of some interest in themselves, but are not directly interpretable. The approach taken most generally is to standardise the observed value by subtracting the analytical expected value, and dividing the difference by the square root of the analytical variance for the spatial weights used, for a set of assumptions. The result is a standard deviate, and is compared with the Normal distribution to find the probability value of the observed statistic under the null hypothesis of no spatial dependence for the chosen spatial weights – most often the test is one-sided, with an alternative hypothesis of the observed statistic being significantly greater than its expected value.

As we see, outcomes can depend on the choices made, for example the style of the weights and to what extent the assumptions made are satisfied. It might seem that Monte Carlo or equivalently bootstrap permutation-based tests, in which the values of the variable of interest are randomly assigned to spatial entities, would provide protection against errors of inference. In fact, because tests for spatial autocorrelation are sensitive to spatial patterning in the variable of interest from any source, they are not necessarily – as we saw above – good guides to decide what is going on in the data generation process. Parametric bootstrapping or tests specifically tuned to the setting – or better specification of the variable of interest – are sometimes needed.

A further problem for which there is no current best advice is how to proceed if some areal entities have no neighbours. By default, test functions in **spdep** do not accept spatial weights with no-neighbour entities unless the `zero.policy` argument is set to `TRUE`. But even if the analyst accepts the presence of rows and columns with only zero entries in the spatial weights matrix, the correct size of n can be taken as the number of observations, or may be reduced to reflect the fact that some of the observations are effectively being ignored. By default, n is adjusted, but the `adjust.n` argument may be set to

`FALSE`. If n is not adjusted, for example for Moran's I, the absolute value of the statistic will increase, and the absolute value of its expectation and variance will decrease. When measures of autocorrelation were developed, it was generally assumed that all entities would have neighbours, so what one should do when some do not, is not obvious. The problem is not dissimilar to the choice of variogram bin widths and weights in geostatistics (Sect. 8.4.3).

We have already used the New York state eight-county census tract data set for examining the construction of neighbour lists and spatial weights. Now we introduce the data themselves, based on Waller and Gotway (2004, pp. 98, 345–353). There are 281 census tract observations, including as we have seen sparsely populated rural areas contrasting with dense, small, urban tracts. The numbers of incident leukaemia cases are recorded by tract, aggregated from census block groups, but because some cases could not be placed, they were added proportionally to other block groups, leading to non-integer counts. The counts are for the five years 1978–1982, while census variables, such as the tract population, are just for 1980. Other census variables are the percentage aged over 65, and the percentage of the population owning their own home. Exposure to TCE waste sites is represented as the logarithm of 100 times the inverse of the distance from the tract centroid to the nearest site. We return to these covariates in the next chapter.

The first example is of testing the number of cases by census tract (following Waller and Gotway (2004, p. 231)) for autocorrelation using the default spatial weights style of row standardisation, and using the analytical randomisation assumption in computing the variance of the statistic. The outcome, as we see, is that the spatial patterning of the variable of interest is significant, with neighbouring tracts very likely to have similar values for whatever reason.

```
> moran.test(NY8$Cases, listw = nb2listw(NY_nb))

	Moran's I test under randomisation

data:  NY8$Cases
weights: nb2listw(NY_nb)

Moran I statistic standard deviate = 3.978, p-value = 3.477e-05
alternative hypothesis: greater
sample estimates:
Moran I statistic       Expectation          Variance
         0.146883         -0.003571          0.001431
```

Changing the style of the spatial weights to make all weights equal and summing to the number of observations, we see that the resulting probability value is reduced about 20 times – we recall that row-standardisation favours observations with few neighbours, and that styles `'B'`, `'C'`, and `'U'` 'weight up' observations with many neighbours. In this case, style `'S'` comes down between `'C'` and `'W'`.

```
> lw_B <- nb2listw(NY_nb, style = "B")
> moran.test(NY8$Cases, listw = lw_B)

	Moran's I test under randomisation

data:  NY8$Cases
weights: lw_B

Moran I statistic standard deviate = 3.186, p-value = 0.0007207
alternative hypothesis: greater
sample estimates:
Moran I statistic       Expectation          Variance
         0.110387         -0.003571          0.001279
```

By default, `moran.test` uses the randomisation assumption, which differs from the simpler normality assumption by introducing a correction term based on the kurtosis of the variable of interest (here 3.63). When the kurtosis value corresponds to that of a normally distributed variable, the two assumptions yield the same variance, but as the variable departs from normality, the randomisation assumption compensates by increasing the variance and decreasing the standard deviate. In this case, there is little difference and the two return similar outcomes.

```
> moran.test(NY8$Cases, listw = lw_B, randomisation = FALSE)

	Moran's I test under normality

data:  NY8$Cases
weights: lw_B

Moran I statistic standard deviate = 3.183, p-value = 0.0007301
alternative hypothesis: greater
sample estimates:
Moran I statistic       Expectation          Variance
         0.110387         -0.003571          0.001282
```

It is useful to show here that the standard test under normality is in fact the same test as the Moran test for regression residuals for the model, including only the intercept. Making this connection here shows that we could introduce additional variables on the right-hand side of our model, over and above the intercept, and potentially other ways of handling misspecification.

```
> lm.morantest(lm(Cases ~ 1, NY8), listw = lw_B)

	Global Moran's I for regression residuals

data:
model: lm(formula = Cases ~ 1, data = NY8)
weights: lw_B
```

```
Moran I statistic standard deviate = 3.183, p-value = 0.0007301
alternative hypothesis: greater
sample estimates:
Observed Moran's I        Expectation           Variance
          0.110387          -0.003571           0.001282
```

Using the same construction, we can also use a Saddlepoint approximation rather than the analytical normal assumption (Tiefelsdorf, 2002), and an exact test (Tiefelsdorf, 1998, 2000; Hepple, 1998; Bivand et al., 2008). These methods are substantially more demanding computationally, and were originally regarded as impractical. For moderately sized data sets such as the one we are using, however, need less than double the time required for reaching a result. In general, exact and Saddlepoint methods make little difference to outcomes for global tests when the number of spatial entities is not small, as here, with the probability value only changing by a factor of two. We see later that the impact of differences between the normality assumption and the Saddlepoint approximation and exact test is stronger for local indicators of spatial association.

```
> lm.morantest.sad(lm(Cases ~ 1, NY8), listw = lw_B)

    Saddlepoint approximation for global Moran's I
    (Barndorff-Nielsen formula)

data:
model:lm(formula = Cases ~ 1, data = NY8)
weights: lw_B

Saddlepoint approximation = 2.993, p-value = 0.001382
alternative hypothesis: greater
sample estimates:
Observed Moran's I
            0.1104

> lm.morantest.exact(lm(Cases ~ 1, NY8), listw = lw_B)

    Global Moran's I statistic with exact p-value

data:
model:lm(formula = Cases ~ 1, data = NY8)
weights: lw_B

Exact standard deviate = 2.992, p-value = 0.001384
alternative hypothesis: greater
sample estimates:
[1] 0.1104
```

We can also use a Monte Carlo test, a permutation bootstrap test, in which the observed values are randomly assigned to tracts, and the statistic of interest computed `nsim` times. Since we have enough observations in the global case, we can repeat this permutation potentially very many times without repetition.

```
> set.seed(1234)
> bperm <- moran.mc(NY8$Cases, listw = lw_B, nsim = 999)
> bperm

	Monte-Carlo simulation of Moran's I

data:  NY8$Cases
weights: lw_B
number of simulations + 1: 1000

statistic = 0.1104, observed rank = 998, p-value = 0.002
alternative hypothesis: greater
```

Waller and Gotway (2004, p. 231) also include a Poisson constant risk parametric bootstrap assessment of the significance of autocorrelation in the case counts. The constant global rate `r` is calculated first, and used to create expected counts for each census tract by multiplying by the population.

```
> r <- sum(NY8$Cases)/sum(NY8$POP8)
> rni <- r * NY8$POP8
> CR <- function(var, mle) rpois(length(var), lambda = mle)
> MoranI.pboot <- function(var, i, listw, n, S0, ...) {
+     return(moran(x = var, listw = listw, n = n, S0 = S0)$I)
+ }
> set.seed(1234)

> boot2 <- boot(NY8$Cases, statistic = MoranI.pboot,
+     R = 999, sim = "parametric", ran.gen = CR,
+     listw = lw_B, n = length(NY8$Cases), S0 = Szero(lw_B),
+     mle = rni)

> pnorm((boot2$t0 - mean(boot2$t))/sd(boot2$t), lower.tail = FALSE)

[1] 0.1472
```

The expected counts can also be expressed as the fitted values of a null Poisson regression with an offset set to the logarithm of tract population – with a log-link, this shows the relationship to generalised linear models (because `Cases` are not all integer, warnings are generated):

```
> rni <- fitted(glm(Cases ~ 1 + offset(log(POP8)), data = NY8,
+     family = "poisson"))
```

These expected counts `rni` are fed through to the `lambda` argument to `rpois` to generate the synthetic data sets by sampling from the Poisson distribution. The output probability value is calculated from the same observed Moran's *I*

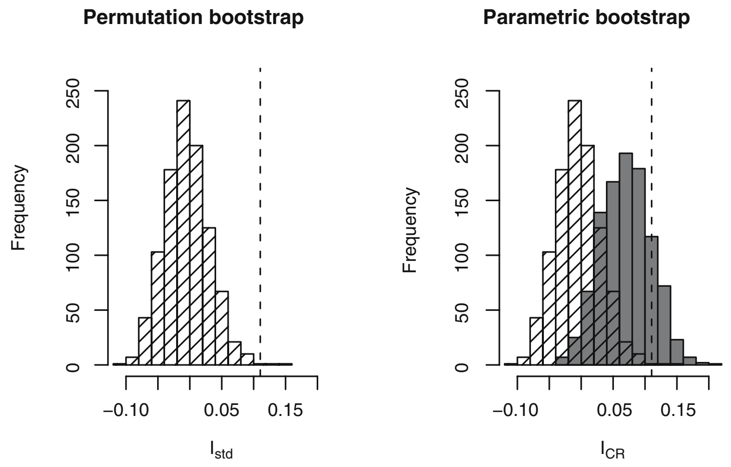

Fig. 9.10. Histograms of simulated values of Moran's I under random permutations of the data and parametric samples from constant risk expected values; the observed values of Moran's I are marked by vertical lines

minus the mean of the simulated I values, and divided by their standard deviation. Figure 9.10 corresponds to Waller and Gotway (2004, p. 232, Fig. 7.8), with the parametric simulations shifting the distribution of Moran's I rightwards, because it is taking the impact of the heterogeneous tract populations into account.

There is a version of Moran's I adapted to use an Empirical Bayes rate by Assunção and Reis (1999) that, unlike the rate results above, shrinks extreme rates for tracts with small populations at risk towards the rate for the area as a whole – it also uses Monte Carlo methods for inference:

```
> set.seed(1234)
> EBImoran.mc(n = NY8$Cases, x = NY8$POP8, listw = nb2listw(NY_nb,
+     style = "B"), nsim = 999)

    Monte-Carlo simulation of Empirical Bayes Index

data:  cases: NY8$Cases, risk population: NY8$POP8
weights: nb2listw(NY_nb, style = "B")
number of simulations + 1: 1000

statistic = 0.0735, observed rank = 980, p-value = 0.02
alternative hypothesis: greater
```

The results for the Empirical Bayes rates suggest that one reason for the lack of significance of the parametric bootstrapping of the constant risk observed and expected values could be that unusual and extreme values were observed in tracts with small populations. Once the rates have been smoothed, some global autocorrelation is found.

```
> cor8 <- sp.correlogram(neighbours = NY_nb, var = NY8$Cases,
+     order = 8, method = "I", style = "C")

> print(cor8, p.adj.method = "holm")

Spatial correlogram for NY8$Cases
method: Moran's I
   estimate expectation  variance standard deviate Pr(I) two sided
1  0.110387   -0.003571  0.001279             3.19         0.01009 *
2  0.095113   -0.003571  0.000564             4.16         0.00026 ***
3  0.016711   -0.003571  0.000348             1.09         0.83111
4  0.037506   -0.003571  0.000255             2.57         0.06104 .
5  0.026920   -0.003571  0.000203             2.14         0.12960
6  0.026428   -0.003571  0.000175             2.27         0.11668
7  0.009341   -0.003571  0.000172             0.98         0.83111
8  0.002119   -0.003571  0.000197             0.41         0.83111
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Another approach is to plot and tabulate values of a measure of spatial autocorrelation for higher orders of neighbours or bands of more distant neighbours where the spatial entities are points. The **spdep** package provides the first type as a wrapper to `nblag` and `moran.test`, so that here the first-order contiguous neighbours we have used until now are 'stepped out' to the required number of orders. Figure 9.11 shows the output plot in the left panel, and suggests that second-order neighbours are also positively autocorrelated (although the probability values should be adjusted for multiple comparisons).

The right panel in Fig. 9.11 presents the output of the `correlog` function in the **pgirmess** package by Patrick Giraudoux; the function is a wrapper for `dnearneigh` and `moran.test`. The function automatically selects distance bands of almost 10 km, spanning the whole study area. In this case, the first two bands of 0–10 and 10–20 km have significant values.

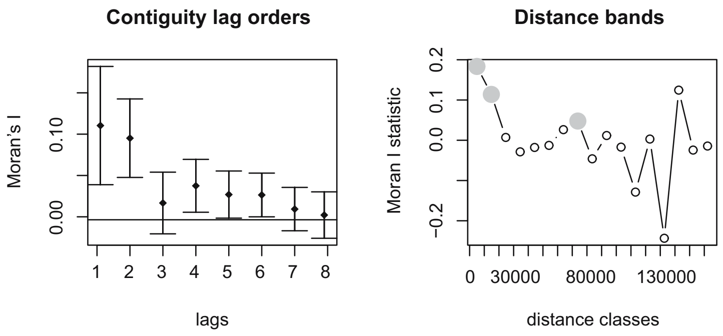

Fig. 9.11. Correlograms: (*left*) values of Moran's I for eight successive lag orders of contiguous neighbours; (*right*) values of Moran's I for a sequence of distance band neighbour pairs

```
> library(pgirmess)
> corD <- correlog(coordinates(NY8), NY8$Cases, method = "Moran")

> corD

Moran I statistic
      dist.class      coef   p.value     n
 [1,]       4996  0.183373 3.781e-09 10728
 [2,]      14822  0.113940 1.128e-08  9248
 [3,]      24649  0.007028 3.025e-01  5718
 [4,]      34475 -0.028733 8.968e-01  5376
 [5,]      44302 -0.017871 7.686e-01  5342
 [6,]      54128 -0.012729 6.940e-01  5578
 [7,]      63954  0.026370 5.040e-02  5524
 [8,]      73781  0.047751 3.620e-03  5976
 [9,]      83607 -0.046052 9.730e-01  4334
[10,]      93434  0.011773 2.632e-01  3862
[11,]     103260 -0.017014 7.074e-01  8756
[12,]     113086 -0.128775 1.000e+00  5816
[13,]     122913  0.003007 4.246e-01  1958
[14,]     132739 -0.243455 9.987e-01   320
[15,]     142566  0.124519 8.951e-02    92
[16,]     152392 -0.024368 4.435e-01    44
[17,]     162218 -0.014310 1.495e-01     6
```

9.4.2 Local Tests

Global tests for spatial autocorrelation are calculated from the local relationships between the values observed at a spatial entity and its neighbours, for the neighbour definition chosen. Because of this, we can break global measures down into their components, and by extension, construct localised tests intended to detect 'clusters' – observations with very similar neighbours – and 'hotspots' – observations with very different neighbours. These are discussed briefly by Schabenberger and Gotway (2005, pp. 23–25) and O'Sullivan and Unwin (2003, pp. 203–205), and at greater length by Waller and Gotway (2004, pp. 236–242) and Fortin and Dale (2005, pp. 153–159). They are covered in some detail by Lloyd (2007, pp. 65–70) in a book concentrating on local models.

First, let us examine a Moran scatterplot of the leukaemia case count variable. The plot (shown in Fig. 9.12) by convention places the variable of interest on the x-axis, and the spatially weighted sum of values of neighbours – the spatially lagged values – on the y-axis. Global Moran's I is a linear relationship between these and is drawn as a slope. The plot is further partitioned into quadrants at the mean values of the variable and its lagged values: low–low, low–high, high–low, and high–high.

```
> moran.plot(NY8$Cases, listw = nb2listw(NY_nb, style = "C"))
```

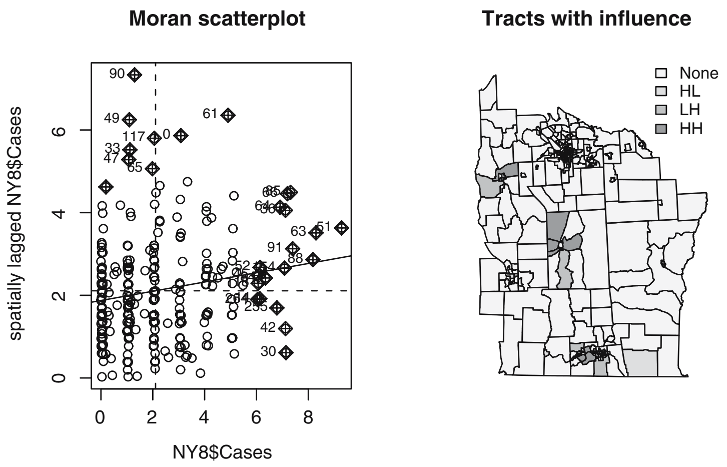

Fig. 9.12. (*Left*) Moran scatterplot of leukaemia incidence; (*right*) tracts with influence by Moran scatterplot quadrant

Since global Moran's I is, like similar correlation coefficients, a linear relationship, we can also apply standard techniques for detecting observations with unusually strong influence on the slope. Specifically, `moran.plot` calls `influence.measures` on the linear model of `lm(wx` $\sim$ `x)` providing the slope coefficient, where `wx` is the spatially lagged value of `x`. This means that we can see whether particular local relationships are able to influence the slope more than proportionally. The map in the right panel of Fig. 9.12 shows tracts with significant influence (using standard criteria) coded by their quadrant in the Moran scatterplot.

Local Moran's I_i values are constructed as the n components summed to reach global Moran's I:

$$I_i = \frac{(y_i - \bar{y}) \sum_{j=1}^{n} w_{ij}(y_j - \bar{y})}{\frac{\sum_{i=1}^{n}(y_i - \bar{y})^2}{n}},$$

where once again we assume that the global mean $\bar{y}$ is an adequate representation of the variable of interest y. The two components in the numerator, $(y_i - \bar{y})$ and $\sum_{j=1}^{n} w_{ij}(y_j - \bar{y})$, appear without centring in the Moran scatterplot.

As with the global statistic, the local statistics can be tested for divergence from expected values, under assumptions of normality, and randomisation analytically, and using Saddlepoint approximations and exact methods. The two latter methods can be of importance because the number of neighbours of each observation is very small, and this in turn may make the adoption of the normality assumption problematic. Using numerical methods, which would previously have been considered demanding, the Saddlepoint approximation or exact local probability values can be found in well under 10 s, about

20 times slower than probability values based on normality or randomisation assumptions, for this moderately sized data set.

Trying to detect residual local patterning in the presence of global spatial autocorrelation is difficult. For this reason, results for local dependence are not to be seen as 'absolute', but are conditioned at least by global spatial autocorrelation, and more generally by the possible influence of spatial data generating processes at a range of scales from global through local to dependence not detected at the scale of the observations.

```
> lm1 <- localmoran(NY8$Cases, listw = nb2listw(NY_nb,
+     style = "C"))
> lm2 <- as.data.frame(localmoran.sad(lm(Cases ~ 1, NY8),
+     nb = NY_nb, style = "C"))
> lm3 <- as.data.frame(localmoran.exact(lm(Cases ~ 1, NY8),
+     nb = NY_nb, style = "C"))
```

Waller and Gotway (2004, p. 239) extend their constant risk hypothesis treatment to local Moran's I_i, and we can follow their lead:

```
> r <- sum(NY8$Cases)/sum(NY8$POP8)
> rni <- r * NY8$POP8
> lw <- nb2listw(NY_nb, style = "C")
> sdCR <- (NY8$Cases - rni)/sqrt(rni)
> wsdCR <- lag(lw, sdCR)
> I_CR <- sdCR * wsdCR
```

Figure 9.13 shows the two sets of values of local Moran's I_i, calculated in the standard way and using the Poisson assumption for the constant risk hypothesis. We already know that global Moran's I can vary in value and in

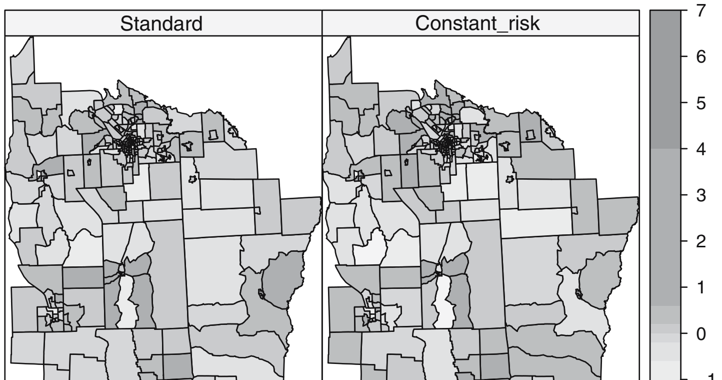

Fig. 9.13. Local Moran's I_i values calculated directly and using the constant risk hypothesis

inference depending on our assumptions – for example that inference should take deviations from our distributional assumptions into account. The same applies here to the assumption for the Poisson distribution that its mean and standard deviation are equal, whereas over-dispersion seems to be a problem in data also displaying autocorrelation. There are some sign changes between the maps, with the constant risk hypothesis values somewhat farther from zero.

We can also construct a simple Monte Carlo test of the constant risk hypothesis local Moran's I_i values, simulating very much as in the global case, but now retaining all of the local results. Once the simulation is completed, we extract the rank of the observed constant risk local Moran's I_i value for each tract, and calculate its probability value for the number of simulations made. We use a parametric approach to simulating the local counts using the local expected count as the parameter to `rpois`, because the neighbour counts are very low and make permutation unwise. Carrying out permutation testing using the whole data set also seems unwise, because we would then be comparing like with unlike.

```
> set.seed(1234)
> nsim <- 999
> N <- length(rni)
> sims <- matrix(0, ncol = nsim, nrow = N)
> for (i in 1:nsim) {
+     y <- rpois(N, lambda = rni)
+     sdCRi <- (y - rni)/sqrt(rni)
+     wsdCRi <- lag(lw, sdCRi)
+     sims[, i] <- sdCRi * wsdCRi
+ }
> xrank <- apply(cbind(I_CR, sims), 1, function(x) rank(x)[1])
> diff <- nsim - xrank
> diff <- ifelse(diff > 0, diff, 0)
> pval <- punif((diff + 1)/(nsim + 1))
```

The probability values shown in Fig. 9.14 are in general very similar to each other. We follow Waller and Gotway (2004) in not adjusting for multiple comparisons, and will consequently not interpret the probability values as more than indications. Values close to zero are said to indicate clusters in the data where tracts with similar values neighbour each other (positive local autocorrelation and a one-sided test). Values close to unity indicate hotspots where the values of contiguous tracts differ more than might be expected (negative local autocorrelation and a one-sided test). Of course, finding clusters or hotspots also needs to be qualified by concerns about misspecification in the underlying model of the data generation process.

Finally, we zoom in to examine the local Moran's I_i probability values for three calculation methods for the tracts in and near the city of Binghampton (Fig. 9.15). It appears that the use of the constant risk approach handles the heterogeneity in the counts better than the alternatives. These results broadly

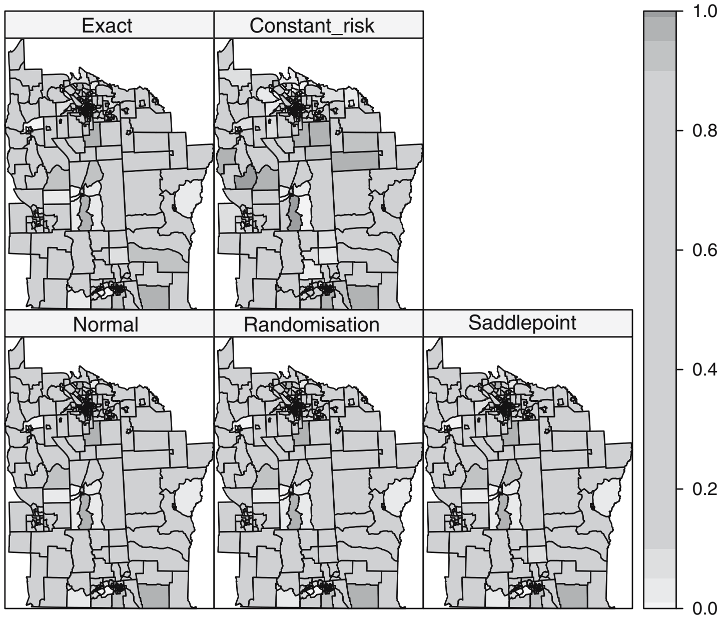

Fig. 9.14. Probability values for all census tracts, local Moran's I_i: normality and randomisation assumptions, Saddlepoint approximation, exact values, and constant risk hypothesis

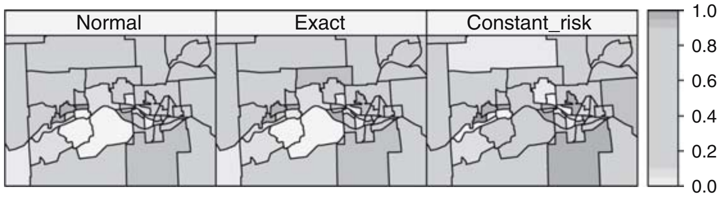

Fig. 9.15. Probability values for census tracts in and near the city of Binghampton, local Moran's I_i: normality assumption, exact values, and constant risk hypothesis

agree with those reached by Waller and Gotway (2004, p. 241), but we note that our underlying model is very simplistic. Finding spatial autocorrelation is not a goal in itself, be it local or global, but rather just one step in a process leading to a proper model. It is to this task that we now turn.