Analyzing Spatial Autocorrelation with R

Applied Spatial Econometrics

Lecture 3 (1.5h)

Thibault LAURENT
thibault.laurent@univ-tlse1.fr

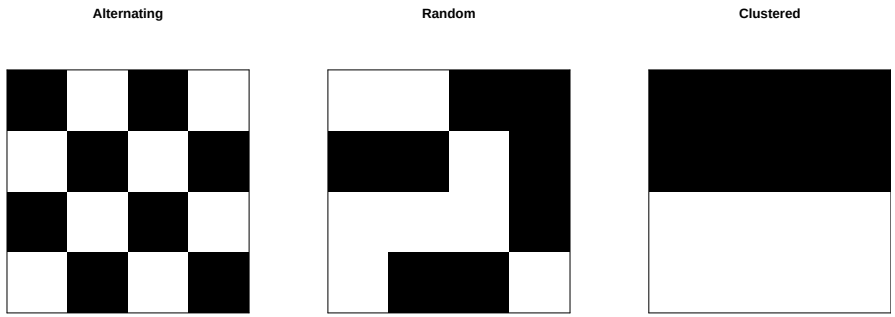
Toulouse School of Economics

17th February of 2016

Introduction

What is spatial autocorrelation?

- Tobler: "Everything is related to everything else, but near things are more related than distant things."
- Spatial autocorrelation helps understand the degree to which one object is similar to other nearby objects.
- Illustration of negative, null and positive spatial autocorrelation.



Spatial autocorrelation + econometrics = Spatial econometrics

- The data observations are not truly independent.
- Models incorporating spatial autocorrelation or neighborhood effects can be estimated using spatial econometric methods.
- Many applications in regional science, real estate economics, and education economics.

Case study used in this lecture

- Harisson and Rubeinfield (1978), Hedonic Housing Prices and the Demand for Clean Air, *Journal of Environmental Economics and Management* (**5**).
- 1st part of the study: Use of a hedonic housing price model and data for the Boston metropolitan area.
- Response variable Y: housing prices (median values of owner-occupied housing in USD 1000).
- Explanatory variables X: housing characteristics (including air pollution).

Preparation of the data

- Install the following packages:

```
> install.packages(c("spdep", "dismo", "RColorBrewer", "classInt",
  "GISTools", "maptools", "GeoXp", "xtable"))
```

- Import the data:

```
> require("maptools")
> boston.tr <- readShapePoly(system.file("etc/shapes/boston_tracts.
   package="spdep")[1], ID="poltract",
   proj4string=CRS(paste("+proj=longlat +datum=NAD27 +no_defs +ellp
   "+nadgrids=@conus,@alaska,@ntv2_0.gsb,@ntv1_can.dat")))
```

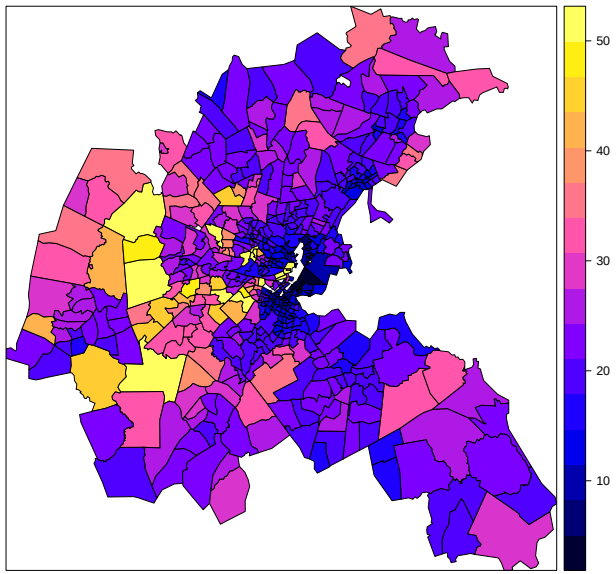
- Define the IDs, the number of observations and the response variable:

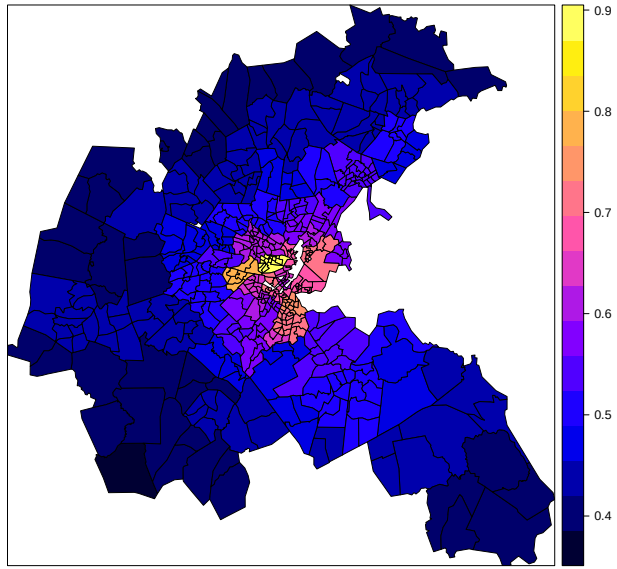
```
> ID <- row.names(boston.tr)
> n.boston <- length(ID)
> y <- boston.tr$MEDV
```

Mapping

Representation of the response variable and one explanatory variable:

```
> spplot(boston.tr, "MEDV")
> spplot(boston.tr, "NOX")
```





Vizualize the geographic situation of Boston

- Color choices:

```
> require("RColorBrewer")
> require("GISTools")
> plotclr <- rev(add.alpha(brewer.pal(5,"RdBu"), 0.7))
```

- Transform Y into 5 classes:

```
> require("classInt")
> bk <- round(classIntervals(y, 5,"kmeans")$brks, digits=1)
> ind <- findInterval(y, bk, all.inside=TRUE)
```

- Import the contour map and change the CRS of the contours:

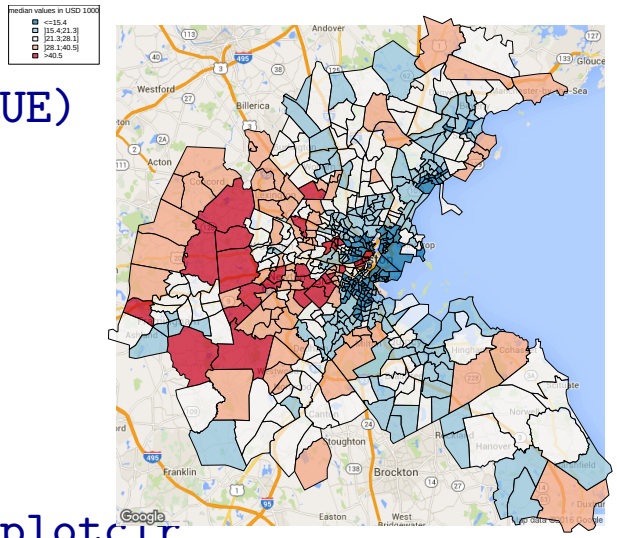
```
> require("dismo")
> g <- gmap(boston.tr, type = "roadmap", zoom = 10)
> projMer <- projection(g)
> boston.tr <- spTransform(boston.tr, CRS(projMer))
```

- Possibility to take into account extra data:
  `https://worldmap.harvard.edu/maps/3948`

Vizualize the geographic situation of Boston (2)

Plotting the map :

```
> # we plot the information
> plot(g)
> plot(boston.tr, col=plotclr[ind], add=TRUE)
> # the legend
> decoup <- c(paste("<=",bk[2],sep=""),
  paste("]",bk[2],";",bk[3],"]",sep=""),
  paste("]",bk[3],";",bk[4],"]",sep=""),
  paste("]",bk[4],";",bk[5],"]",sep=""),
  paste(">",bk[5], sep=""))
> legend("topleft", legend = decoup, fill=plotclr,
  cex = 0.8,title="median values in USD 1000")
```



Model used in the paper

- Ordinary Least-Squares regression:

```
> boston.lm <- lm(log(MEDV) ~ CRIM + ZN + INDUS + CHAS +
  I(NOX^2) + I(RM^2) +  AGE + log(DIS) + log(RAD) + TAX +
  PTRATIO + B + log(LSTAT), data=boston.tr@data)
```

- Regression validation:

```
> summary(boston.lm)
> plot(boston.lm)
```

- Spatial analysis of the residuals :

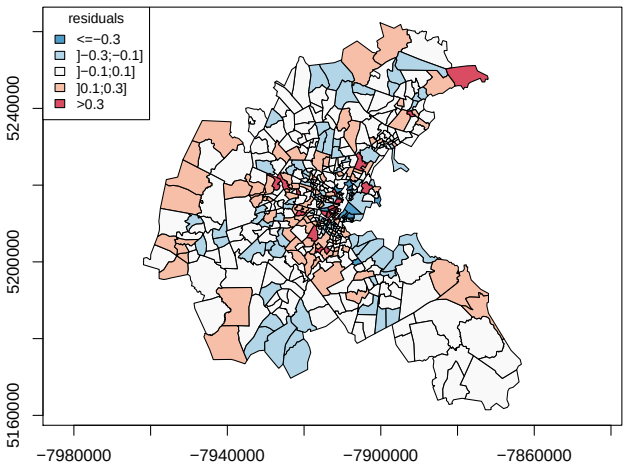
```
> res.boston <- residuals(boston.lm)
```

- We transform the residuals into 5 classes:

```
> bk <- round(classIntervals(res.boston, 5,"kmeans")$brks,
  digits=1)
> ind <- findInterval(res.boston, bk, all.inside=TRUE)
```

Representation of the residuals

```
> plot(boston.tr, col=plotclr[ind], axes=T
> decoup <- c(paste("<=",bk[2],sep=""),
  paste("]",bk[2],";",bk[3],"]",sep=""),
  paste("]",bk[3],";",bk[4],"]",sep=""),
  paste("]",bk[4],";",bk[5],"]",sep=""),
  paste(">",bk[5], sep=""))
> legend("topleft", legend = decoup,
  cex = 0.8, title="residuals",
  fill=plotclr)
```



Objectives of this lecture

How can we determine if the residuals are not truly independent?

1. How can we define the spatial proximity? Creation of spatial neighbours and spatial weights.
2. How can we measure the spatial autocorrelation? Global VS Local indexes depending on continuous/discrete variables.

Preparation of a toy example

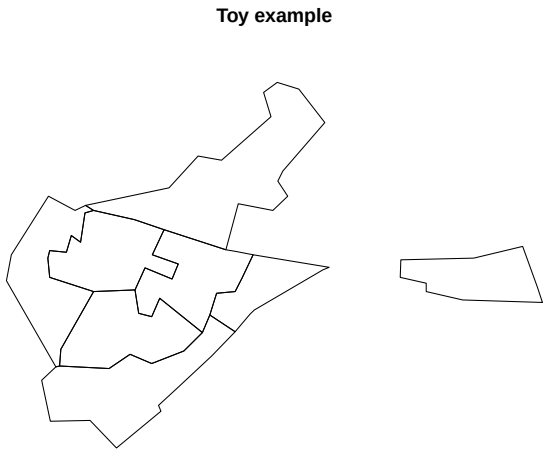
- We define the centroid of the Boston tracts:

```
> coord <- coordinates(boston.tr)
```

- We select only 8 tracts which will be used as a toy example:

```
> poly.ex <- boston.tr[c(1:7,9),]
> coord.ex <- coord[c(1:7,9),]

> plot(poly.ex, main="Toy example")
```



Stockage of a sparse matrix (1)

- Mathematically:

$$W = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

- Numerically: W will be represented as a sparse matrix: less memory demands.

Stockage of a sparse matrix (2)

- The neighbours are stocked in a `nb` object which is a list containing for each observation, the indexes of the neighbours.
- The weights are stocked in a `listw` object which contains both a `nb` object and a list of the weights.
- Many operations are available for these classes of object (**spdep** package).
- For computational matrix algebra, the package **Matrix** will be used.

a) Neighbours based on contiguity

The function `poly2nb` allows to construct a neighbour object based on the following principle: locations *i* and *j* are neighbors if they share a common border.

- Creation:

```
> require("spdep")
> poly.ex_nb.1 <- poly2nb(poly.ex)
```

- Structure

```
> str(poly.ex_nb.1)
```

- Number of neighbours per observations:

```
> card(poly.ex_nb.1)
```

- Symetric:

```
> is.symmetric.nb(poly.ex_nb.1)
```

- Summary:

```
> summary(poly.ex_nb.1)
```

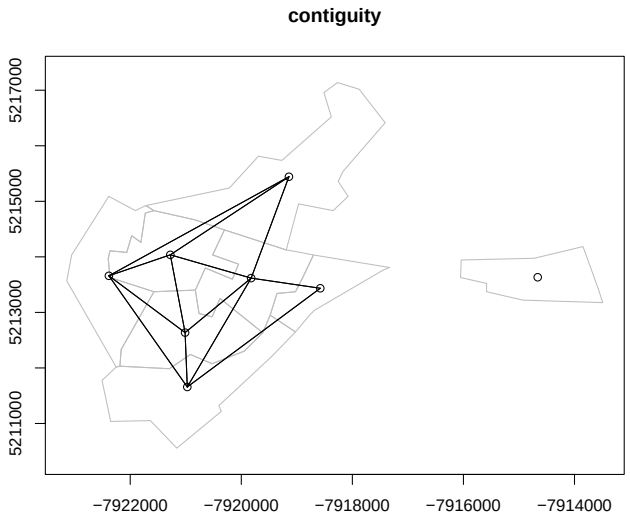
- Drop some nodes:

```
> subset(poly.ex_nb.1,
  row.names(poly.ex)!="101")
```

Visualizing the network

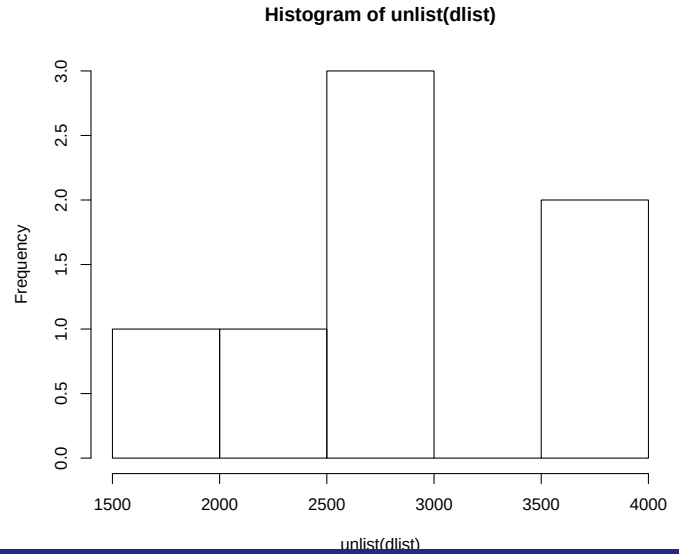
The network:

```
> plot(poly.ex, border="grey",
  axes=TRUE, main="contiguity")
> plot(poly.ex_nb.1, coord.ex, add=T)
```



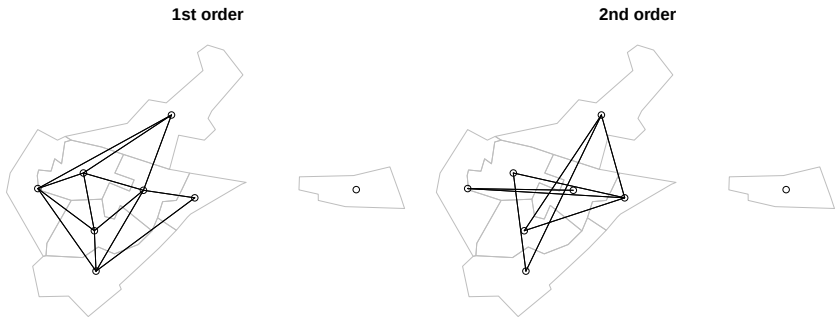
The distribution of the distances between centroids:

```
> dlist <- nbdists(poly.ex_nb.1,
  coord.ex, longlat=FALSE)
> dlist <- lapply(dlist, max)
> hist(unlist(dlist))
```



The neighbours of the neighbours

```
> lag.nb <- nblag(poly.ex_nb.1, 4)
> op <- par(mfrow=c(1,2), oma=c(0,0,0,0), mar=c(0,0,0,0))
> plot(poly.ex, border="grey", main="1st order")
> plot(lag.nb[[1]], coord.ex, add=TRUE)
> plot(poly.ex, border="grey", main="2nd order")
> plot(lag.nb[[2]], coord.ex, add=TRUE)
> par(op)
```



b) Neighbours based on Delaunay triangulation

- Creation:

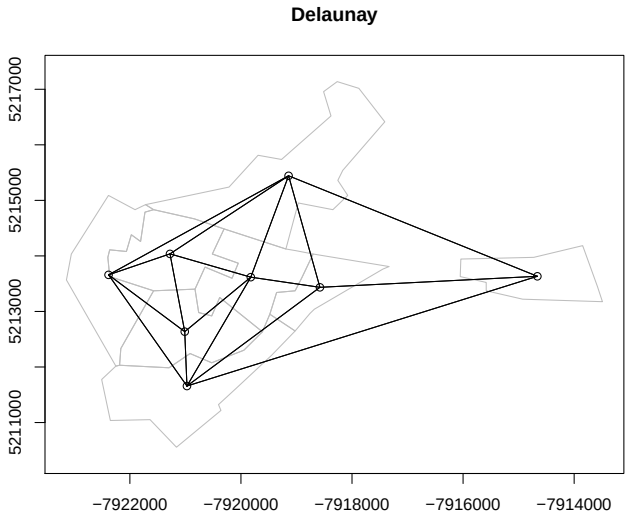
```
> poly.ex_nb.2 <- tri2nb(coord.ex)
```

- Some characteristics

```
> is.symmetric.nb(poly.ex_nb.2)
> summary(poly.ex_nb.2)
```

- Representation:

```
> plot(poly.ex, border="grey", axes=T,
  main="Delaunay")
> plot(poly.ex_nb.2, coord.ex, add=T)
```



c) *K*-nearest neighbours

- Creation in 2-steps:

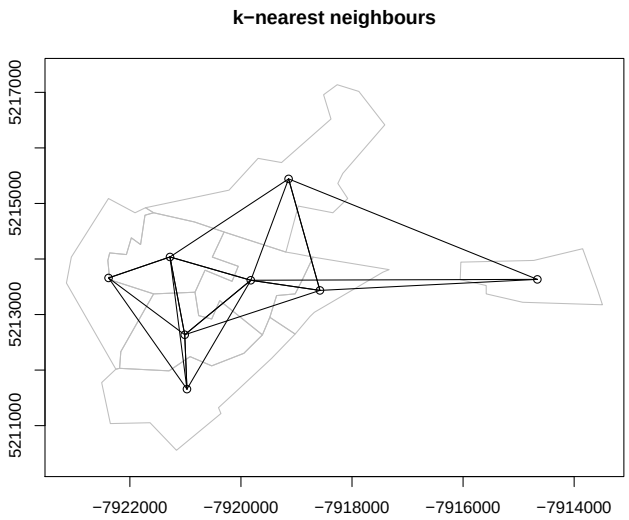
```
> poly.ex_knn <- knearneigh(coord.ex,
  k=3, longlat = FALSE)
> poly.ex_nb.3 <- knn2nb(poly.ex_knn)
```

- Some characteristics

```
> is.symmetric.nb(poly.ex_nb.3)
> summary(poly.ex_nb.3)
```

- Representation:

```
> plot(poly.ex, border="grey", axes=T,
  main="k-nearest neighbours")
> plot(poly.ex_nb.3, coord.ex, add=T)
```



d) Neighbours based on distance between $[l; u]$ bound

- Creation:

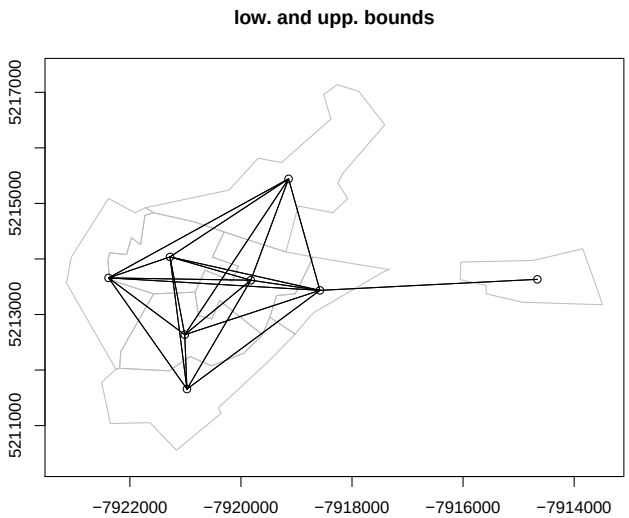
```
> poly.ex_nb.4 <- dnearneigh(coord.ex,
  0, 4000, longlat = FALSE)
```

- Some characteristics

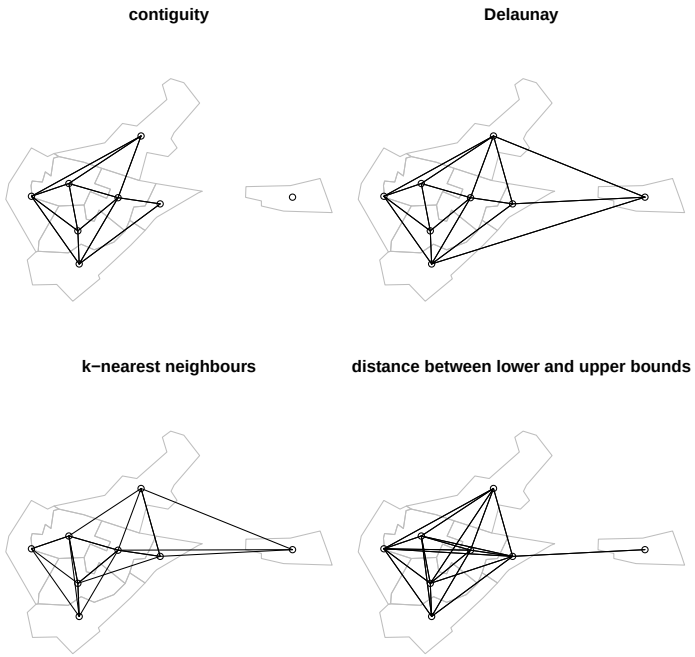
```
> is.symmetric.nb(poly.ex_nb.4)
> summary(poly.ex_nb.4)
```

- Representation:

```
> plot(poly.ex, border="grey", axes=T,
  main="low. and upp. bounds")
> plot(poly.ex_nb.4, coord.ex, add=T)
```



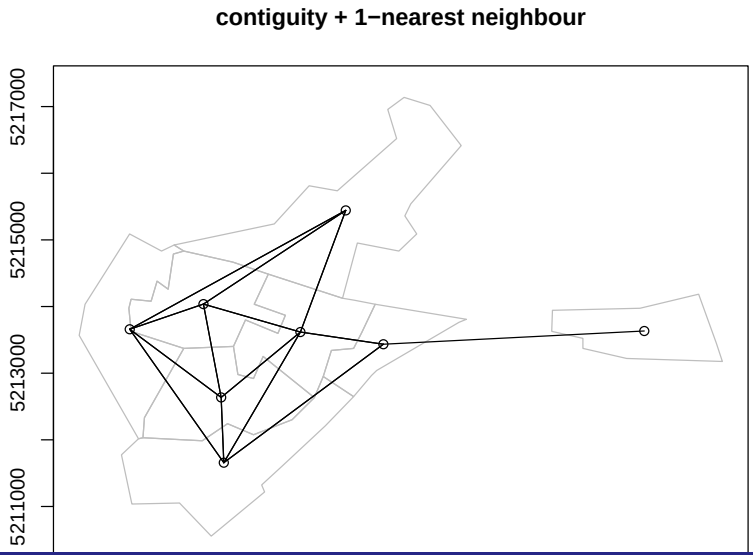
Summary of the neighbours types



Combining some neighbours structure (1)

- By using the functions `intersect.nb`, `union.nb`, `setdiff.nb`
- e) Contiguity ∪ 1-nearest neighbour

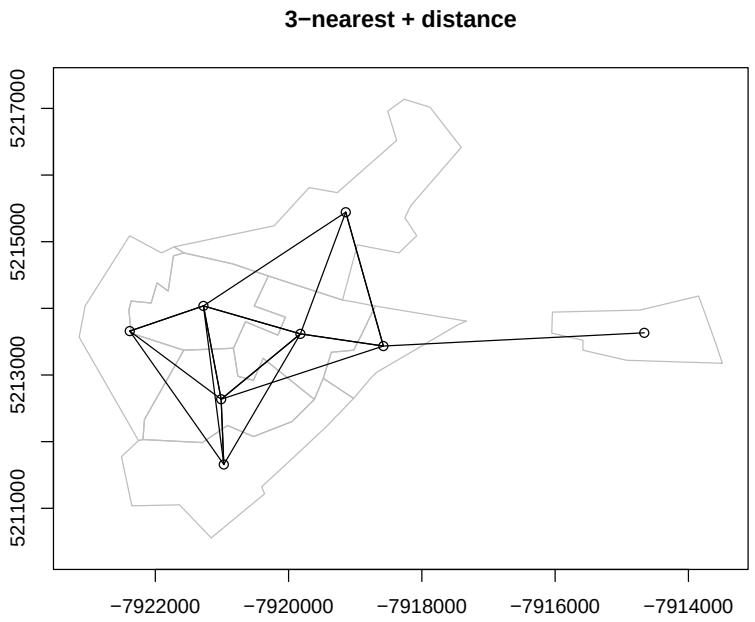
```
> poly.ex_nb.5 <- union.nb(poly.ex_nb.1, knn2nb(knearneigh(
  coord.ex, k=1, longlat=F), row.names = row.names(poly.ex)))
> plot(poly.ex, border="grey", axes=T, main="contiguity + 1-nearest
> plot(poly.ex_nb.5, coord.ex, add=T)
```



Combining some neighbours structure (2)

- f) Distance between $[l; u]$ bound $\cap$ 3-nearest neighbours

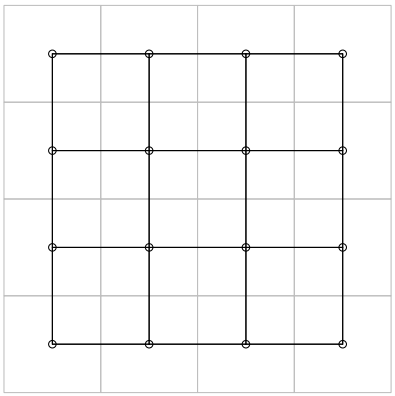
```
> poly.ex_nb.6 <- intersect.nb(poly.ex_nb.3, poly.ex_nb.4)
> plot(poly.ex, border="grey", axes=T, main="3-nearest + distance")
> plot(poly.ex_nb.6, coord.ex, add=T)
```



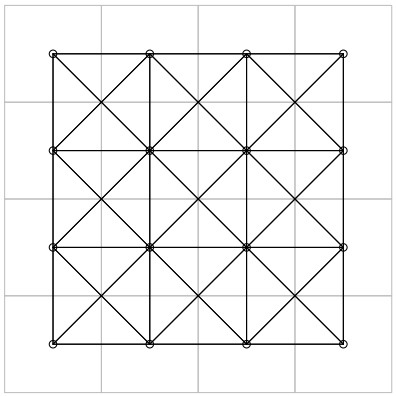
Neighbours on grid data

```
> x = GridTopology(c(2,2), c(1,1), c(4,4))
> xy <- coordinates(x)
> nb.rook <- cell2nb(4, 4, type="rook")
> nb.queen <- cell2nb(4, 4, type="queen")
```

rook



queen



More examples:

```
> vignette("nb", package="spdep")
```

a) Binary matrix

- The same weight equal to 1 is given to each vertice:

```
> poly.ex_listw.a <- nb2listw(
  poly.ex_nb.5, style="B")
```

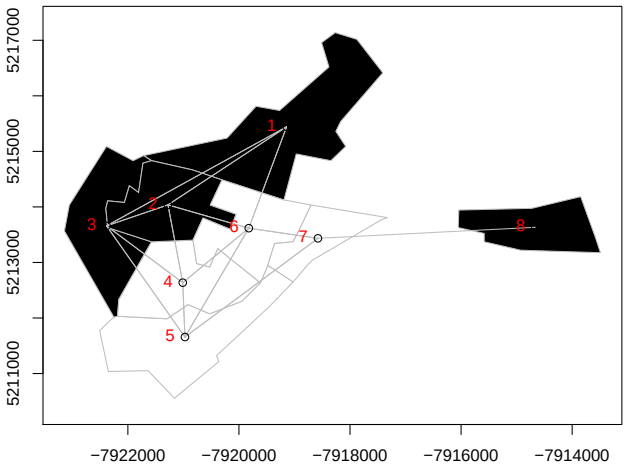
- The structure of that object:

```
> str(poly.ex_listw.a)
```

- In that case, the spatially lagged variable WX gives the sum of X observed at the neighbours.

```
> lag(poly.ex_listw.a,
  as.numeric(poly.ex@data$CHAS)-1)

[1] 2 2 2 2 1 2 0 0
```



b) Row-normalized matrix

- Economic motivation (Lesage and Pace, 2009)

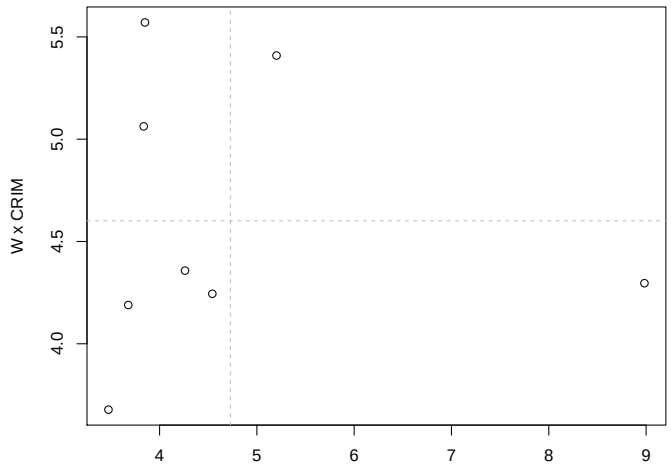
```
> poly.ex_listw.b <- nb2listw(
  poly.ex_nb.5, style="W")
```

- The sums of weights for each areal entity are unity:

$$W = \begin{pmatrix} 0 & \frac{1}{3} & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 & 0 \\ \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} & 0 & 0 \\ \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 \\ 0 & \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 \\ 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} & 0 \\ \frac{1}{5} & \frac{1}{5} & 0 & \frac{1}{5} & \frac{1}{5} & 0 & \frac{1}{5} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

- In that case, the spatially lagged variable WX gives the average mean observed at the neighbours:

```
> x <- poly.ex@data$CRIM
> wx <- lag(poly.ex_listw.b, x)
```



c) Weights proportionnal to the inverse distance

1. Choose one neighbourhood structure and compute the distance between centroids:

```
> dlist <- nbdists(poly.ex_nb.3, coord.ex)
```

2. Choose the power of the inverse distance:

```
> p <- 1
> dlist.c <- lapply(dlist, function(x) 1/x^p)
```

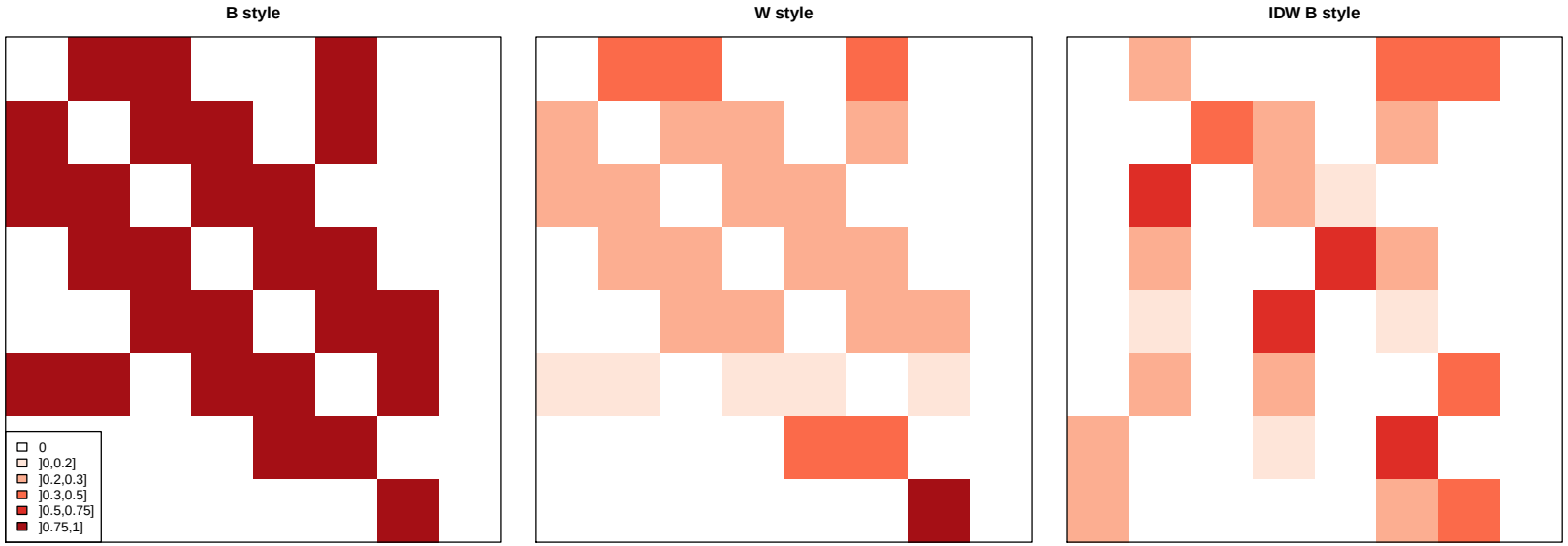
3. Row-normalize the weight matrix:

```
> poly.ex_listw.c <- nb2listw(poly.ex_nb.3,
  glist=dlist.c, style="W")
```

To get the distribution of the weights :

```
> summary(unlist(poly.ex_listw.c$weights))
```

Representation of the weight matrices



Choice of the neighborhood structure

- Contiguity:

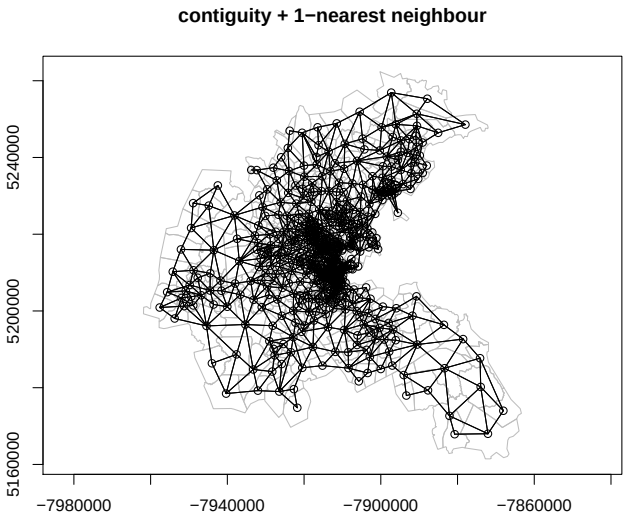
```
> boston_nb.1 <- poly2nb(boston.tr)
```

- Distance between $[0; 2000]m$:

```
> boston_nb.2 <- dnearneigh(coord,
  0, 2000, longlat=F, row.names=ID)
```

- Union of the 2 structures:

```
> boston_nb <- union.nb(boston_nb.1,
  boston_nb.2)
```



Analysis of the neighborhood structure

- Distribution of the distances between centroids:

```
> require("GeoXp")
> histnbmap(boston.tr, boston_nb)
```

- Distribution of the number of neighbours per tract:

```
> barnbmap(boston.tr, boston_nb)
```

- Comparing the coordinates for a particular site with these of its neighbouring sites:

```
> neighbourmap(boston.tr, "LAT", boston_nb)
```

Goals: check that a observation has not too much (or too few) neighbours; the distance between neighbours is not too large, etc.

Choice of the weights

- Proportional to the inverse-distance between centroid:

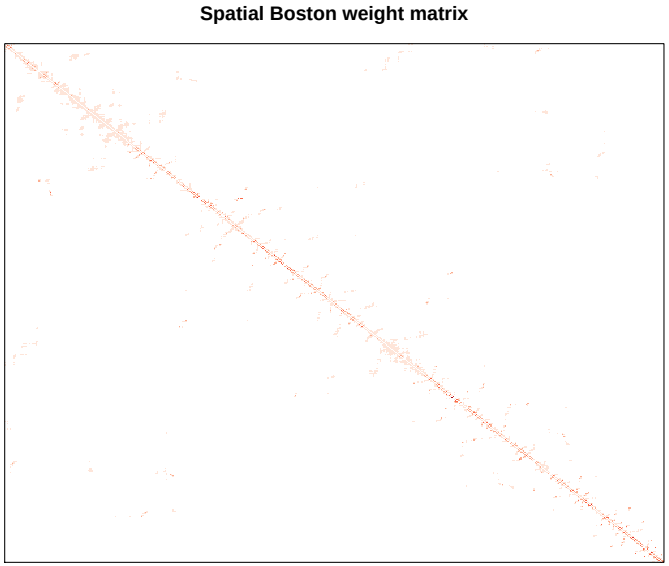
```
> dlist <- lapply(nbdists(boston_nb,
  coord), function(x) 1/x)
```

- W row-normalized:

```
> boston_listw <- nb2listw(boston_nb,
  glist=dlist, style="W")
```

- Some statistics:

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



Comparison of W memory size

- Full matrix:

```
> boston_mat <- listw2mat(boston_listw)
> object.size(boston_mat)
```

```
2077032 bytes
```

- `listw` object:

```
> object.size(boston_listw)
```

```
386912 bytes
```

- A sparse `Matrix` object:

```
> boston_sparse <- as_dgRMatrix_listw(boston_listw)
> object.size(boston_sparse)
```

```
106328 bytes
```

Simulate spatial autocorrelation

Goal: simulate the following SAR process

$$X = (I - \rho W)^{-} 1\epsilon$$

with $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$

1. Simulate ϵ (whose observations are independent by definition):

```
> uncorr_x <- rnorm(n.boston)
```

2. Define a value of ρ:

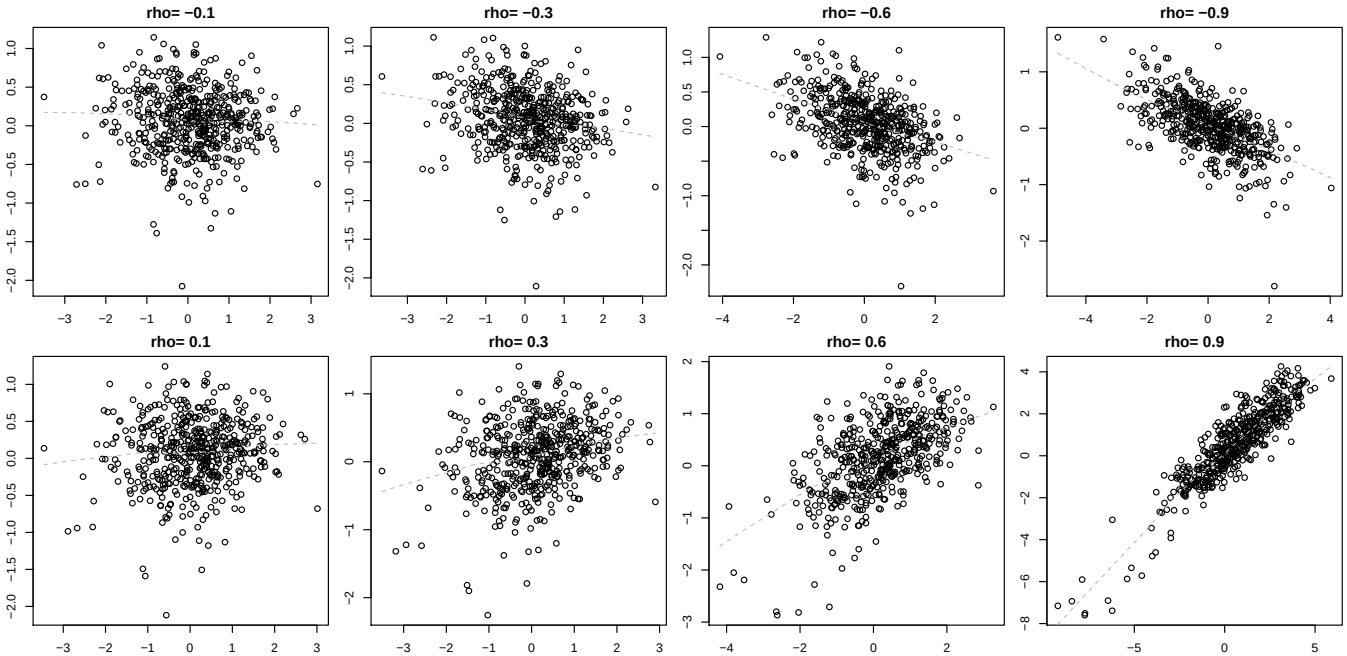
```
> rho = 0.5
```

3. Compute the inverse of $(I - \rho W)$ and multiply by ϵ:

```
> autocorr_x <- invIrW(boston_listw, rho) %*% uncorr_x
```

Mapping of the simulated variable X

Scatterplot of WX against X



Moran's I index

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

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.

- $I > -1/(n-1) \Longrightarrow$ positive spatial autocorrelation: spatial clustering of high and/or low values (no distinction between high or low)
- $I < -1/(n-1) \Longrightarrow$ negative spatial autocorrelation : checkerboard pattern, "competition".

Moran's I test (1)

Null and alternative hypothesis:

- H_0 : absence of spatial autocorrelation
- H_1 : presence of spatial autocorrelation

1st Alternative: Inference

- randomization "non free sampling"

```
> res.unc.1 <- moran.test(uncorr_x, boston_listw, randomisation=T)
> res.unc.1a <- moran.test(autocorr_x, boston_listw,
  randomisation=T)
```

- normal distribution "free sampling"

```
> res.unc.2 <- moran.test(uncorr_x, boston_listw, randomisation=F)
> res.unc.2a <-moran.test(autocorr_x, boston_listw,
  randomisation=F)
```

Moran's I test (2)

- Asymptotic results:

	I	$E(I)$	$var(I)$	st. deviate	p-value
rando uncorr	0.02551	-0.00198	0.00065	1.07	0.14
rando autocorr	0.24955	-0.00198	0.00065	9.83	<1e-08
norm uncorr	0.02551	-0.00198	0.00065	1.07	0.14
norm autocorr	0.24955	-0.00198	0.00065	9.83	<1e-08

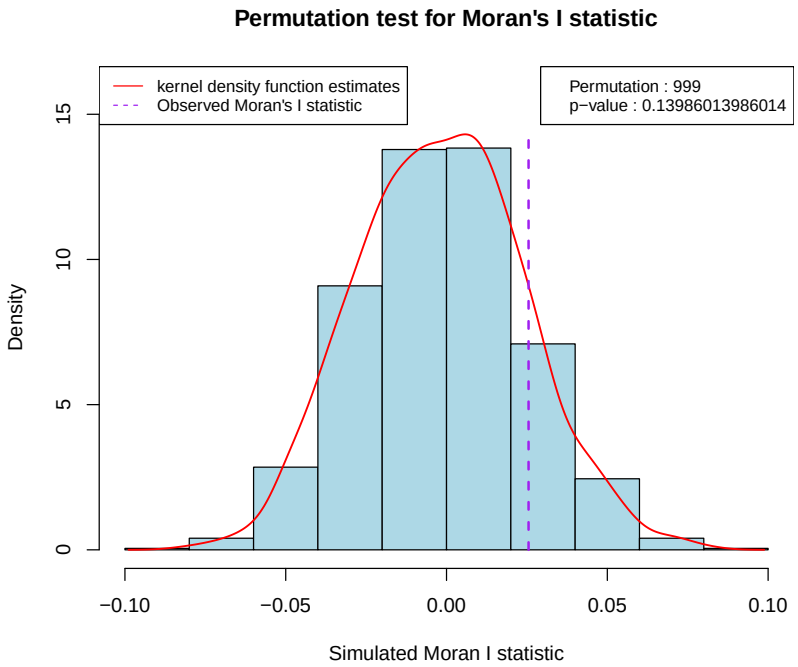
2nd Alternative: Moran's I permutation test

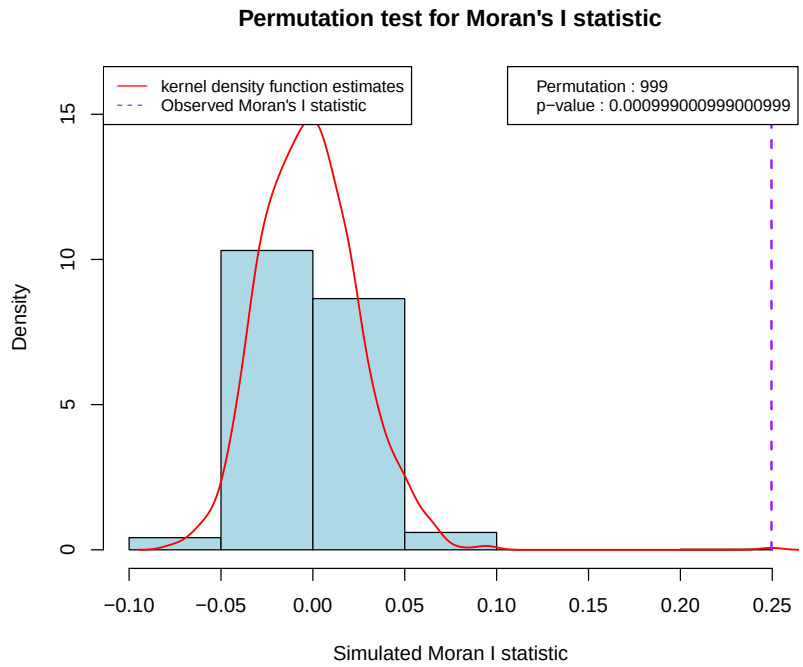
- Algorithm. Repeat B times:
  1. Permute randomly the values of X
  2. Compute the Moran's I statistic

```
> res.unc <- moran.mc(uncorr_x, boston_listw, nsim=1000)
> res.aut <- moran.mc(autocorr_x, boston_listw, nsim=1000)
```

Moran's I test (3)

- Compare the True Moran's I statistic to the simulated distribution





Moran's I test (4)

Special cases:

- For rate, use an empirical Bayes index modification of Moran's I for testing for spatial autocorrelation in a rate (`EBImoran.mc`)
- For residuals of an OLS regression, the residuals arise from a first step of estimation of the model and hence one must recalculate the moments in that case in the free sampling model:

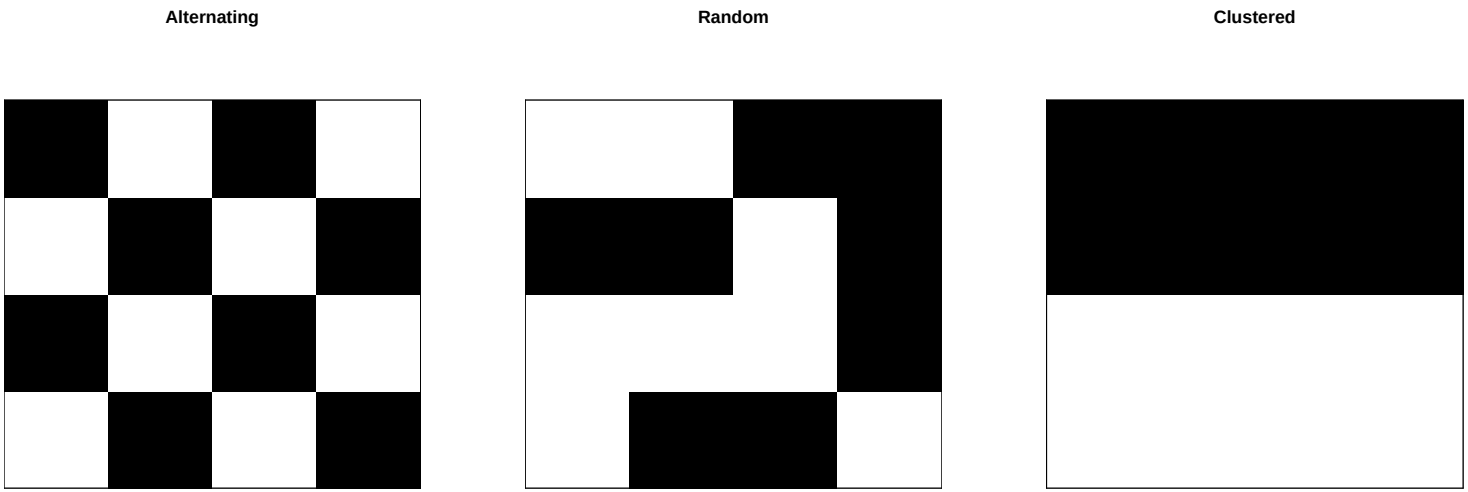
```
> lm.morantest(boston.lm, boston_listw)
```

Alternatives :

- Geary's C, global G test and a Mantel test: They are implemented as `geary.test` and `globalG.test`, with permutation bootstrap variants: `geary.mc` and `sp.mantel.mc`.

Join Count test (1)

Example: a grid of size 4×4. Possible value 0 (white) or 1 (black).
Spatial weights matrix: rook binary. $BB = (1/2) \sum_i \sum_j x_i x_j w_{ij}$,
$WW = (1/2) \sum_i \sum_j (1 - x_i)(1 - x_j) w_{ij}$,
$BW = (1/2) \sum_i \sum_j (x_i - x_j)^2 x_{ij}$



Join Count test (2)

- 1st alternative: Inference "non free sampling"

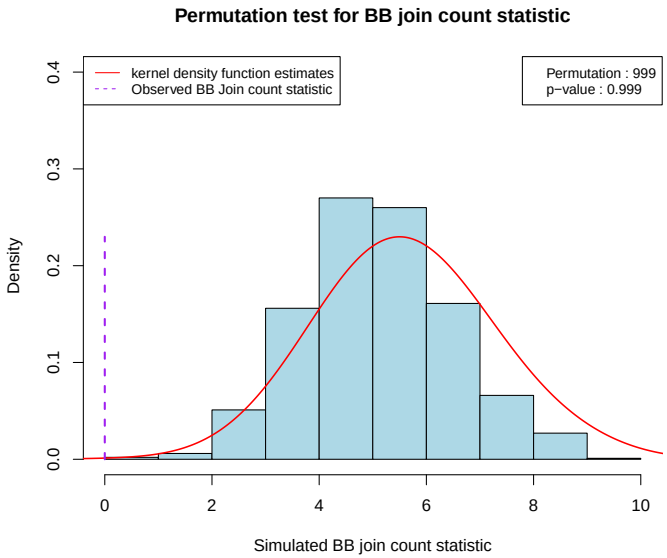
```
> joincount.multi(factor(as.vector(box1), labels=c("B","W")),
  nb2listw(nb.rook, style="B"))
```

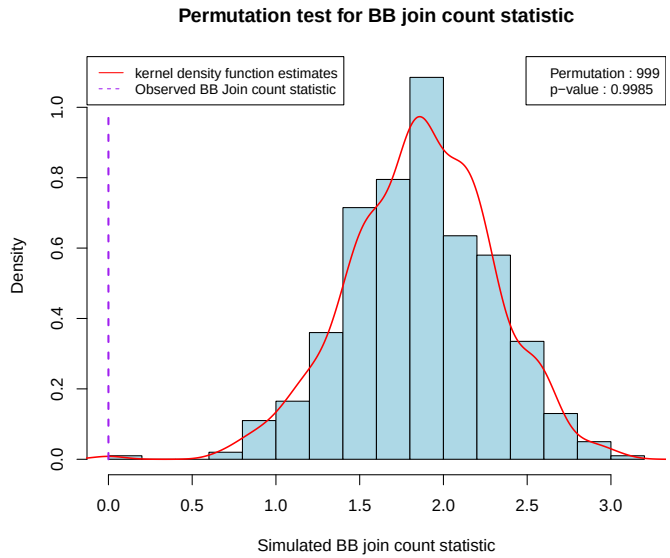
	Joincount	Expected	Variance	z-value
B:B	0.00	5.60	1.87	-4.09
W:W	0.00	5.60	1.87	-4.09
W:B	24.00	12.80	5.35	4.84
Jtot	24.00	12.80	5.35	4.84

Join Count test (3)

- 2nd alternative: permutation test

```
> res.BB<-joincount.mc(factor(as.vector(box1), labels=c("B","W")),
  nb2listw(nb.rook, style="B"), nsim=999)
```



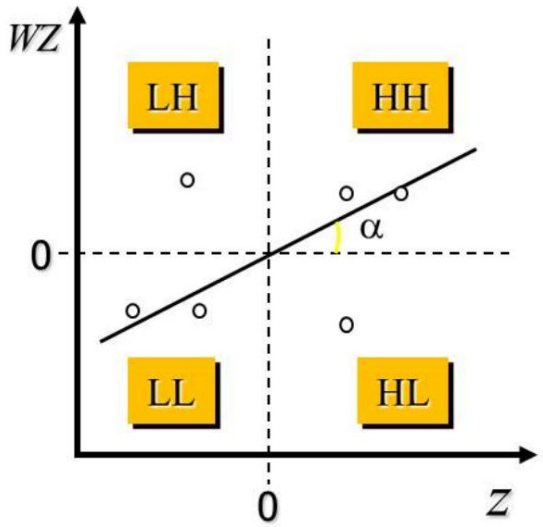


On the left, choice of a binary rook matrix; on the right, choice of a row-normalized rook matrix

Moran scatterplot (1)

Principles:

- Linear spatial autocorrelation : linear association between value at i and weighted average of neighbors (WX VS X)
- Four quadrants : high-high and low-low = spatial clusters. high-low and low-high = spatial outliers
- Moran's I : slope of linear scatterplot smoother
- Uusual influence on the Moran's I line marked graphically



Moran scatterplot (2)

Aims:

- Classification of spatial autocorrelation
- Local nonstationarity : outliers, high leverage points
- Local Moran I's

$$I_i = \frac{(x_i - \bar{x}) \sum_j w_{ij}(x_j - \bar{x})}{\frac{\sum_k (x_k - \bar{x})^2}{n}}$$

Application (1)

Application on the residuals of the Boston data:

- Use the function `moranplotmap` (package **GeoXp**):

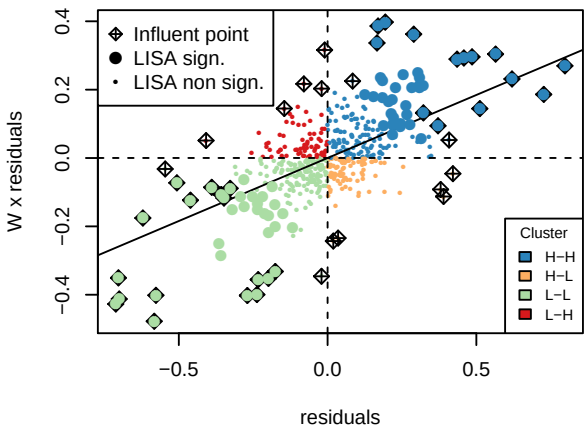
```
> boston.tr$res <- residuals(boston.lm)
> moranplotmap(boston.tr, "res", boston_listw)
```

- Use the function `plot.M` given with the R code of this lecture:

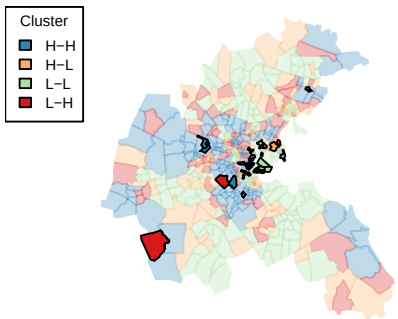
```
> plot.M(boston.tr, boston_listw, res.boston,
  rev(brewer.pal(4,"Spectral")), "residuals")
```

Application (2)

Moran plot of residuals



Influent points cluster map



Significant LISA cluster map