

## Areal data

# Quantitative Methods in Geography

Geo 340

William D. McCoy

Department of Geosciences  
University of Massachusetts, Amherst

Autumn, 2007

- ▶ location quotient ( $LQ$ )
- ▶ coefficient of localization ( $CL$ )
- ▶ Lorenze curve
- ▶ Gini coefficient (index of dissimilarity)

## Working with spatial data

- ▶ areal data
- ▶ point data
- ▶ directional data

## Areal data: location quotient

The location quotient is a measure of the proportion of an activity in a particular area compared to the proportion of some base or aggregate of activities in that area. The location quotient is defined as

$$LQ_i = \frac{A_i / \sum_{i=1}^n A_i}{B_i / \sum_{i=1}^n B_i},$$

where  $A_i$  is the amount of some activity  $A$  in area  $i$  and  $B_i$  is the measure of some base or aggregate  $B$  in area  $i$ .

## Location Quotient: Example

State	Ill. Pop. ( $A_i$ )	$A_i / \sum A_i$	Pop. ( $B_i$ )	$B_i / \sum B_i$	LQ
CT	34,100	0.271	3,100,000	0.254	1.06
ME	7,406	0.059	1,058,000	0.087	0.68
MA	63,954	0.507	5,814,000	0.477	1.06
NH	5,684	0.045	812,000	0.067	0.68
RI	12,103	0.096	931,000	0.076	1.26
VT	2,832	0.022	472,000	0.039	0.58
Totals	126,079	1.000	12,187,000	1.000	

## Location Quotient: R code

It's easy to write a function in **R** to calculate location quotients. Note how the following code corresponds to the formula for the location quotient:

```
> "LQ" <- function(activity, base) {  
+   act <- activity/sum(activity)  
+   b <- base/sum(base)  
+   act/b  
+ }
```

## Location Quotient: executing the code

And to execute the code we just need to pass two vectors to our new function. A vector of LQs is returned.

```
> LQ(ne.illiteracy, ne.pop)  
  
Connecticut      Maine Massachusetts  
  1.0632778      0.6766313      1.0632778  
New Hampshire Rhode Island      Vermont  
  0.6766313      1.2566010      0.5799697
```

## Areal data: coefficient of localization

The coefficient of localization expresses the extent or degree of localization on a scale of zero to one. Zero represents no localization, but rather a uniform distribution of the activity relative to some base. A coefficient of one would represent extreme localization. The coefficient can be defined as

$$CL = \frac{1}{2} \sum_{i=1}^n \left| \frac{A_i}{\sum_{i=1}^n A_i} - \frac{B_i}{\sum_{i=1}^n B_i} \right|.$$

## Coefficient of localization: R code

We can reuse some of the **R** code in our `LQ()` function when writing a function to calculate the coefficient of localization.

```
> "CL" <- function(activity, base) {  
+   act.frac <- activity/sum(activity)  
+   base.frac <- base/sum(base)  
+   sum(abs(act.frac - base.frac))/2  
+ }
```

This code returns a single number.

```
> CL(ne.illiteracy, ne.pop)  
[1] 0.06588609
```

## Areal data: Lorenz curve

To construct a Lorenz curve, the first thing we must do is order the areas in decreasing order of their *LQ*s. Then we keep track of the cumulative sums of the percentages of the activity and the base.

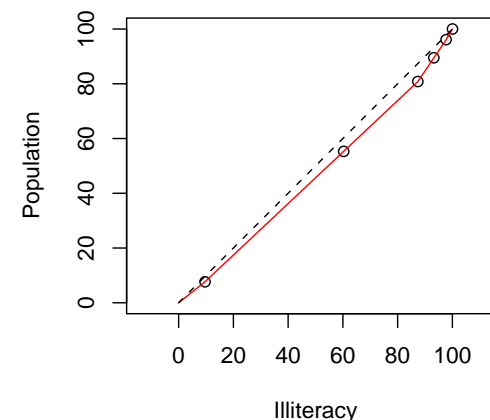
State	LQ	Activity		Base	
		Ill%	Cum%	Pop%	Cum%
RI	1.26	9.6	9.6	7.6	7.6
CT	1.06	27.1	36.7	25.4	33.0
MA	1.06	50.7	87.4	47.7	80.7
ME	0.68	5.9	93.3	8.7	89.4
NH	0.68	4.5	97.8	6.7	96.1
VT	0.58	2.2	100.0	3.9	100.0

## Lorenz curve: R code

```
> "Lorenz" <- function(activity, base, ...) {  
+   act.pc <- (100 * activity)/sum(activity)  
+   base.pc <- (100 * base)/sum(base)  
+   act.order <- act.pc[rev(order(LQ(activity,  
+     base)))]  
+   base.order <- base.pc[rev(order(LQ(activity,  
+     base)))]  
+   cs.act.order <- cumsum(act.order)  
+   cs.base.order <- cumsum(base.order)  
+   plot(cs.act.order, cs.base.order,  
+     xlim = c(0, 100), ylim = c(0,  
+     100), type = "p", ...)  
+   lines(c(0, cs.act.order), c(0, cs.base.order),  
+     col = 2, type = "l")  
+   lines(c(0, 100), c(0, 100), lty = 2)  
+   max(cs.act.order - cs.base.order)  
+ }
```

## Lorenz curve: Illiteracy in New England

```
> Lorenz(ne.illiteracy, ne.pop, xlab = "Illiteracy",  
+   ylab = "Population", asp = 1)  
[1] 6.588609
```



## Areal data: Gini coefficient

Note the last line of code in our `Lorenz()` function. The line produces an extra calculation after the Lorenz curve is drawn.

```
> max(cs.act.order - cs.base.order)
```

This code subtracts the vector of cumulative sums of the base from the vector of cumulative sums of the activity. Then it finds the maximum difference. That largest difference is called the Gini coefficient or index of dissimilarity.

Notice that the Gini coefficient is the same number as the coefficient of localization, except that it is calculated using percentage units and so is 100 times as large.

## Point data: measures of central tendency

- ▶ mean center
- ▶ weighted mean center
- ▶ Manhattan median
- ▶ Euclidean median

## Mean center

The mean center is simply the mean of the  $x$ -coordinates (usually representing longitude) and the mean of the  $y$ -coordinates (usually representing latitude).

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i, \quad \bar{y} = \frac{1}{n} \sum_{i=1}^n y_i$$

The mean center minimizes the sum of the squared distances between itself and each point. Mathematically, it minimizes

$$\sum_{i=1}^n (x_i - \bar{x})^2 + (y_i - \bar{y})^2.$$

## Weighted mean center

Sometimes we will want to weight each point relative to some attribute of interest, such as population. In those cases, we can generalize the mean center to include the weights.

$$\bar{x}_w = \frac{\sum_{i=1}^n w_i x_i}{\sum_{i=1}^n w_i}, \quad \bar{y}_w = \frac{\sum_{i=1}^n w_i y_i}{\sum_{i=1}^n w_i}$$

## Manhattan median

The Manhattan median is simply the median of the  $x$ -coordinate and the median of the  $y$ -coordinate. The Manhattan median minimizes the sum of the Manhattan (N-S and E-W) distances between itself and each of the points. Mathematically, it minimizes

$$\sum_{i=1}^n |x_i - x_m| + |y_i - y_m|,$$

where  $x_m$  and  $y_m$  are the medians of the  $x$ - and  $y$ -coordinates, respectively.

## Euclidean median

The Euclidean median minimizes the sum of the Euclidean distances between itself and all the points. That is, it minimizes

$$\sum_{i=1}^n \sqrt{(x_i - x_e)^2 + (y_i - y_e)^2}.$$

The Euclidean median is not so easy to calculate. One must use an iterative process as described in the book. That should be easy to program in **R**. You can do it for extra credit.

## Point data: measure of dispersion

The standard distance is a measure of the dispersion of points around the mean center,

$$SD = \sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n} + \frac{\sum_{i=1}^n (y_i - \bar{y})^2}{n}}.$$

As you can see, this is just the square root of the sum of the variances in the  $x$ - and  $y$ -coordinates,

$$SD = \sqrt{\sigma_x^2 + \sigma_y^2}.$$

## Directional data: directional mean

To calculate the directional mean, we must find the ratio of the sums of sines and cosines of our angular data. The sine divided by the cosine of an angle is the tangent. Therefore, we must find the angle whose tangent is this ratio,

$$\theta_R = \tan^{-1} \frac{\sum_{i=1}^n \sin \theta_i}{\sum_{i=1}^n \cos \theta_i}.$$

To use this in **R**, use the function `atan2()` with the sum of the sines as the first argument and the sum of the cosines as the second argument.

## Directional data: circular variance

As a measure of the dispersion of bearings around a directional mean, we can use the circular variance,

$$S_0 = 1 - \frac{1}{n} \sqrt{\left(\sum_{i=1}^n \sin \theta_i\right)^2 + \left(\sum_{i=1}^n \cos \theta_i\right)^2}.$$

## Using circular data in R

To use circular data in R, you can load the *circular* package and enter data of class circular.

```
> library(circular)
> wmn <- c(119, 162, 221, 259, 270, 29,
+ 97, 292, 40, 313, 94, 45, 47, 108,
+ 221, 270, 119, 248, 270, 45, 23)
> windMilwNoon <- circular(wmn, type = "directions",
+ units = "degrees", template = "geographics")
```

## Finding the directional mean and variance

```
> mean(windMilwNoon)
Circular Data:
Type = directions
Units = degrees
Template = geographics
Modulo = asis
Zero = 1.570796
Rotation = clock
[1] -24.46567
> var(windMilwNoon)
[1] 0.9310339
> summary(windMilwNoon)
          n          Mean          Rho
21.00000000 -24.46567151  0.06896613
```

## Plotting directional data

```
> plot(windMilwNoon, stack = TRUE, sep = 0.075)
```

