Analysing Spatial Data in R: Worked example: spatial autocorrelation

Roger Bivand

Department of Economics
Norwegian School of Economics and Business Administration
Bergen, Norway

31 August 2007

Areal data

- Spatial data often take the form of polygon entities defined by boundaries for which observations are available
- The observed data are frequently aggregations within the boundaries, such as population counts
- Exceptionally, the areal entities themselves constitute the units of observation, for example when studying local government behaviour where decisions, such as local taxes, are taken at the level of the entity
- By and large, though, areal entities are aggregates used to tally measurements, like voting results at polling stations

Spatial autocorrelation

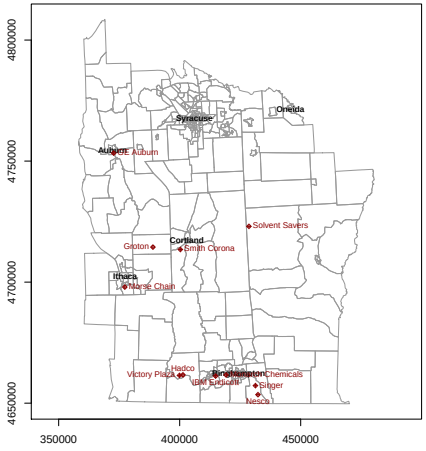
- A wide range of scientific disciplines have encountered spatial autocorrelation among areal entities, with the term "Galton's problem" used in several
- In his exchange with Tyler in 1889, Galton questioned whether observations of marriage laws across areal entities constituted independent observations, 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
- Here we will be concerned with areal entities that are defined as neighbours, for chosen definitions of neighbours; we will use the **spdep** package

Reading the NY data

```
> library(rgdal)
> NY8 <- readOGR(".", "NY8_utm18")

OGR data source with driver: ESRI Shapefile
Source: ".", layer: "NY8_utm18"
with  281  rows and  17  columns

> TCE <- readOGR(".", "TCE")

OGR data source with driver: ESRI Shapefile
Source: ".", layer: "TCE"
with  11  rows and  5  columns

> cities <- readOGR(".", "NY8cities")

OGR data source with driver: ESRI Shapefile
Source: ".", layer: "NY8cities"
with  6  rows and  1  columns
```

The 281 census tract data set for 8 central New York State counties featured prominently in Waller & Gotway (2004) will be used in the examples, supplemented with tract boundaries derived from TIGER 1992 and distributed by SEDAC/CIESIN; the boundaries have been projected from geographical coordinates to UTM zone 18

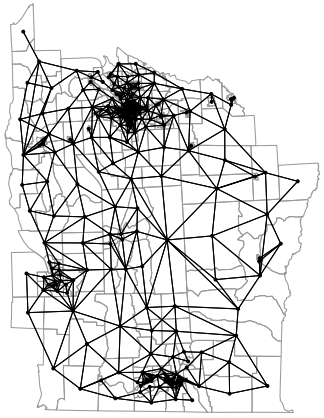
New York State study area



The data set includes adjusted counts of leukemia 1978–1982, and observations on a number of other variables, including distances to 11 hazardous waste sites

For comparison, we'll read in the contiguity neighbours defined by Waller & Gotway for their 281 census tracts:

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



Spatial weights

- Creating spatial weights is a necessary step in using areal data, perhaps just to check that there is no remaining spatial patterning in residuals
- Making the weights is, however, not easy to do, 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 first step is to determine the sets of neighbours for each observation, the second to assign weights to each neighbour relationship

- In the **spdep** package, neighbour relationships between n observations are represented by an object of class `nb`
- This 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 indentifiers, and a logical value indicating whether the relationships are symmetric

Subsetting to Syracuse City

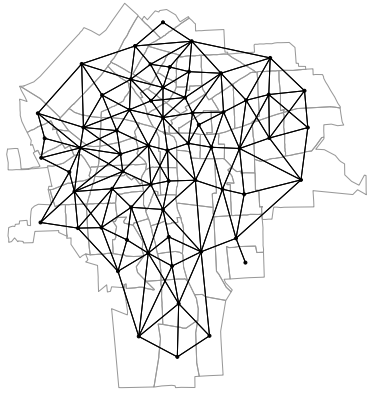
The `nb` objects are old-style class objects, unlike the `Spatial*` objects presented earlier. They have a number of methods, including `print`, `plot`, `summary`, and `subset` methods. We'll return to the full dataset later, but for our present purposes, the census tracts in Syracuse City will be enough:

```
> 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
```

Contiguity queen neighbours

We begin by reproducing the contiguity neighbours used by Waller & Gotway; tracts sharing boundary points are taken as neighbours, using the `poly2nb` function, which accepts a `SpatialPolygonsDataFrame` as its (first) argument:

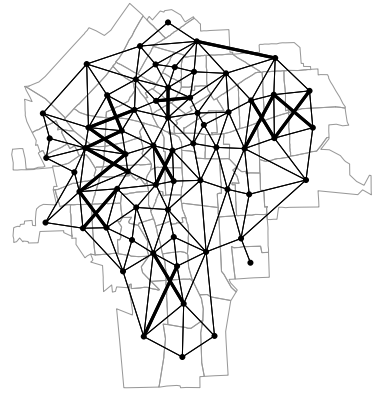
```
> Sy1_nb <- poly2nb(Syracuse)
> isTRUE(all.equal(Sy0_nb, Sy1_nb,
+     check.attributes = FALSE))

[1] TRUE
```



Contiguity rook neighbours

If contiguity is defined as tracts sharing more than one boundary point, the `queen=` argument is set to `FALSE`, and some differences from the book neighbours arise, marked by thicker lines:

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

[1] FALSE
```

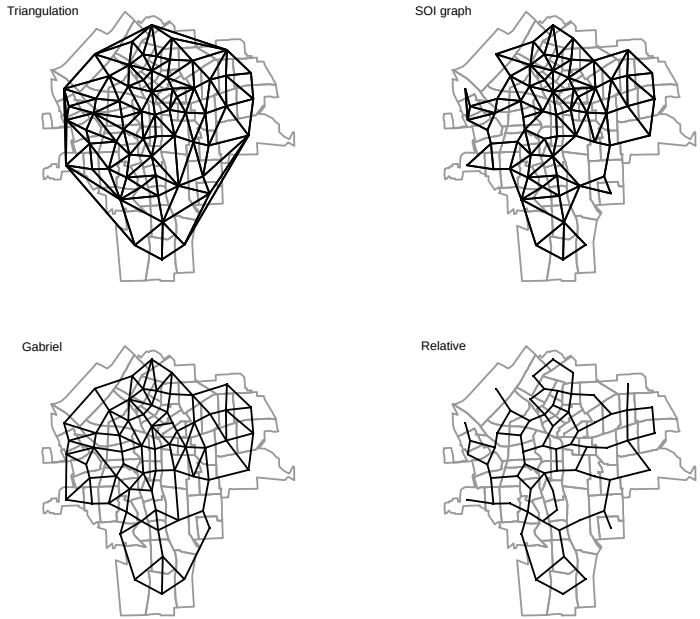


Graph neighbours

Neighbours can be defined using points to represent areas in a number of ways. First graph-based neighbours, starting with a Delaunay triangulation of the tract centroids:

```
> 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)
> 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
```

Graph neighbours



K-nearest neighbours

K-nearest neighbours

K-nearest neighbours are also typically not symmetrical, but can be, if required, constructed for 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
```

Distance-based neighbours

Distance-based neighbours are defined as points representing areas within distance bands — a problem is that unless the defining distance threshold is greater than the maximum first nearest neighbour distance, some observations will not have any neighbours:

```
> 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
```

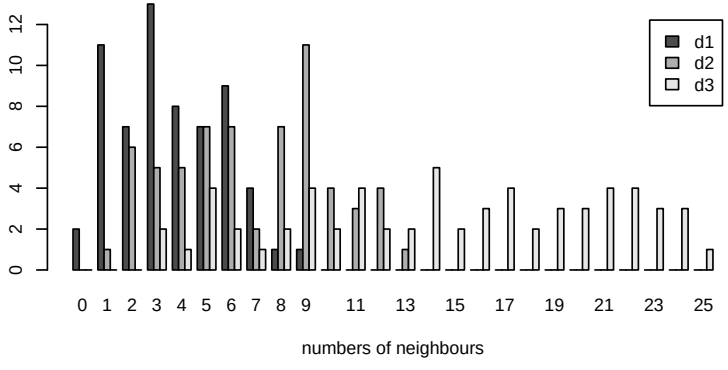
Distance-based neighbours

Higher order neighbours

If we say that a second order neighbour is a first order neighbour of a first order neighbour, and its itself not the tract in question nor already a lower order neighbour of that tract, we can define higher order neighbours. We'll take higher order neighbours over the queen-based contiguity neighbours for Syracuse, using `nblag`, which returns a list of `nb` objects, here up to ninth order:

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

Distance-based neighbours



Higher order neighbours

The largest numbers of neighbours are found for the third order, and by the ninth order, no tract has any neighbours left:

	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

Grid neighbours

If the data are in a full grid, `cell2nb` can be used to generate a neighbours list, on a torus if required; this form is most useful for testing statistics as it is the most neutral form of tesselated field with no edges:

```
> 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
```

Spatial weights styles

- Spatial weights are in part difficult to use because the identified neighbour relationships need to be given a value
- Even when the values are binary 0/1, the issue of what to do with no-neighbour observations arises
- Then the weights can be standardised, for example so that row sums are unity, or that the weights sum to the number of observations
- Finally, the weights may be generalised to reflect something that we "know" about the spatial dependency process, for example that it falls with increasing distance

Grid neighbours

In other settings, we can readily generate neighbour lists from the value of the `cellsize` slot of the `GridTopology` object included in `SpatialGrid*` and `SpatialPixels*` objects:

```
> 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
```

Row-standardised weights

```
> 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
```

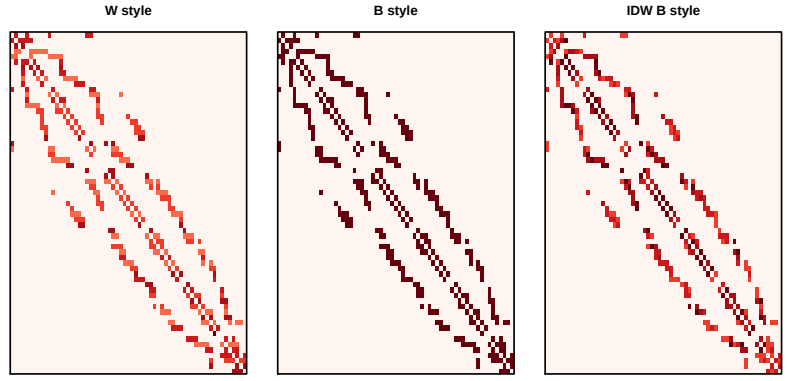
Row-standardised weights are the default style in the `nb2listw` function used for converting neighbour lists into spatial weights objects (made up of a neighbours list, a corresponding weights list, and a style); here, the row sums of the weights are unity

Binary weights

```
> Sy0_lw_B <- nb2listw(Sy0_nb,
+     style = "B")
> Sy0_lw_B

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: B
Weights constants summary:
   n   nn  S0  S1   S2
B 63 3969 346 692 8248
```

Row-standardised weights increase the influence of links from observations with few neighbours, while binary weights vary the influence of observations — those with many neighbours are up-weighted compared to those with few

Images of weights



General binary weights

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

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: B
Weights constants summary:
   n   nn       S0       S1
B 63 3969 344.7038 778.2078
        S2
B 8465.642
```

General weights use the `glist=` argument to use extra a-priori information about the relationships, here that influence is inversely proportional to distance

No-neighbour weights

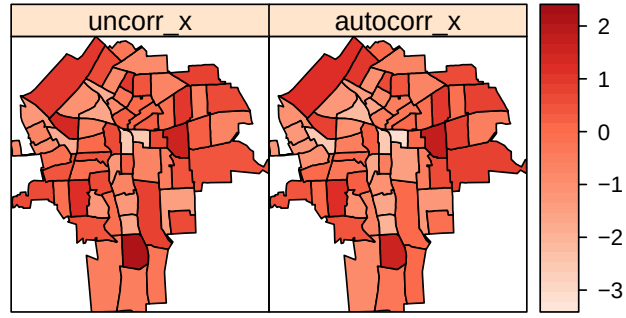
```
> Sy0_lw_D1 <- nb2listw(Sy11_nb,
+     style = "B")

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
```

Recalling that our first set of distance-based neighbours left some observations without neighbours, we see that this has to be declared every time the neighbours or weights are used, with the `zero.policy=` argument

Importing and exporting; convenience functions

- We have already seen the `read.gal` function used to import GAL format neighbours lists; the can be exported with a matching `write.nb.gal` function, and GWT format files can also be handled
- Similar import/export functions are provided for the Matlab Spatial Econometrics Library, which uses a three-column sparse representation of spatial weights that can also make exchange with the S–PLUS SpatialStats module possible
- There is a similar `listw2WB` for writing weights for WinBUGS
- The convenience functions `listw2mat` and `mat2listw` can be used to move to and from a matrix representation, and also to output weights for the Stata spatwmat command

Uncorrelated and autocorrelated maps

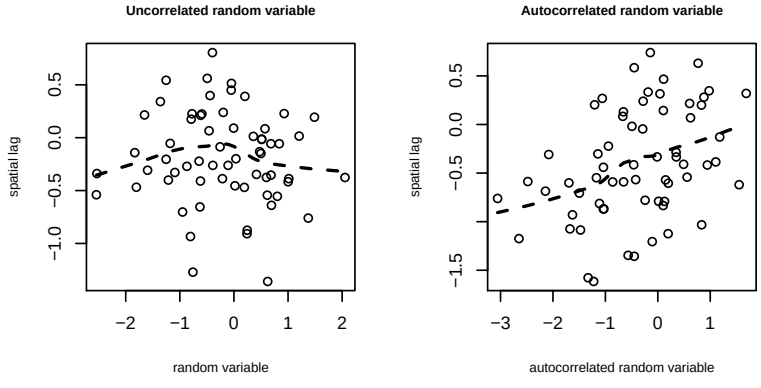


Using a weights list as a dense matrix

```
> set.seed(987654)
> n <- length(Sy0_nb)
> uncorr_x <- rnorm(n)
> cor(uncorr_x, lag(Sy0_lw_W,
+     uncorr_x))

[1] -0.07328062

> rho <- 0.5
> IrW <- diag(n) - rho * listw2mat(Sy0_lw_W)
> SARcov <- solve(t(IrW) %*% IrW)
> SARcovL <- chol((SARcov + t(SARcov))/2)
> autocorr_x <- t(SARcovL) %*%
+     uncorr_x
> cor(autocorr_x, lag(Sy0_lw_W,
+     autocorr_x))

          [,1]
[1,] 0.3863454
```

This example using `listw2mat` simulates spatial autocorrelation in the map of Syracuse using a SAR covariance matrix, with identity observation weights; this approach (see Haining 1990) differs from that used by Cliff and Ord (1973)

Autocorrelation (Moran) scatterplots



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
- We begin with the simulated variable for the Syracuse census tracts
- Since the input variable is known to be drawn at random, 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*

Moran's *I*

We'll start using the `moran.test` function, which takes a variable of interest and a `listw` object as its required arguments; it returns an `htest` object, and uses by default the randomisation assumption. The third case test using a different `listw` object than the one used to introduce dependency. The final pair of tests show how Moran's *I* can detect model misspecification — a gentle trend induces apparent spatial autocorrelation, unmasked when the correct model is fitted and tested using the `lm.morantest` function for model residuals:

```
> 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"))
> 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)
```

Moran's *I*

Moran's *I* 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 i-th 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. Centering on the mean is equivalent to asserting that the correct model has a constant mean, and that any remaining patterning after centering is caused by the spatial relationships encoded in the spatial weights

Moran's *I*

The table shows that the correctly specified test rejects the null hypothesis of no spatial autocorrelation clearly, but with the wrong spatial weights, the outcome is borderline. When apparent dependence is introduced by the spatial trend, the test reports spatial autocorrelation, so it is worth remembering that the test is sensitive to misspecification in general:

	I	*E*(*I*)	*var*(*I*)	st. deviate	p-value
uncorr_x	-0.03329	-0.01613	0.00571	-0.227	0.59
autocorr_x	0.21260	-0.01613	0.00572	3.02	0.0012
autocorr_x k=1	0.1590	-0.0161	0.0125	1.56	0.059
trend_x	0.23747	-0.01613	0.00575	3.34	0.00041
lm(trend_x ẽt)	-0.0538	-0.0309	0.0054	-0.312	0.62

Global tests

- Global tests are tests that test the whole study area for spatial autocorrelation, assuming the spatial process is the same everywhere
- Moran's *I* and Geary's *C* are established tests for continuous variables, but there is also a global *G* test and a Mantel test, which generalises the others
- They are implemented as `moran.test`, `geary.test` and `globalG.test`, with permutation bootstrap variants: `moran.mc`, `geary.mc` and `sp.mantel.mc`.
- There are also join coint tests for same colour joins: `joincount.test` and `joincount.mc`, and a multi-way join count test `joincount.multi`

Global tests in practice

The differences in the results above suggests that we are measuring the wrong things, and that we need to be careful. We could try the incidence proportion, but the Empirical Bayes *I* tells yet another story:

```
> Ip <- NY8$Cases/NY8$POP8
> set.seed(987)
> moran.mc(Ip, listw = lw, nsim = 999)

Monte-Carlo simulation of Moran's I

data:  Ip
weights: lw
number of simulations + 1: 1000

statistic = 0.0399, observed rank = 896, p-value = 0.104
alternative hypothesis: greater

> EBImoran.mc(n = NY8$Cases, x = NY8$POP8, listw = lw, nsim = 999)

Monte-Carlo simulation of Empirical Bayes Index

data:  cases: NY8$Cases, risk population: NY8$POP8
weights: lw
number of simulations + 1: 1000

statistic = 0.0735, observed rank = 981, p-value = 0.019
alternative hypothesis: greater
```

Global tests

The tests return `"htest"` objects, like other hypothesis tests in R. We can also try to reproduce Waller & Gotway's results for parametric bootstrapping from independent Poisson counts, reaching a similar result:

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

Moran's I test under normality

data:  NY8$Cases
weights: lw

Moran I statistic standard deviate = 3.1825, p-value = 0.0007301
alternative hypothesis: greater
sample estimates:
Moran I statistic       Expectation          Variance
      0.110387402      -0.003571429       0.001282228

> 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)$I)
+ }
> r <- sum(NY8$Cases)/sum(NY8$POP8)
> rni <- r * NY8$POP8
> set.seed(1)
> boot2 <- boot(NY8$Cases, statistic = MoranI.pboot, R = 999, sim = "parametric",
+     ran.gen = CR, mle = rni, listw = lw, n = length(NY8$Cases),
+     S0 = Szero(lw))
> pnorm((boot2$t0 - mean(boot2$t))/sd(boot2$t), lower.tail = FALSE)

[1] 0.1467508
```

Global tests in practice

As Waller & Gotway show later on, the incidence proportion is affected by three outliers, so a transformation is called for; note that Moran's *I* under the normality assumption is the same as the Moran test of regression residuals for a model with only the constant term:

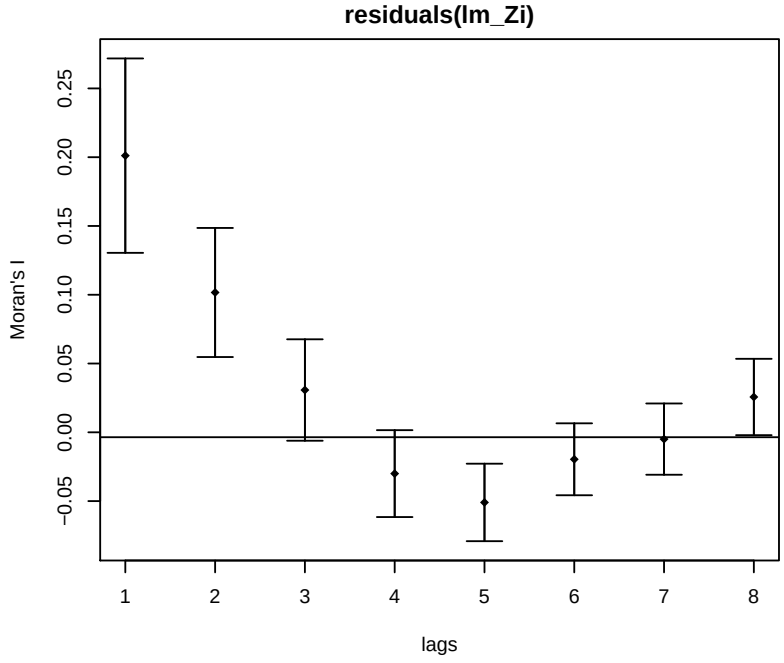
```
> NY8$Zi <- log((1000 * (NY8$Cases + 1))/NY8$POP8)
> moran.test(NY8$Zi, listw = lw, randomisation = FALSE)

Moran I statistic standard deviate = 5.7167, p-value = 5.431e-09
alternative hypothesis: greater
sample estimates:
Moran I statistic       Expectation          Variance
      0.201133022      -0.003571429       0.001282228

> lm.morantest(lm(Zi ~ 1, data = NY8), listw = lw)

Moran I statistic standard deviate = 5.7167, p-value = 5.431e-09
alternative hypothesis: greater
sample estimates:
Observed Moran's I        Expectation           Variance
       0.201133022       -0.003571429        0.001282228
```

Global tests in practice

Since we are now using regression, we can follow Waller & Gotway in moving to weighted least squares, to address the issue of heteroskedasticity in the variance of the Z_i depending on the population sizes of the tracts:

```
> lm_Zi <- lm(Zi ~ 1, data = NY8, weights = POP8)
> lm.morantest(lm_Zi, listw = lw)

Global Moran's I for regression residuals

data:
model: lm(formula = Zi ~ 1, data = NY8, weights = POP8)
weights: lw

Moran I statistic standard deviate = 3.1031, p-value = 0.0009574
alternative hypothesis: greater
sample estimates:
Observed Moran's I        Expectation           Variance
       0.107765109       -0.003390201        0.001283092
```

Moran correlogram



Moran correlogram

Taking an ad-hoc approach, we can "step out" across the graph of neighbours to higher-order lags as described earlier. This also lets us calculate a Moran correlogram — here with an inappropriate standard deviate. The function has print and plot methods:

```
> res <- sp.correlogram(NY_nb, residuals(lm_Zi), order = 8, method = "I",
+     style = "B")
> res

Spatial correlogram for residuals(lm_Zi)
method: Moran's I
     estimate expectation    variance standard deviate Pr(I) two sided
1  0.20113302 -0.00357143  0.00124837           5.7937       6.885e-09 ***
2  0.10164973 -0.00357143  0.00055031           4.4854       7.279e-06 ***
3  0.03074475 -0.00357143  0.00033988           1.8614        0.062690 .
4 -0.03001318 -0.00357143  0.00024947          -1.6741        0.094111 .
5 -0.05098154 -0.00357143  0.00019836          -3.3662        0.000762 ***
6 -0.01961033 -0.00357143  0.00017087          -1.2270        0.219831
7 -0.00494214 -0.00357143  0.00016781          -0.1058        0.915732
8  0.02567845 -0.00357143  0.00019241           2.1087        0.034974 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Global tests

- Global tests detect a range of misspecifications, including missing variables, and variables with the wrong functional form
- They can also pick up divergences between the data and our distributional assumptions about the data
- There are additional Lagrange Multiplier tests for regression residuals typically used in spatial econometrics
- The analytical results of the tests are often close to simulations and approximations, where available

Local tests

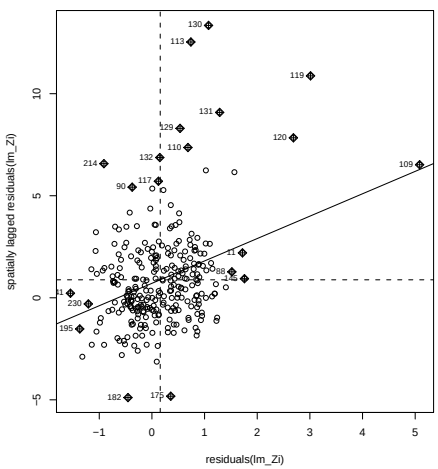
- Local tests also pick up global spatial dependency and other omitted trends, so that finding "hotspots" is a black art; local Moran's *I* and local *G* are available in **spdep**
- One useful step is to look at a Moran scatterplot of the apparent spatial process, especially if the local test to be used is local Moran's *I*
- In addition, it is worth recalling that local indicators of spatial association involve multiple comparisons, "burning up" degrees of freedom
- Finally, should simulations be used to test the "significance" of local indicators, or perhaps other approximations?

Local Moran's *I*

The `localmoran` function takes a variable and a spatial weights list, and can return adjusted p-values. The `localmoran.sad` function uses a local saddlepoint approximation, and can use the `select=` argument to return test results for selected spatial units. Here we will plot comparisons of the local Moran's *I* unadjusted p-values for the count of cases, and for the null transformed and weighted model. Note that `localmoran.sad` can also take a `sarlm` model object — but not yet an `spautolm` model object.

```
> res <- localmoran(x = NY8$Cases, listw = nb2listw(NY_nb, style = "W"))
> colnames(res)

[1] "Ii"        "E.Ii"      "Var.Ii"    "Z.Ii"      "Pr(z > 0)"

> NY8$Cases_locI_R_pv <- res[, 5]
> res0 <- localmoran(NY8$Zi, nb2listw(NY_nb, style = "W"))
> NY8$Zi_locI_R_pv <- res0[, 5]
> res1 <- localmoran.sad(model = lm_Zi, select = 1:length(NY_nb),
+     nb = NY_nb, style = "W")
> res1s <- summary(res1)
> names(res1s)

 [1] "Local Morans I"  "Stand. dev. (N)" "Pr. (N)"         "Saddlepoint"
 [5] "Pr. (Sad)"       "Expectation"     "Variance"        "Skewness"
 [9] "Kurtosis"        "Minimum"         "Maximum"         "omega"
[13] "sad.r"           "sad.u"

> NY8$Zi_locI_N_pv <- res1s[, 3]
> NY8$Zi_locI_S_pv <- res1s[, 5]
```

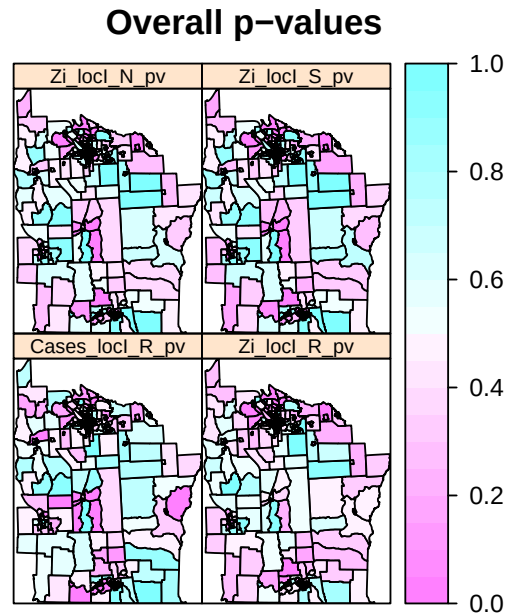
Moran scatterplot



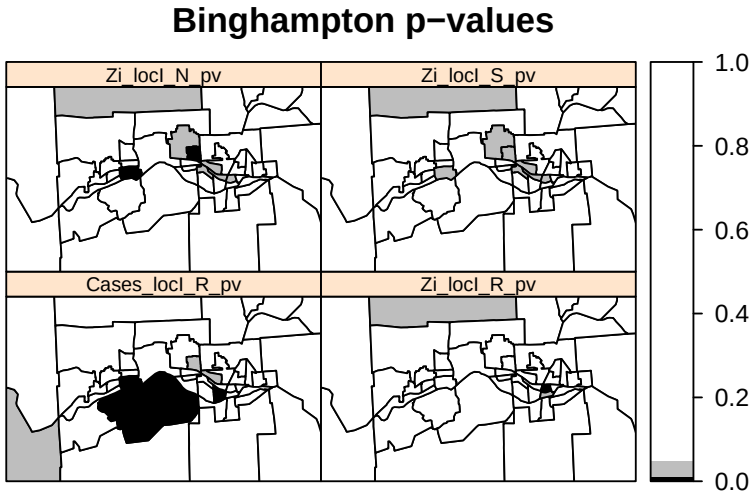
We'll start by looking at a Moran scatterplot of the residuals of the null transformed and weighted model; the standard output also includes a listing of cases exterting unusual influence on the Moran's *I* line, here marked graphically:

```
> moran.plot(residuals(lm_Zi),
+     lw, quiet = TRUE)
```

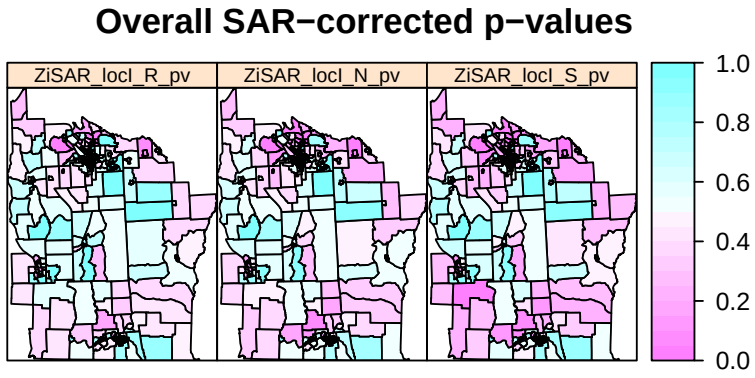
Overall p-values, local Moran's *I*



Binghampton p-values, local Moran's *I*



Overall SAR-corrected p-values



Local Moran's *I*

Note that `localmoran.sad` can also take a `sarlm` model object — but not yet an `spautolm` model object. It is often interesting to see how the analyses behave when global spatial autocorrelation is removed, but there is little guidance about how actual dependence processes interact with model misspecifiation. Here we have not been able to use case weights to control for population size (yet):

```
> ZiSAR <- errorsarlm(Zi ~ 1, NY8, lw)
> res2 <- localmoran(residuals(ZiSAR), nb2listw(NY_nb, style = "W"))
> NY8$ZiSAR_locI_R_pv <- res2[, 5]
> res3 <- localmoran.sad(model = ZiSAR, select = 1:length(NY_nb),
+     nb = NY_nb, style = "W")
> res3s <- summary(res3)
> NY8$ZiSAR_locI_N_pv <- res3s[, 3]
> NY8$ZiSAR_locI_S_pv <- res3s[, 5]
```