Charles DiMaggio, PhD
Center for Injury Epidemiology and Prevention
Columbia University, NY
cjd11@columbia.edu

Spatial Epidemiology Notes

Applications and Vignettes in R

July 25, 2014

Contents

Chapter 1

Introduction to Spatial Analysis in R

1.1 Introduction

These notes, and this first section in particular, are taken in large measure (OK almost completely) from the excellent "Applied Spatial Data Analysis with R" by Roger Bivand, Edzer Pebesma and Virgilio Gomez-Rubio. Buy the book. It is worth it. I also stole some code from a course Dr. Bivand put online.

My intent is to present a relatively brief, non-jargony overview of how practicing epidemiologists can apply some of the extremely powerful spatial analytic tools that are easily available to them. I try to define terms and concepts as simply as possible or at least simply enough so that *I* understand them. I try to provide enough examples and code so that someone with at least a master's degree level of training in epidemiology, and some experience and familiarity with R, can start applying these tools almost immediately to their own data and problems.

The material is unapologetically based on the kinds of issues and topics which interest me, and which I've spent some time working on. You will find a lot of references to trauma and injury and disease outcomes. You will find a lot of ecological or areal analyses. You will not find much, if anything, about geostatistics. This is the application of spatial statistics to interpolate and predict values and includes topics like kriging. It is simply not something that I've had to deal with in my epidemiological work. Similarly, I consider point process analysis another highly specialized topic that I've not had to apply in my practice. If confronted with such data, I would likely seek help and collaboration from a *'real'* medical geographer or spatial analyst. It is though a topic to which epidemiologists are likely to see reference, so I've tried to perform due diligence and included some information and an example later in these notes.

Probably the most important step in spatial epidemiology is to decide, early on, if a spatial component can add *anything* to your epidemiological analysis. An example where spatial data can add to descriptive analyses is determining how many people may have been exposed to an environmental agent based on scattered air quality measures and residence. Spatial data may add to analytic questions if the analysis involves whether outcomes are clustered geographically. Once you've made the decision that there is an important spatial component to your epidemiological problem, the question is then how to address address and incorporate that spatial information. That's what these notes are about.

1.1.1 Spatial Data and Epidemiology: The Open-Source Way

GIS (Graphical Information Systems) have really been around for as long as there have been maps. Computing just makes them more comprehensive, accurate and available to non-cartographic folks. There are a number of open-source and free tools available to epidemiologists to help incorporate GIS into their armamentarium. The two most important are GRASS [1] and R. There has also been some movement away from toolbox programs (like ArcGIS) to more service-oriented tools, like Google Earth (to which R and GRASS can also interface).

[1] See Neteler and Mitasova (2008) for an excellent introductory text

1.1.2 GIS and Epidemiology

GIS has a language all its own. The World Geodetic System (WGS84) is a graphical representation of the world, that transforms a 3D sphere to 2D map made of points, lines, polygons, (vectors), grids (rasters) , and their associated data attributes.

In general, you can represent topography with a series points (each with its own value), in a grid (a tesselated plane of pixel-like data, e.g. digital camera images and computer screens), or with contour lines (polygons)

1.1.3 R and GIS

R has some excellent spatial analysis tools. To get a sense of some of its capabilities, check out the Task View on the CRAN (Comprehensive R Archive Network) website.

If you'd like to work through some of the examples in these notes, your first step is to install the "ctv" package, open it, and install all the spatial packages available for R:

```
> install.packages("ctv")
> library(ctv)
> install.views("Spatial")
```

1.2 About R spatial data

1.2.1 R data classes

In R, data classes refer to data structure, organization and storage. For spatial classes, this generally refers to things like points, lines, polygons and grids. R data methods are the functions that can be applied to those classes. The important thing is that the class of an R data object determines the methods that can be used. [2] When you ask R to return a "result" of an operation, it applies the print method associated with the data object that results from the method for that class.

If a class doesn't have an explicit method for an operation, it gets passed to a generic function. For example, print(), plot() , summary() and str() [3]are generic functions that work on most any R object and are good ways to begin to get information on an object .

Classes were not introduced to S (on which R is based) until version 3 in 1992. These are known as S3 or old-style classes. They are not formally defined, and are usually just lists with attributes. These S3 classes grew over time without very much organization. In 1994, new-style S4 classes were introduced. They have formally defined "slots" for components, are faster to read and to write, and (unlike S3 class objects) the user can't redefine their attributes. We'll see later that these slots are how we can access some of the rich data elements like the area of a spatial polygon, that are part of formally defined spatial objects.

[2] If you are less than comfortable with R in general, and data objects in particular you can look at some slides I've posted
[3] 'str' stands for 'structure'. It is an extraordinarily useful function that you should use all the time.

1.2.2 R spatial data classes

While you may never have to create classes directly, it definelty helps to have a sense of the kinds of classes you will be working with when doing spatial epidemiolgy in R.

The R spatial “sp” classes are new-style S4 classes. The foundation *spatial* class has only 2 slots: a bounding box with 2 rows (matrix eastings and northings (x and y)) and a slot for the coordinate reference system (CRS). The functions *getClass()* and *slots()* will return some information on spatial classes. You can create data objects from the slots e.g. with bbox() or proj4string() and manipulate it with R indexing functions

The *SpatialPoints* class extends the *spatial* class by adding a slot for a matrix of points. The SpatialPointsDataFrame() class extends the SpatilPoints class by associating a row of data with each point. When creating a SpatialPointsDataFrame, if both the points and their associated data have row names and the variable *match.ID* is set to *TRUE*, you don’t need to sort them to match them up. [4]

The *SpatialLines* slot is a matrix of coordinates that defines a two-dimensional line. It can be converted to a *SpatialLinesDataFrame* object by using the maptools package function *map2SpatialLines()*.

A *SpatialPolygons* object is based on a closed line (i.e with the same 1st and last coordinate) with extra slots for a centroid and area. A *SpatialPolygonsDataFrame* matches the polygon ID to a data frame row that includes variables like a ’hole’ slot which is set to TRUE or FALSE for whether the polygon is filled (like a land mass) or empty (like a lake) and a ’ring’ slot (clockwise vs. counterclockwise), and for whether there is aboundary between land, water.

A *SpatialGrid* object, as its name implies, is a regular grid or lattice of spaces. It is defined by its origin, the size of each cell, its overall dimensions, and is extended to a *SpatialGridDataFrame* by adding a data slot. [5]

For rasters, R uses the *SpatialPixel* class. A *SpatialPixelsDataFrame* object associates data with centroids which are saved as a sequence. This can be more efficient than a SpatialGridDataFrame object, and can be coerced from a grid object using *as()* or can be created from a SpatialPoints object if the points lie on a grid.

If you’ve not worked with R object classes before, this is a lot of information all at one time. The important take-away point about these spatial objects, particularly the spatial data frame objects, is that *they behave and can be manipulated and indexed like regular R data frames*. The important important exception being that do not (yet) have merge() capabilities. Another very nice feature of these spatial objects is that they are (generally) compatible with GIS systems like ArcGIS and GRASS.

1.3 Creating spatial objects

We can create spatial objects from data sets. Here we use the “meuse” data set of top soil heavy metal concentrations that comes with the sp package[6] to create points and lines and then use the plot() function to visualize them. We begin by assigning coordinates using the *coordinates()* function to create a SpatialPoints object and then joining up the points as lines using *SpatialLines()* function to create a SpatialLines object.

We begin by loading the sp package and the meuse data set and then finding out a little about the structure and variables that make up the data set.

```
> library(sp)
> data(meuse)
> str(meuse)
```

[4] By the way, as an S4 type class, a SpatialPointsDataFrame inherits all objects from SpatialPoints and Spatial objects upon which it is based

[5] Starting to notice a pattern?

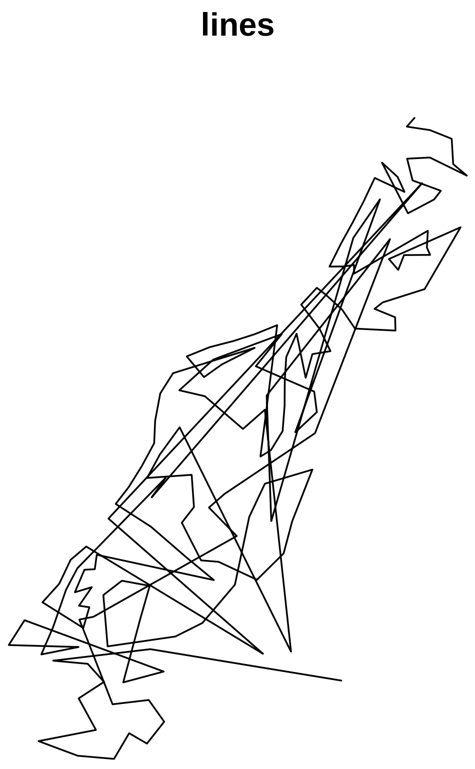
[6] Use ?meuse to find out more about the data...

```
data.frame:        155 obs. of  14 variables:
 $ x      : num  181072 181025 181165 181298 181307 ...
 $ y      : num  333611 333558 333537 333484 333330 ...
 $ cadmium: num  11.7 8.6 6.5 2.6 2.8 3 3.2 2.8 2.4 1.6 ...
 $ copper : num  85 81 68 81 48 61 31 29 37 24 ...
 $ lead   : num  299 277 199 116 117 137 132 150 133 80 ...
 $ zinc   : num  1022 1141 640 257 269 ...
 $ elev   : num  7.91 6.98 7.8 7.66 7.48 ...
 $ dist   : num  0.00136 0.01222 0.10303 0.19009 0.27709 ...
 $ om     : num  13.6 14 13 8 8.7 7.8 9.2 9.5 10.6 6.3 ...
 $ ffreq  : Factor w/ 3 levels "1","2","3": 1 1 1 1 1 1 1 1 1 1 ...
 $ soil   : Factor w/ 3 levels "1","2","3": 1 1 1 2 2 2 2 1 1 2 ...
 $ lime   : Factor w/ 2 levels "0","1": 2 2 2 1 1 1 1 1 1 1 ...
 $ landuse: Factor w/ 15 levels "Aa","Ab","Ag",..: 4 4 4 11 4 11 4 2 2 15 ...
 $ dist.m : num  50 30 150 270 380 470 240 120 240 420 ...
```

The ‘x’ and ‘y’ variables are coordinates. We will use them to plot these data by assigning them to the *coordinates()* function.

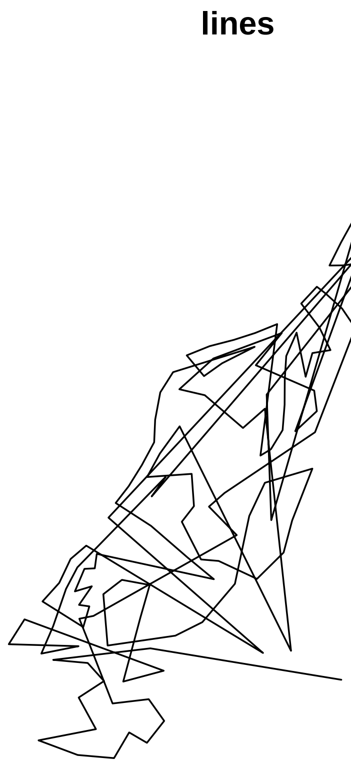
```
> coordinates(meuse) <- c("x", "y")
```

```
> plot(meuse)
> title("points")
```



We can join up the lines into a SpatialLines object:

```
> cc <- coordinates(meuse)
> m.sl <- SpatialLines(list(Lines(list(Line(cc)), "1")))
> plot(m.sl)
> title("lines")
```



We could also have added our points to the original data frame and, using the *SpatialPointsDataFrame()* function in *sp* created a SpatialPointsDataFrame object. And, we can similarly use functions like *SpatialPolygons()*, *SpatialPixels()*, SpatialPixelsDataFrame, and *SpatialGridDataFrame()*. The good news is that, as epidemiologists, we will rarely have cause to primarily create or manipulate spatial objects in this way. But it is good to know that we can.

1.4 Plotting spatial data in R

Visualizing spatial data is in many respects the main attraction of working with them. R sp classes can be visualized with both traditional (plot, image, lines, points, etc.) and trellis (spplot) graphs.

We begin by creating a SpatialPoints object out of the muese data

```
> library(sp)
> data(meuse)
> coords <- SpatialPoints(meuse[, c("x", "y")])
> summary(coords)
```

```
Object of class SpatialPoints
Coordinates:
     min    max
x 178605 181390
y 329714 333611
Is projected: NA
proj4string : [NA]
Number of points: 155
```

We add the SpatialPoints object to the original data to make a SpatialPointsDataFrame object. [7]

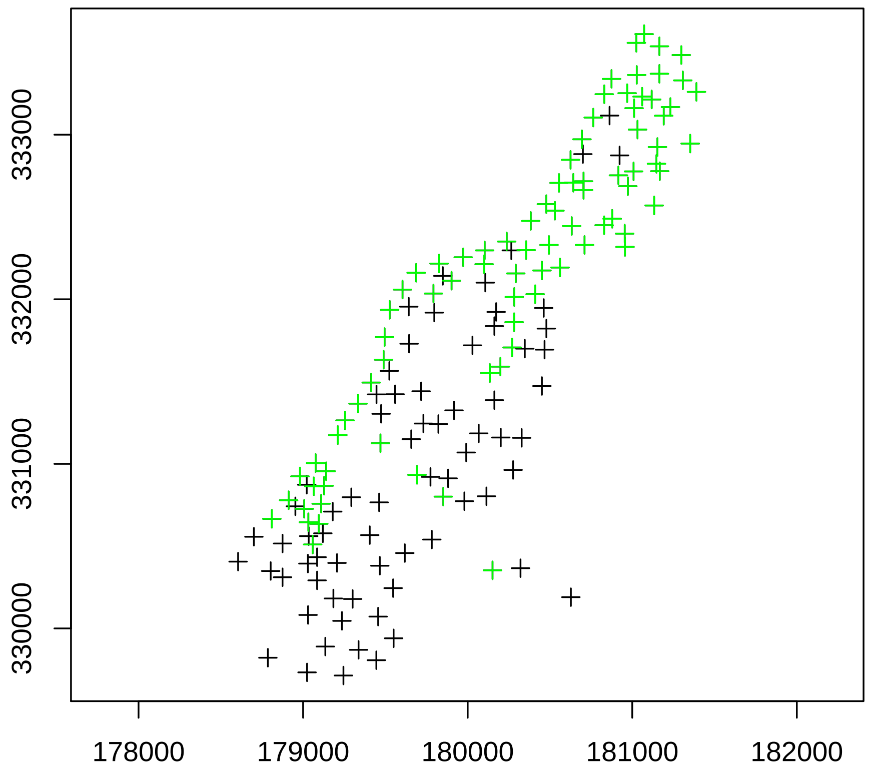
```
> meuse1 <- SpatialPointsDataFrame(coords, meuse)
> names(meuse1)
```

```
 [1] "x"       "y"       "cadmium" "copper"  "lead"    "zinc"    "elev"
 [8] "dist"    "om"      "ffreq"   "soil"    "lime"    "landuse" "dist.m"
```

Now let's plot a variable form these data, the risk of flooding. First, we create the axes, then we overlay (using add=T) all the points using the default symbol, and finally, we overlay those points with a flood frequency value equal to 1 with the color green.

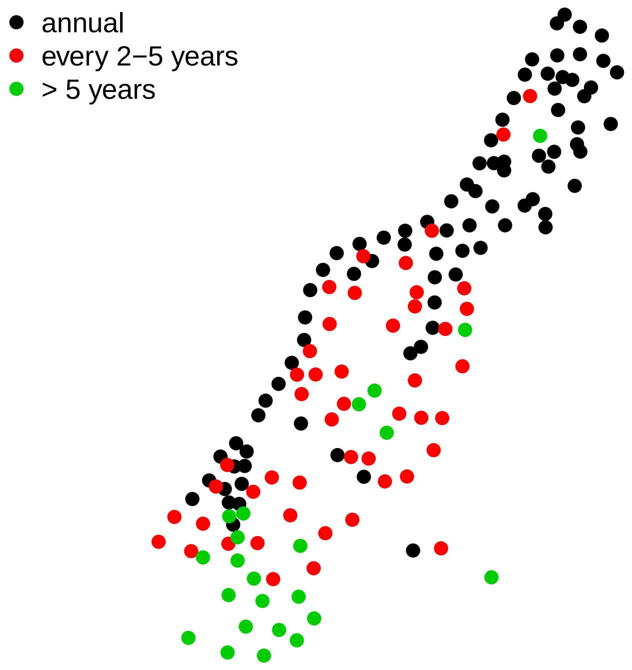
```
> plot(as(meuse1, "Spatial"), axes = TRUE)
> plot(meuse1, add = TRUE)
> plot(meuse1[meuse1$ffreq == 1, ], col = "green", add = TRUE)
```

[7] Don't worry about the reference to projections. We'll get into that soon.



This simple dichotomous representation is nice enough, but we will frequently want to categorize data points. In this next chunk of code, we first create a numeric object based on flood frequencies. We then plot these frequencies using default colors. The last three lines of code create a legend for our map.

```
> meuse1$ffreq1 <- as.numeric(meuse1$ffreq)
> plot(meuse1, col = meuse1$ffreq1, pch = 19)
> labs <- c("annual", "every 2-5 years", "> 5 years")
> cols <- 1:nlevels(meuse1$ffreq)
> legend("topleft", legend = labs, col = cols, pch = 19, bty = "n")
```



We can have a lot more control over class intervals and colors. The classInt package can be used to break up continuous values into categories. RColorBrewer is a good source for color palettes. [8]

```
> library(classInt)
> library(RColorBrewer)
```

We first define class intervals to divide up our data. We can use simple fixed values, quantiles and natural breaks. We then assign those intervals to a set of colors.[9] Here we create a set of 5 cateories based on quantiles of the zinc ppm variable, and define a color palette.

```
> q5 <- classIntervals(meuse1$zinc, n = 5, style = "quantile")
> q5
```

```
style: quantile
  one of 14,891,626 possible partitions of this variable into 5 classes
  [113,186.8) [186.8,246.4) [246.4,439.6) [439.6,737.2)  [737.2,1839]
           31            31            31            31            31
```

```
> pal <- brewer.pal(3, "Blues")
> pal
```

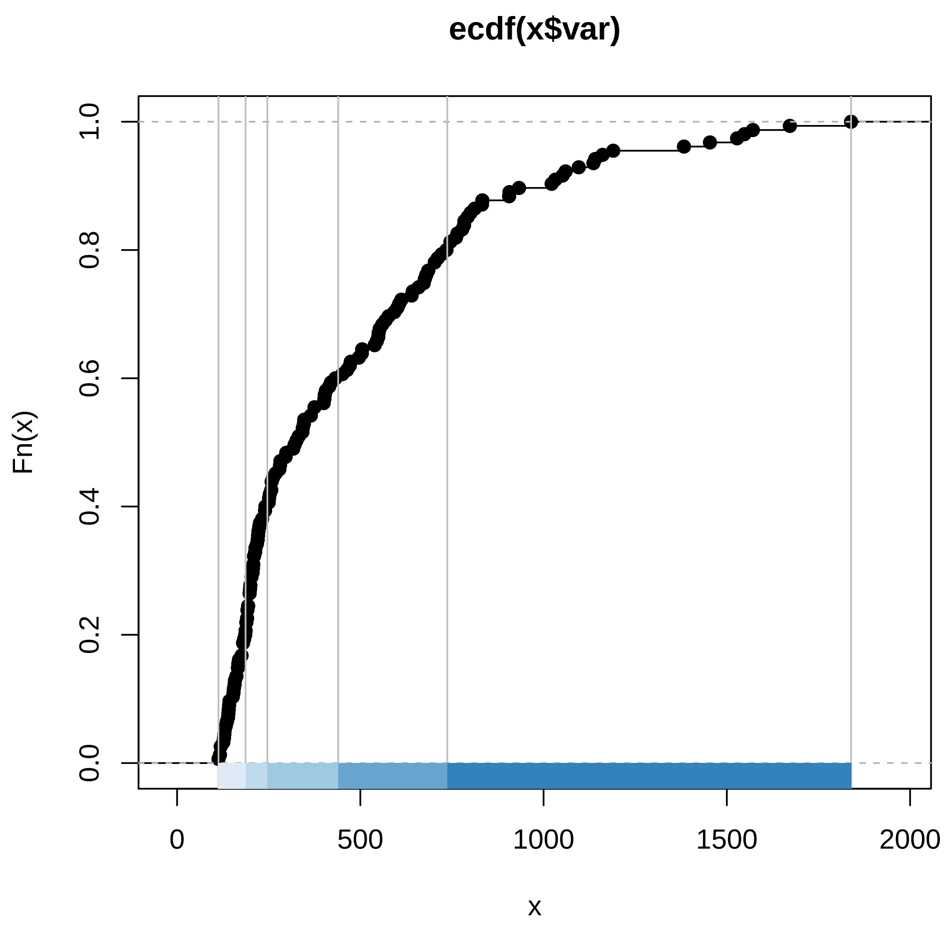
```
[1] "#DEEBF7" "#9ECAE1" "#3182BD"
```

[8] Type in *example(brewer.pal)* for info. Use *colorRampPalette()* to create shades of single color for a quantitative variable.

[9] The choice of symbols and colors for spatial analysis is a field unto itself and well worth exploring in more detail than is possible here.

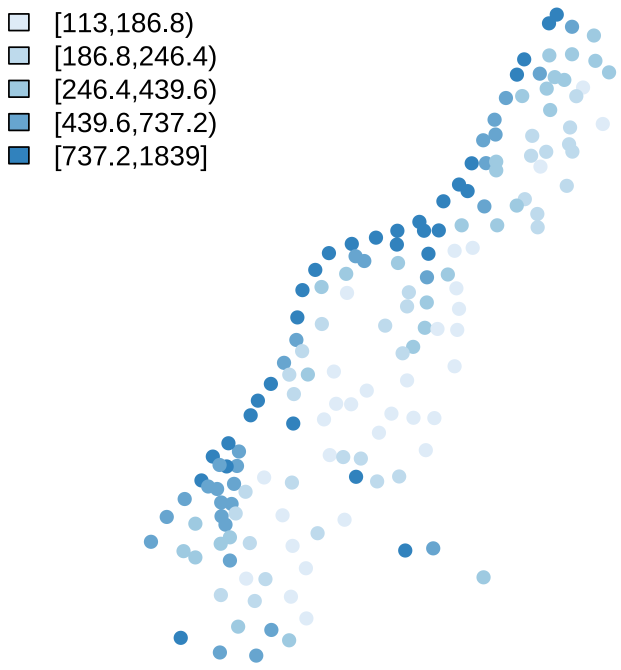
We can plot the quantiles against the color palette to get a sense of how they will break out:

```
> plot(q5, pal = pal)
```



Finally, we define a color variable based on the quantiles of zinc concentration and our choice of color palette, and plot the zinc concentrations with an informative legend.

```
> q5Colours <- findColours(q5, pal)
> plot(meuse1, col = q5Colours, pch = 19)
> legend("topleft", fill = attr(q5Colours, "palette"), legend = names(attr(q5Colours,
+     "table")), bty = "n")
```

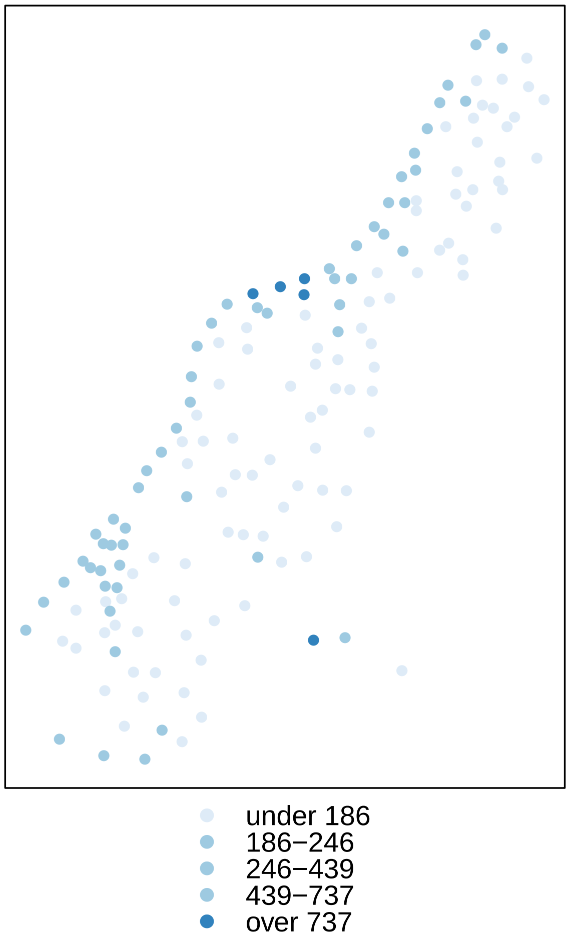


1.4.0.1 spplot()

As with most things in R, there are usually more than one way to do something. The *spplot()* method uses the lattice package, which is the R implementation of the Trellis graphic system. The first argument is a spatial object, second is the variable to plot (the default is to plot all the variables). Use *sp.layout()* to control points, lines, annotations (it takes a list as an argument). Use *layout* = , *skip* = to control how the panels are arranged.

Here, we plot the zinc parts per million variable using the q5 categories and pal color palette we created above.

```
> meuseZincSP <- spplot(meuse1, "zinc", at = q5, col.regions = pal,
+     legendEntries = c("under 186", "186-246", "246-439", "439-737",
+         "over 737"))
> print(meuseZincSP)
```



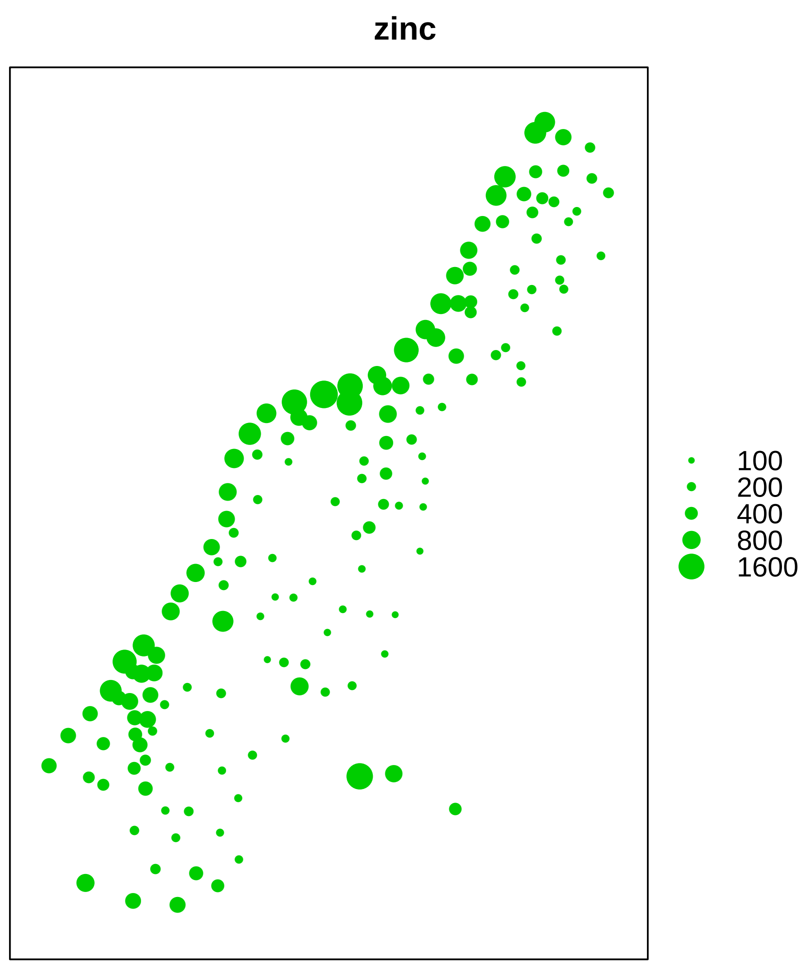
When using spplot(), pass the argument *pretty* = *TRUE*, to ensure colour breaks coincide with legend values.

1.4.0.2 Bubble plots

Bubble plots are a way of representing value by the size of a symbol. They are most conveniently available using the lattice package:[10]

```
> library(lattice)
> bubble1 <- bubble(meuse1, "zinc", maxsize = 2, key.entries = 100 *
+     2^(0:4))
> print(bubble1)
```

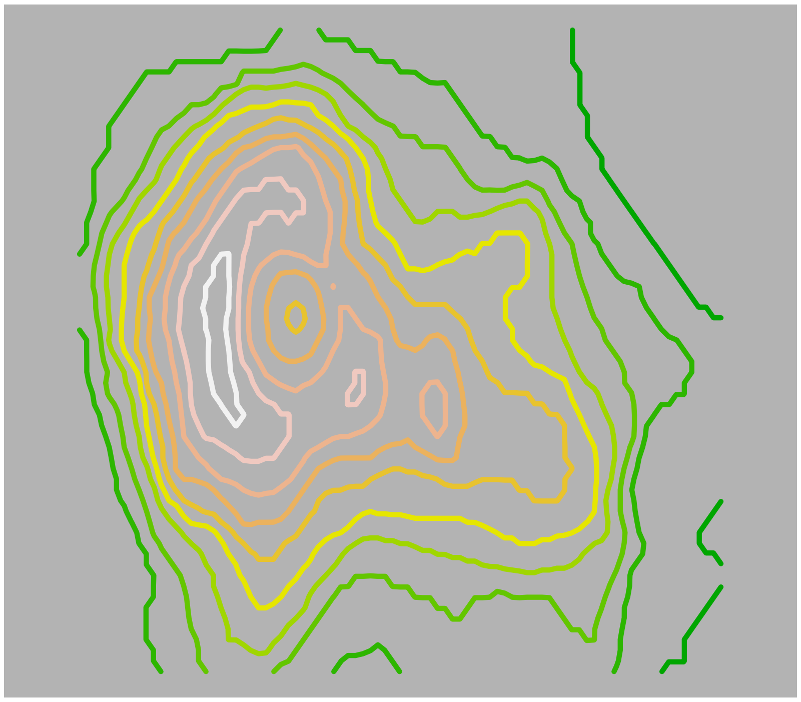
[10] There is code for the plot() methods that I've used to create bubble plots from areal data. It involves extracting the centroids from areas and treating them as points.



1.4.1 Plotting contour lines and gridded data

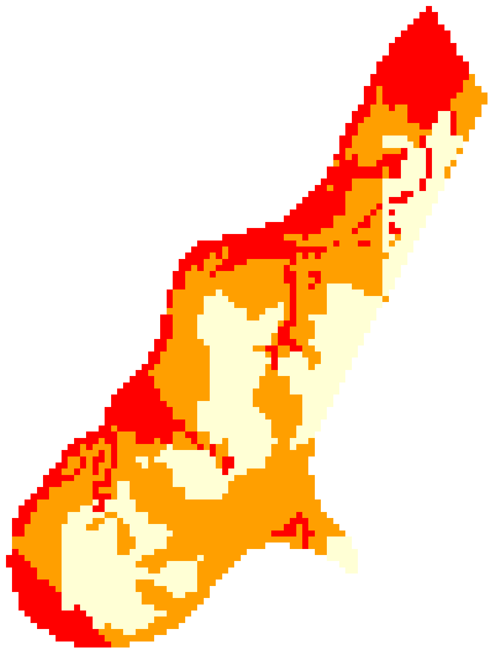
Plotting contour lines, which often represent elevation, is a relatively straightforwad process. Here we plot a volcano data set that comes with the maptools package. We begin by taking advantage of a function called *countourLines2SLDF* that converts a list of contours to a SpatialLinesDataFrame object. Then we apply the plotting techniques we used for points.

```
> library(maptools)
> volcano_sl <- ContourLines2SLDF(contourLines(volcano))
> volcano_sl$level1 <- as.numeric(volcano_sl$level)
> pal <- terrain.colors(nlevels(volcano_sl$level))
> plot(volcano_sl, bg = "grey70", col = pal[volcano_sl$level1],
+      lwd = 3)
```



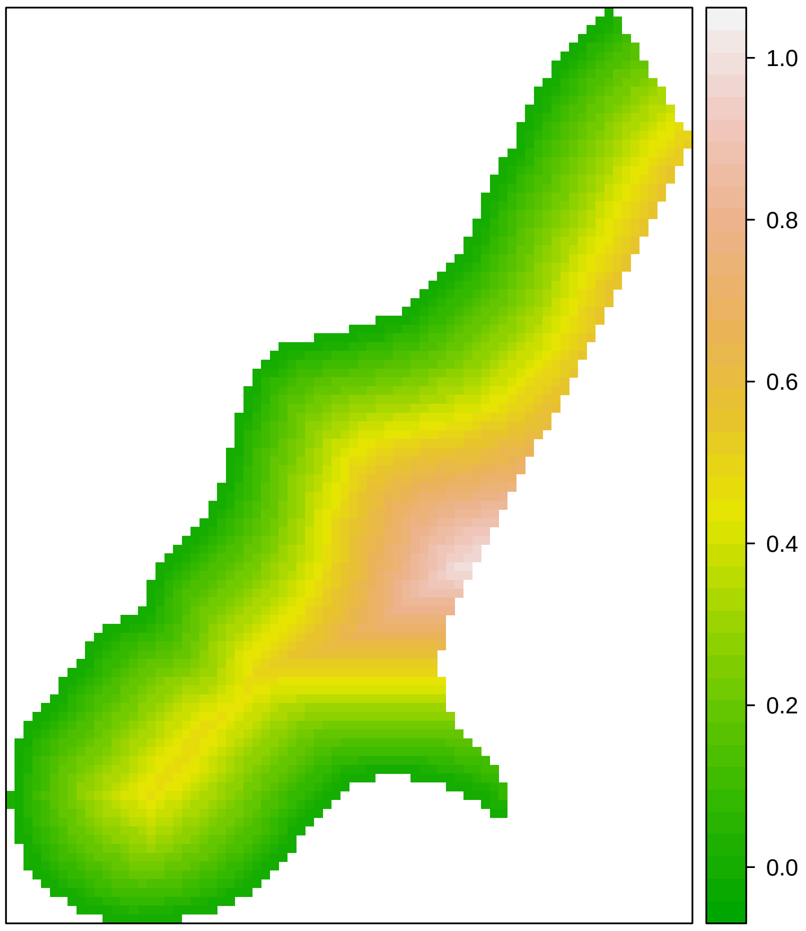
Raster data is displayed as an sp class grid object. Here we create a spatial grid object from the meuse data and then plot a raster image of flood frequencies using the *image()* method.

```
> data(meuse.grid)
> coords <- SpatialPixels(SpatialPoints(meuse.grid[, c("x", "y")]))
> meuseg1 <- SpatialPixelsDataFrame(coords, meuse.grid)
> meuseg1$ffreq1 <- as.numeric(meuseg1$ffreq)
> image(meuseg1, "ffreq1")
```



You can get a sense of some of the advantages of the spplot() method over the plot() method by comparing the above plot to the following one. Note particularly the automatic generation of the legend.

```
> bpal <- colorRampPalette(pal)(41)
> gridPlotSP <- spplot(meuseg1, "dist", col.regions = bpal, cuts = 40)
> print(gridPlotSP)
```



Now that we have a sense of how R spatial data are structured, can be created, and how they can be plotted, we can move on to the more likely scenario of entering, transferring and minipulating existing spatial data.

Chapter 2

Data and Tools

2.1 Getting spatial data into and out of R

Spatial data import and export is intimately tied to Coordinate Reference Systems (CRS). [1] CRS are needed to so things like calculate distances, and (critical for spatial epidemiology) combining maps and data from different sources. They consist of a set of definitions for the *reference ellipsoid* (a mathematical representation of the shape of the earth), *datum* (a set of reference points on the earths surface against which measurements are made), the unit of measurement (e.g. meters, miles) and *projection* (how the spherical earth is represented in two-dimensions, e.g. equirectangular, cylindrical or Mercator's , conical) for geographic data. Many modern data sources are standardized to the *WGS84* coordinate frame. The *PROJ.4 library* of projections is used with CRS's to translate between one projection and another.

After installing the programs and packages described below, you will have a number of tools at your disposal to read in and convert spatial data files. For example, *char2dms()* will convert characters to degree-minute-seconds. You can then use *as()* to coerce degree-minute-seconds to numeric decimal degrees. Maps from the *maps()* package can be converted to *sp* objects with *map2SpatialPolygons()* function in the *maptools* package.

One type of file with which you should become familiar is the ESRI *shapefile*. This common exchange format uses at least three different kinds of files to represent data, **.shp* (geometries) **.shx* (index to geometries) and **.dbf* (DBF III database or attribute data).

Spatial data import and export in R is made immeasurably easier with the *rgdal()* package, (Geospatial Data Abstraction Library) which interfaces with the open-source GIS GRASS. [2] GRASS and rgdal both use the PROJ4 projections library. The *spgrass6()* package allows communication between R and GRASS. [3] Installation is a bit of a bear and involves installing GDAL, GRASS, PROJ4, rgdal and spgrass.

2.1.1 Installing GDAL, GRASS, PROJ4, rgdal() and spgrass6()

These instructions are for Mac OS X.[4] They are based on some hard-won knowledge culled from several incredibly helpful online sources.

[1] I know I promised to stay away from jargon, but it's unavoidable here.

[2] Much more on GRASS soon

[3] Don't worry about this alphabet soup of packages yet. If you need to use them, they will all become familiar soon enough. Like character names in a Tolstoy novel.

[4] If you are one of the many folks who live in the PC world, take heart. By all accounts, the process is easier and there is much more online documentation available. Use these instructions as a way to guide yourself to the tools for a Windows installation. If you use one of the Linux distributions, be happy. All these tools were created by folks who live in your world, so installation should be even easier.

2.1.1.1 Install GDAL and associated frameworks for OS X

You first need to install frameworks files [5] into the /Library/Frameworks folder that allows UNIX programs to work in OS X.

Go to William Kyngesburye's site and link to the frameworks page Download and install the required frameworks: (1) GDAL Complete, (2) FreeType and (3) Cairo frameworks.

William makes this process relatively painless.

2.1.1.2 Install PROJ4

This needs to be installed from source. Download it from the PROJ4 site. Then fire up your terminal and install it with the following series of commands:

```
cd ~/Downloads/
tar -xzvf proj-4.7.0.tar.gz
cd proj-4.7.0
./configure
make && make test
sudo make install
```

This should install the file to your /usr/local/lib directory.

2.1.1.3 Install GRASS

As of this writing, the current version of GRASS is 6.4. Version 6.3 is still available, notably on Lorenzo Moretti's site. I like the 6.3 interface, and Lorenzo has a nice 5 minute tutorial, but rgdal() seems to really want 6.4.

You can get GRASS 6.4 (again) from William Kyngesburye's site. You will need to need to install both the GRASS package and the GRASS GDAL PLUGIN that comes on the disk image. Thanks to William (again) the process is relatively painless. After installation, create a folder called "grassdata" in your home directory, e.g /Users/Charlie/grassdata/. Download the ubiquitous spearfish tutorial zip file so you have some data to play with immediately.

2.1.1.4 Install rgdal()

This file also has to be installed from source because it needs some special configuration utilities. You can do this with *install.packages()* or right from your terminal. I generally use the terminal. Go to this site and download the latest version of rgdal_0.X-X.tar.gz. Start up your terminal, change directories to the folder to which you downloaded the tar.gz file, and enter this syntax (all one line) to install it: [6]

```
R CMD INSTALL --configure-args="--with-gdal-config=/Library/Frameworks/GDAL.framework/unix/bin/gdal-conf
```

[5] Frameworks are bundles or sets of files that programs can access as a kind of shared library

[6] In an additional bit of annoyance, I recently found out that the latest version of rgdal will only work under 64bit R. If you get an error message like, âĂŸrgdalâĂŹ is not installed for 'arch i386' , That's what's probably happening. You will need to run it under a 64 bit version of R (whether your machine is actually *running* under a 64 bit kernel or not) to get it to work. Go figure.

2.1.1.5 Install spgrass6()

Another package that needs to be installed from source. This one, though, can be installed from within R. Download the source file from CRAN and then, within R, type:

```
install.packages("spgrass6", type="source", dependencies = TRUE)
ibrary(sp)
library(rgdal)
```

2.2 Reading in Vector and Raster Data

After you've installed all the programs you will need to display and analyze spatial data, you can start considering how they can be used to read in the different kinds of spatial data with which you may have to work.

2.2.1 Vector Data

Vector files consist of points, lines and polygons and their associated data attribute tables. OGR Drivers in *rgdal* can read many spatial vector formats. The *readOGR()* function takes two arguments, the data source name (dsn) and and the map layer to be imported. For shapefiles, the dsn is usually name of the directory (or folder in Windows-speak) containing the three (or more) files to be imported [7], and the layer is name of the shapefile without the ".shp" extension) the *writeOGR()* function is used to export vector files, specifying drivers e.g. KML (keyhole markup language) for google earth, or ESRI Shapefile.

If you are unable (or unwilling) to install rgdal() there is still hope. The *read.shape()* function in the maptools package will read in ESRI shape files as well. Maptools has number of other helpful functions. *sp2WB()* exports a SpatialPolygons object as a text file in S-PLUS map format to be imported by WinBUGS

2.2.2 Raster Data

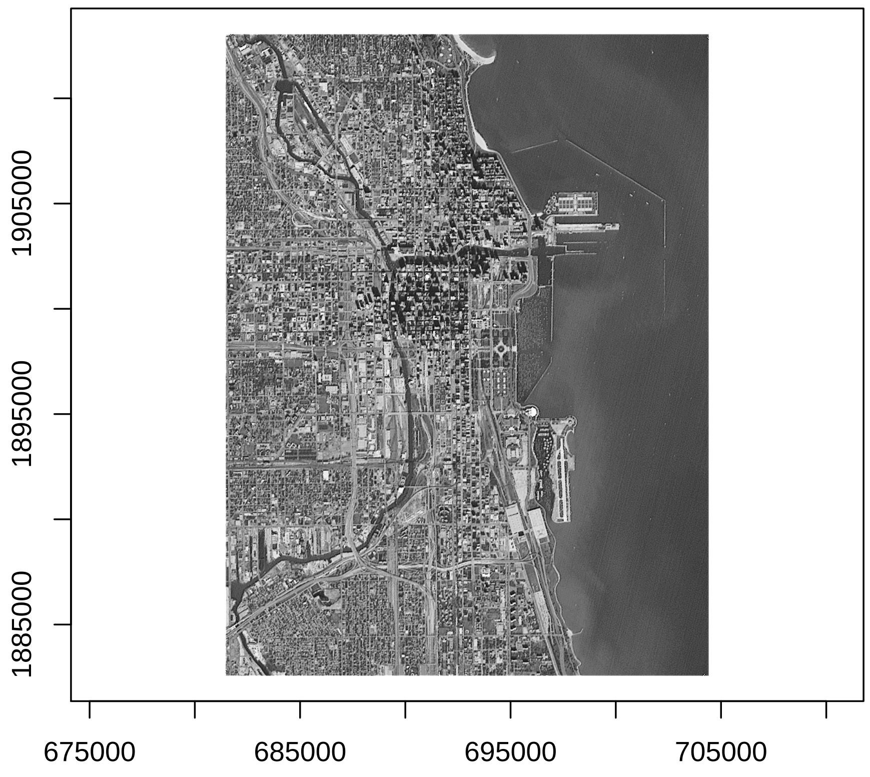
Raster files consist of tessellated grids, each cell of which has a value associated with it. The *readGDAL()* function of rgdal will read in rasters. There are a lot of raster data available on the web. Here is an image downloaded from OSGEO.

```
> library(rgdal)
> sp27gtiff <- readGDAL("/Users/charlie/Dropbox/asdarHowTo/asdarExamples/SP27GTIF.TIF")
```

```
/Users/charlie/Dropbox/asdarHowTo/asdarExamples/SP27GTIF.TIF has GDAL driver GTiff
and has 929 rows and 699 columns
```

```
> image(sp27gtiff, col = grey(1:99/100), axes = T)
```

[7] You can use a dot (".") if the files are in your current working directory



The *writeGDAL()* function will export rasters.

2.2.3 Using Maptools to read in spatial data

As an alternative to rgdal, Maptools has some nice functions to read in spatial data including readShapPoly (for shapefiles ending in .shp), readShapeLines and readShapePoints. They do not, however, read the CRS which can be a drawback if you combining data sources or moving data from R to a GIS. The syntax is simple:

```
library(maptools)
list.files()
scot<-readShapePoly("scot_BNG.shp")
```

For raster files, you can use *readAsciiGrid()* in maptools to reads Arc ASCII grids into a SpatialGridDataFrame objects, though again it does not handle CRS as does readGDAL in the rgdal package.

Another function, *GE_SpatialGrid()*, can be used to create a .png image that can be overlaid on google earth with *image()* and *kmlOverlay()* [8]. There are also functions in maptools for writing sp objects to shapefiles, e.g. writePolyShape.

[8] see pp 97-98 of Bivand

2.3 GRASS and R

GRASS (Geographic Resources Analysis Support System) is a major open source GIS that originated with the US Army. It uses GDAL and OGR as its main import/export engine. Here's the exciting [9] part: you can connect GRASS to R with the spgrass6() package (which itself uses sp ,rgdal etc...).

You start R from within GRASS at the command line by simply typing R and hitting enter. You will then be at an R prompt. Load the spgrass6() library to complete the connection. Type *gmeta6()* to get information on your current GRASS "location" (a GRASS concept for related set of maps, files and their CRS)

The two main spgrass6() functions are the readRAST6() function to read GRASS rasters into R, and the readVECT6() function for importing vector data from GRASS into R.

A typical workflow might consist of using GRASS to import data and do geoprocessing. Then reading the resulting files into R for spatial analysis.

2.4 The Broad Street Pump Example

Roger Bivand has a neat example of using GRASS with R to update John Snow's classic Broad Street pump investigation.[10] At issue is whether cholera is due to miasma or person-to-person transmission. If miasmatic emanations are the culprit, the closer one lives to the Broad Street pump "as the crow flies", the greater the association between straight-line areal distance and cholera-related death. If, though, there were person-to-person transmission of an infectious agent, street-level *walking* distance from the Broad Street pump should be the operative risk factor.

Dr. Bivand has posted data files on his book site. He prepared the files initially by doing the geoprocessing in GRASS. He started with a georeferenced image file of the Broad Street area and vector shapefiles of the buildings, nearby cholera deaths, and the locations of the Broad Street pump and the non Broad Street pumps. He read these into a GRASS location based on the British National Grid CRS. He then used the GRASS *r.cost* module to calculate the walking distances between the site of deaths and the different pumps.

To demonstrate how to access R from GRASS, [11] we begin by starting up GRASS then choosing the rsb mapset of the snow2 location. Start up a monitor and view some of the vector maps in the mapset.

```
system("/Applications/GRASS-6.4.app/Contents/MacOS/grass.sh")
g.list vect
d.mon x0
d.erase
d.vect vsnow4
```

We see that Dr. Bivand has created a georeferenced image file of the Broad Street area and vector shapefiles of the buildings, nearby cholera deaths, and the locations of the Broad Street pump and the non Broad Street pump.

From the GRASS prompt, start R, load the spgrass6() library, and get some information on the current GRASS location:

```
> R
> library(spgrass6)
> gmeta6()
```

Which returns the following information:

```
gisdbase    /Users/charlie/grassdata
location    snow2
```

[9] To some.

[10] See "Applied Spatial Data Analysis" by Bivand, Pebesma and Gomez-Rubio. pp104-106

[11] I have a set of notes on how to use GRASS that make this code more intelligible.

```
mapset      rsb
rows        314
columns     342
north       181380
south       180595
west        528915
east        529770
nsres       2.5
ewres       2.5
projection  +proj=tmerc +lat_0=49.00000000000002 +lon_0=-2 +k=0.999601 +x_0=400000
+y_0=-100000 +no_defs +a=6377563.396 +rf=299.3249646
+towgs84=446.448,-125.157,542.060,0.1502,0.2470,0.8421,-20.4894
+to_meter=1
```

Next, let's read the vector and raster maps into R. Using the readVECT6() function, we import "buildings", a vector map outlining the buildings near the Broad Street pump, and the locations of both the Broad Street and non-Broad Street pumps. We use readRAST6 to import"sohoSG" as an R object containing the two distance rasters (areal distance vs walking distance) which are the variables in which we are interested. Finally, "deaths" is the death coordinates and counts. We save these objects as an R data file that we can load for later analysis.

```
> buildings <- readVECT6("vsnow4")
> nb_pump <- readVECT6("vpump_not_broad")
> b_pump <- readVECT6("vpump_broad")
> sohoSG <- readRAST6(c("snowcost_broad", "snowcost_not_broad"))
> deaths <- readVECT6("deaths3")
> save.image("/Users/charlie/Dropbox/grassHowTo/snowFiles.RData")
```

Now, let's reload the R data file and maptools and do some analyses [12]

We begin by overlaying the death coordinates on the distances file so we can get distances to each house, and create a logical variable in that file indicating if the Broad Street pump was closer or farther away than a non-Broad Street pump in terms of street distance. Then we get the sum of deaths that were nearer and farther away from Broad Street. We see there were more deaths closer to the Broad Street pump (357), than there were farther away (221).

```
> load("/Users/charlie/Dropbox/spatialEpiHowTo/spatialEpiData/spatialEpiData.Rdata")
> library(maptools)
> o <- overlay(sohoSG, deaths)
> deaths <- spCbind(deaths, as(o, "data.frame"))
> deaths$b_nearer <- deaths$snowcost_broad < deaths$snowcost_not_broad
> by(deaths$Num_Cases, deaths$b_nearer, sum)
```

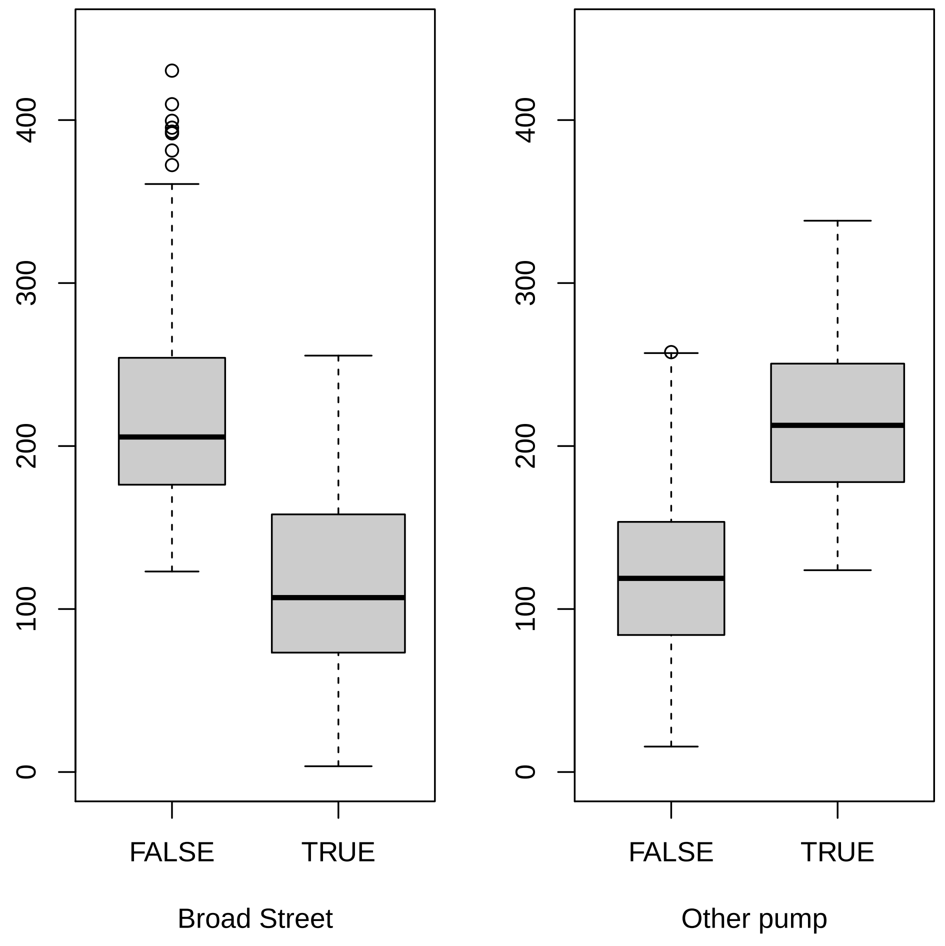
```
deaths$b_nearer: FALSE
[1] 221
------------------------------------------------------------
deaths$b_nearer: TRUE
[1] 357
```

In this next bit of code, we compare box plots for the walking distances for death locations to the Broad Street pump and to non-Broad Street pumps. We note that the interquartile ranges do not overlap, though some deaths occured in folks who lived closer to another pump.

```
> oopar <- par(mfrow = c(1, 2), mar = c(5, 3, 1, 1) + 0.1)
> b_wid <- table(deaths$b_nearer)
> boxplot(snowcost_broad ~ b_nearer, deaths, width = b_wid, ylim = c(0,
+     450), ylab = "distance", xlab = "Broad Street", col = grey.colors(1,
+     0.8, 0.8, 2.2))
> boxplot(snowcost_not_broad ~ b_nearer, deaths, width = b_wid,
+     ylim = c(0, 450), xlab = "Other pump", col = grey.colors(1,
```

[12] We could also, very easily have continued our R session within GRASS

```
+          0.8, 0.8, 2.2))
> par(oopar)
```

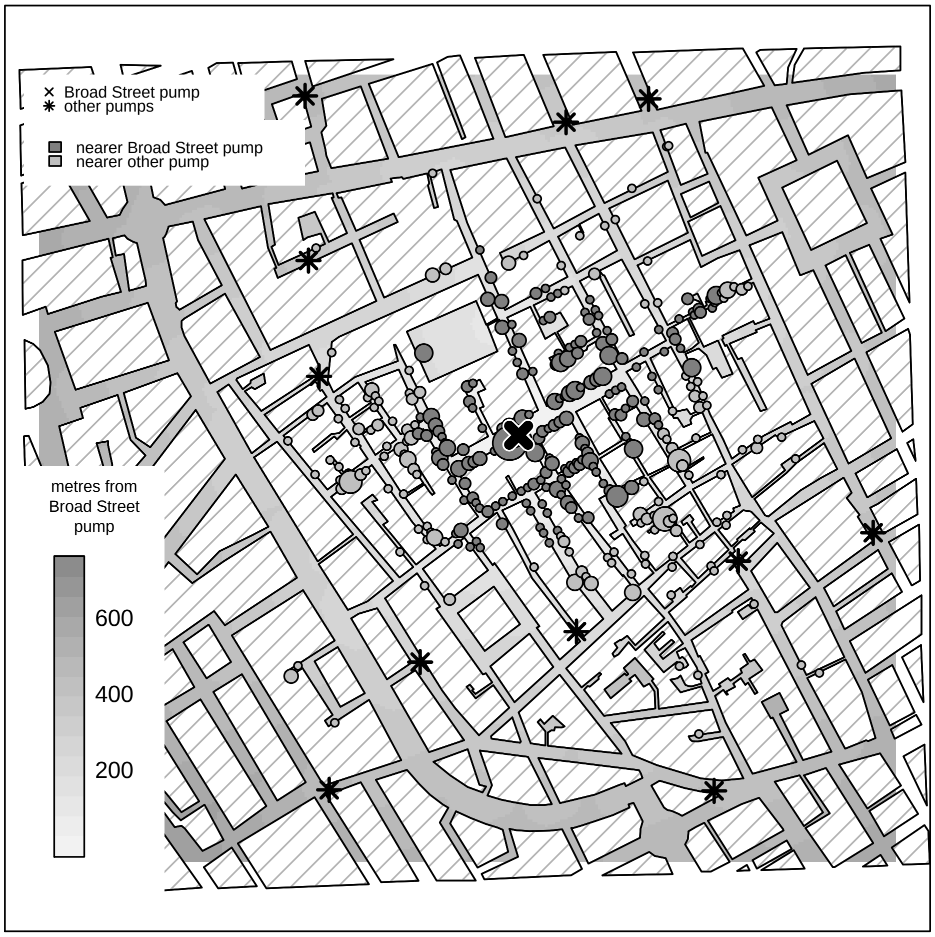


Finally, we plot a map illustrating the number of mortalities by proportional symbols, coded for whether they were closer or farther away from Broad Street with the gray-scaled streets indicating proximity to Broad Street. [13]

```
> oopar <- par(mar = c(1, 1, 1, 1) + 0.1)
> library(RColorBrewer)
> gcols <- grey.colors(15, 0.95, 0.55, 2.2)
> image(sohoSG, "snowcost_broad", breaks = seq(0, 750, 50), col = gcols)
> plot(buildings, col = "white", add = TRUE)
> plot(buildings, angle = 45, density = 10, col = "grey70", add = TRUE)
> symbols(coordinates(deaths), circles = 4 * sqrt(deaths$Num_Cases),
+     inches = FALSE, add = TRUE, bg = c("grey75", "grey50")[deaths$b_nearer +
+         1])
> source("/Users/charlie/Dropbox/spatialEpiHowTo/spatialEpiData/legend_image.R")
> rect(528900, 180550, 529040, 180990, border = NA, col = "white")
> text(528970, 180950, "metres from\nBroad Street\npump", cex = 0.6)
> legend_image(c(528930, 528960), c(180600, 180900), sohoSG$snowcost_broad,
+     vertical = TRUE, breaks = seq(0, 750, 50), col = gcols)
```

[13] The legend image file to which the source() function refers is from the geoR package. Dr. Bivand has a copy of the source code on his site

```
> plot(nb_pump, add = TRUE, pch = 8, cex = 1.3, lwd = 2)
> plot(b_pump, add = TRUE, pch = 4, cex = 1.5, lwd = 8, col = "white")
> plot(b_pump, add = TRUE, pch = 4, cex = 1.5, lwd = 6)
> rect(528900, 181330, 529140, 181380, border = NA, col = "white")
> legend(c(528910, 529100), c(181350, 181380), legend = c("Broad Street pump",
+     "other pumps"), pch = c(4, 8), bty = "n", cex = 0.6, y.inter = 0.7)
> rect(528900, 181270, 529180, 181335, border = NA, col = "white")
> legend(c(528910, 529100), c(181275, 181325), legend = c("nearer Broad Street pump",
+     "nearer other pump"), fill = c("grey50", "grey75"), bty = "n",
+     cex = 0.6, y.inter = 0.7)
> box()
> par(oopar)
```



2.5 The Scottish Lip Cancer Example

Let’s work through another example to illustrate reading in and analyzing spatial data that does not require us to use GRASS or any other GIS.

The Scottish Lip Cancer data set consists of the number of cases of lip cancer in 56 Scottish counties. The data come as shape files with no CRS, so the first order of business is reading in the shape files and defining the CRS.

```
> require(sp)
> require(rgdal)
> scot_LL <- readOGR("/Users/charlie/Dropbox/spatialEpiHowTo/spatialEpiData/scot/scot.shp",
+     "scot")
```

```
OGR data source with driver: ESRI Shapefile
Source: "/Users/charlie/Dropbox/spatialEpiHowTo/spatialEpiData/scot/scot.shp", layer: "scot
with 56 features and 2 fields
Feature type: wkbPolygon with 2 dimensions
```

```
> proj4string(scot_LL) <- CRS("+proj=longlat ellps=WGS84")
> scot_LL$ID
```

```
 [1] 12 13 19  2 17 16 21 50 15 25 26 29 43 39 40 52 42 51 34 54 36 46 41 53 49
[26] 38 44 30 45 48 47 35 28  4 20 33 31 24 55 18 56 14 32 27 10 22  6  8  9  3
[51]  5 11  1  7 23 37
```

```
> scot_dat <- read.table("/Users/charlie/Dropbox/spatialEpiHowTo/spatialEpiData/scot/scot.
+     skip = 1)
> names(scot_dat) <- c("District", "Observed", "Expected", "PcAFF",
+     "Latitude", "Longitude")
> scot_dat$District
```

```
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
[51] 51 52 53 54 55 56
```

```
> library(maptools)
> scot_dat1 <- scot_dat[match(scot_LL$ID, scot_dat$District), ]
> row.names(scot_dat1) <- sapply(slot(scot_LL, "polygons"), function(x) slot(x,
+     "ID"))
> scot_LLa <- spCbind(scot_LL, scot_dat1)
> all.equal(scot_LLa$ID, scot_LLa$District)
```

```
[1] TRUE
```

```
> names(scot_LLa)
```

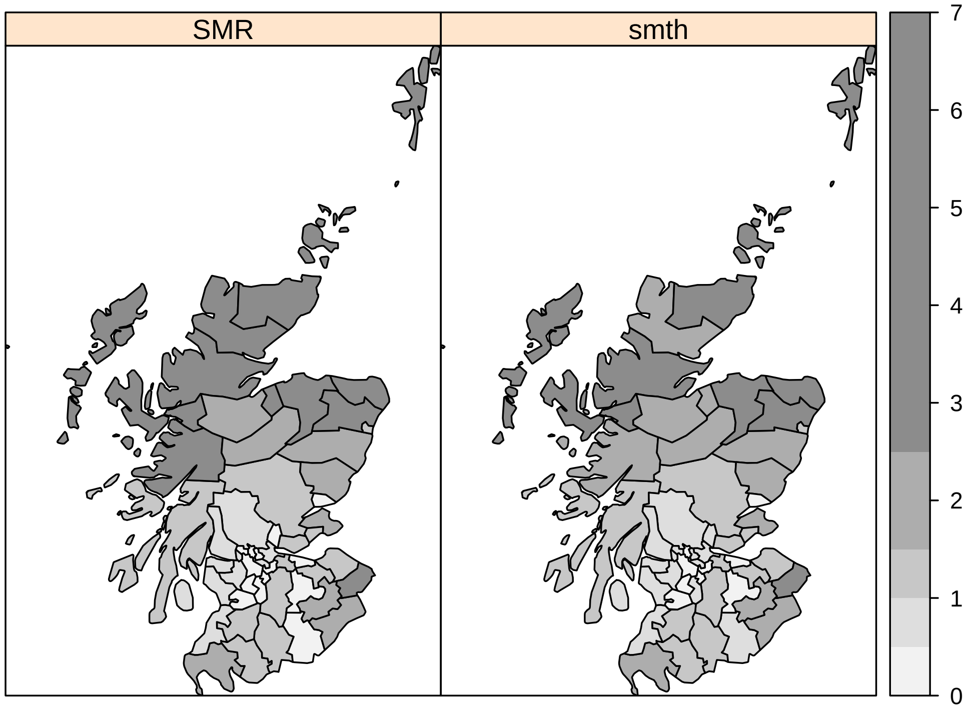
```
[1] "NAME"      "ID"        "District"  "Observed"  "Expected"  "PcAFF"
[7] "Latitude"  "Longitude"
```

In the following code we calculate both unadjusted and empirical Bayesian-smoothed SMRs. Our interest here is simply in the mechanics of getting data into R and mapping it. We won't go into the details of the analysis just now. Suffice it to say that that unadjusted estimates are potentially unstable. The Bayesian estimates are "smoothed" or shrunken toward the global estimate.

```
> library(spdep)
> O <- scot_LLa$Observed
> E <- scot_LLa$Expected
> scot_LLa$SMR <- probmap(O, E)$relRisk/100
> library(DCluster)
> scot_LLa$smth <- empbaysmooth(O, E)$smthrr
```

Now, we reproject the map file to the British National Grid CRS (which will be needed to export the file later) and plot the smoothed rates.

```
> scot_BNG <- spTransform(scot_LLa, CRS("+init=epsg:27700"))
> library(lattice)
> library(RColorBrewer)
> ratesMap <- spplot(scot_BNG, c("SMR", "smth"), at = c(0, 0.5,
+     1, 1.5, 2.5, 7), col.regions = grey.colors(5, 0.95, 0.55,
+     2.2))
> print(ratesMap)
```



Finally, we export the data in two formats using *writeOGR()*. First as a JLM (Keyhole Markup Language) file that can be overlaid on a Google Earth image which uses a WGS84 datum. Then we export the centroids as a SpatialPointsDataFrame object, which consists of the district name, the observed and expected values and the unadjusted and adjusted SMRs.

```
> writeOGR(scot_LLa["ID"], dsn = "scot_district.kml", layer = "borders",
+     driver = "KML")
> llCRS <- CRS("+proj=longlat ellps=WGS84")
> scot_SP_LL <- SpatialPointsDataFrame(coordinates(scot_LLa), proj4string = llCRS,
+     data = as(scot_LLa, "data.frame")[c("NAME", "Observed", "Expected",
+         "SMR", "smth")])
> writeOGR(scot_SP_LL, dsn = "scot_rates.kml", layer = "rates",
+     driver = "KML")
> drv <- "ESRI Shapefile"
> writeOGR(scot_BNG, dsn = ".", layer = "scot_BNG", driver = drv)
```

2.6 Spatial data manipulation

While the heavy lifting of geoprocessing is most properly done in a GIS like GRASS, R has tools to allow you to complete some tasks directly.

2.6.1 Overlaying spatial data

The *sp* package *overlay()* function matches one kind of spatial object with another, for example points within spatial polygons. In the following example, ‘x’ could be a SpatialGridDataFrame object of elevation values and ‘y’ a SpatialPoints object of transect values. We can then query grid cells of the resulting the ‘x_y’ grid SpatialPointsDataFrame object.

```
x_y <- overlay(x,y)
```

2.6.2 Combining spatial data

The function ‘spRbind()’ combines positional data objects with matching column names, such as two SpatialPoints objects or two SpatialPointsDataFrame objects. It only takes two arguments so you need to run it multiple times to bind more than two spatial objects. Note that it does not check for duplicates.

Chapter 3

Areal Data and Spatial Neighbors

3.1 About Areal data

Areal data are, in epidemiological terms, essentially ecological data. More often than not, in the spatial epidemiology work I've done at least, I find myself working with these with these so-called "areal units" rather than with the discrete point processes against they are distinguished. Partly because they are more readily available and easier to collect (e.g. state-level data, ZIP codes) and partly because they are the kinds of data I am most likely to be interested.[1] Areal spatial units are polygons. These are ecological entities in epidemiological terms, so the areal entities are themselves are the units of analysis.

Areal data are subject to many analytic difficulties. They are characterized by irregular geographic categorizations like ZIP code tabulation areas (ZCTAs). They also are likely to have a high degree of autocorrelation, with entities mirroring and influencing each other, for example by adopting similar policies and practices in response to similar issues as nearby areal units. Frequently, the way the areal units themselves are defined, such as their shapes and boundaries, is arbitrary and not related to the spatial distribution of the problem under analysis. They may more properly be thought of as convenient "bins" into which spatial data are gathered, rather than as objects of interest in themselves. When arbitrary boundaries are used to divide a study area, positive spatial dependence (proximate observations can in part be used to predict each other) reduces information, raising the related issue of how many effectively independent observations are actually present.

Areal units are prone to all the biases of other ecologic data, such as the well-known ecologic fallacy, as well as to some less well known biases peculiar to spatial data, such as the so-called *modifiable areal unit problem*, where a change in how we define an areal unit can affect and even reverse the direction of our results. A commonly known example of a modifiable area unit bias, is gerrymandering political districts to enhance the fortunes of a political party.

With spatial points processes, essentially all points can be considered are neighbors of each other with the weight of influence decreasing by distance. We can control for these distance-based weights with variagrams. With areal units, we define neighbors using rules like contiguity. We can give each neighbor relationship equal weight, or vary the weights describing this spatial dependence. It is essentially a two step-process: define neighbors , assign weights. A number of tools are available in R to help us define and weight neighbors, such as the *nb2neig()* and *neig2nb()* functions in either the *spdep* package or the *ade4* packages respectively .

I tend to work with the *spdep* package which has a *nb* class to define neighbors. The nb class is a list object with an index vector of neighbors for each unit, (the number zero is, common sensically enough, no neighbors), character

[1] In fact, I prefer to work with areal units. I find point-process data to be somewhat unsatisfying when working with health outcomes. It assumes a level of precision in terms of geographic location that I find inconsistent with the way health outcomes are actually collected. Also, outside of exposures that are clearly related to geographic proximity, such as environmental exposures with which I rarely work, I find the degree of statistical sophistication required for point-process work is rarely warranted. I prefer to leave those analyses to folks who are more familiar and comfortable with them

identifiers , and total (cardinal) number neighbors for each area unit on a map. [2] We can use the *card()* function and *summary()* or *print()* methods to get at an informative list of neighbor objects.

3.2 Kinds of spatial neighbors

3.2.1 Contiguity neighbors

There are a couple of ways to define neighbors, the first and perhaps most common is based on contiguity, i.e. the two neighboring areal entities share a border.

The *poly2nb()* function in spdep takes a SpatialPolygons object as its first argument and returns a list of neighbors. The optional “snap” argument allows control over how close 2 polygons with a shared border need to be to qualify as a neighbor. Use “queen=TRUE”, [3] to define a neighbor when at least *one* point on the boundary of one polygon is within the snap distance of at least *one* point of its neighbor. Note that when more than 2 polygons meet at a single point, they all meet this contiguity condition, giving rise to crossed links. With the stricter “queen=FALSE” or “rook” relationship, at least *two* boundary points must be within the snap distance of each other.

Alternatively, we could export a SpatialPolygonsDataFrame object to a GIS such as GRASS or ArcGIS, and use the topology engine in the GIS to find contiguities in the graph of polygon edges.

3.2.2 Graph-based neighbors

We can apply graph theory and use centroids (which can be population weighted) to define neighbors. The *tri2nb()* function in the *tripack* package uses triangulation methods to calculate the distances between areal centroids.

There are a few approaches to defining neighbors and establishing their distance from each other. Delaunay triangulation [4] is based on the location of points on a tangent sphere. It maximizes the angles of the triangles in the triangulation, avoiding skinny triangles.

A Sphere-of-Influence (SOI) graph is a subset of a Delaunay graph that removes links that are based on relatively long distances. Points are SOI neighbors “if circles centered on the points, are of a radius equal to the points’ nearest neighbour distances, and intersect in two places” (Avis and Horton, 1985)

Finally, Gabriel graphs are a further restriction on Delaunay graphs where two points are neighbors only if there are no other points in their line set.

[2] Depending on the method, a neighbor may be asymmetric i.e. ‘a’ may be a neighbor of ‘b’, but ‘b’ doesn’t have to be a neighbor of ‘a’. The unrequited love of geography.

[3] The term comes from chess...

[4] Triangulation is one of those terms that gets bandied about. It has a very specific meaning as a way to determine the distance between two points using the angles in a triangle, rather than physically measuring the distance. The concept is actually quite clever and elegant. Imagine you are at some point and there is a far-off object that you want to know the distance to. Begin by measuring off a short line parallel to the distant object, through the point on which you are standing, using a unit like meters. Next, measure the angles of two lines from the ends of your parallel baseline directed toward the distant object. On a piece of paper (now-a-days on a computer...) draw your partial triangle, and extend the lines. Measure the distance from the point at which they meet, to your original baseline. That’s the distance from where you are to the object. An exceedingly simple, but rather effective YouTube video illustrates the process nicely.

Fig. 3.1 Delauny Graph

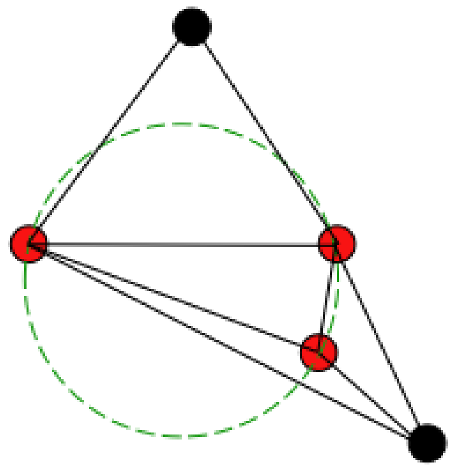
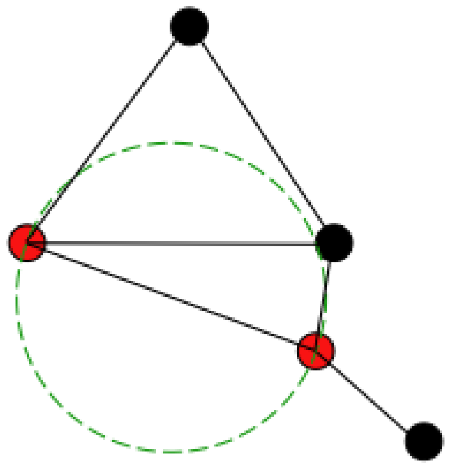


Fig. 3.2 Gabriel Graph



3.2.3 *Distance-Based Neighbors*

Distance-based approached may be useful if areal entities are (approximately) regularly spaced. In this approach you select the k nearest points as neighbors (where k is number neighbors you want for esch unit). The function *knearneigh()* returns an intermediate form which is converted to an "nb" object with *byknn2nb()*. *knearneigh()* can also take a longitude / latitude argument to handle geographical coordinates.

Running *knearneigh()* with k =1 is useful to get the minimum distance at which all areas have a distance-based neighbor. With *nbdists()* we can calculate the list of vectors of distances. The greatest value will be the minimum distance needed to make sure all areas are linked to at least one neighbor.

We can use *dnearneigh()* to find neighbors with some inter-point distance. The arguments d1 and d2 set the lower and upper distance bounds (they can take longitude and latitude as parameters). The number of distance-based neighbors will (necessarily) increase with increasing distance.

3.2.4 Other Approaches

You can define a set of higher-order neighbors (i.e. two or more links away from the origin) with correlograms. You input a list of neighbors as first-order sets, then step out across a resulting graph to second-, third-, and higher-order neighbors based on number of links traversed. The function *nblag()* takes an existing neighbor list and returns a list of lists, from first to the maximum lag. You can also create a regular, rectangular grid of neighbors using the function *cell2nb()*

3.3 New York City ZIP Code Neighbors

3.3.1 The New York City ZIP Code Tabulation Area Data Set

In the United States, a lot of health-related data can be obtained for the ZIP Code Tabulation Area (ZCTA) level. I most often work with New York City data, so the first step in my spatial analyses is to create a geo-referenced New York City ZCTA map.[5]

In GRASS, I imported a shape file of census tracts and then geoprocessed them to create a shape file of ZIP Code Tabulation Areas.[6] In an emacs terminal session [7] I fire up GRASS, and use the spgrass6 library to import the ZIP Code file into R. I save the file as an R data object for later loading.

```
system("/Applications/GRASS-6.4.app/Contents/MacOS/grass.sh")
g.list vect
R
library(spgrass6)
nycZIPS <- readVECT6("nycZips")
save(nycZIPS,file="/Users/charlie/Dropbox/spatialEpiHowTo/spatialEpiAreal/spatialEpiAreal.RData")
q()
exit
```

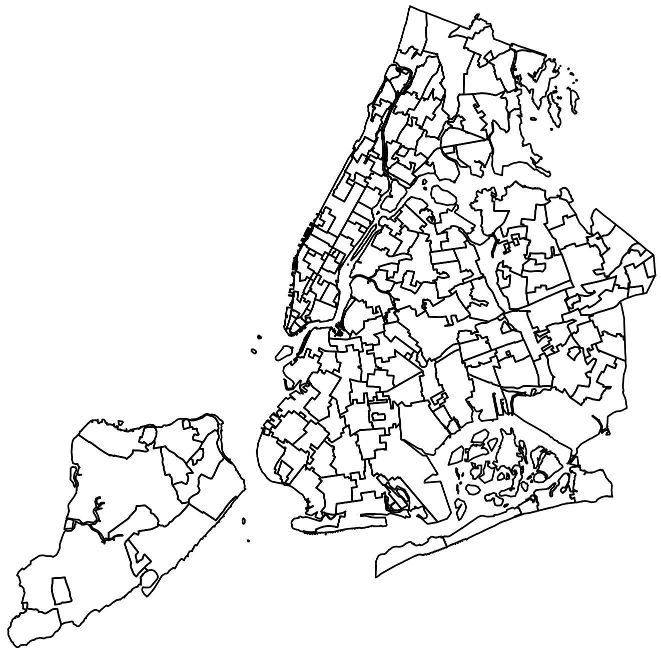
Now we switch back to R exclusively. Load the spatial data file called 'nycZIPS' which we created in GRASS. Open the sp library and plot the nycZIPS object which represents the ZIP Code tabulation areas in New York City.

```
> load("/Users/charlie/Dropbox/spatialEpiHowTo/spatialEpiNeighbors/spatialEpiNeighbors.RDat
> library(sp)
> plot(nycZIPS)
```

[5] There is, to my knowledge anyway, no standard such map. I have created a number of these ZCTA maps over the years with varying degrees of success. The approach I describe here is so far the most accurate.

[6] This is the kind of thing for which GIS is made. I can get this GRASS code for this process to you if you need it.

[7] I recently realized this little "R to GRASS to R" trick only works with emacs...



It's always gratifying to see a procedure work. This map of New York City ZCTAs' appears reasonably complete. But, notice those little spatial polygons in the northeast off the 'coast' of northern Manhattan and the Bronx, and in the southwest between Staten Island and Manhattan. They may prove problematic. To continue our due diligence at this point, we look at the dataframe object itself. We are interested at this point in the ZCTA's, so let's start looking at them. Here's where knowing a little R makes the transition to spatial analysis easier. Although it's a spatial object, we can treat it as a dataframe , which allows us to run some simple summary procedures.

In the next few lines of code, we determine that there are 295 ZCTA's in the spatial object. This is, immediately, a problem that needs to be addressed. There are, in fact, a couple of conflicting sources about the actual number of ZCTA's in New York City. None of them are close to the 295 we have in our map. One source states there are 176 ZCTA's in New York City (41 in Manhattan, 37 in Brooklyn, 61 in Queens, 25 in the Bronx, and 12 on Staten Island). Other sources cite a total of 177 or 178. Why the discrepancies? This is part of the problem of using administrative areal units. ZCTA's are, in essence, a filing system for the US Postal Service. They may not behave 'nicely' in an analytic sense. So, while we think of ZIP Codes as geographic entities, in New York City, some *buildings* in Manhattan (like the Federal Reserve Bank at 33 Liberty Street) have their own ZIP Codes, some ZIP Codes are reserved for Post Office Boxes, and the World Trade Center 'footprint' retains its former ZIP Code of 10048, which is being reserved for future use. As I said, areal data can be messy. We can, though, wrestle them to acceptably reliable and valid uses.

```
> length(nycZIPS$zcta5)
```

```
[1] 295
```

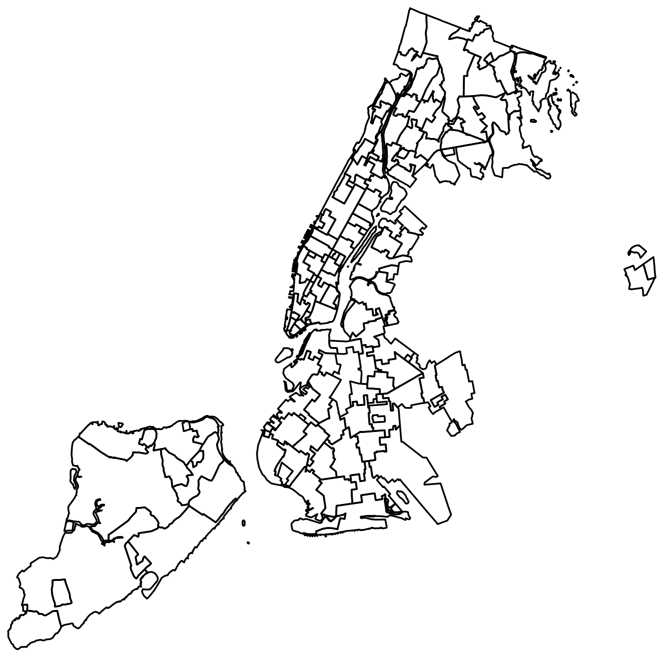
To begin, we see that a number of ZIP Codes are repeated, some many times.

```
> table(nycZIPS$zcta5)
```

```
10001 10002 10003 10004 10007 10009 10010 10011 10012 10013 10014 10016 10017
    1     1     1     2     2     1     2     2     1     1     1     1     2
10018 10019 10021 10022 10023 10024 10025 10026 10027 10028 10029 10030 10031
    2     1     1     1     1     1     1     1     1     1     1     1     1
10032 10033 10034 10035 10036 10037 10038 10039 10040 10044 10048 10069 10128
    1     1     1     2     1     1     2     1     1     3     1     1     1
10162 10171 10280 10282 10301 10302 10303 10304 10305 10306 10308 10309 10310
    1     1    11     1     3     1     2     1     1     3     1     2     1
10312 10314 10451 10452 10453 10454 10455 10456 10457 10458 10459 10460 10461
    2     2     1     1     1     1     1     1     1     1     1     2     1
10462 10463 10464 10465 10466 10467 10468 10469 10470 10471 10472 10473 10474
    1     1    21     1     1     1     1     2     1     1     1     2     1
10475 11004 11005 11040 11101 11102 11103 11104 11105 11106 11201 11203 11204
    3     1     1     1     2     1     1     1     1     1     1     1     1
11205 11206 11207 11208 11209 11210 11211 11212 11213 11214 11215 11216 11217
    1     1     1     1     1     1     1     1     1     1     1     1     1
11218 11219 11220 11221 11222 11223 11224 11225 11226 11228 11229 11230 11231
    1     1     1     1     1     2     1     1     1     1     1     1     1
11232 11233 11234 11235 11236 11237 11238 11239 11354 11355 11356 11357 11358
    1     1     1     3     1     1     1     1     2     1     1     1     1
11360 11361 11362 11363 11364 11365 11366 11367 11368 11369 11370 11372 11373
    1     1     1     1     1     1     1     1     1     1     3     1     1
11374 11375 11377 11378 11379 11385 11411 11412 11413 11414 11415 11416 11417
    1     1     1     1     1     1     1     2     1     1     2     1     1
11418 11419 11420 11421 11422 11423 11426 11427 11428 11429 11430 11432 11433
    1     1     1     1     1     2     2     1     1     1     1     2     1
11434 11435 11436 11691 11692 11693 11694 11697
    1     2     1     1     1     2     2    51
```

These may be those little ‘islands’ of ZCTAs. There are a couple of R functions that explicitly address duplicate values. As a first approach, use the *unique()* function.

```
> nycZIPSunique <- nycZIPS[unique(nycZIPS$zcta5), ]
> plot(nycZIPSunique)
```

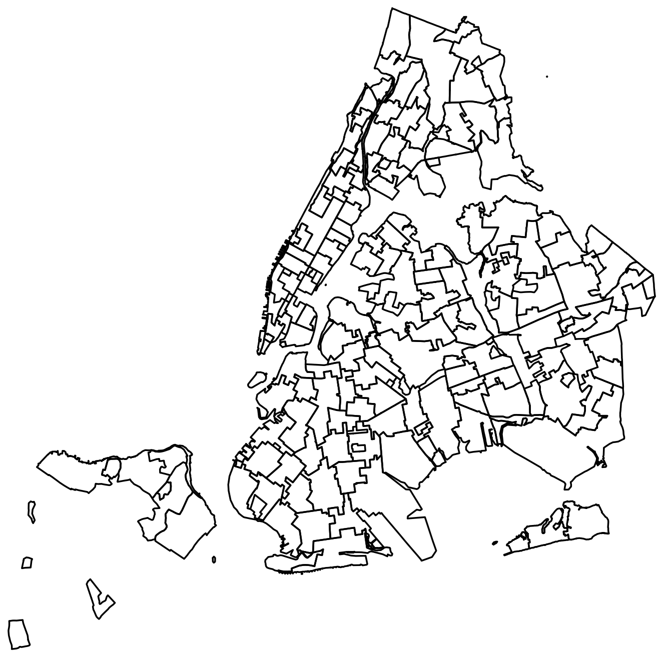


That is not at all what we want. The unique() function *deletes*, all the duplicated values. Another approach could be to create a logical vector of duplicate values using the *duplicated()* function,

```
nycZIPS$dup<-duplicated(nycZIPS$zcta5)}
```

Or, more directly, restricting to non-duplicate values.

```
> nycZIPSnodup <- nycZIPS[!duplicated(nycZIPS$zcta5), ]
> length(nycZIPSnodup)

[1] 177

> plot(nycZIPSnodup)
```



This is getting closer to what we want, returning 178 ZCTA's, which is more in line with what we know about ZIP Codes in New York City. But the plotted map doesn't *look* like New York City. Why? R is choosing the first occurrence of the duplicates. Some of these are smaller polygons contained within the same-named larger polygons, others are those small islands [8]. Because this approach does not represent the geography correctly, neighbor objects will be similarly incorrect. Were on the right track, but not quite there yet.

We need a way to choose the larger polygons. Since we are working with a spatial object, we can use the *area* of the polygons, which are conveniently one of the slots of the S4 sp object. Getting at slot information can be a bit tedious. We could specify each geography using slot notation:

```
> nycZIPS@polygons[[1]]@Polygons[[1]]@area
```

```
[1] 15066358
```

A better approach, since we are dealing with fairly simple polygon data, is to use *sapply()* to return a vector of areas, and then append them to the ZCTA data frame.

```
> areas <- sapply(slot(nycZIPS, "polygons"), slot, "area")
> nycZIPS$area <- areas
```

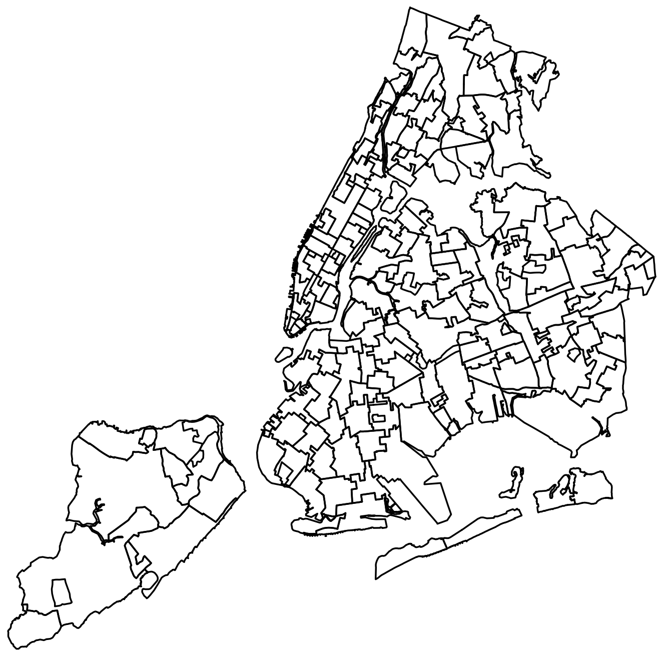
The next step involves creating a logical vector of TRUES and FALSES based on whether the area is the maximum area for the set of ZCTAs. We use the *ave()* ('average')function to generate a vector that contains the *maximum* of the area variable for every ZCTA. We set this equivalent to the area variable (using the = = operator) to create the logical

[8] Of which there are surprisingly many in New York City, many with colorful names like Rat Island, and Chimney Sweeps Island

vector called ‘sel’. Then we use the logical vector to select rows from our spatial data frame. The following plot shows that this is a much better approach.

```
> sel <- ave(nycZIPS$area, nycZIPS$zcta5, FUN = max) == nycZIPS$area
> nycZIPS2 <- nycZIPS[sel, ]
> plot(nycZIPS2)
> length(nycZIPS2)
```

```
[1] 177
```



The take-away message is that spatial data, like any data, requires care at the earliest stages to ensure valid analyses. Just because you get a map, doesn’t mean the data are correct.

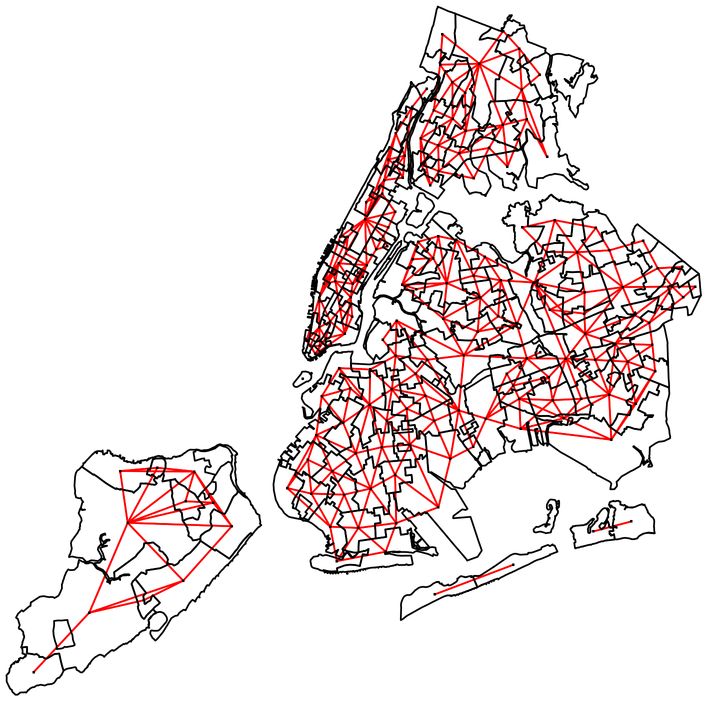
3.3.2 NYC ZCTA Contiguity Neighbors

To create neighbors, we use the *poly2nb()* function in the *spdep* package which outputs the list object nycZIPS.nb. We print out the first few observations of this list, and add this to our plot and note the myriad tangled relationships based on the simple contiguity rule of sharing any spot on a border. We create a second neighbor object (nycZIPS.nb2) using the stricter “rook” definition of contiguity neighbors, where neighbors have to meet on more than one point, by setting“queen” to false.

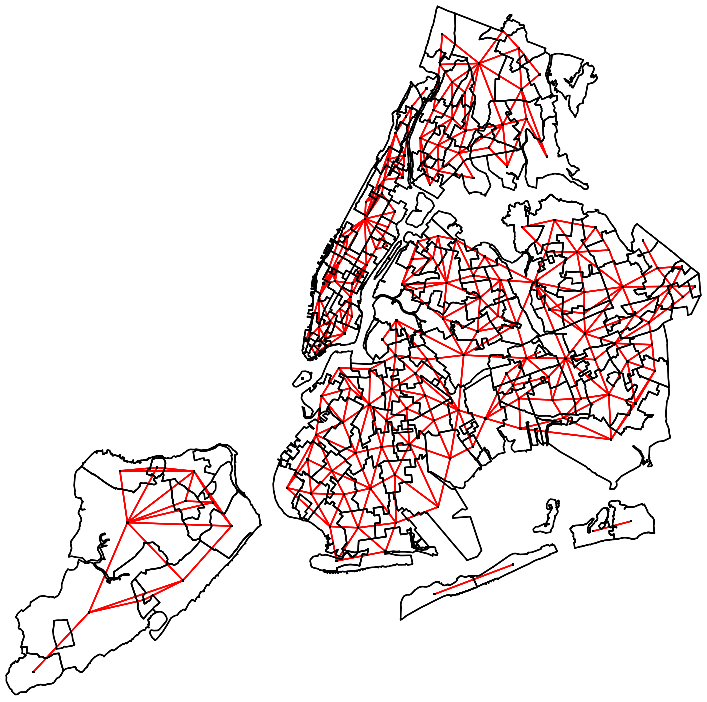
```
> library(spdep)
> nycZIPS.nb <- poly2nb(nycZIPS2)
> head(nycZIPS.nb)
```

```
[[1]]
[1]  3  7  8 12 14 31

[[2]]
[1]  3  5  6  7  9 42

[[3]]
[1] 1 2 6 7 9

[[4]]
[1] 0

[[5]]
[1]  2  9 10 33 42

[[6]]
[1] 2 3 7
```

```
> plot(nycZIPS.nb, coordinates(nycZIPS2), col = "red", pch = ".")
> plot(nycZIPS2, add = T)
```



```
> nycZIPS.nb2 <- poly2nb(nycZIPS2, queen = F)
> plot(nycZIPS.nb2, coordinates(nycZIPS2), col = "red", pch = ".")
> plot(nycZIPS2, add = T)
```

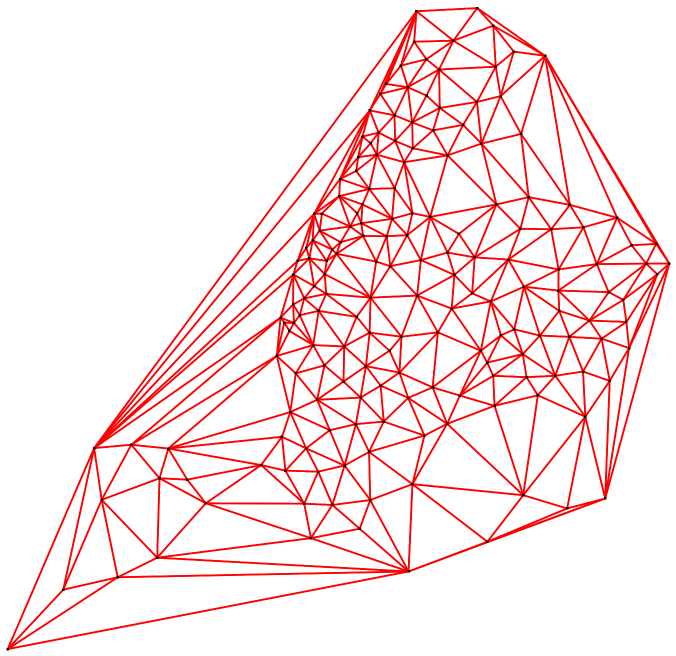


There is not a big difference between the queen or rook definitions, but both are unsatisfactory in that they do not account for cross-county relationships that are separated by water. New York City is so well knit by bridges, roads and public transportation like the subway that such geographic boundaries are relatively unimportant and are likely not to strictly define neighborhoods. Graph-based neighbor definitions may address this shortcoming.

3.3.3 NYC ZCTA Graph-Based Neighbors

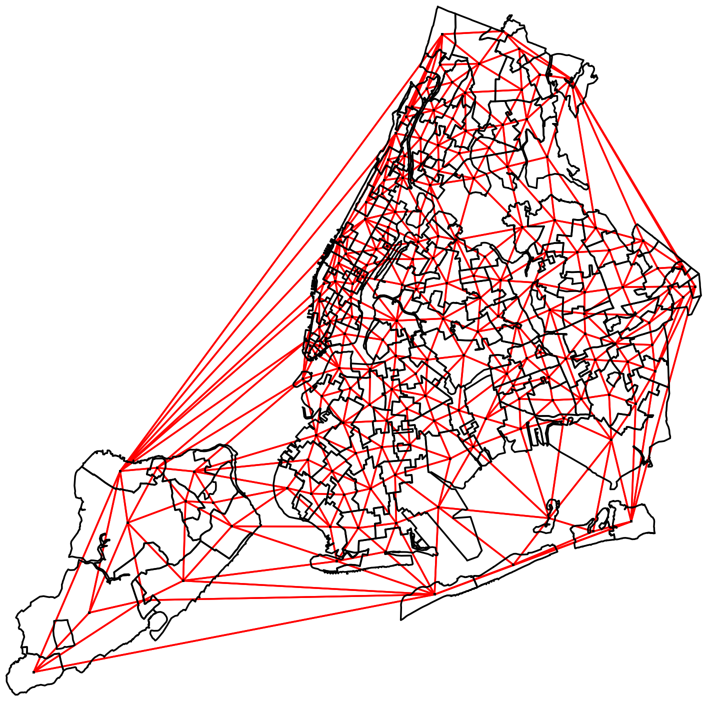
In this next set of code, we apply graph-based neighbor definitions to our New York City ZIP Code tabulation areas. We begin by loading the tripack package and extracting the coordinates and row names for the coordinates. We then create and plot Delaunay, Sphere-of-Influence and Gabriel graphs.

```
> library(tripack)
> coords <- coordinates(nycZIPS2)
> IDs <- row.names(coords)
> nycZIPS.nb4 <- tri2nb(coords, row.names = IDs)
> plot(nycZIPS.nb4, coords, pch = ".", col = "red")
```



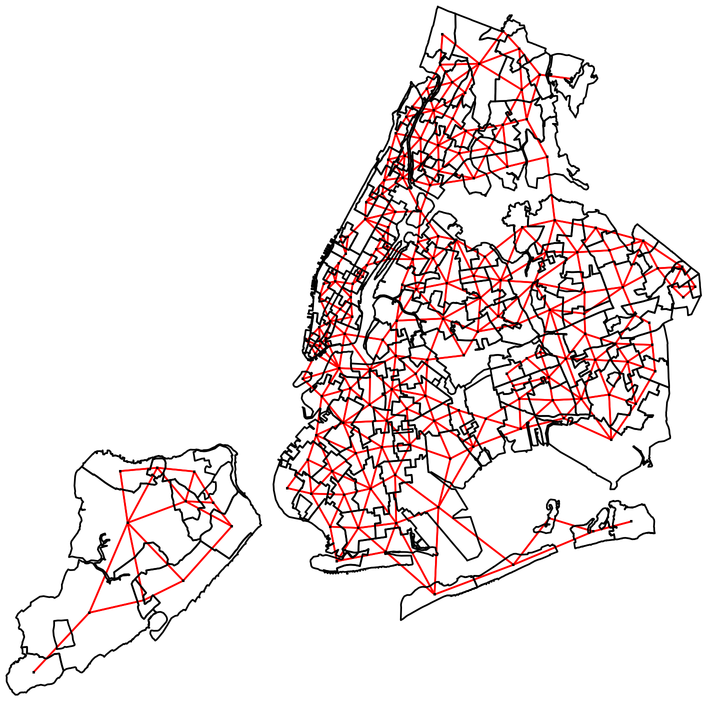
I just like leaving that up there for a moment because it is so pretty. Now let’s overlay the map of New York City:

```
> plot(nycZIPS.nb4, coords, pch = ".", col = "red")
> plot(nycZIPS2, add = T)
```

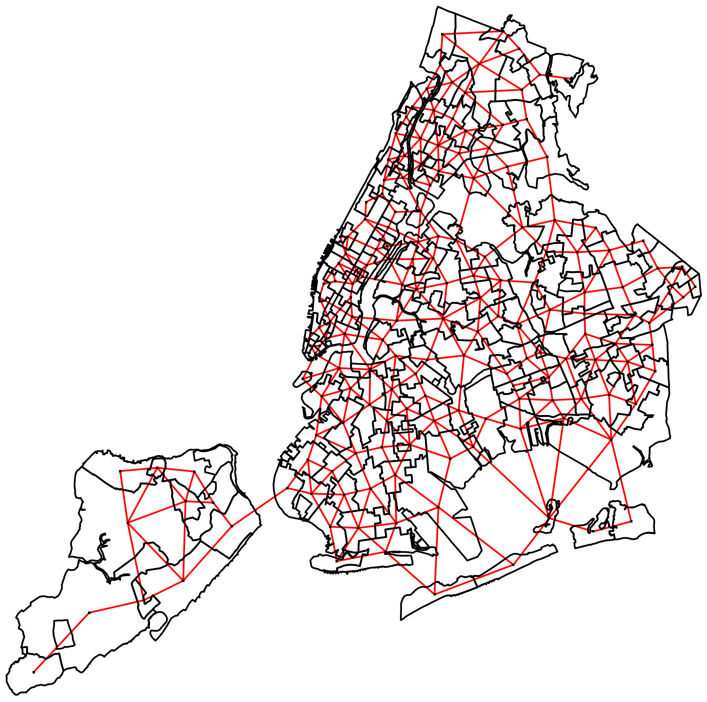


The Delaunay graph is indeed pretty, and it does ignore water boundaries, but it results in some odd neighbors. ZCTA's in northern Manhattan and the Bronx are "neighbors" of Staten Island, which is over 25 miles away, and differs in important geographic and social ways. Sphere-of-Influence and Gabriel graphs may address this inconsistency.

```
> nycZIPS.nb5 <- graph2nb(soi.graph(nycZIPS.nb4, coords), row.names = IDs)
> plot(nycZIPS.nb5, coords, pch = ".", col = "red")
> plot(nycZIPS2, add = T)
```



```
> nycZIPS.nb6 <- graph2nb(gabrielneigh(coords), row.names = IDs)
> plot(nycZIPS.nb6, coords, pch = ".", col = "red")
> plot(nycZIPS2, add = T)
```



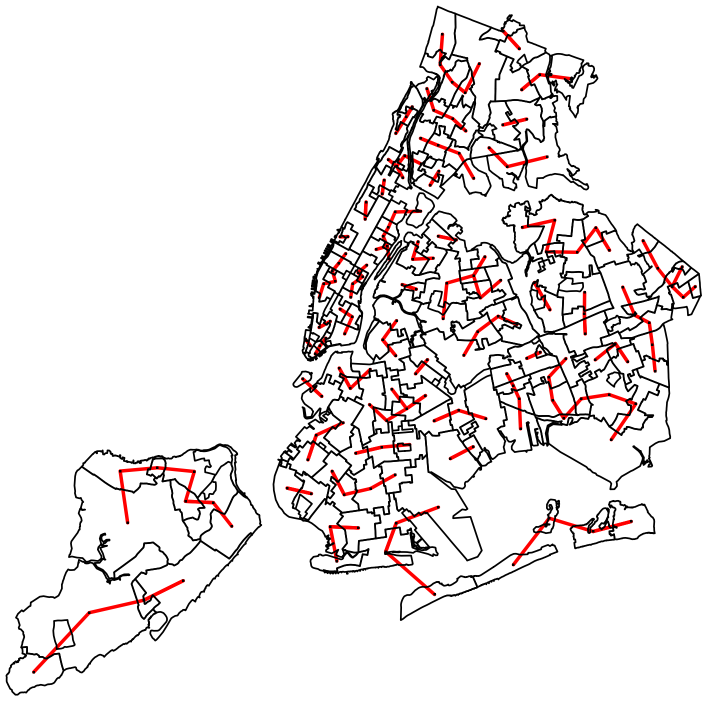
the Gabriel graph is, so far, the most satisfactory.

3.3.4 NYC ZCTA Distance-Based Neighbors

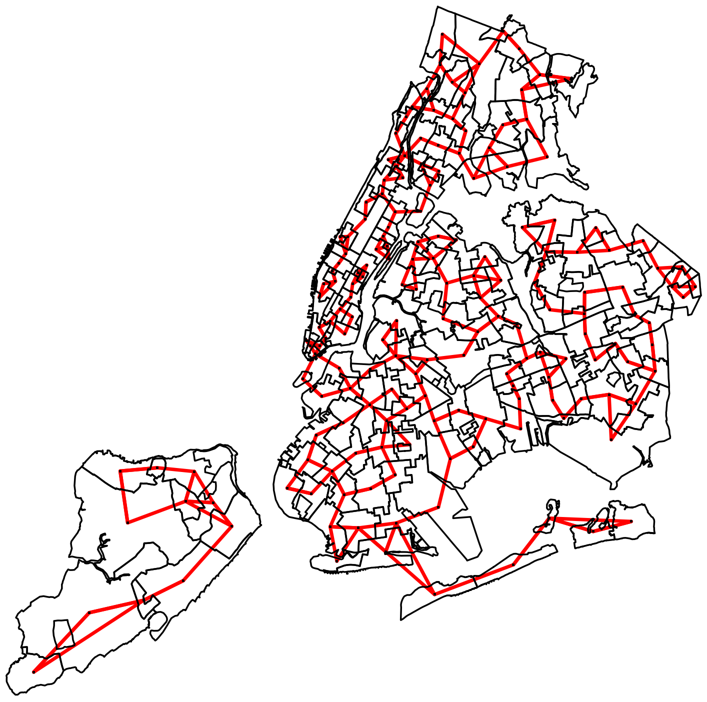
Next we look at distance-based neighbors as alternatives to contiguity or graph-based neighbors. We begin by defining *k* number of neighbors using the *knn2nb()* function [9] We can plot them, but this first step is mostly to define a vector of distances that ensures some explicit number of neighbors. From this vector of distances, we select a maximum distance and neighbors based on some proportion of that distance. In this example, we look at 50%, 75% and 100% of the maximum distance. We will see how the number of connections can quickly become unmanageable.

```
> nycZIPS.nb7 <- knn2nb(knearneigh(coords, k = 1), row.names = IDs)

> plot(nycZIPS.nb7, coords, pch = ".", col = "red", lwd = 2)
> plot(nycZIPS2, add = T)
```

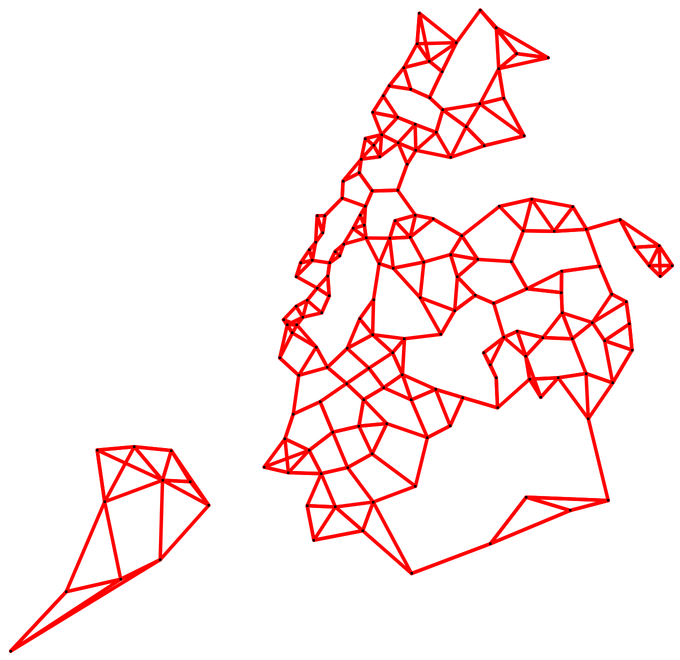
[9] We first have to install the RANN package.



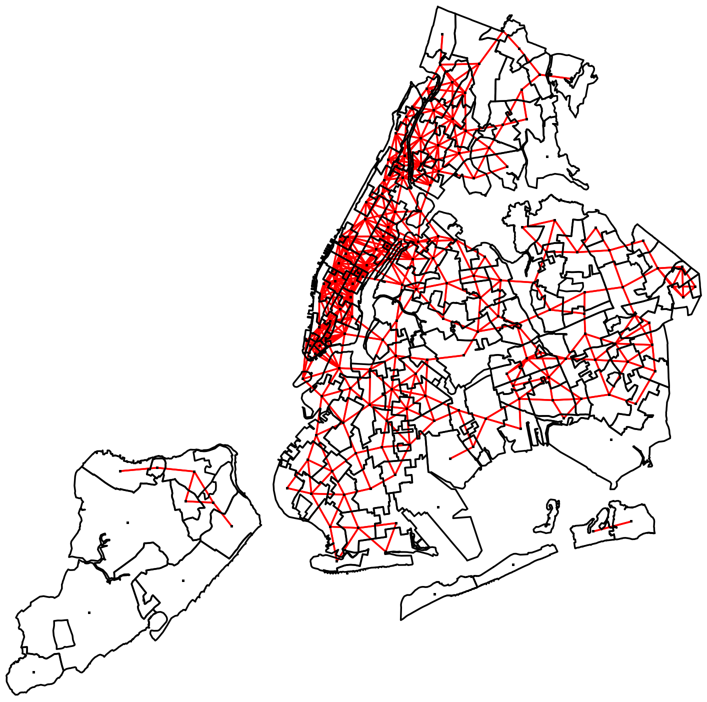
```
> nycZIPS.nb8 <- knn2nb(knearneigh(coords, k = 2), row.names = IDs)
> plot(nycZIPS.nb8, coords, pch = ".", col = "red", lwd = 2)
> plot(nycZIPS2, add = T)
```



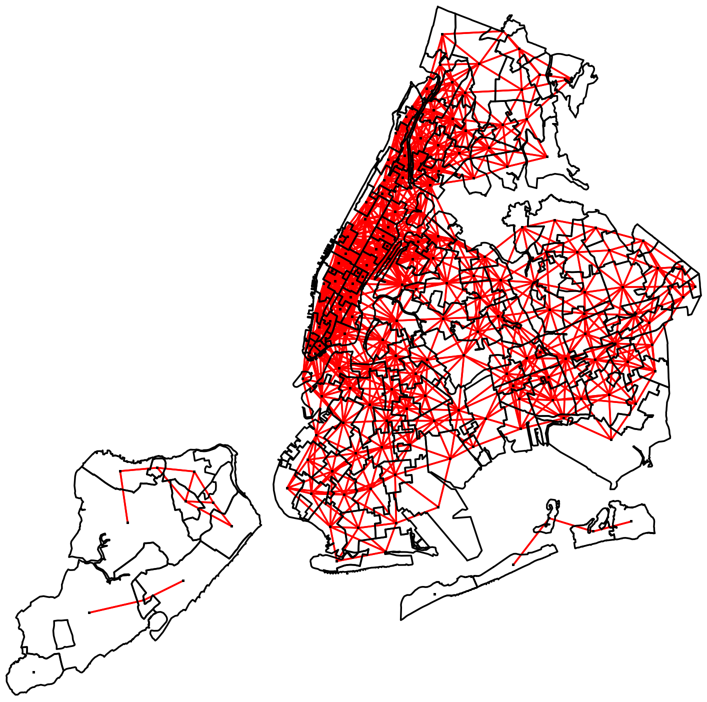
```
> nycZIPS.nb9 <- knn2nb(knearneigh(coords, k = 3), row.names = IDs)
> plot(nycZIPS.nb9, coords, pch = ".", col = "red", lwd = 2)
> plot(nycZIPS2)
```



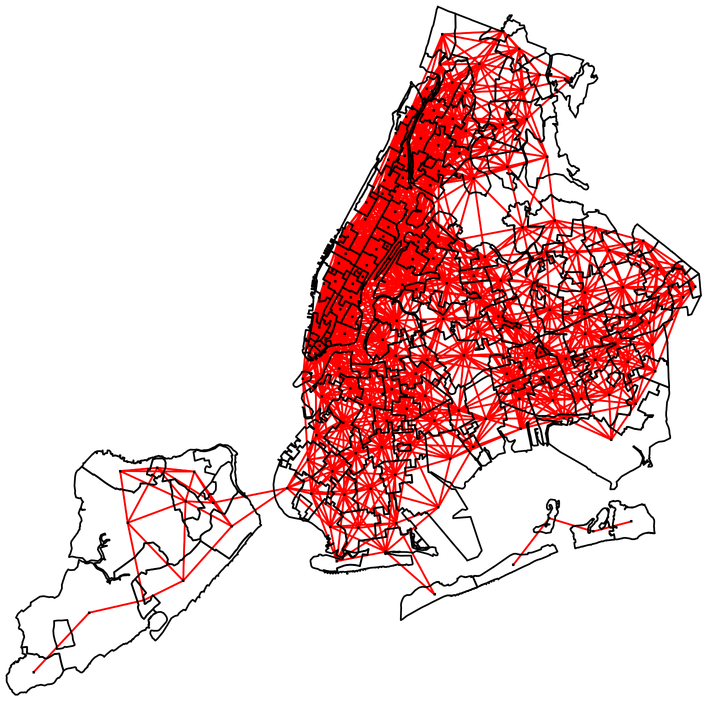
```
> dsts <- unlist(nbdists(nycZIPS.nb7, coords))
> summary(dsts)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1357    3584    5095    5163    6152   18050

> max.dst <- max(dsts)
> max.dst

[1] 18047.65

> nycZIPS.nb10 <- dnearneigh(coords, d1 = 0, d2 = 0.5 * max.dst,
+      row.names = IDs)
> plot(nycZIPS.nb10, coords, pch = ".", col = "red")
> plot(nycZIPS2, add = T)
```



```
> nycZIPS.nb11 <- dnearneigh(coords, d1 = 0, d2 = 0.75 * max.dst,
+      row.names = IDs)
> plot(nycZIPS.nb11, coords, pch = ".", col = "red")
> plot(nycZIPS2, add = T)
```



```
> nycZIPS.nb12 <- dnearneigh(coords, d1 = 0, d2 = 1 * max.dst,
+      row.names = IDs)
> plot(nycZIPS.nb12, coords, pch = ".", col = "red")
> plot(nycZIPS2, add = T)
```



This chapter should give you a sense of how spatial neighbors can be defined. For the purposes of the New York City ZCTA analysis we will be using, Gabriel graphs make the most sense. What is most appropriate to your analysis may differ. In the next chapter, we move on to how we assign weights to neighbor relationships and test for spatial autocorrelation.

Chapter 4
Spatial Weights and Autocorrelation

4.1 Spatial Weights

Once the list of sets of neighbors for study area is set, you can assign spatial weights to each relationship. These weights can be used to test for the presence of spatial autocorrelation.

We'll use the function *nb2listw()* converts a neighbors list object, to a weights object. The weights object is a set of weights indexed by a list of neighbors. The weight of the link between i (on the row) and j (on the column) is the k^{th} element of the i^{th} (row). If j is not on the $i^t h$ row, the weight for that link is zero, i.e. they are not neighbors.

The default is *style="W"*, where weights for each areal unit (or row) are standardized to sum to one (aka "row standardization"). Note that under this setting, the more neighbors a unit has, the smaller each individual weight will be compared to units with fewer neighbors. As an alternative, *style="B"* sets binary weights, where the weight of *each* neighbor relationship to 1. The sums of the weights for areas will differ according to the numbers of neighbors areas have.[1] Lastly, if you're brave, you can use the *glist* argument to set your own weights. For example, calculating inverse distances using nbdists() and lapply() like this:

```
dsts <- nbdists(nycZIPS.nb, coordinates(nycZIPS))
idw <- lapply(dsts, function(x) 1/(x/1000))
nycZIPS.lw.idwB <- nb2listw(nycZIPS.nb, glist = idw, style = "B")
```

One important note: *nb* objects that have areal units with no neighbors will return on error. You will need to set *zero.policy=TRUE every time you run the function.*

There are import / export functions available for weights. One useful utility for future Bayesian Hierarchical Modeling in R using WinBUGS is the *listw2WB()* function for writing weights for use in WinBUGS. Another helpful function in the same package, *nb2lines()*, can be used to create shapefiles.

4.2 Spatial autocorrelation

Observations near each other (either spatially or in time) tend to be like each other. In statistical terms, this is said to "reduce the amount of information" in these observations because they can be used, in part, to predict each other. Spatial autocorrelation is more complicated than one-dimensional autocorrelation because it is multidimensional and multidirectional. In R, we use the *spdep* package to assess and analyze spatial autocorrelation.

A number of important caveats accompany the use of statistical tests for spatial autocorrelation. First, there are underlying assumptions of homoskedasticity for normal models and no overdispersion for Poisson models. Second, the

[1] When we move on to defining neighborhoods for conditional autoregression models in WinBUGS, it is usually best use binary weights of 1 for neighbors, and 0 otherwise.

models are highly sensitive to outliers. Third, results may vary based on choice of model and on choice of weighting structure. For all these reasons, it is a good idea to take a number of approaches and see how robust your results actually are.

4.2.1 *Moran's I*

Moran's I is a global test of autocorrelation. Moran's I is based on the weight between two areal units *i* and *j* times the mean-adjusted product of the outcome of interest for those two units divided by the squared mean difference of the index point *i*. This result is further adjusted by multiplying it by the number of weights divided by the total sum of weights *n*.

$$\frac{n}{\sum w_{ij}} \sum \frac{w_{ij}(y_i-\bar{y})(y_j-\bar{y})}{(y_i-\bar{y})^2} \tag{4.1}$$

The statistic is centered on the mean, so we are assuming the correct model has a constant mean. Trend in the data violates this assumption. If present, we can transform the data with a regression that controls for the trend.

The expected value for Moran's I is:[2]

$$\frac{-1}{n-1} \tag{4.2}$$

Moran's I can be obtained with the moran.test() function in spdep which takes as it's arguments a list of weights, and a variable of interest. It is probably the most commonly used global test of autocorrelation, though you will also see reference to Geary's C (geary.test()), which is the inverse of inverse of Moran's I for continuous variables .

Values of these statistics are not directly interpretable. A common approach is to standardize the observed value by subtracting the expected value, divide the difference by the square root of the variance for the spatial weights used, and compare this standard deviate with the Normal distribution for a p value.[3]

The syntax for Moran's I is relatively straightforward:

```
moran.test(outcome, listw = nb2listw(neighbor.list))
```

A statistically significant result indicates that neighboring tracts are likely to have similar values for whatever variable you are testing. Like non-spatial tests of significance, *why* the values are similar, is a matter for the analyst to pursue.

4.2.2 *Other Tests for Autocorrelation*

4.2.2.1 Empirical Bayes

There is a version of Moran's I that uses Empirical Bayes to shrink extreme rates for tracts with small populations towards the rate for the area as a whole. Once the rates are smoothed, you look for global autocorrelation.

```
set.seed(1234)
EBImoran.mc(n = outcome, x = population, listw = nb2listw(neighbor.list
style = "B"), nsim = 999)
```

[2] You can look up the somewhat unruly formula for the variance in Bivand, or by bringing up the code for moran.test() in R

[3] Bivand notes that units without neighbors will cause errors, but that there is currently no good advice on how to address this issue. The spdep tests, though have options to ignore units without neighbors.

4.2.2.2 Correlograms

The *correlog()* function in the *pgirmess* package returns a correlogram [4]:

```
library(pgirmess)
corD <- correlog(coordinates(map.object), outcome.variable, method = "Moran")
```

4.2.3 Moran Scatterplot

A Moran scatterplot can be used to look for areas of *local* autocorrelation or clusters:

```
moran.plot(outcome.variable, listw = nb2listw(neighbor.list))
```

4.3 Autocorrelation in New York City ZIP Codes

To illustrate these methods, we return to our map of New York City ZIP Code Tabulation Areas. If you recall, the Gabriel graph approach returned the most satisfactory result. We begin by loading that neighbor object into our analysis space.

```
> load("/Users/charlie/Dropbox/spatialEpiHowTo/spatialEpiAutocorr/spatialEpiAutocorr.RData
```

Next, we create a list of weights based on the neighbor object. We will use a binary definition for neighbors. As noted above, we must address the issue of zero neighbors. [5] The resulting weight list object includes a vector of "nieghbors" and their associated "weights".

```
> library(sp)
> library(spdep)
> nycZIPS.wts <- nb2listw(nycZIPS.nb6, zero.policy = T, style = "B")
> print(nycZIPS.wts, zero.policy = T)
```

```
Characteristics of weights list object:
Neighbour list object:
Number of regions: 177
Number of nonzero links: 379
Percentage nonzero weights: 1.209742
Average number of links: 2.141243
23 regions with no links:
11 38 41 43 52 53 54 72 75 79 82 88 113 114 116 118 119 124 134 160 166 172 177
Non-symmetric neighbours list

Weights style: B
Weights constants summary:
    n    nn  S0  S1   S2
B 154 23716 379 379 3482
```

```
> names(nycZIPS.wts)
```

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

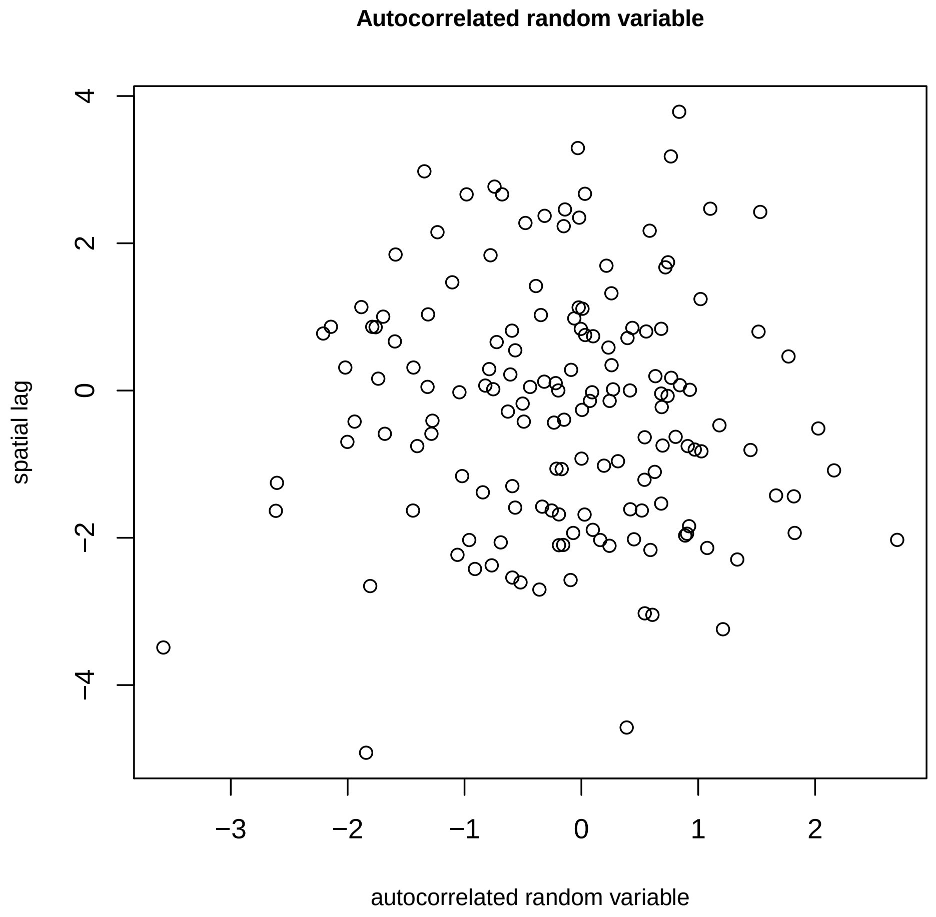
[4] A autocorrelation plot often used in time series analysis

[5] Note that when we want to do *anything* with this object, e.g print , we have to re-invoke the zero.policy argument.

In the next bit of code, we model the weights to see if they are autocorrelated with a autocorrelation or Moran scatter-plot. The operative function is *invIrW()* which calculates a spatial autocorrelation matrix using our weights, a spatial autocorrelation coefficient (ρ) and a simulated uncorrelated identity matrix. [6] We then plot the autocorrelation matrix against a vector of spatial lag values we create using the *lag()* function on the weight values.

```
> set.seed(987654)
> n <- length(nycZIPS.nb6)
> uncorr.x <- rnorm(n)
> rho <- 0.05
> autocorr.x <- invIrW(nycZIPS.wts, rho, feasible = TRUE) %*% uncorr.x
```

```
> plot(autocorr.x, lag(nycZIPS.wts, autocorr.x), xlab = "autocorrelated random variable",
+      ylab = "spatial lag", main = "Autocorrelated random variable",
+      cex.main = 0.8, cex.lab = 0.8)
```



The points are fairly-well scattered with some intimation of a positive relationship between lag and autocorrelation. We can test this apparent lack of global autocorrelation with Moran's I.

```
> moran.x <- moran.test(autocorr.x, listw = nycZIPS.wts, zero.policy = T)
> moran.x
```

[6] Note, that I used the option to set the variable "feasible" to TRUE. This surpresses error messages from a test to see if rho falls between 1/min(eigen(V)) and 1/max(eigen(V)). Roger Bivand seems alright with this approach, and who am I to argue with the master?

```
        Morans I test under randomisation

data:  autocorr.x
weights: nycZIPS.wts

Moran I statistic standard deviate = -0.327, p-value = 0.6282
alternative hypothesis: greater
sample estimates:
Moran I statistic       Expectation          Variance
     -0.023079946      -0.006535948       0.002559748
```

```
> summary(moran.x)
```

```
            Length Class  Mode
statistic   1      -none- numeric
p.value     1      -none- numeric
estimate    3      -none- numeric
alternative 1      -none- character
method      1      -none- character
data.name   1      -none- character
```

```
> moran.x$p.value
```

```
[1] 0.6281644
```

The p-value confirms our initial impression of a lack of autocorrelation from the plot.

4.4 Spatial Correlation of New York City Pediatric Traumatic Brain Injury

After looking at the autocorrelation of neighbor weights in our spatial units, we can look at the autocorrelation of the outcomes in those units. We now (finally) move onto some disease data. In this example, we'll look at New York City traumatic brain injury diagnoses among a cohort of children younger than 9 years of age. The object 'tbizip' is a dataframe with columns for ZCTA estimates of age and gender-specific TBI diagnoses and population as well as gender-standardized morbidity ratios.[7]

4.4.1 Merging the data file with the map file

We want to merge our disease data with the the map object nycZIPS2 with which we have been working. We take a quick look at our disease data set, and realize that there are 195 rows of data in 'nyczip' compared to 177 ZCTAs in 'nycZIPS2'.

```
> length(nycZIPS2$zcta5)
```

```
[1] 177
```

```
> length(tbizip$zips)
```

```
[1] 195
```

[7] These data come from New York City Medicaid records.

The observations in ‘tbizip’ that do not correspond to valid ZCTA in our map object appear to be empty. [8]

```
> tbizip$tbi_tot[!tbizip$zips %in% nycZIPS2$zcta5]
```

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

```
> tbizip$smr[!tbizip$zips %in% nycZIPS2$zcta5]
```

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

We will merge the two data objects into a single dataframe object called nycZIPS.df, restricting the merge to only the observations that are present in the ZCTA map object. The ID variables are tbizip$zips and nycZIPS$zcta5. We perform a left merge that ensures the result is keyed to the map object.

```
> library(sp)
> nycZIPS.df <- merge(nycZIPS2, tbizip, all.x = T, all.y = F, by.x = "zcta5",
+     by.y = "zips")
```

4.4.2 Plotting a choropleth

In preparation for plotting a choropleth of TBI rates, we will need to do some data manipulaiton. We begin by creating variables for counts of diagnoses and population at risk. We deal with missing data by assigning zero to areas with no diagnoses, and adding 0.1 (to avoid dividing by zero) to ZCTA’s for which there are no records of children enrolled in Medicaid. NB: We assign these new variables to our *spatial polygons map object* nycZIPS2, so we can work with the sp object. Otherwise, methods like spplot will not work on the dataframe object.

```
> cases <- nycZIPS.df$tbi_tot78 + nycZIPS.df$tbi_tot012 + nycZIPS.df$tbi_tot34 +
+     nycZIPS.df$tbi_tot56
> cases[is.na(cases)] <- 0
> pop <- nycZIPS.df$pop_tot78 + nycZIPS.df$pop_tot012 + nycZIPS.df$pop_tot34 +
+     nycZIPS.df$pop_tot56
> pop[is.na(pop)] <- 0.1
> nycZIPS2$cases <- cases
> nycZIPS2$pop <- pop
```

We calculate a rate per 10,000 population and again assign that variables to the map file.

```
> rate <- cases/pop * 10000
> nycZIPS2$rate <- rate
```

Based on a summary of the rate outcome variable, we create four categories using the *cut()* function, and map the cateories with colors chosen from RColorBrewer.

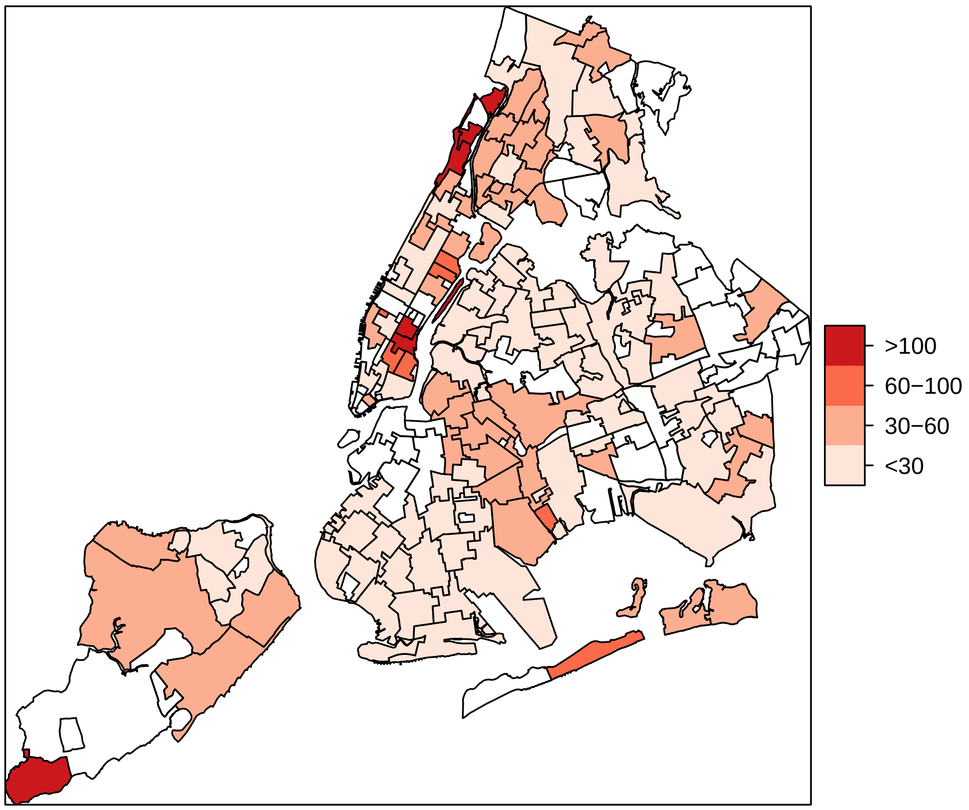
```
> summary(nycZIPS2$rate)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   0.00    0.00   18.86   25.75   36.10  191.10
```

```
> nycZIPS2$rate.cat <- cut(nycZIPS2$rate, breaks = c(0, 30, 60,
+     100, 200), labels = c("<30", "30-60", "60-100", ">100"))
```

```
> library(RColorBrewer)
> tbiplot1 <- spplot(nycZIPS2, "rate.cat", col.regions = brewer.pal(4,
+     "Reds"))
> print(tbiplot1)
```

[8] I used the *%in%* operator which returns of logical vector of TRUES and FALSES for whether one vector is in another to index one file with the other



After this preliminary work, we can turn our attention to statistical tests of global autocorrelation for the ZCTA-based pediatric TBI injury rates.

4.4.3 *Global tests of autocorrelation in the NYC TBI data*

We now apply global tests of spatial autocorrelation to our TBI data. Here we use three variants of Moran's I as a way to test the sensitivity of our results and check for possible problems with model mispecification. We first use the *moran.test()* function we encountered above. Note that we, again, have to set the zero.policy to TRUE, to account for regions with no neighbors.

```
> moran.cases <- moran.test(nycZIPS2$cases, listw = nycZIPS.wts,
+      zero.policy = TRUE)
> moran.cases

        Morans I test under randomisation

data:  nycZIPS2$cases
weights: nycZIPS.wts

Moran I statistic standard deviate = 8.7553, p-value < 2.2e-16
```

```
alternative hypothesis: greater
sample estimates:
Moran I statistic       Expectation          Variance
      0.418263797      -0.006535948       0.002354097
```

```
> moran.cases$p.value
```

```
[1] 1.017617e-18
```

The test returns a statistically significant p-value indicating areas of autocorrelation or clustering in our data.

Next, we use a Monte Carlo simulation version of this test on the TBI rate variable. This also returns a statistically significant result.

```
> set.seed(987)
> moran.mc(rate, listw = nycZIPS.wts, nsim = 999, zero.policy = TRUE)
```

```
        Monte-Carlo simulation of Morans I

data:  rate
weights: nycZIPS.wts
number of simulations + 1: 1000

statistic = 0.1878, observed rank = 1000, p-value = 0.001
alternative hypothesis: greater
```

Lastly, we use the Empirical Bayes version, specifying both the count and the population at risk.

```
> EBImoran.mc(n = nycZIPS2$cases, x = nycZIPS2$pop, listw = nycZIPS.wts,
+     nsim = 999)
```

```
        Monte-Carlo simulation of Empirical Bayes Index

data:  cases: nycZIPS2$cases, risk population: nycZIPS2$pop
weights: nycZIPS.wts
number of simulations + 1: 1000

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

We see that all three tests returned results indicating similar levels of correlation. Clearly, areas with increased numbers or proportions of pediatric TBI are near each other.

Now that we have a sense of what spatial areal units are, how neighbor relationships are defined and weighted, and how we can test for spatial autocorrelation, we turn our attention to spatial modeling of health outcomes.

Chapter 5
Spatial Models

As I mentioned in the previous chapter, there is usually a marked lack of independence between observations in spatial data. While covariates may help us explain some of this spatial autocorrelation (which we consider later), the mainstay of characterizing these non-independent values is through their residuals. [1]

5.1 Autoregressive Models

Two approaches seen in spatial modeling are Simultaneous Autoregressive (SAR) and Conditional Autoregressive (CAR) Models. Note that these models may require that we transform our data to meet the underlying assumption of normally distributed outcomes. Let's consider them each in turn.

5.1.1 Simultaneous Autoregressive Models

5.1.1.1 The Model

First, a little matrix stuff to set the stage. This is for completeness sake, and you may want to skip over it the first time around.

To reflect the non-independent nature of the data, we can define error terms so that they depend on each other:

$$e_i = \sum b_{ij} e_i + \varepsilon_i \tag{5.1}$$

ε_i is assumed *iid*, and can be formulated in a matrix form that allows us to specify spatial dependence:

$$(I - B)(Y - X^{\tau\beta}) \tag{5.2}$$

Here, I is an identity matrix, and B is a matrix that contains the dependence parameters b_{ij}. This allows re-parameterization in terms of W the spatial dependence matrix, which can be written $B = \lambda W$, where λ is a spatial autocorrelation parameter for the spatial dependence matrix W. Enough of that.

[1] As a reminder, residuals, $\hat{r}$, are a a vector of random errors, most often assumed to be normally distributed. A general model can be represented as an outcome set equal to some mean effect (or intercept), plus the coefficient for differential effect of an explanatory variable plus error: $y = \beta_0 + \beta_1 + e$. The only random variable on the right side of the equation is the error term, and it is usually the most efficient way of characterizing the outcome variable.

5.1.1.2 Invoking the model

In R, we can specify a SAR model with the *spautolm()* function in the spdep package. We write the formula for a linear predictor as usual and include the *W* or spatial weights matrix as a *listw* object. Here we specify a model with some proportion of the population in an area exposed and two additional explanatory variables.

```
my.sar.model <- spautolm(Z ~ pct.exposed + pct.var1 + pct.var2, data = my.data,
listw = my.listw)
```

5.1.2 Conditional Autoregressive Models

In a CAR model, instead of a global ε vector, we may specify a model based on the *neighbors* of area *i*, defined in a chosen way (e_{ji}). Such CAR specifications rely on the distribution of the spatial error terms e_i conditioned on ε, the vector of all random error terms minus e_i itself.

We'll use the function *spautolm()* again, this time setting the argument *family=”CAR”*:

```
my.car.model <- spautolm(Z ~ pct.exposed + pct.var1 + pct.var2,
data = my.data, listw = my.listw, family="CAR")
```

5.2 Non-Autoregressive Approaches

Two other approaches to modeling spatial data deserve mention because of their familiarity to epidemiologists and usefulness for disease modeling in general.

5.2.0.1 Poisson Models

Poisson models are appropriate for count data, and may make more sense than other approaches when we, as we often do in epidemiology, are tallying up disease outcomes. Whereas we my need to transform data to meet the assumptions of the linear models above, this is rarely the case for a Poisson model of count data. Invoking the model proceeds as it does in non-spatial settings. We use the standard R call to *glm()*, specify outcome and explanatory variables, an offset variable for the population at risk, and invoke a Poisson link function.

```
my.pois.model <- glm(cases ~ pct.exposure + pct.var1 + pct.var2
+ offset(log(pop.var)), data = my.data, family="poisson")
```

Unfortunately, there is currently no satisfactory spatial autocorrelation test for the residuals of such a straightforward GLM (as there is for the autoregressive linear models), but as we will see, Poisson models play an important role in Bayesian approaches to spatial hierarchical models.

5.2.0.2 Generalised Estimating Equations

GEE can be used as an alternative to GLMs when the data are correlated, e.g. longitudinal data with several observations for the same subject. This approach has been used with clusters of grid cells, but has not yet been extended to irregular areal data or polygons.

5.3 Modeling Pediatric Traumatic Brain Injury in New York City

Taking up where we left off, after determining that pediatric traumatic brain injury in New York City is spatially dependent, we might reasonably be interested in knowing how a variable like housing characteristics is related to this distribution. Our first step then is to get some housing data for our ZCTA map.

5.3.1 Entering and Cleaning the Data

The US Census Bureau is a compendious (if not always user friendly) source of such information. To create the kind of custom geographically-referenced data set we need, we can navigate from the Census Bureau homepage to the American Factfinder data tool. On that page, we choose the Decennial Census from among the data sets under the "Data Sets" menu on the left and select "Custom Table" from the list of options for the "2000 Census Summary File 1". On the next page, under "Choose a selection method", click the "geo within geo" tab. [2] We then use the series of drop-down menus to select 5-digit ZIP Code tabulation areas within New York City, and on the resulting page, we select (and individually add) each variable we want. Finally, we save the resulting table as a .csv file, which we read into R after doing some data cleaning, dropping some unnecessary variables, identifying odd or inappropriate ZCTA's and removing those observations.

```
> housing <- read.table(file = "/Users/charlie/Dropbox/spatialEpiHowTo/spatialEpiModels/ce
+     header = F, sep = ",", skip = 2, col.names = c("x1", "zcta",
+         "x2", "x3", "x4", "house.tot", "pop.tot", "house.rent",
+         "house.occ"), colClasses = c("character", "character",
+         "character", "character", "character", "numeric", "numeric",
+         "numeric", "numeric"))
> housing <- housing[, -c(1, 3:5)]
> housing$zcta <- as.numeric(housing$zcta)
> complete <- complete.cases(housing)
> housing <- housing[complete, ]
> head(housing)

   zcta house.tot pop.tot house.rent house.occ
1 10001      9806   17310       6971      8941
2 10002     32965   84870      27366     31512
3 10003     31154   53673      21686     29516
4 10004       779    1225        489       622
5 10005       562     884        489       524
6 10006       978    1447        803       808
```

In the section on spatial neighbors, we learned how to extract the areas from the spatial polygon object. If we did not have that data, we could get it as well from the US Census Bureau in their US Gazatteer files, and read it into an R data set:

```
> area <- read.table(file = "/Users/charlie/Dropbox/spatialEpiHowTo/spatialEpiModels/censu
+     header = T)
> area <- area[, -c(2, 3, 5:7)]
> head(area)
```

[2] I found these links only after numerous frustrating and time consuming dead ends. The Census Bureau is "improving" their site and will be transitioning to a new American Factfinder2 tool, with which I have yet to be able to create these kinds of tables.On the plus side, I recently found out that there is an R package of 2000 census data. If 2000 data work for you, then this would be the way to go. I've included a brief introduction to the package as an appendix.

```
   zcta sq.miles
1 10001    0.621
2 10002    0.879
3 10003    0.576
4 10004    0.560
5 10005    0.074
6 10006    0.092
```

We load our our NYC ZIP Codes map and weight objects which we created in our previous session. Recall that if we just use merge() the resulting simple dataframe object will not be an spatial polygon dataframe and will not work with sp methods. As in the previous set of ntoes, we add variables to the map object after indexing by a logical vector to restrict the entries to our 177 ZCTA's. [3]

```
> load("/Users/charlie/Dropbox/spatialEpiHowTo/spatialEpiModels/SpatialEpiModels.RData")
> library(sp)
> nycZIPS3 <- nycZIPS2
> housing2 <- housing[housing$zcta %in% nycZIPS3$zcta5, ]
> housing2 <- housing2[order(housing2$zcta), ]
> nycZIPS3 <- nycZIPS2[order(nycZIPS3$zcta5), ]
> nycZIPS3$house.pop <- housing2$house.pop
> nycZIPS3$pop.tot <- housing2$pop.tot
> nycZIPS3$house.rent <- housing2$house.rent
> nycZIPS3$house.occ <- housing2$house.occ
```

Now we do the same thing with the square mile area variable from the US census. One important note, the US Census file of area by square miles does not include ZIP code 10048, the old World Trade Center ZIP Code. We add it to the areal sp object data as a value of 0.0001. [4] We also create a variable called "rent.dense". This is measure of rental housing density in a ZCTA and it is based on the number of rental units in a ZCTA divided by the area in square miles. The choice of this variable is motivated by the hypothesis that, in New York City at least, rental density is an indirect measure of the number of high-rise apartments in an area, which in turn may be related to TBI risk through falls.

```
> area2 <- area[area$zcta %in% nycZIPS3$zcta5, ]
> wtc <- data.frame(zcta = "10048", sq.miles = 1e-04)
> area2 <- rbind(area2, wtc)
> area2 <- area2[order(area2$zcta), ]
> nycZIPS3$sq.miles <- area2$sq.miles
> nycZIPS3$rent.dense <- nycZIPS3$house.rent/nycZIPS3$sq.miles
```

One more bit of data housekeeping. We will be conducting analyses that will return errors if thye lead to division by or a log transformation of a zero value. [5] In the previous bit of code, we added a 0.0001 to ZCTA 10048 to avoid a missing value. In the case of our outcome and predictor variables, we will convert zero values to missing (NA). We do this for two reasons. First, spatial areas with no people (or at least no people who meet our study criteria) are essentially undefined, so NA makes sense. And, R has many options for dealing with NA values which will allow us some flexibility down the line.

```
> nycZIPS3$rate[which(nycZIPS3$rate == 0)] <- NA
> nycZIPS3$house.rent[which(nycZIPS3$house.rent == 0)] <- NA
> nycZIPS3$house.occ[which(nycZIPS3$house.occ == 0)] <- NA
```

[3] I found this alternative approach online which seems like a much more elegant way to do the same thing, but I couldn't get it to work.

[4] A bit more on this in the following paragraph

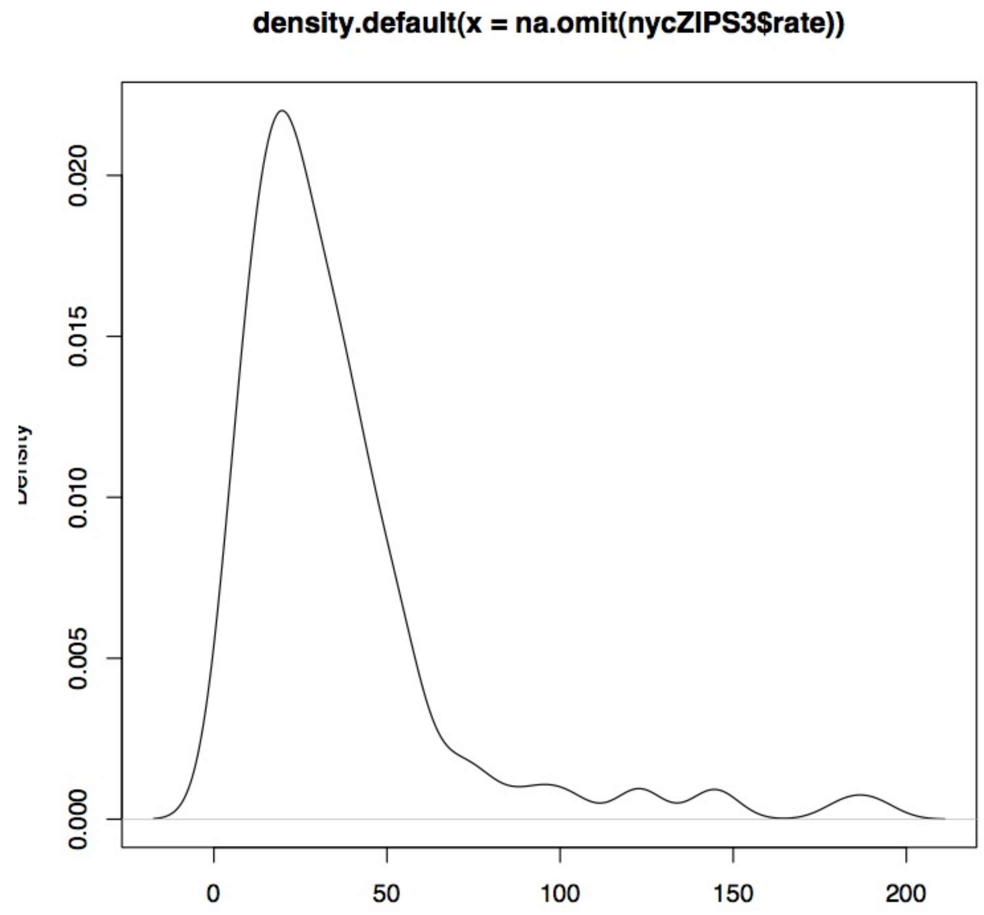
[5] Both of which are undefined or infinite

5.3.2 Exploring the Data

We begin our analyses as we do with all data, whether they be spatial or not: exploration, cleaning, simple descriptive analyses. In the interest of time and space, I'll omit the initial analyses [6]

Our outcome variable is the rate of pediatric TBI among the population of children enrolled in Medicaid. Here, we simply plot that rate

```
> plot(density(na.omit(nycZIPS3$rate)))
```



As we noted above, if we plan to use autoregressive procedures, there is an underlying assumption of normality, which these data do not appear to meet. A simple log transformation may be helpful:

```
> plot(density(log(na.omit(nycZIPS3$rate))))
```

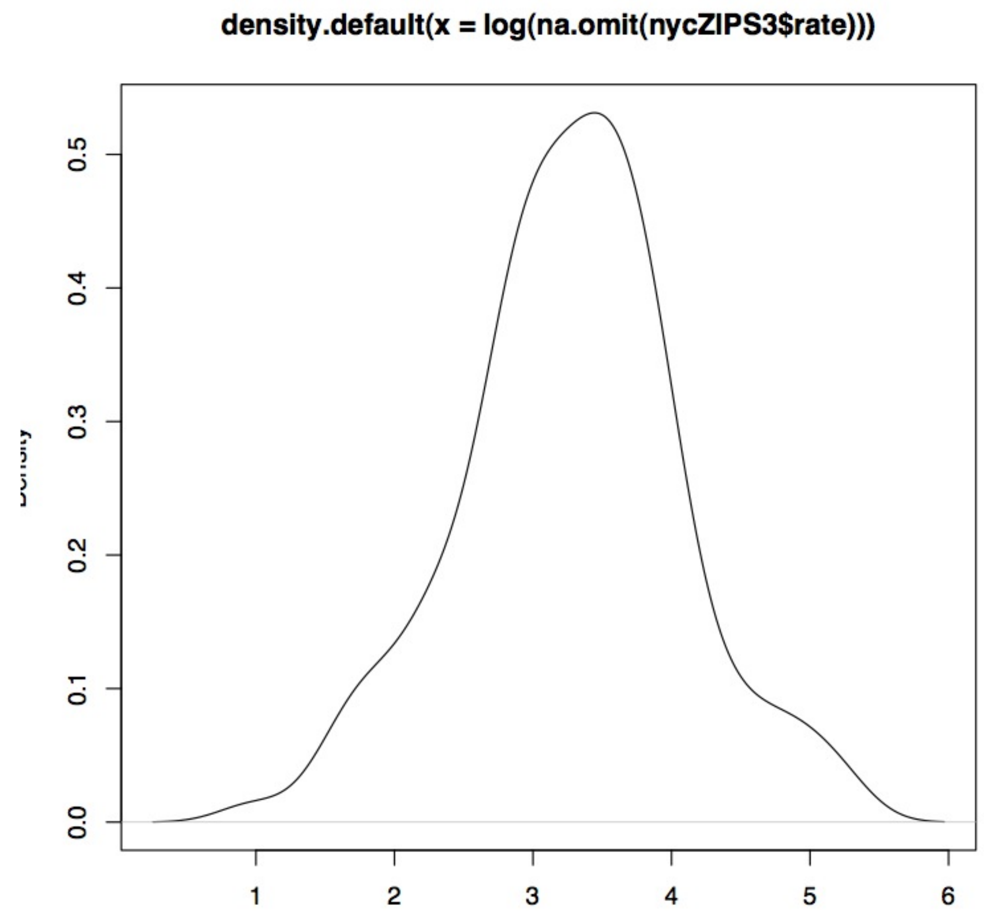
This is closer to what we need to meet our assumptions.

Next, we graph a simple matrix of scatterplots of the numeric variables in our spatial data frame. We first create a data set restricted to just the variables we are interested in. The we use the pairs() procedure to plot them.

```
> scatterData <- cbind(log(nycZIPS3$rate), nycZIPS3$house.rent,
+      nycZIPS3$house.occ, nycZIPS3$rent.dense)
```

```
> colnames(scatterData) <- c("log TBI rate", "rentals", "owned",
+      "rental density")
```

[6] I, in fact, wrote an entire paper just looking at the descriptive statistics of this data set.



```
> library(lattice)
> print(splom(~na.omit(scatterData), panel = function(x, y) {
+     panel.xyplot(x, y)
+     panel.lmline(x, y)
+ }))
```

Turn your attention to the upper left (and conversely the lower right) scatterplots of the association between the log of the TBI rate and rental density. We will explore this possible association.

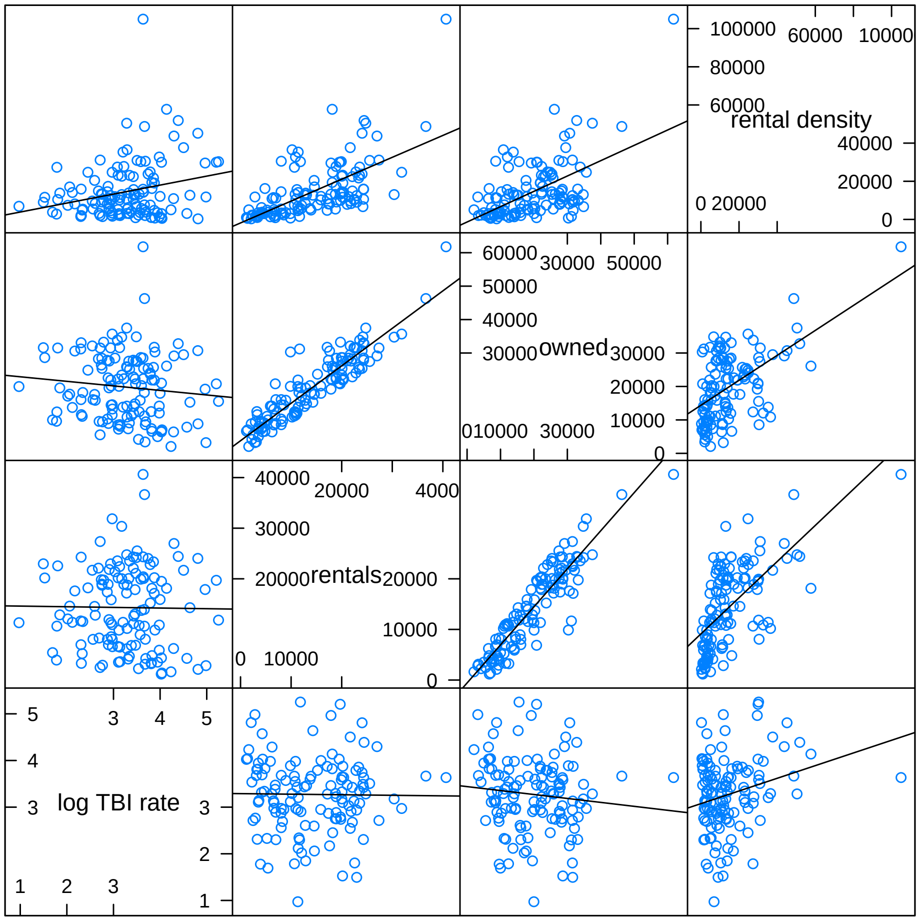
5.3.3 *Modeling the Data*

5.3.3.1 Non-Spatial Models

As I mentioned on the very first page of these sets of notes, whether a spatial component adds anything your analysis is not a given.[7] With that in mind, let's first model the ZCTA-based TBI rates with two models that do not take the spatial component into account.

We first perform a simple linear regression model of the log of the TBI rates by the rental density variable.

[7] To be honest, I think its fairly clear that there is *some* spatial component to these data juse based on our previous Moran's test. But bear with me.



Scatter Plot Matrix

```
> linear.model <- lm(log(rate) ~ rent.dense, data = nycZIPS3)
> summary(linear.model)

Call:
lm(formula = log(rate) ~ rent.dense, data = nycZIPS3)

Residuals:
     Min       1Q   Median       3Q      Max
-2.19566 -0.42572  0.01101  0.48516  1.76676

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 3.075e+00  9.617e-02  31.971  < 2e-16 ***
rent.dense  1.358e-05  4.648e-06   2.922  0.00413 **
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1    1

Residual standard error: 0.77 on 125 degrees of freedom
  (50 observations deleted due to missingness)
Multiple R-squared: 0.06394,        Adjusted R-squared: 0.05646
F-statistic: 8.539 on 1 and 125 DF,  p-value: 0.004127
```

```
> exp(linear.model$coefficients[2])
```

```
rent.dense
  1.000014
```

The model is statistically significant (p=0.004) with each one unit increase in the number of rental units per square mile, we can expect a 1 unit increase (notice we exponentiated back to our original scale) in the TBI rate. Although the adjusted R^2 reflects a poor model fit. Let's see what a Poisson model of the case count returns.

```
> poisson.model <- glm(cases ~ rent.dense + offset(log(pop)), data = nycZIPS3,
+     family = "poisson")
> summary(poisson.model)
```

```
Call:
glm(formula = cases ~ rent.dense + offset(log(pop)), family = "poisson",
    data = nycZIPS3)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-5.1611  -1.2442  -0.0263   0.6799  11.4359

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -6.180e+00  4.548e-02  -135.9   <2e-16 ***
rent.dense   2.399e-05  2.222e-06    10.8   <2e-16 ***
---
Signif. codes:  0 âĂŸ***âĂŹ 0.001 âĂŸ**âĂŹ 0.01 âĂŸ*âĂŹ 0.05 âĂŸ.âĂŹ 0.1 âĂŸ âĂŹ 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 733.89  on 176  degrees of freedom
Residual deviance: 636.63  on 175  degrees of freedom
AIC: 1115.1

Number of Fisher Scoring iterations: 5
```

```
> exp(poisson.model$coefficients[2])
```

```
rent.dense
  1.000024
```

The results are not too very different from those of the linear model. Now let's see what, if anything, the spatial component adds.

5.3.4 Spatial Models

We begin by loading the required R packages and then [8] create spatial neighbor objects we can use with nb2listw() in the spautolm() call.

names(nycZIPS3)

```
> library(sp)
> library(spdep)
> library(tripack)
```

[8] See the previous set of notes on spatial neighbors.

```
> coords <- coordinates(nycZIPS3)
> IDs <- row.names(coords)
> nycZIPS.delauny <- tri2nb(coords, row.names = IDs)
> nycZIPS.soi <- graph2nb(soi.graph(nycZIPS.delauny, coords), row.names = IDs)
> nycZIPS.gabriel <- graph2nb(gabrielneigh(coords), row.names = IDs)
```

```
> tbi.sar.model <- spautolm(log(rate) ~ rent.dense, data = nycZIPS3,
+     nb2listw(nycZIPS.delauny), zero.policy = TRUE)
> summary(tbi.sar.model)
```

```
Call:
spautolm(formula = log(rate) ~ rent.dense, data = nycZIPS3, listw = nb2listw(nycZIPS.delaun
    zero.policy = TRUE)

Residuals:
      Min        1Q    Median        3Q       Max
-2.077661 -0.396714  0.022739  0.428378  2.054764

Regions with no neighbours included:
 133

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) 3.2277e+00 1.4191e-01 22.7455   <2e-16
rent.dense  8.0893e-06 5.2861e-06  1.5303   0.1259

Lambda: 0.5122 LR test value: 21.599 p-value: 3.3595e-06

Log likelihood: -135.2055
ML residual variance (sigma squared): 0.45884, (sigma: 0.67737)
Number of observations: 127
Number of parameters estimated: 4
AIC: 278.41
```

```
> exp(tbi.sar.model$fit$coefficients[2])
```

```
rent.dense
  1.000008
```

While the point estimate remains unchanged from our previous, non-spatial, models, we now see that taking spatial structure and non-independence of the observations into account results in a non-statistically significant p-value.

You may notice that I based the spatial structure on a Delauny graph. We should check the sensitivity of our results to the choice of neighbor structure. Let's see what the results would be with a sphere-of-influence set of neighbors.

```
> tbi.sar.model2 <- spautolm(log(rate) ~ rent.dense, data = nycZIPS3,
+     nb2listw(nycZIPS.soi), zero.policy = TRUE)
> summary(tbi.sar.model2)
```

```
Call:
spautolm(formula = log(rate) ~ rent.dense, data = nycZIPS3, listw = nb2listw(nycZIPS.soi),
    zero.policy = TRUE)

Residuals:
      Min        1Q    Median        3Q       Max
-2.002893 -0.415003  0.019227  0.373568  1.939308

Regions with no neighbours included:
 51 133

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) 3.3484e+00 1.4161e-01 23.6448   <2e-16
rent.dense  1.7235e-06 5.1786e-06  0.3328   0.7393

Lambda: 0.53536 LR test value: 25.189 p-value: 5.1978e-07

Log likelihood: -133.4107
ML residual variance (sigma squared): 0.43512, (sigma: 0.65963)
Number of observations: 127
Number of parameters estimated: 4
AIC: 274.82
```

```
> exp(tbi.sar.model2$fit$coefficients[2])
```

```
rent.dense
  1.000002
```

Clearly, the results are quite sensitive to our choice of spatial structure. Given the apparent clustering of TBI outcomes in our maps, and the fact that New York City neighborhoods tend to vary in similarly non-homogenous ways in terms of socioeconomics and housing, we could legitimately challenge the choice of a global or similtaneous autoregression model. Let's consider a conditional autoregression model.

```
> tbi.car.model <- spautolm(log(rate) ~ rent.dense, data = nycZIPS3,
+     nb2listw(nycZIPS.delauny), zero.policy = TRUE, family = "CAR")
> summary(tbi.car.model)
```

```
Call:
spautolm(formula = log(rate) ~ rent.dense, data = nycZIPS3, listw = nb2listw(nycZIPS.delaun
    family = "CAR", zero.policy = TRUE)

Residuals:
       Min         1Q     Median         3Q        Max
-1.9466438 -0.3952981  0.0064588  0.3869216  2.4550617

Regions with no neighbours included:
 133

Coefficients:
              Estimate Std. Error z value  Pr(>|z|)
(Intercept) 3.1820e+00 1.4210e-01 22.3926 < 2.2e-16
rent.dense  1.7961e-05 5.4514e-06  3.2947 0.0009851

Lambda: 0.76152 LR test value: 19.247 p-value: 1.1487e-05

Log likelihood: -136.3819
ML residual variance (sigma squared): 0.45678, (sigma: 0.67585)
Number of observations: 127
Number of parameters estimated: 4
AIC: 280.76
```

```
> exp(tbi.car.model$fit$coefficients[2])
```

```
rent.dense
  1.000018
```

Modeling the data this way, returns a statistically significant result.

```
> tbi.car.model2 <- spautolm(log(rate) ~ rent.dense, data = nycZIPS3,
+     nb2listw(nycZIPS.soi), zero.policy = TRUE, family = "CAR")
> summary(tbi.car.model2)
```

```
Call:
spautolm(formula = log(rate) ~ rent.dense, data = nycZIPS3, listw = nb2listw(nycZIPS.soi),
    family = "CAR", zero.policy = TRUE)

Residuals:
      Min        1Q    Median        3Q       Max
-1.769910 -0.424080  0.048574  0.420993  2.209880

Regions with no neighbours included:
 51 133

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) 3.6011e+00 1.7096e-01 21.0647   <2e-16
rent.dense  3.6750e-06 5.2747e-06  0.6967    0.486

Lambda: 0.89041 LR test value: 27.628 p-value: 1.4703e-07

Log likelihood: -132.1912
ML residual variance (sigma squared): 0.38861, (sigma: 0.62339)
Number of observations: 127
Number of parameters estimated: 4
AIC: 272.38
```

```
> exp(tbi.car.model2$fit$coefficients[2])
```

```
rent.dense
  1.000004
```

Again, our results are hardly robust to our assumptions.

In the next set of notes, we introduce an approach that may help address some of the infelicities of more traditional linear spatial models.

Chapter 6
Bayesian Hierarchical Spatial Modeling I: Introduction to the Method

6.1 Introduction

The motivation for Bayesian approaches to spatial modeling lies in the difficulties of spatial data that we've discussed. Data points near each other will be very similar in terms of the kinds of variables, like demographics, SES and geographic features, in which we are likely to be interested as epidemiologists, making familiar approaches like linear or logistic regression inappropriate. Poisson models of the kinds of count data we find in spatial epidemiology while an attractive option, are subject to their own difficulties. The data tend to be over-dispersed, meaning that the variance is greater than the mean [1] While a number of effective approaches to spatial data analysis exist, the spatial data we work with as epidemiologists are most often not the kind of highly ordered, 'lattice' or point-process data for which many spatial analytic techniques have been developed.

In this chapter, we'll try to tackle Bayesian Hierarchical Modeling of spatial data. Bayesian analysis is a vast and rapidly expanding field. Space constraints here preclude a more general and thorough treatment of the topic of Bayesian epidemiological analysis. [2] We will for now limit ourselves to focused introduction and then (in the next chapter) return to applications in the New York City TBI vignette.

Most of this section of the notes are based on Andrew Lawson's texts and workshops which are well-worth pursuing if you would like a more thorough treatment of the subject. I particularly recommend Bayesian Disease Mapping. And if you have the opportunity, by all means attend one of Dr. Lawson's workshops.

We'll first consider the types of data and questions for which Bayesian approaches are suited. Then we'll introduce the basic theory of Bayesian statistics, and proceed to describe the gamma-Poisson model as a flexible and useful approach in the multi-level setting frequently encountered in spatially distributed count data. I'll present an example of the methods, using data from Hurricane Katrina in Orleans Parish, Louisiana, and consider some of the advantages and limitations of Bayesian spatial analysis for the practicing epidemiologist.

6.2 The problem with place

As we have seen, aggregating data to the group level based on geography is not simple. Bringing data together spatially frequently results in heterogeneous and arbitrary groupings that may be too large and undifferentiated to capture risk appropriately. Analyses that rely on variable specification based on irregular geographic units, such as ZIP Codes, may be affected by extreme values based on a few cases in small populations. Rare events contribute to more heterogeneity than is assumed by commonly used epidemiological methods like the Poisson models. Finally, epidemiologically

[1] A Poisson distributed variable has a single parameter, λ for both its mean and variance

[2] There are some excellent such texts. I highly recommend Andrew Gelman's introductory text and David Speigelhalter's excellent early book on the subject

influential covariates of an outcome, which may be unmeasured, are likely to be similar in adjacent areas resulting in spatial autocorrelation.

A a basic measure of increased occurrence might be a ratio that compares observed to expected counts of an outcome in a geographic area like a census block. We could then calculate some risk that explains any change from the expected number to the observed number. So, for example, if there were no risk in a particular area, this risk factor would be equal to 1, and the observed number would be equal to the expected number. If, on the other hand, there was some increased or decreased risk of in a particular area this number would be greater than or less than 1. This is a standardized mortality (or morbidity) ratio (SMR), where for region i:

- y_i is the observed count for some outcome
- e_i is *expected* count
- θ_i is the (unknown) parameter for the relative risk

A crude estimate of the risk θ_i would be $smr_i = \frac{y_i}{e_i}$

As noted, this kind of data is subject to the problems of non-independence. Census blocks near each other are likely to be similar in important ways based on geography and population demographics. The areal units are also often defined in irregular and arbitrary ways unrelated to their potential use for health outcomes analyses. In the United States ZIP Codes are intended as a convenient means of delivering mail. They are far from regularly arranged yet they have been treated as lattice-like for point-process data. Finally, count data of the kind often used in spatial injury epidemiology are subject to over dispersion and instability. Small expected numbers in the denominator e.g. say only two household in a census block, can lead to large inflated risk estimates if only one household is affected. These characteristics call into question the suitability of such approaches as simple Poisson models.

6.2.1 The Rev. Bayes meets Dr. Snow

A century before John Snow mapped cholera deaths in London Thomas Bayes, an English minister and mathematician, sparked an approach to conditional probability that bears his name[3] and that addresses, in many respects, the problems inherent in spatial epidemiological analysis. At it's most basic level, the Bayesian approach to knowledge asks: How do we combine what we expect with what we see? Or put somewhat differently, how do we learn from the data to update our knowledge?

Clinicians, who I believe tend to be natural Bayesians, are taught 'hoof beats usually mean a horse is approaching', and that much more information is needed before concluding it is a zebra. We turn to this mode of thinking in our approach to spatial analysis. How does what we expect to see in a region, based on the surrounding regions, impact our conclusions about what we actually see? Bayes Theorem formalizes and quantifies this common-sense approach to evidence and expectations.

6.2.2 From common sense, to numbers.

Statistically, in a Bayesian approach, we base our conclusions about the probability of a risk estimate given our data, $Pr[\theta|y]$, on a combination of our prior expectation, expressed as the probability of observing some risk estimate $Pr[\theta]$, and the likelihood of observing the data with which we are presented, $Pr[y|\theta]$:

$$Pr[\theta|y] \propto Pr[\theta] \cdot Pr[y|\theta] \tag{6.1}$$

[3] Stigler's Law of Eponomy states that 'No scientific discovery is named after its original discoverer'. In this case Thomas Bayes clearly got the ball rolling (if you know his original thought experiment, then pun intended) but folks like Richard Price and especially Pierre Simon Laplace should rightfully have their names attached to the theory. Still, 'Bayesian' has a certain ring to it.

When we have a lot of data based on our observations, the likelihood of that data tends to overwhelm any prior expectations we might have had to the contrary. The less data we have, the more influence our prior expectation will have.

Our *prior distribution* essentially dictates how we believe the parameter θ would behave if we had no data from which to make our decision. What, for example, might we expect is the probability that someone living within 3 miles of a certain location would die from a gun shot wound? Our best guess might be, for example, 1 in 20 or about 5%, and that this probability varies around this point estimate in a normal fashion with a variance of say 0.01 or 1%. This estimate may be based on previous studies, law enforcement data, clinical experience or a combination of sources. What then, if we conduct a study that indicates the risk of firearm related fatality within 3 miles of the location is 45%? How do we revise what we think about the risk of firearm-related deaths in this area? Any revision will depend in large measure on how *likely* these data or observations are given our expectations (and model). This, then, is the second bit of information on the right side of the equation, referred to as the *likelihood* or the *likelihood function* which represents the probability we assign our observed data given the postulated parameters represented by our prior. Our *posterior distribution*, or revised probability of the parameter θ, is a combination of these two probabilities. [4] In a very common sense way, it tells us, for example, that if the results of our study differ markedly from our best existing information we should perhaps be somewhat skeptical.

6.2.3 From numbers to BUGS

For many years, approaches to combining the prior and the likelihood were restricted to a set of situations where the prior distribution was conjugate to, or in the same family, as the likelihood and could be derived by some fairly simple updating of the likelihood function. So, for example, in the setting of an standardized mortality ratio (SMR), a $\Gamma(\alpha,\beta)$ prior is conjugate to a $Pois(\lambda)$ likelihood and can be updated with the information in the likelihood function through the simple formula $\Gamma(\alpha+y,\beta+e)$ [5]

Many reasonably realistic problems, though, are not amenable to conjugate analyses. There may not be an appropriately conjugate prior. They may require higher order differential equations that do not have closed or simple solutions. [6] In these cases, simulation approaches can be use to solve for the parameter estimates, but for may years the required computing power for the kinds of simulations necessary, and more importantly, an approach or algorithm to define a valid sample space from which to simulate were not widely available or widely appreciated.

Software developed over the past decade or so, of which the WinBUGS package developed in the UK is a notable example, takes advantage of both advances in computing and in the understanding of constructing Monte Carol Markov Chains to make such simulations approach. WinBUGS (which stands for Windows Bayes Using Gibbs Sampling) samples from a proposed posterior distribution using either Gibb's (for which it's named) or Metropolis Hasting algorithms.

The details of the sampling schemes used in packages like WinBUGS is beyond the scope of this short description, [7] but it is important to note that this approach carries with it additional responsibilities for the analyst.

---

[4] And leads to the essential mantra of Bayesian analysis: 'The posterior is proportional to the prior times the likelihood.' Which should be invoked like an incantation when doubt and confusion arise.

[5] Don't worry too much if this isn't entirely clear at this point.

[6] For a meaningful appreciation of Bayesian analysis, you may have to review some (though not necessarily a lot) of your college calculus. In this case, you may think of "higher order" as referring to equations that require multiple factors to solve, and "simple or closed" as equations that have only one solution or that can be integrated to 1. This last point is critical, since we're dealing with probabilities which, by definition must sum or integrate to 1.

[7] In addition to Lawson, a couple of excellent references and introductory texts for all this stuff are: *(1)* Albert, J. (2009) Bayesian Computation with R. New York: Springer. *(2)* Banerjee, S., Carlin, B. P., and Gelfand, A. E. (Eds.). (2004). Hierarchical modeling and analysis for spatial data. Boca Raton, Chapman and Hall/CRC. *(3)* Gelman A, Carlin JB. (2009) Bayesian Data Analysis. Boca Raton: Chapman and Hall/CRC. *(4)* Greenland, S. (2006). Bayesian perspectives for epidemiological research: I. Foundations and basic methods. Int J Epidemiol, 35(3), 765-775. and *(5)* Spiegelhalter D, Abrams K, and Myles J. (2004) Bayesian Approaches to Clinical Trials and Healthcare Evaluation. West Sussex, John Wiley and Sons.

First, because these are Markov Chains, each value is dependent on the prior value in the chain; we must assess correlation of sample values. We do this by reviewing autocorrelation graphs and statistics. Ideally, we would like any correlation among values to drop off after the first lag.

Second, since we are trying to sample the (posterior) target distribution as fully and as efficiently as possible, our proposal distribution should sample an appropriately wide area of the proposal distribution. This part of practical Bayesian analysis can get messy, particularly if our proposal distribution is too narrow or our starting value is off somewhere in the hinterlands of a large multi-dimensional target posterior distribution. One approach to this task is to calculate acceptance rates. For random walk chains with normal proposal densities, ideal acceptance rates would be about 50% for one parameter model and about 25% for multi-parameter models. Gibb's sampling, which is based on defining a multi-parameter distribution conditional on one parameter given all the others, can help obviate some of this fine tuning necessary for more traditional Metropolis-Hastings algorithms, which require that the proposal distribution be specified *a priori*.

Third, to help ensure that we are, indeed, sampling from the stable underlying target distribution, we also do things like evaluate whether the chain of sample values actually *converges* to a stable distribution that is consistent with the posterior distribution in which we are interested. Practically, we do this by running 2 or 3 *chains* of samples each starting with an *initial value* chosen from widely dispersed areas of the target distribution, [8] and then evaluating the chains to make sure they are all sampling form the same distribution. This evaluation can be as simple as looking at the kernal density of the distribution graphically to make sure they are reasonably overlapping, to calculating statistics such as the Brooks-Gellman-Rubin (BGR) that compares within chain variation to across chain variation.

Finally, and in some respects most critically, we are obligated to evaluate our choice of prior distributions, some of which may be more influential or appropriate than others. This can be accomplished with sensitivity analyses substituting and evaluating the effects of different prior distributions.

6.2.4 *From BUGS to a model*

Bayesian thinking lends itself naturally to the kind of hierarchical models suited to areal spatial analysis. We can specify not only a distribution for how we believe *individual* risk (θ_i) is distributed, but also, by specifying an additional set of parameters, how we believe θ varies across higher levels of organization, such as geographic units. One could, for example, say that y_i is the empirical (observed) rate of some event in a geographic area i, θ is the true underlying rate, and some additional parameter(s) how that true rate varies across all such areas in which we are interested.

To begin building a model, we must define our prior and our likelihood. Let's start with the likelihood, or data component of the model. For count data of the kind with which we frequently work in epidemiology, we assume an underlying Poisson distribution. Often described as the distribution of rare events, the Poisson distribution is characterized by a single parameter, λ (i.e., both $\mu = \lambda$, and $var = \lambda$). λ is the rate per unit time at which some event k occurs, and the Poisson distribution is defined as:

$$Pois(\lambda) = e^{\lambda^k}/k! \tag{6.2}$$

The y_i counts in area i, are independently identically Poisson distributed and have an expectation in area i of e_i , the expected count, times θ_i , the risk for area i :

$$y_i, iid \sim Pois(e_i\theta_i) \tag{6.3}$$

Having defined our likelihood, we next define a prior distribution for this likelihood. A useful and commonly used prior distribution for θ in the setting of spatial analyses is the gamma (Γ) distribution:

[8] If this sounds a lot easier said than done, it is.

$$\theta \sim \Gamma(\alpha, \beta) \tag{6.4}$$

where, $\mu = \alpha/\beta$, and $var = \alpha/\beta^2$

The Gamma distribution consists of a very flexible class of probability distributions. For example, $\Gamma(1,b)$ is exponential with $\mu = 1/b$, $\Gamma(\frac{v}{2}, \frac{1}{2})$ is chi square distributed with v degrees of freedom. Gamma distributions are constrained to be positive, which is necessary when dealing with count data, and setting the two parameters equal to each other $\alpha = \beta$ results in a null value of 1, which is useful for modeling risk estimates. Finally, the Gamma distribution is conjugate to the Poisson distribution, making our prior and our likelihood of the same family which not only allows for simplified statistics in one parameter problems, but carries with it additional advantages in terms of a valid choice of prior in more complicated analyses.

6.2.4.1 The Poisson-gamma model

Since a basic Bayesian assumption is that any parameter in a problem has a prior distribution of its own, the α and β parameters in the gamma also have prior distributions. The usual approach is to put exponential distributions on α and β.[9]:

$$y_i \sim Pois(e_i \theta_i) \tag{6.5}$$
$$\theta \sim \Gamma(\alpha, \beta) \tag{6.6}$$
$$\alpha \sim exp(v), \tag{6.7}$$
$$\beta \sim exp(\rho) \tag{6.8}$$

Below is an example of the code for this hierarchical model in BUGS language. The code looks a lot like R. But not really.

```
model
{
For (i in 1:m)               # loop to repeat for every spatial level
{
       y[i]~dpois(mu[i])      # Poisson likelihood for observed counts
      mu[i]<-e[i]*theta[i]
      theta[i]~dgamma(a,b)    # relative risk
}
        a~dexp(0.1)           # hyperprior distributions
        b~dexp(0.1)
}
```

Following that is an illustration of the basic hierarchy in a directed acyclic graph (called a 'Doodle' in the BUGS world) for the model (on the left) and a description of the components of the graph (on the right).

[9] At this point we begin to see the inherently hierarchical nature of the Bayesian approach

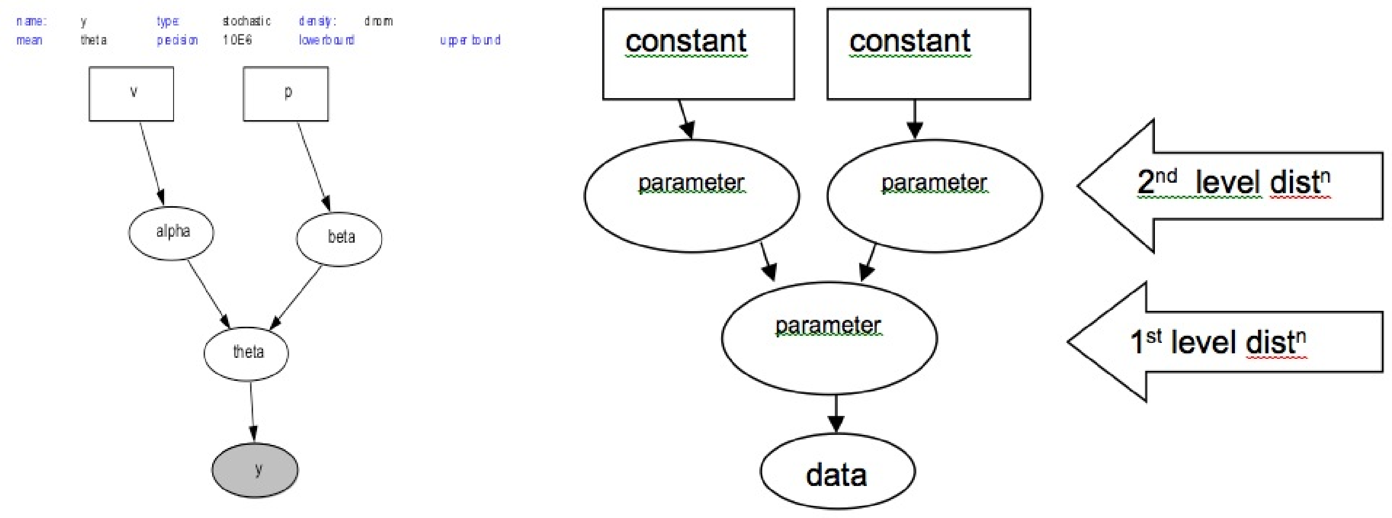


Fig. 6.1 Poisson-gamma Spatial Model Hierarchy

6.2.4.2 Random and spatial effects

Taking the natural logarithm of our hierarchical model allows the inclusion of linear regression terms, a random effects term and a spatial effects term:

$$y_i \sim Pois(\mu_i) \tag{6.9}$$
$$log(\mu_i) = \beta_n + T_1 + T_2 \tag{6.10}$$
$$\tag{6.11}$$

Where β_n represents a vector a log-linear regression terms for variables that might capture potential confounders like age, gender, or socio-economic status, T_1 represents a random effect term, and T_2 represents a spatial effects term.

Random effects terms have been proposed [10] as a useful way to account for group-level heterogeneity . Basically, when there is more variation, or 'noise', in data than is accounted for by the individual-level model and error, we separate out part of the error or residual variance ($r \sim nl(0,\sigma)$) that we believe is due to the groups that give rise to or within which the individuals are nested ($v \sim nl(0,\sigma)$). This variance or heterogeneity though, is not explicitly spatially structured.

The explicit spatial effects component in the model can be represented by a conditional autoregressive (CAR) term, which we first encountered in our discussion of spatial linear models. As you may recall, a CAR model is based on a set of spatial neighborhoods. In the usual formulation, each neighborhood consists of adjacent spatial shapes that share a common border. The mean θ_j in neighborhood j is normally distributed with its parameters defined as μ_j the

[10] Though not always accepted...

average of the μ_{ij}'s in the neighborhood and σ_j equal to the σ's of the neighborhood μ_{ij}'s divided by the number (δ_j) of spatial shapes in the neighborhood. [11]

$$\mu_j \sim nl(\bar{\mu_\delta}, \tau_\mu / n_\delta) \tag{6.12}$$

In much the same way the random effect term captures unstructured heterogeneity, the CAR term captures spatially structured heterogeneity or variance in the data that is not captured by your risk model.

Our updated model, then looks like this:

$$\begin{aligned} y_i &\sim Pois(e_i\theta_i = \mu_i) &(6.13)\\ log(\mu_i) &= \beta_n + \nu_i + \upsilon_i &(6.14)\\ \nu &\sim nl(0, \tau_\nu) &(6.15)\\ \upsilon &\sim nl(\bar{\upsilon_\delta}, \tau_\upsilon / n_\delta) &(6.16) \end{aligned}$$

The WinBUGS package makes it relatively easy to specify a CAR model. You first define an adjacency matrix, which is basically a list of all block groups that share an adjacency. You then define a set of weights for those adjacencies. The most straightforward and most commonly used approach is a weight of 1 when two spatial shapes share an adjacency and a weight of zero when they do not. The following code demonstrates what a model might look like in BUGS:

```
model
{

for( i in 1 : m ) {
      y[i] ~ dpois(mu[i])
      mu[i] <- e[i] * rr[i]
      log(rr[i]) <- b0 + b1*variable1 + b2*variable2 + b3*variable3 +v[i] + h[i]
                                          # h is CAR term
                                          # v is random effects term
r[i]<-(y[i]-mu[i])                        # r is residual

v[i]~dnorm(0,tau.v)        # prior on random effects needs to be inside loop
   }
# priors
b0~dflat()
b1~dnorm(0, 0.001)
b2~dnorm(0, 0.001)
b3~dnorm(0, 0.001)

h[1:m]~car.normal(adj[], weights[], num[], tau.h)     #prior on CAR
 tau.v~dgamma(0.001,0.001)                # priors on v and h
             tau.h~dgamma(0.001,0.001)
}
}
```

6.3 From theory to practice: Post-Katrina Repatriation Model

Before applying these methods to our ongoing example about traumatic brain injury in New York City, I'll illustrate the basic approach with a model I developed to try to document and help explain why some New Orleans neighborhoods seemed to rebound relatively quickly, while others languished (and some continue to languish) for years following

[11] The spatially structured component CAR is sometimes described as a Gaussian process $\lambda \sim Nl(W, \tau_\lambda)$ where W represents the matrix of neighbors that defines the neighborhood structure, and the conditional distribution of each λ_i , given all the other λ_i 's, is normal with μ = the average λ of itÃŢs neighbors and a precision (τ_λ).

Hurricane Katrina. The data is US Postal Service counts of the number of households in a census block actively receiving mail. We compare pre-hurricane counts in June 2005 (our expected number) with June 2009 counts (observed number). θ_i is an estimate for the increase or decrease in the observed vs. the expected number of households receiving mail in a census block group using a Bayesian smoothed estimate of y_i/e_i. (Note that we are interested primarily in the decreased 'risk' of mail delivery in a census block). Based on some preliminary, non-spatial, analyses, we are interested in the role of income and geographic elevation above sea-level as predictors of repatriation.

We first plot the unadjusted observed vs. the expected proportion of households receiving mail in block groups pre and post hurricane. The results are fairly heterogeneous, with some areas actually experiencing a net increase in population following the disaster.

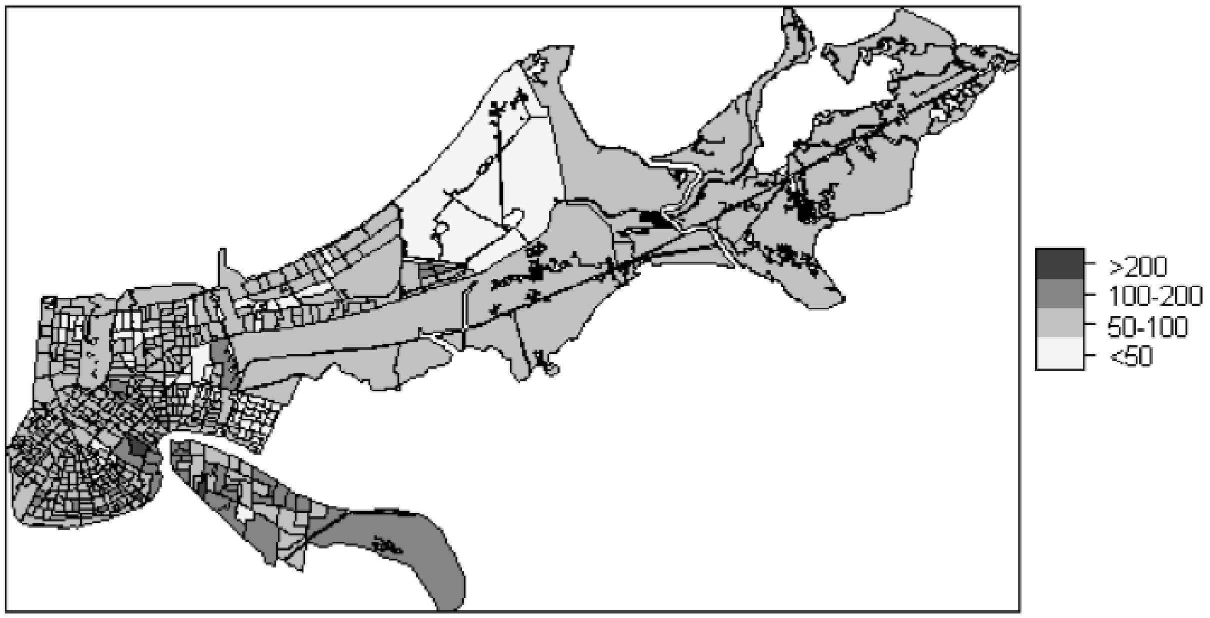


Fig. 6.2 Percent Observed vs. Expected Households Receiving Mail, Orleans County, LA, June 2005 vs. May 2009, by Census Block Group.

We then map our social and environmental variables to see how they vary geographically and if that variation may be consistent with changes in repatriation.

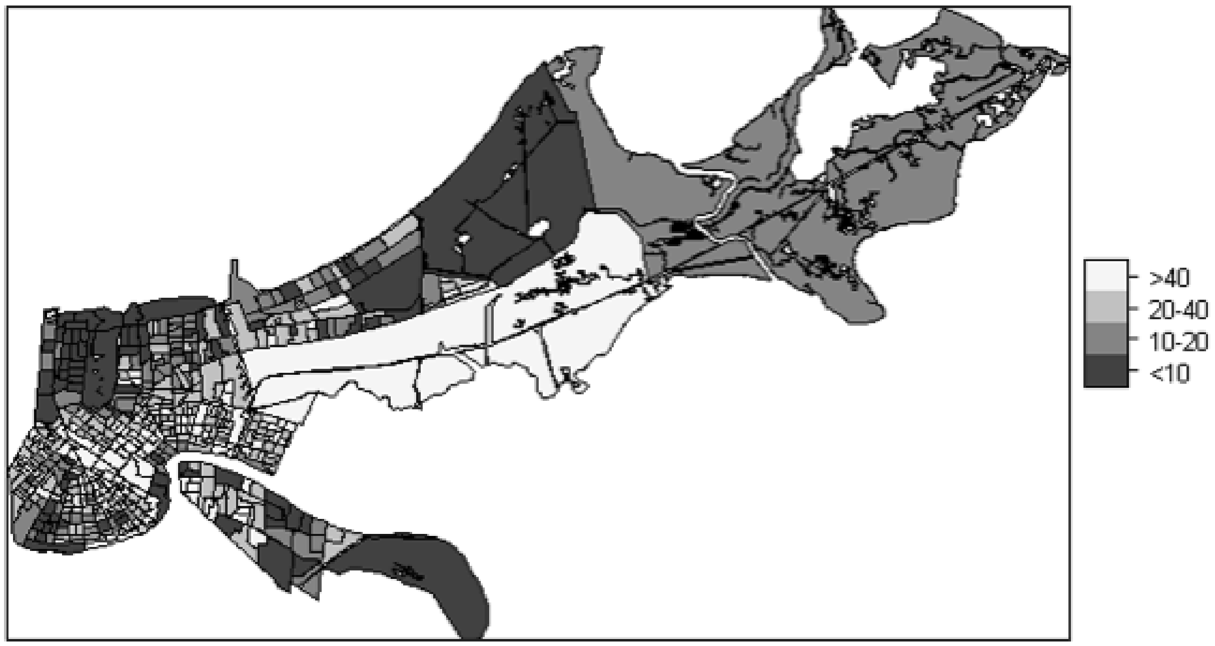


Fig. 6.3 Proportion of Population Living Below Federal Poverty Level, Orleans County, LA, 2000, by Census Block Group.

There appears to be, even on this subjective level, an inverse relationship between these two variables. Our next step, is to examine their relationship to repatriation more rigorously. For this, we turn to a Bayesian hierarchical model.

In addition to assessing whether poverty and/or elevation above sea level explains whether folks returned to their homes in Orleans Parish after Hurricane Katrina we also include a conditional auto-regression (CAR) term to 'smooth' the outcome estimate and a a random effects term to capture additional non-modeled variability across the ecologic units of census block groups. We compared models using the Deviance Information Criterion (DIC) which is a Bayesian analogue of the Akaike Information Criterion (AIC), where a relatively smaller value is considered a 'better' model.

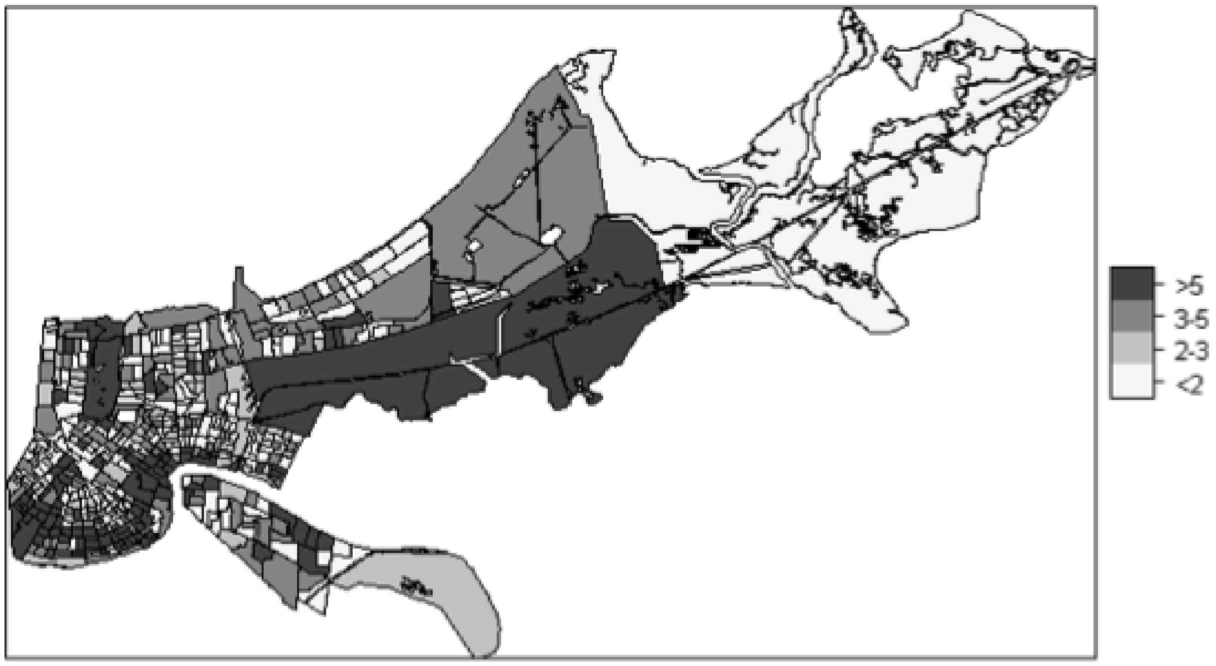


Fig. 6.4 Elevation (in Meters) Above Sea Level. Orleans County, LA, by Census Block Group

We found that by this criterion, a model consisting of the CAR term, poverty, elevation and an interaction term for poverty and elevation is the 'winner'.

```
  model
{
for( i in 1 : m ) {
      y[i] ~ dpois(mu[i])
      mu[i] <- e[i] * rr[i]
log(rr[i]) <- b0 + b1*((elevation[i]-3.7)/3.6) + b2*((poverty[i]-28.4)/18.2) + b3*((elevation[i]-3.7)/3.
r[i]<-(y[i]-mu[i])                                  # r is residual
prob50[i]<-step(0.5-rr[i])                # step function to calculate and plot exceedence
   }
h[1:m]~car.normal(adj[], weights[], num[], tau.h) #no need weights in model get from matrix  #define CAR
                         b0~dflat()
b1~dnorm(0, 0.001)
b2~dnorm(0, 0.001)
b3~dnorm(0, 0.001)
tau.h~dgamma(0.001,0.001)    }
```

Notice that we transformed or re-parameterized our poverty and elevation variables by subtracting the mean dividing by the standard error. The explanation for this lies in assessing convergence to the assumed posterior distribution. Our first plot tracings of the Markov chains appear to be 'wandering' and not efficiently sampling the the sample space for the posterior distribution.

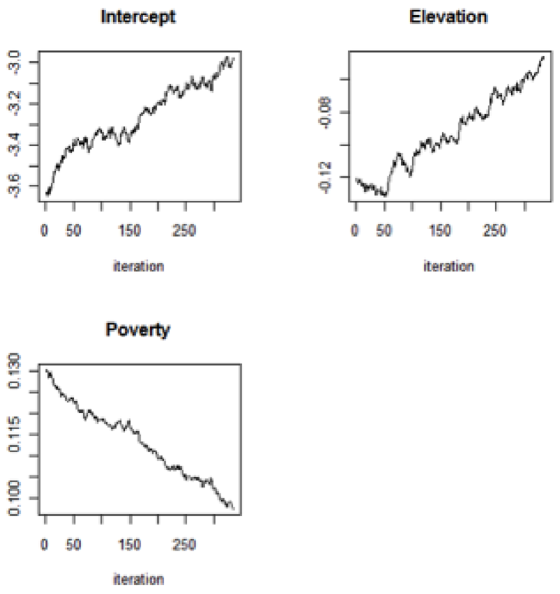


Fig. 6.5 Sample tracings, before re-parameterization.

This issue of potential non-convergence prevents us from making valid inferences on the parameters. There are a few ways to address it. We can re-parameterize the problematic variables by centering them (subtracting the mean value), standardize them (dividing by the standard error) or transform them (taking the log of the value). We find that centering and standardizing the values returns plot tracings much more consistent with convergence. We can test this assessment through use of a statistic, such as the Brooks-Gellman-Rubin statistic, which in this case indicated adequate convergence.

Having fit the model, we now proceed to make inferences and apply the results to smoothing the mapped risk estimates and identifying areas of unusually low repatriation rates. We begin by plotting the probability distributions for our parameter estimates. These are probability density plots of the beta coefficients. That last statement should give you some pause, and actually represents perhaps the most profound difference between a Bayesian approach and the more commonly encountered frequentist or Pearson-Neyman approach to biostatistics. From these data, we can directly calculate the probability of a parameter value. [12]

[12] Take a moment to think about that, and ponder wither the p-value.

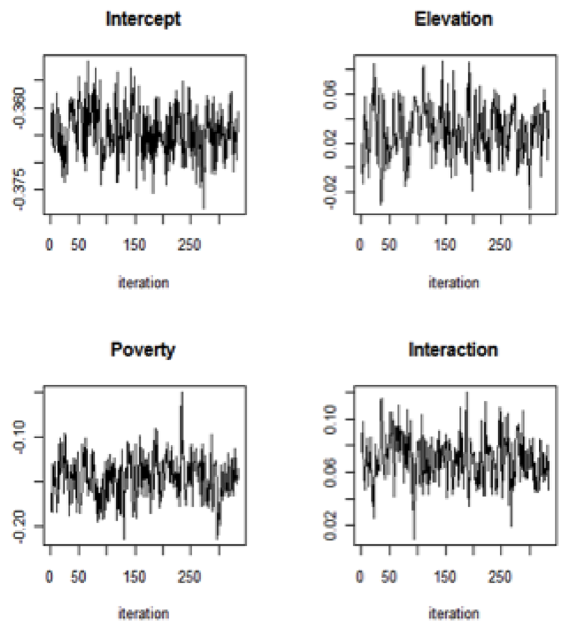


Fig. 6.6 Sample tracings, after re-parameterization.

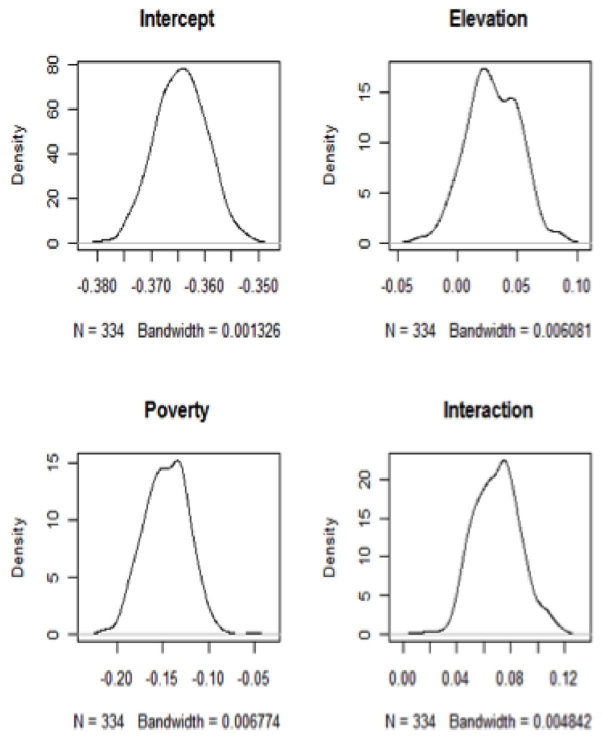


Fig. 6.7 Kernel density plots of model parameters.

Based on these probability distributions, we can easily calculate the mean and upper and lower credible intervals at the 2.5% and 97.5 probability level for our parameters.

We see these estimates in the following table.

Node (parameter)	mean	sd	MC error	2.5%	97.5%
Intercept	-0.3643	0.004737	1.777E-4	-0.3739	-0.3552
Elevation	0.02958	0.02042	9.978E-4	-0.00882	0.06738
Poverty	-0.1443	0.025	0.001272	-0.1913	-0.09626
PovertyElevation	0.07187	0.01735	8.302E-4	0.04037	0.1067

The MC error refers to the Monte Carlo error. It is the standard deviation divided by the number of sampling iterations, so is an indication of the increasing precision of our estimates as our sampling increases. As a rule of thumb, the Monte Carlo error should be 1% to 5% of the posterior standard deviation. We conclude from these results that the percentage of individuals below the poverty level was the most consistent predictor of repatriation following this disaster, even more so than elevation above sea level.

The CAR model also returns risk estimates for each census block in our data, conditioned on all the surrounding risk estimates. We can map these results as smoothed risk estimates.

Comparing this map to our previous un-smoothed map (Figure 2) we see there has been some ‘shrinkage’ toward local mean values.

With this admittedly healthy preamble, we turn our attention once again to traumatic brain injury in New York City to see if a Bayesian approach returns more valid or reliable results than those of non-Bayesian approaches.

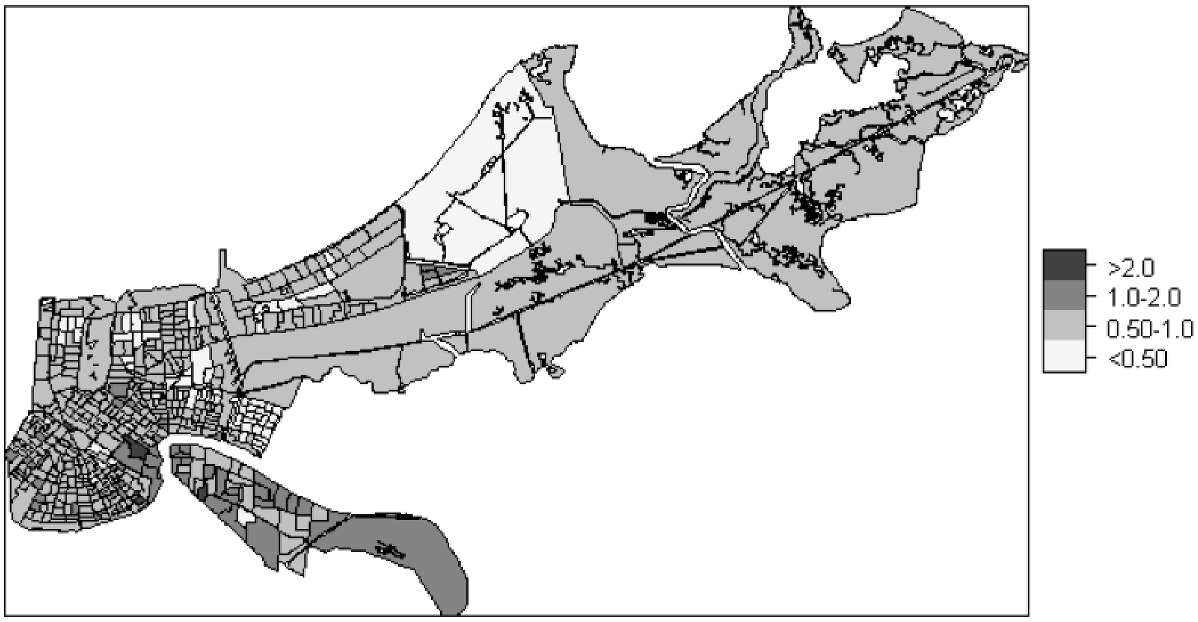


Fig. 6.8 Bayesian Smoothed Risk Estimates Orleans Parish Census Block-Group Level Depopulation September 2008 vs. August 2005 Based on Number of Household Receiving Mail Delivery

Chapter 7

Bayesian Hierarchical Spatial Modeling II: From Theory to Application

7.1 Back to the TBI Data

Let's apply what we've learned about Bayesian spatial modeling to the TBI example we've been working through. We begin by loading the TBI spatial object we've been painstakingly building up.

Even after all our efforts, there is additional data manipulation required. Up to now, we've been working with rates. The Poisson model upon which we will build our hierarchical Bayesian model, is based on counts. So, it's back to the original data file to extract the counts upon which our rates are based. We create two variables (observed and expected) from counts for children under age 3, age 3 to 4, 5 to 6 and 7 to 8. The observed count is simply the number of TBI diagnoses in the ZCTA. The expected count is calculated based on age and gender specific rates standardized to the city as a whole.[1]

Next, we have to add those counts to our spatial dataframe object. Again, some care is required. We first compare the ZCTA's in the data object to the spatial object. There are 172 matching ZCTA's, which is good. Looking for NA's in the existing SMR variable in the spatial object,[2] we can identify the 5 ZCTA's that do not have count data. We use a logical vector based on this variable to index the spatial object to remove the ZCTA's that have no data, creating a new spatial object called nycZIPS4.

```
> library(sp)
> tbizip$zips[tbizip$zips %in% nycZIPS3$zcta5]
```

```
  [1] "10001" "10002" "10003" "10004" "10007" "10009" "10010" "10011" "10012"
 [10] "10013" "10014" "10016" "10017" "10018" "10019" "10021" "10022" "10023"
 [19] "10024" "10025" "10026" "10027" "10028" "10029" "10030" "10031" "10032"
 [28] "10033" "10034" "10035" "10036" "10037" "10038" "10039" "10040" "10044"
 [37] "10128" "10280" "10301" "10302" "10303" "10304" "10305" "10306" "10308"
 [46] "10309" "10310" "10312" "10314" "10451" "10452" "10453" "10454" "10455"
 [55] "10456" "10457" "10458" "10459" "10460" "10461" "10462" "10463" "10464"
 [64] "10465" "10466" "10467" "10468" "10469" "10470" "10471" "10472" "10473"
 [73] "10474" "10475" "11004" "11005" "11040" "11101" "11102" "11103" "11104"
 [82] "11105" "11106" "11201" "11203" "11204" "11205" "11206" "11207" "11208"
 [91] "11209" "11210" "11211" "11212" "11213" "11214" "11215" "11216" "11217"
[100] "11218" "11219" "11220" "11221" "11222" "11223" "11224" "11225" "11226"
[109] "11228" "11229" "11230" "11231" "11232" "11233" "11234" "11235" "11236"
[118] "11237" "11238" "11239" "11354" "11355" "11356" "11357" "11358" "11360"
[127] "11361" "11362" "11363" "11364" "11365" "11366" "11367" "11368" "11369"
[136] "11370" "11372" "11373" "11374" "11375" "11377" "11378" "11379" "11385"
[145] "11411" "11412" "11413" "11414" "11415" "11416" "11417" "11418" "11419"
```

[1] You may have to go back to your introductory public health textbook to remind yourself how to standardize rates. I know I did.

[2] I created this in an earlier section.

```
[154] "11420" "11421" "11422" "11423" "11426" "11427" "11428" "11429" "11430"
[163] "11432" "11433" "11434" "11435" "11436" "11691" "11692" "11693" "11694"
[172] "11697"
```

```
> nycZIPS4<-nycZIPS3[!is.na(nycZIPS3$smr),]
```

Then we add the expected and observed counts to the new spatial dataframe object. For good measure, we check that the ZCTA's do, indeed, line up. To make sure all our numbers are internally consistent, we re-calculate the SMR variable. Then we save the new data file.

```
> nycZIPS4$expect<-tbizip$expect[tbizip$zips %in% nycZIPS4$zcta5]
> nycZIPS4$observe<-tbizip$observe[tbizip$zips %in% nycZIPS4$zcta5]
> as.numeric(tbizip$zips[tbizip$zips %in% nycZIPS4$zcta5])
```

```
  [1] 10001 10002 10003 10004 10007 10009 10010 10011 10012 10013 10014 10016
 [13] 10017 10018 10019 10021 10022 10023 10024 10025 10026 10027 10028 10029
 [25] 10030 10031 10032 10033 10034 10035 10036 10037 10038 10039 10040 10044
 [37] 10128 10280 10301 10302 10303 10304 10305 10306 10308 10309 10310 10312
 [49] 10314 10451 10452 10453 10454 10455 10456 10457 10458 10459 10460 10461
 [61] 10462 10463 10464 10465 10466 10467 10468 10469 10470 10471 10472 10473
 [73] 10474 10475 11004 11005 11040 11101 11102 11103 11104 11105 11106 11201
 [85] 11203 11204 11205 11206 11207 11208 11209 11210 11211 11212 11213 11214
 [97] 11215 11216 11217 11218 11219 11220 11221 11222 11223 11224 11225 11226
[109] 11228 11229 11230 11231 11232 11233 11234 11235 11236 11237 11238 11239
[121] 11354 11355 11356 11357 11358 11360 11361 11362 11363 11364 11365 11366
[133] 11367 11368 11369 11370 11372 11373 11374 11375 11377 11378 11379 11385
[145] 11411 11412 11413 11414 11415 11416 11417 11418 11419 11420 11421 11422
[157] 11423 11426 11427 11428 11429 11430 11432 11433 11434 11435 11436 11691
[169] 11692 11693 11694 11697
```

```
> nycZIPS4$smr<-(nycZIPS4$observe/nycZIPS4$expect)*100
> save(nycZIPS3, nycZIPS4, file = "/Users/charlie/Dropbox/spatialEpiHowTo/spatialEpiBayes/
```

7.2 Unadjusted SMRs

Let's take a quick look at the newly created SMR's by printing them out. Even this cursory look shows how unstable they are.

```
> nycZIPS4$smr
```

```
  [1]    99.601594    49.993751   398.406375     0.000000     0.000000
  [6]   233.177881 75000.000000   199.203187    99.601594    66.577896
 [11]     0.000000   498.007968     0.000000 25000.000000    99.800399
 [16]    99.601594     0.000000    99.601594    99.601594   128.498001
 [21]   199.885780    72.700836   199.203187   115.765102    85.665334
 [26]   187.453137   593.601600   508.163945   329.868053   149.925037
 [31]     0.000000   199.600798   199.600798     0.000000     0.000000
 [36] 50000.000000   498.007968     0.000000    66.622252    49.950050
 [41]   159.872102    71.387778   174.825175   149.700599     0.000000
 [46]   398.406375    74.925075     0.000000   179.856115   107.659182
 [51]   103.434009   144.807613    64.267352   107.659182    93.738283
 [56]   156.645781   139.265819   121.393887   178.909703   119.904077
 [61]    49.985718    89.964014     0.000000    49.900200   143.714071
 [66]    80.756807   169.535733    63.613232   199.600798     0.000000
 [71]     0.000000     0.000000   149.850150     0.000000     0.000000
 [76]     0.000000     0.000000    49.966689    28.555111    42.832667
 [81]    19.984013    24.975025    28.555111     0.000000    58.809692
 [86]    88.869140     0.000000   127.979523   111.094653    86.655113
 [91]    22.212350     8.330556   102.929067   129.610428    57.882551
 [96]    73.313783     0.000000   153.798831     0.000000    52.370977
[101]    51.214516    65.449785   126.065032    49.950050    53.319115
[106]    59.976010    78.548986    64.507805    39.968026    29.988005
[111]    61.097534     0.000000    72.700836   105.857445    33.318525
[116]    38.449708   166.622234    99.981821     0.000000   398.406375
[121]    62.468766    14.997001    74.925075     0.000000    74.925075
[126]     0.000000     0.000000 25000.000000     0.000000     0.000000
[131]   124.875125     0.000000    66.622252    59.994001    33.318525
[136]    19.984013    38.880249    33.328889    24.975025    49.950050
[141]    14.997001     0.000000     0.000000   110.503052   398.406375
[146]    39.968026   159.872102     0.000000    99.800399    99.920064
[151]   139.888090    99.955575     0.000000     0.000000     0.000000
[156]    49.950050    19.984013     0.000000     0.000000     0.000000
[161]     0.000000     0.000000    35.704085    87.456272    66.644452
[166]     0.000000     0.000000   128.534704   159.872102   199.203187
[171]   498.007968     0.000000
```

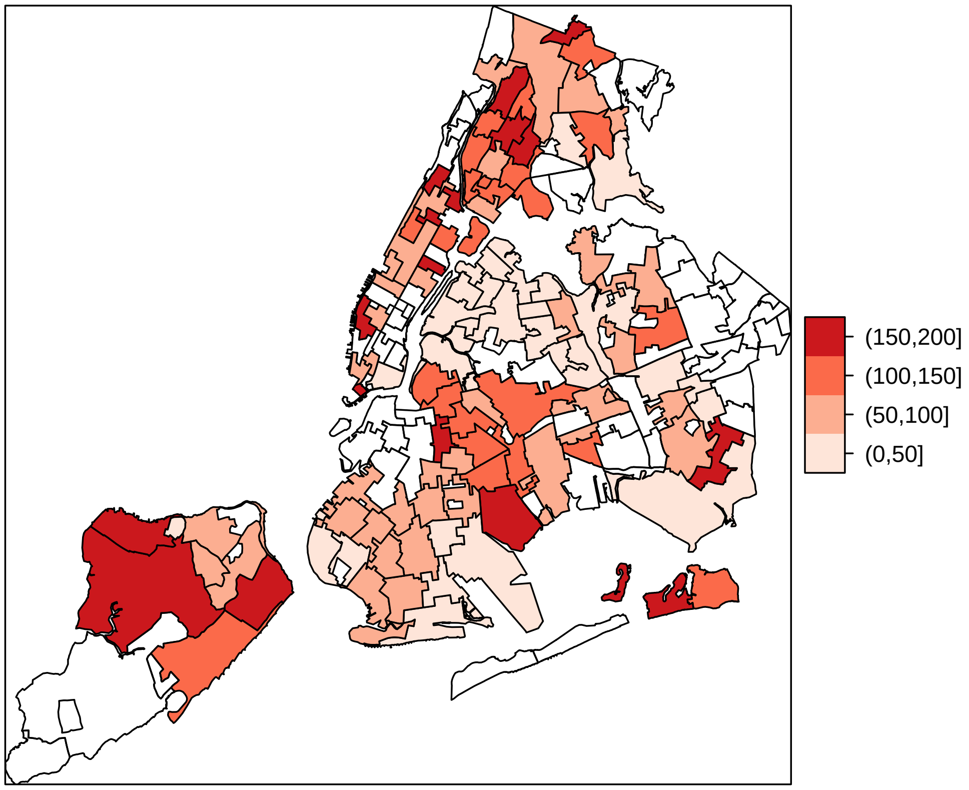
We can get a better sense of the spread with a set of summary statistics.

```
> summary(nycZIPS4$smr)
```

```
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
    0.00     0.00    64.98  1105.00   128.10 75000.00
```

We'll next plot the SMR's using some of the tools we went over in previous chapters.

```
> library(classInt)
> library(RColorBrewer)
> nycZIPS4$smr.cat<-cut(nycZIPS4$smr, breaks=c(0,50,100,150,200))
> smr.plot1<-spplot(nycZIPS4, "smr.cat", col.regions = brewer.pal(4, "Reds"))
> print(smr.plot1)
```



Clearly, some areas of the city appear to pose an increased risk of TBI to children on Medicaid. We know, though, that some of these high risk estimates are based on extreme values, and that some of those extreme values are due to unstable, over-dispersed estimates. We now turn to a Bayesian modeling to address this issue.

7.3 Connecting to and using WinBUGS

WinBUGS, and its open-source cousin OpenBUGS are programs for running Bayesian inference Using Gibbs Sampling (BUGS). They are not the only programs that will run Bayesian analyses. Another stand-alone program, which I like because it plays nicely with Mac OS X, is Just Another Gibbs Sampler (JAGS). There are also an increasing number of R packages and functions for doing Bayesian analysis. Just look at the Bayesian page of the CRAN Task View engine.

As of this writing, though, WinBUGS is the only program in which the spatial CAR model we've been discussing is available. [3]

The WinBUGS program comes with it's own set of issues and requirements. It is, as it's name implies, a Windows-only program. and, it's a very "point and click" experience. Almost to the point of distraction. There's a way to write

[3] I really do like Martyn Plummer's JAGS program. Dr. Plummer describes it as "not unlike BUGS" but it does most anything WinBUGS does. And, in my opinion, much more quickly with a minimum of fuss. I anxiously await the possibility that he will include a CAR spatial model in a future release. Though I'm not holding my breath.

scripts, but you'll have to learn (yet) another programming language. It does have a nice graphical interface based on directed acyclic graphs (called "Doodles" in BUGS world) where you can actually draw your model, that is great for illustrating your model, though not so great for actually writing it and tweaking it. OpenBUGS operates very much like WinBUGS. JAGS is syntax driven. The model description language looks very much like WinBUGS, but the scripting has quirks and idiosyncrasies of its own.

Fortunately, there a way around all of these issues because you can actually call WinBUGS, OpenBUGS and JAGS from within R itself, using familiar (by now) R language. Also, by working in R, you create R objects that integrate seamlessly with all the spatial tools we've been discussing.

First you have to install the BUGS program on you computer. Then you write a model in a text file and save it in a file somewhere. Then, you make a connection to the BUGS program from within R by using a function in a package like *R2WinBUGS*() or *R2JAGS*() that tells BUGS where the model file is, what the data are, and what parameters you want to define the variables. If all goes well, R connects to BUGS, supplies it with the information you provided to the R function, runs the simulation, and returns the results as an object that can be analyzed in R.

There are some computer platform issues that you may have to address. If you work exclusively in Windows, then the process is actually quite straightforward and will work "out of the box". If, though, (like me) you use a Mac or linux machine, and you want to use WinBUGS, you can only access the program with an emulator or virtual machine. I've found Parallels to be the easiest solution. It costs money, but it works. With Parallels I can also easily burrow through from my virtual Windows machine, where WinBUGS is running, to my files on the Mac side. Freeware approaches exist. They work for many folks. Not for me, though.

My approach is to do the analysis by running both R and WinBUGS in a virtual Windows machine, using *choose.files()* to track down my data files on the Mac side. Again, there open-source approaches And, again, I've never been able to get them to work. [4]

Again, the only reason I've found to *have* to use WinBUGS is for the spatial CAR model. Otherwise, I use JAGS and R2JAGS which work natively on Mac OS X.

7.3.1 Installing WinBUGS

Installing and setting up WinBUGS is fairly simple.

- First, assuming you have a Windows machine or some kind of virtualization software on a non-Windows machine. Go to the WinBUGS homepage and follow the instructions under 'Quick Start'. Unless you are certain you simply must have the 64-bit version (rare) just ignore step two. Make sure you install the update patch and the key.
- Second, after you've installed WinBUGS, install the *R2WinBUGS* package in R. If you are using a non-Windows machine, *you will also have to install a Windows version of R on your virtual machine.* This can be a bare-bones installation. The only package you will need is, again, R2WinBUGS.
- Third, before moving on to the next section, I strongly recommend you spend a little time working through this vignette. It does a much better job than I can do to introduce you to the mechanics of running a WinBUGS simulation from within R. See if you can get the software to work. See if you understand what the analysis is actually doing.

[4] Although, hope springs eternal. While writing this, I came across this claim to have solved the issue for Mac users.

7.4 Bayesian Model of the TBI Data

We will now go through the process of analyzing our TBI data from a hierarchical Bayesian perspective using a spatial CAR model. [5]

7.4.1 Setting up the model, data and parameters

We begin with a basic model of the kind we defined in the previous chapter.

```
model
{
for( i in 1 : m ) {
      y[i] ~ dpois(mu[i])
      mu[i] <- e[i] * rr[i]
      log(rr[i]) <- b0 +v[i] + h[i]

r[i]<-(y[i]-mu[i])
prob2x[i]<-step(rr[i]-2)
v[i]~dnorm(0,tau.v)
   }
b0~dnorm(0,tau.b0)
h[1:m]~car.normal(adj[], weights[], num[], tau.h)

tau.b0~dgamma(0.001,0.001)
tau.v~dgamma(0.001,0.001)
    tau.h~dgamma(0.001,0.001)
}
```

Let's go through the terms to remind ourselves of what we are doing here:

y   is our data, or Likelihood, here the the *Poisson*-distributed TBI count in area *i* defined by the parameter *mu*

mu   continues the definition of the Likelihood by decomposing the *mu* parameter into the expected count *e* and the risk *rr* in area *i*

v   the unstructured heterogeneity or random effects term in our log-transformed model of risk in area *i*

h   the spatially structured component of the random effects term

r   a residual defined by the difference between the observed and modeled counts

prob2x   a step function to calculate (and plot) the "exceedence" or probability of twice the risk in area *i* [6]

dnorm(0,tau.v)   normal prior on the unstructured heterogeneity

dnorm(0,tau.b0)   normal prior on the intercept term in the logistic model

car.normal   the conditional autoregressive prior on the spatial heterogeneity term defined by an adjacency matrix with weights which we will create using *poly2nb()* and *nb2WB* respectively and identify as part of our BUGS call from R

tau.b0/v/h   gamma priors on the precision terms for previously defined variables

We write this up as a text (.txt) file and save it in a directory or folder on our machine.

[5] Refer to the previous set of notes if this doesn't make sense to you.

[6] This approach to mapping out risk is based on Nicky Best, Sylvia Richardson and Andrew Thomson's article "A comparison of Bayesian spatial models for disease mapping" in Statistical Methods in Medical Research 2005; 14: 35-59

Now we turn to setting up the data and variables to run the model. We begin by creating the adjacency matrix from our spatial object and weights from the adjacency object for use with the car.normal prior on the spatial heterogeneity term. [7]

```
> library(sp)
> library(maptools)
> library(maps)
> library(spdep)
> tbinb <- poly2nb(nycZIPS4)          #adjacency matrix
> tbiWBweights<- nb2WB(tbinb)         #weights
>
```

In the code above, we first load the required packages. Then we extract the coordinates and use *graph2nb()* to create the Gabriel graph adjacency matrix. [8] Then we use the utility function *nb2WB()* to create the weight matrix in WinBUGS format. [9]

Next, we create the data objects, variable definitions and initial values [10] for the *bugs()* call in *R2WinBUGS*

```
> m<-nrow(nycZIPS4) # for loop number
> d<-c(list(y = nycZIPS4$observe, e = nycZIPS4$expect, m=nrow(nycZIPS4)))
> inits1<-list(b0 = 1, v = rep(0, m), h = rep(0, m), tau.v = 1, tau.h = 1, tau.b0 = 1)
> inits2<-list(b0 = 10, v = rep(1, m), h = rep(1, m), tau.v = 0.1, tau.h = 0.1, tau.b0 = 0
```

Lastly, we save these objects in an R data file which we'll open in a separate R session under Windows. [11]

```
> save(tbinb, tbiWBweights, d, inits1, inits2, file = "/Users/charlie/Dropbox/spatialEpiHo
```

7.4.2 Running the WinBUGS model

Assuming we are now in a Windows environment, the following code (in R) will run the model we prepared in the previous subsection and save the outputted "bugs" object for later use.

```
choose.files()
load("Z:\\Dropbox\\spatialEpiHowTo\\spatialEpiBayes\\winbug.RData")
library(R2WinBUGS)
tbi.bugs <- bugs(
data = c(d, tbiWBweights),
inits = list(inits1, inits2),
parameters.to.save = c("rr", "r", "v", "h", "prob2x"),
model.file = "Z:\\Dropbox\\spatialEpiHowTo\\spatialEpiBayes\\model_3.txt",
n.chains = 2, n.iter = 40000, debug=TRUE,
bugs.directory = "C:\\WinBUGS14",
)
save(tbi.bugs, file="Z:\\Dropbox\\spatialEpiHowTo\\spatialEpiBayes\\tbibugs.RData")
```

[7] You may note that I am using the function *poly2nb()* to create the adjacency matrix, rather than the *graph2nb(gabrielneigh(coords)* approach we've used before. The spatial CAR model requires the adjacency matrix and weights to be symmetric. *poly2nb()*, while not as optimal, provides good results in a structure that works with the CAR model. I am, though, planning to explore ways to get a good symmetric Gabriel graph matrix to work in WinBUGS.

[8] You will note that I am using See the chapter on Neighbors for details on why we chose this definition of neighbors.

[9] There is another function, *poly2nb()* in (I believe) the maptools package that will also create an adjacency matrix that can be fed to *nb2WB()* which I've used in some analyses. It does not, though, create adjacencies that cross natural barriers like rivers in the New York City spatial object.

[10] We will need to tell BUGS a value to use to start the MCMC. It is best to have a couple of chains running, each starting at a different value. See the previous chapter for details.

[11] If you are working in Windows, this is not necessary...

A couple of points about this code. First, because my data objects are on the Mac side of my machine, I load the file I created. Note that I'm using the *choose.files()* function to locate that file. It's a little trick to get the correct path from my virtual Windows machine to my Mac in the format that R wants it. Next, I load the R2WinBUGS package. The heart of the matter is the call to *bugs()*. If you went through the vignette as I (strongly) suggested this should make sense to you. We identify the data, set initial values, tell WinBUGS what variables we want to monitor. We then identify the location of the model file. Note that the file is saved on the Mac side of my machine, not in my virtual, parallel's machine. We next tell WinBUGS we want to run two chains of 40,000 iterations (a relatively long run, actually). I have set the debug option to TRUE. This is not strictly necessary, but this keeps the WinBUGS program open so I can go over the results and graphs. If the model fails to run, this option allows you to figure out what went wrong. [12] Finally, we point R in the direction of our WinBUGS program. Also, note that I am storing the results of this run in an object called "tbi.bugs".

Once, I've created a "bugs" object (here named "tbi.bugs"), I can save it, and leave the windows environment. The next code represents my return to the Mac side, where I open R2WinBUGS (which I will need to work with the bugs object I created on the Windows side), and load the bugs object tbi.bugs.

```
> library(R2WinBUGS)
> load("/Users/charlie/Dropbox/spatialEpiHowTo/spatialEpiBayes/tbibugs.RData")
```

7.5 Assessing the Output

7.5.1 Using Base R Tools to Assess WinBUGS Output

We can explore and analyzethe tbi.bugs object as we would any R object. We can print it out (*print(tbi.bugs)*) or request the structure (*str(tbi.bugs)*). [13] But I usually start off with a graphical inspection. Let's begin by looking at the named lists that make up tbi.bugs.

```
> names(tbi.bugs)
```

```
 [1] "n.chains"        "n.iter"          "n.burnin"        "n.thin"
 [5] "n.keep"          "n.sims"          "sims.array"      "sims.list"
 [9] "sims.matrix"     "summary"         "mean"            "sd"
[13] "median"          "root.short"      "long.short"      "dimension.short"
[17] "indexes.short"   "last.values"     "isDIC"           "DICbyR"
[21] "pD"              "DIC"             "model.file"      "program"
```

The list objects "sims.array", "sims.list" and "sims.matrix" contain the actual MCMC output or "chains". We will be working with, and mapping, the mean value for each of these chains. But before we do that, we will want to ensure ourselves that the chains themselves can reasonably be accepted as stably converged to the desired posterior distribution. We do this, primarily, by inspecting graphs. There are also some summary statistics we can request.

The three MCMC-chain list objects differ in how they are structured and how many of the chain observations they preserve. Let's start off by looking at sims.array.

```
> dim(tbi.bugs$sims.array)
```

```
[1] 500   2 856
```

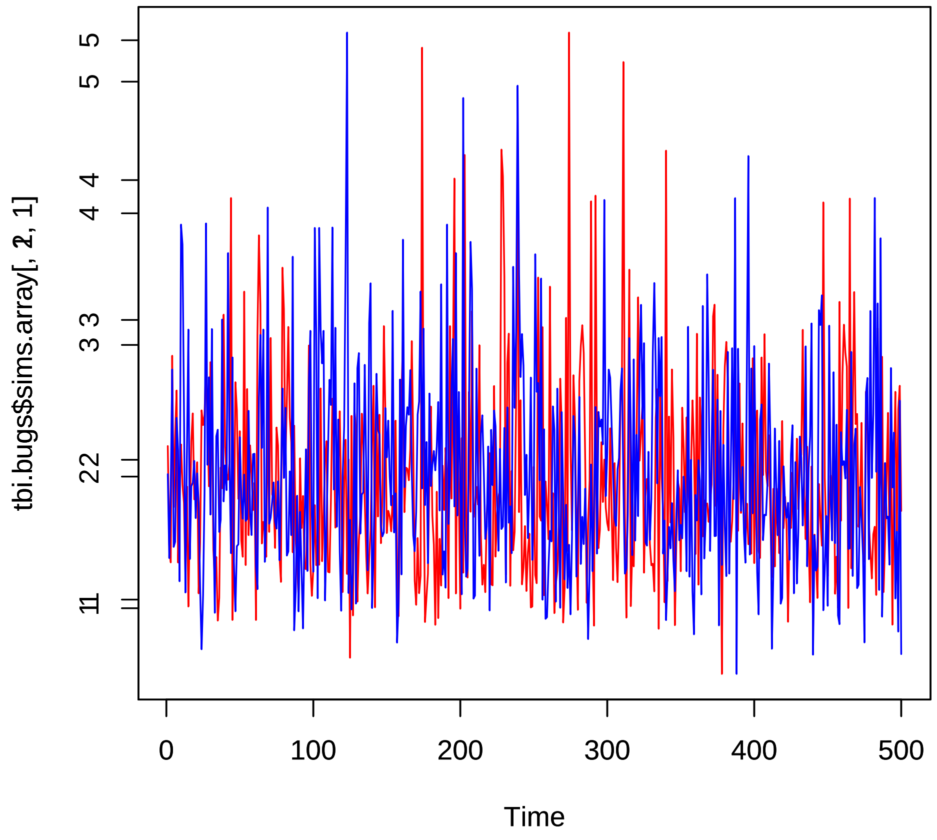
[12] I could also set an option called "codaPkg" to true. This would allow me to analyze the results using the *coda* package which we will discuss in the next section.

[13] In this case, because there are 172 observations for 5 different variables, I won't fill up the page with the voluminous output.

R2WinBUGS kept 500 observation in each of the 2 MCMC chains we requested. The number 856 refers to the 5 outputs we requested in our model, i.e. the relative risk estimate ("rr"), the residual ("r"), the unstructured heterogeneity ("v"), the spatially structured heterogeneity ("h"), and the exceedance probability ("prob2x"). [14]

We can plot the sims.array chains as time series and inspect them to see if they overlap and appear to converge to a stable distribution. To do this, we will have to work with (to me) the somewhat odd 3-dimensional, un-named structure of the list. Here, I plot the first for the first variable (indexed [,1,1]) and the overlay the second chain for this first variable.

```
> ts.plot(tbi.bugs$sims.array[,1,1], col="red")
> par(new=T)
> ts.plot(tbi.bugs$sims.array[,2,1], col="blue")
> par(new=F)
```



By looking at the structure of tbi.bugs, I can figure out that the first variable is the relative risk estimate for the first ZCTA. I find working with the indexes for sims.array off-putting and confusing. By contrast, the list object *sims.list*, contains named elements. [15]

```
> names(tbi.bugs$sims.list)
```

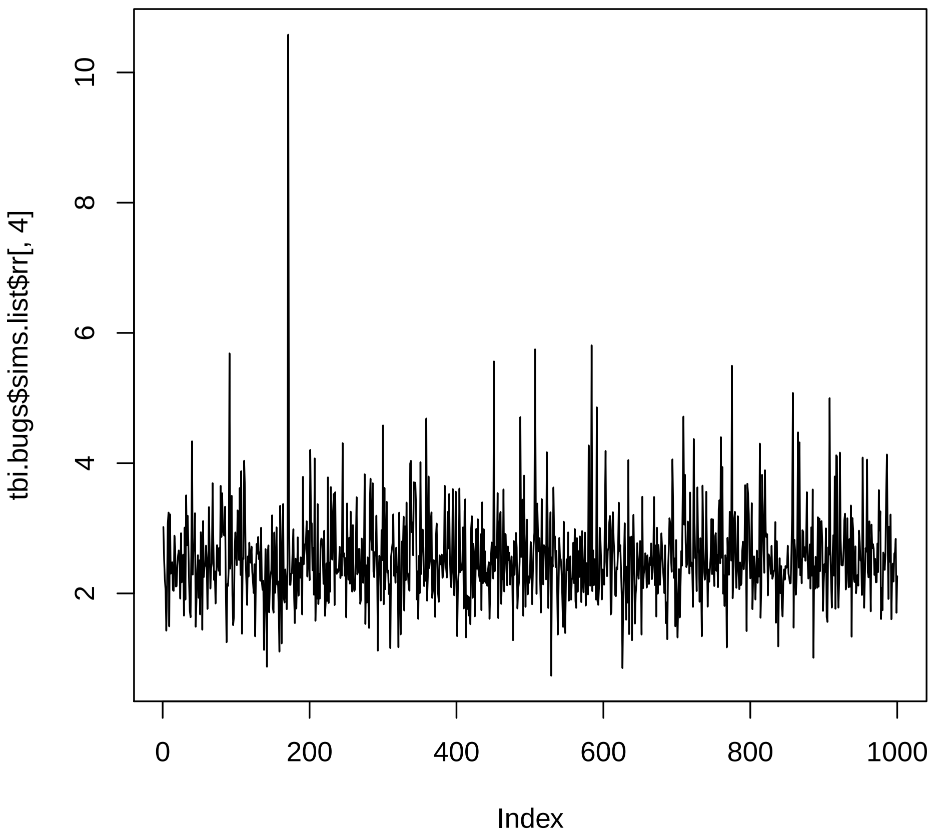
```
[1] "rr"       "r"        "v"        "h"        "prob2x"   "deviance"
```

[14] It doesn't add up to excactly 5x172 because 4 values were undefined in some way.

[15] It also differs from sims.array in that it combines the two chains rather than listing them separately.

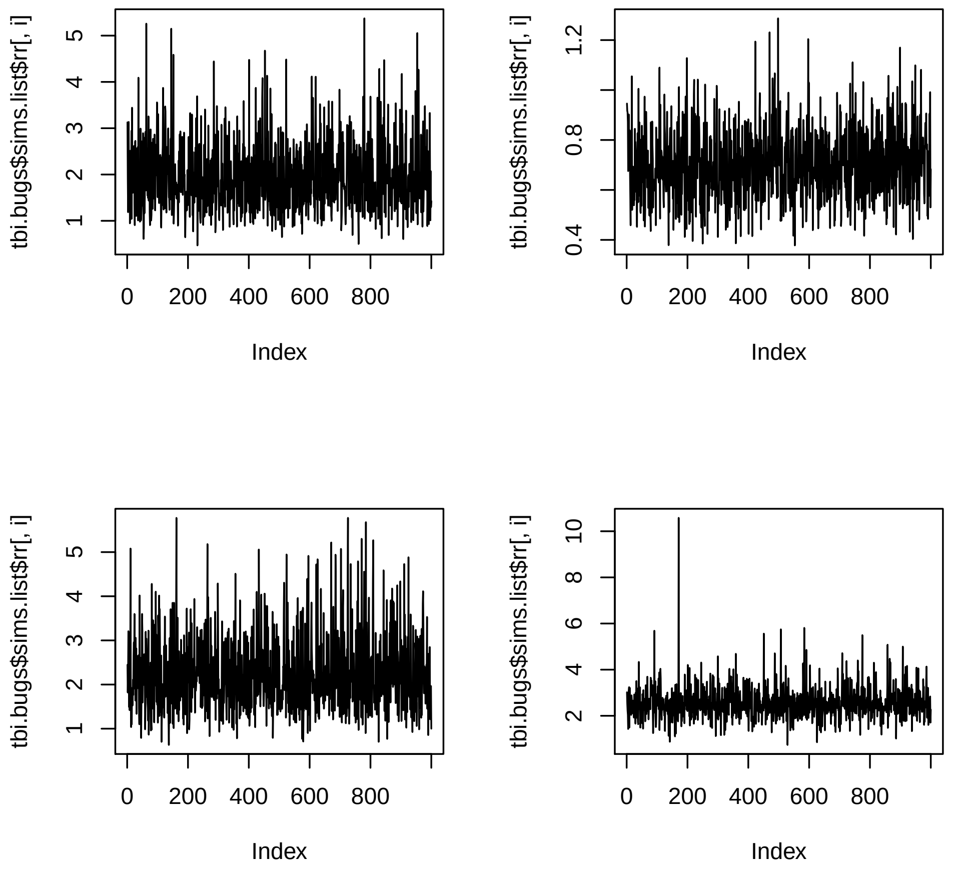
Now, if I want to inspect the plot for (say) the relative risk for the 4^th ZCTA, I can write:

```
> plot(tbi.bugs$sims.list$rr[,4], type="l")
```



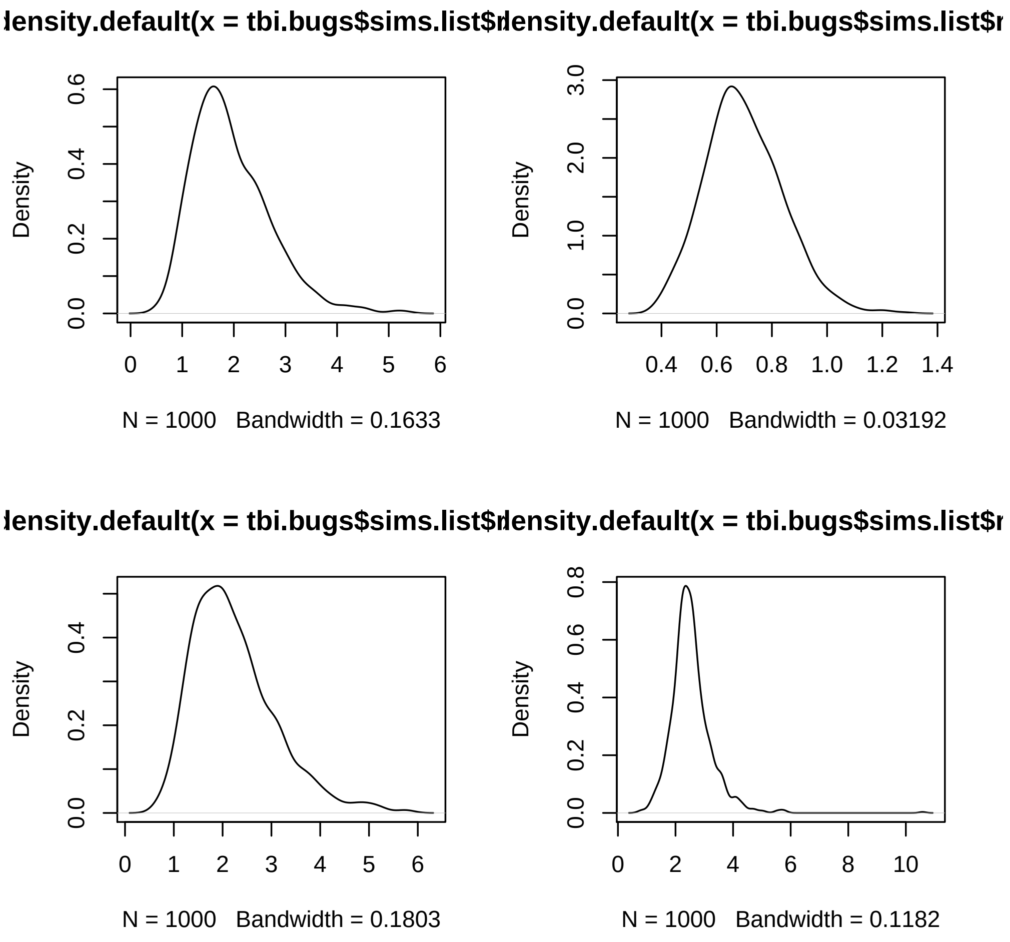
If I want to look at more than one plot at at time, I can loop over the index.

```
> par(mfrow=c(2,2))
> for(i in 1:4){
+         plot(tbi.bugs$sims.list$rr[,i], type="l")
+ }
```



We can see that the plots appear reasonable in that they overlap in an acceptably "spikey" fashion, don't seem to be "wandering off", and look like they've "settled" or converged to a stable pattern. As a next step, we could plot the densities of the chains, to see whether they appear to be normally distributed about a mean that we can use as a summary value for mapping.

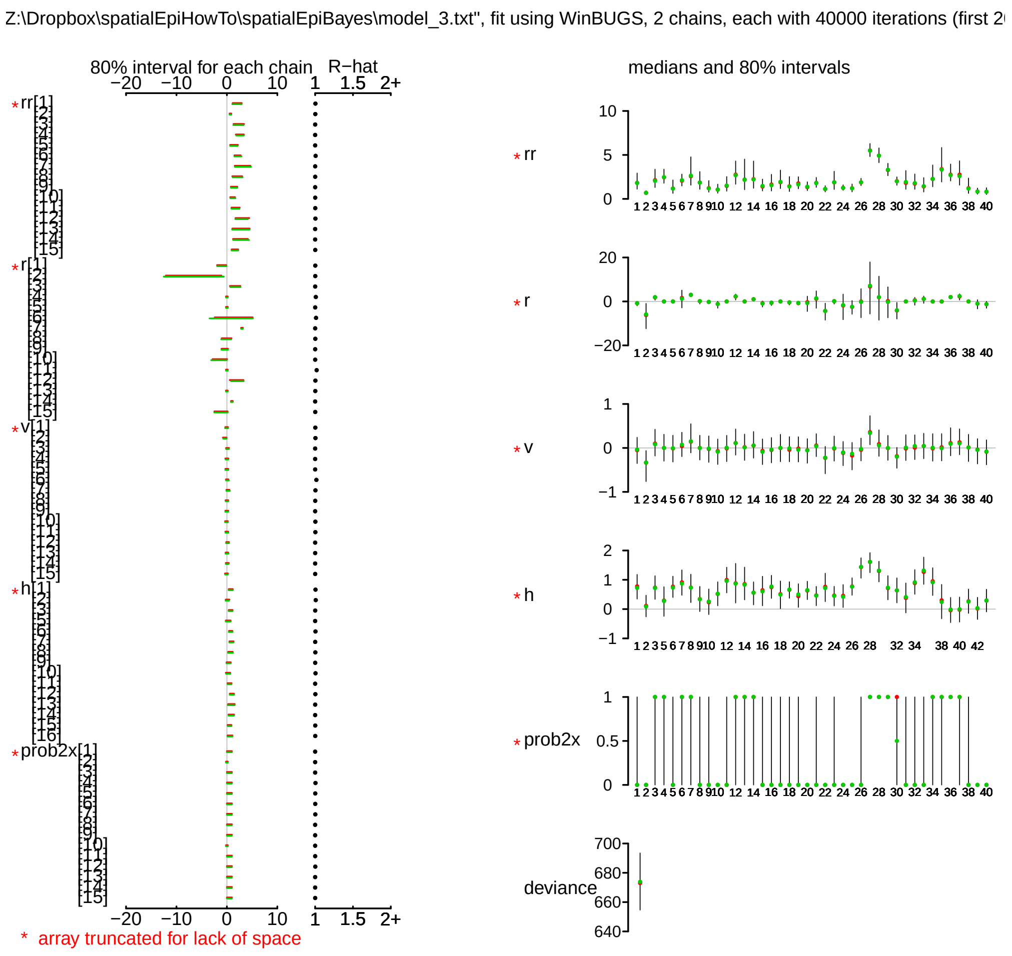
```
> par(mfrow=c(2,2))
> for(i in 1:4){
+         plot(density(tbi.bugs$sims.list$rr[,i]))
+ }
```



Again, the results appear reasonable.

With a couple of variables (which is most often the case) using basic R tools like ts.plot and plot is usually sufficient for a graphical inspection of the MCMC chains. But, again, when you have 5 variables for 172 geographic units, it can get a bit tedious. This is one reason I like to keep the WinBUGS window open and take a quick look at the (very nice) graphs that it automatically outputs. Fortunately, the bugs object created by R2WinBUGS has a very neat graphical summary method.

```
> plot(tbi.bugs)
```



We can see things, even in this truncated output, like the relative risk estimates ("shrunk" down considerably from their original SMR incarnations), the residuals (which appear suitably randomly distributed), and how many of the areas have more than twice the risk of TBI.

7.5.2 *Assessing BUGS Output with CODA*

While you could analyze the results of the BUGS run entirely with tools available in base R, there are a couple of packages specifically designed for this task. They are *CODA* (Convergence Diagnostics and Output Analysis) and *BOA* (Bayesian Output Analysis). They basically do the same things. We'll work with the coda package. [16]

The coda package requires a specially structured file called an *mcmc object*. Getting this file is a two-step process in R2WinBUGS. First, you run the simulation as you normally would, except you include the option *codaPkg=TRUE*. This creates a text file for each chain. Then you run the command *read.bugs()* on the resulting bugs object from that simulation to create the *mcmc object* that the coda package requires. Here is the syntax from my Mac under Parallels. Notice that I again save the results to an R data file that I can open on the Mac side. And, again, that this step is unnecessary if you live in MS Windows.

```
tbi.bugs2 <- bugs(
```

[16] In an odd departure from the syntax-driven spirit of R, both CODA and BOA are usually described as menu-driven. These menus are interactive and fairly self-explanatory. You can access them with the commands *codamenu()* and *boa.menu()* respectively.

```
  data = c(d, tbiWBweights),
  inits = list(inits1, inits2),
  parameters.to.save = c("rr", "r", "v", "h", "prob2x"),
  model.file = "Z:\\Dropbox\\spatialEpiHowTo\\spatialEpiBayes\\model_3.txt",
  n.chains = 2, n.iter = 40000, codaPkg=TRUE, debug=FALSE,
  bugs.directory = "C:\\WinBUGS14",
  )
  tbi.coda<-read.bugs(tbi.bugs2)
  save(tbi.coda, file="Z:\\Dropbox\\spatialEpiHowTo\\spatialEpiBayes\\tbicoda.RData")
```

Once back on my Mac side, I load the mcmc object I created on the Parallel's/Windows side as well as the *coda* package, and take a quick look at the structure of the tbi.coda object.

```
> load("/Users/charlie/Dropbox/spatialEpiHowTo/spatialEpiBayes/tbicoda.RData")
> library(coda)
> str(tbi.coda)
```

```
List of 2
 $ : mcmc [1:500, 1:856] 653 644 669 682 668 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:500] "501" "502" "503" "504" ...
  .. ..$ : chr [1:856] "deviance" "h[1]" "h[2]" "h[3]" ...
  ..- attr(*, "mcpar")= num [1:3] 501 1000 1
 $ : mcmc [1:500, 1:856] 683 650 674 664 669 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:500] "501" "502" "503" "504" ...
  .. ..$ : chr [1:856] "deviance" "h[1]" "h[2]" "h[3]" ...
  ..- attr(*, "mcpar")= num [1:3] 501 1000 1
 - attr(*, "class")= chr "mcmc.list"
```

The mcmc object is a list consisting of two chains, each chain has 500 x 856 dimensions corresponding to 500 replicates of 856 variables (recall there are 172 ZCTA's and each has a relative risk, residual, etc...). The commands in coda will return results for every one of those 856 variables. Though convenient for more manageable output, in this instance it quickly outstrips our ability to appreciate the results. For this demonstration, I'll extract just a small number of those variables. Know, though, that you should look at all the results.

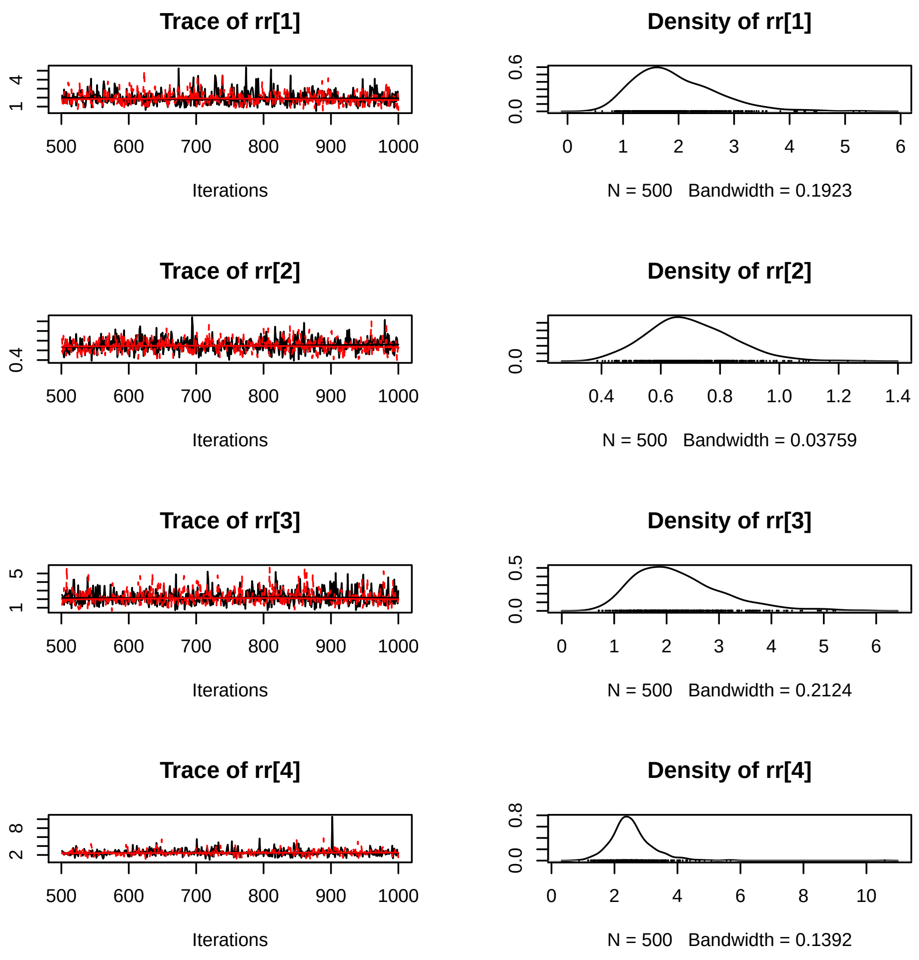
Although it is is a list, there is a direct method for subsetting mcmc objects ([i,j]), where *i* is the row to extract and *j* is the column. I used the *varnames()* function (results not shown) to figure out that columns 513 to 684 correspond to the two chains for the relative risk estimates.

```
> varnames(tbi.coda)
```

Let's see how the plot command works for just the relative risk estimates for the first 4 ZCTA's. [17]

```
> plot(tbi.coda[,513:516])
```

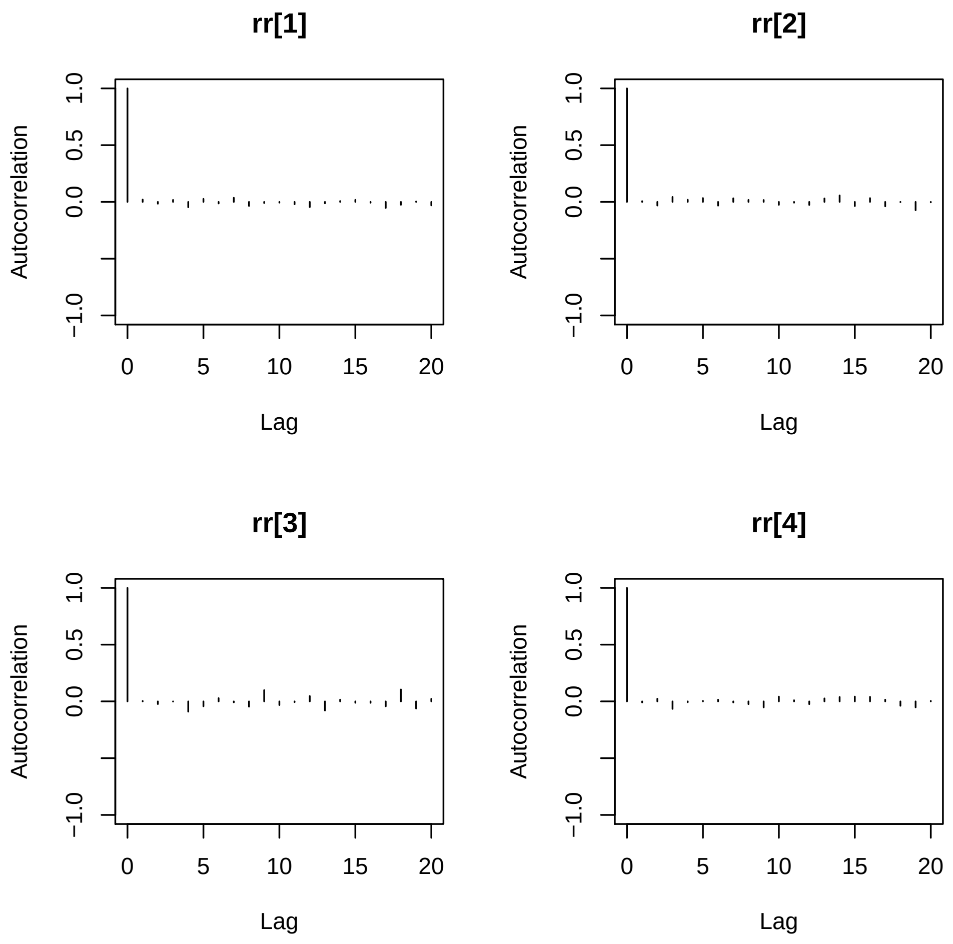
[17] To get all the results you would just use the name of the mcmc object without any subsetting index.



Compared to our efforts to use base R tools, the convenience of coda is readily apparent. Do note that the densities plotted on the right side of the figure represent the actual probability distribution of the relative risk estimates, and that conceptually this is quite different from frequentist statistics.

Another immediately nice feature of coda is that you can easily plot autocorrelations of the chains. Recall that by definition each step in an MCMC chain is based on the immediate previous step, so the simulations are correlated. These autocorrelations should drop off as the simulation traverses the probability space and settles into convergence. Again, there are tools in base R to plot autocorrelations (which actually coda is using), but again, coda makes it easier. Here we see that the autocorrelations drop off quickly and precipitously, indicating adequate convergence.

```
> autocorr.plot(tbi.coda[,513:516])
```



You may by now appreciate that convergence of an MCMC simulation is a critical issue in Bayesian analysis as we've been doing it. Graphical inspection of the is helpful, but there are 4 eponymous statistics available to help us better assess the algorithm. They are, respectively, the (1) Geweke , (2) Gelman-Rubin, (3) Raftery-Lewis and (4) Heidelberger-Welch diagnostic tests.

Geweke's test compares the means of two samples from a single chain with a Z statistic. The samples come from the beginning and end of the chain. CODA/BOA test the first 10% and last 50% of the chain. A $|Z| > 2$ is taken as evidence that the two means are not the same, and the simulation did not converge. The Gelman-Rubin statistic is based on an ANOVA-like approach to the means of two or more chains. A value close to 1 indicates convergence.

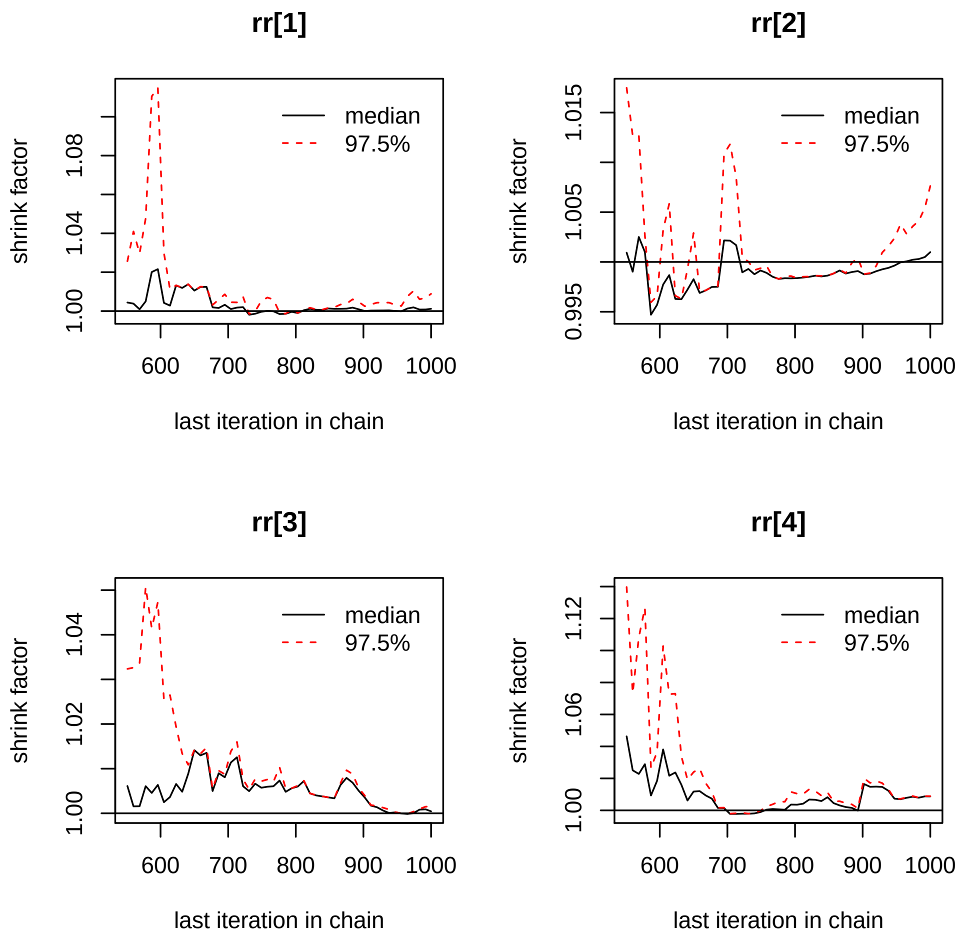
Raftery-Lewis looks at the quantiles of a single chain rather than means. the Heidelberger-Welch diagnostic also looks at a single chain, but takes a somewhat different approach, testing the chain for stationarity, dropping 10% of the observations if it fails, and continuing in this fashion until it (if) it passes.

I'll briefly demonstrate the Gelman-Rubin, but this document give a very nice overview of this important topic, and recommends that the data should pass all 4 tests, and that the failure of any one of them be considered an "alarm" that needs to be investigated. [18] Coda comes with both graphical and statistical version of the Gelman-Rubin diagnostic. We'll do both.

```
> gelman.plot(tbi.coda[,513:516])
> gelman.diag(tbi.coda[,513:516])
```

[18] You may also want to check out this page on mcmc diagnostics with coda in R

```
Potential scale reduction factors:

       Point est. Upper C.I.
rr[1]        1.00       1.01
rr[2]        1.00       1.01
rr[3]        1.00       1.00
rr[4]        1.01       1.01

Multivariate psrf

1
```



Finally, let's take a look at some of the statistical output of the coda results.

```
> summary(tbi.coda[,513:516])

Iterations = 501:1000
Thinning interval = 1
Number of chains = 2
Sample size per chain = 500

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:
        Mean     SD Naive SE Time-series SE
rr[1] 1.9492 0.7508 0.023742       0.027703
rr[2] 0.6995 0.1412 0.004465       0.004636
rr[3] 2.2375 0.8620 0.027260       0.025912
rr[4] 2.5442 0.7109 0.022480       0.021712

2. Quantiles for each variable:

        2.5%    25%    50%    75%  97.5%
rr[1] 0.8778 1.4147 1.8110 2.3825 3.6820
rr[2] 0.4492 0.5998 0.6876 0.7922 0.9919
rr[3] 0.9994 1.6065 2.0740 2.6755 4.3155
rr[4] 1.3749 2.1418 2.4660 2.8425 4.1337
```

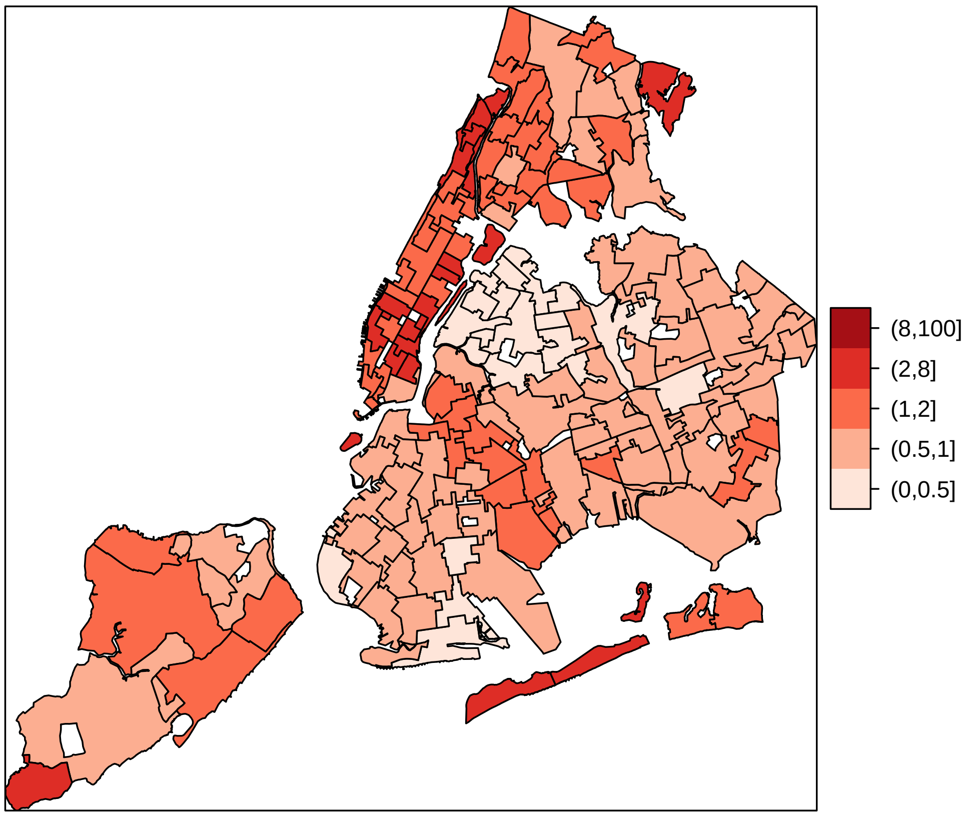
Here we see that the point estimate for the relative risk is 1.9 for ZCTA 1 and 0.7 for ZCTA 2. As for the 95% intervals we would report, again, because we are working with the probability distribution it is simply a matter of "chopping off" the two tails of the distribution at 2.5% and 97.5%.

7.6 Mapping the Output

It is only at this point, after having performed due diligence on the Bayesian output, that we turn to mapping our results. We will map 3 of the outputs, the (smoothed) relative risk estimate, the residuals, and the probability exceedance estimates. We first need to identify those variables in our bugs object. In our bugs object, these correspond to the variables tbi.bugs$mean$rr, tbi.bugs$mean$r, and tbi.bugs$mean$prob2x respectively. We add the categorized version of the variables to our map object, and map with *spplot()* using a set of *brewer.pal()* colors.

We begin with the risk estimates.

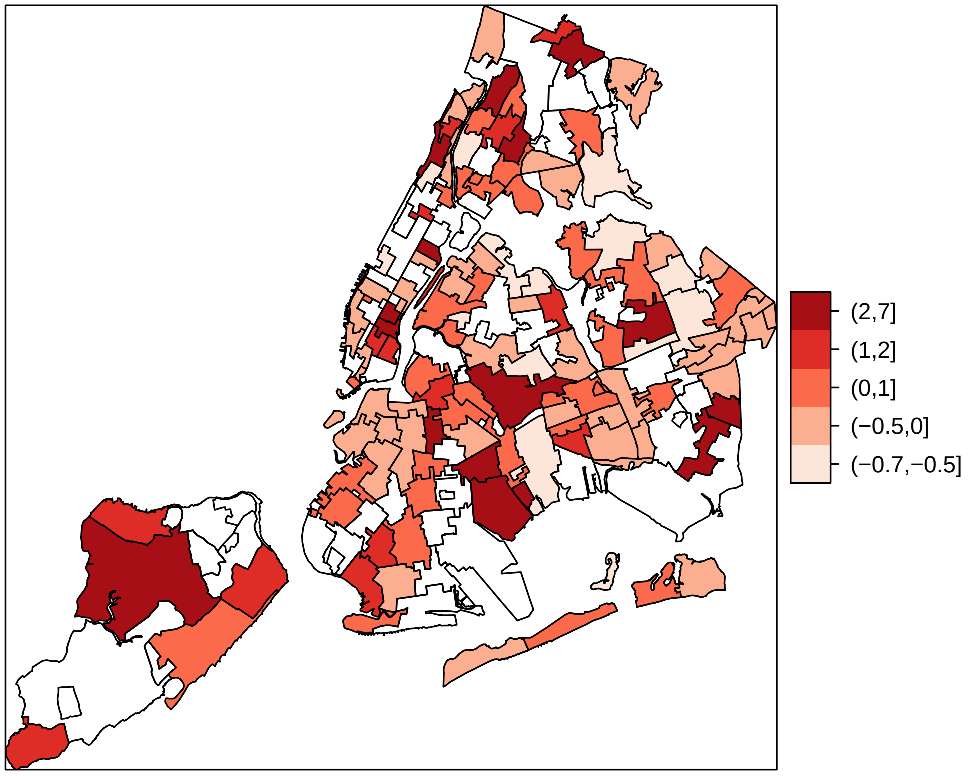
```
> nycZIPS4$rr.cat<-cut(tbi.bugs$mean$rr, breaks=c(0,.5, 1,2, 8,100))
> rr.plot1<-spplot(nycZIPS4, "rr.cat", col.regions = brewer.pal(5, "Reds"))
> print(rr.plot1)
```



A couple of points. First, because of the way we modeled the risk, we are now working on a risk scale rather than an SMR scale. The estimates control for gender and age through the original standardization process of creating the SMRS upon which the data are based. We can see that the estimates, *appear* less unstable with fewer extreme values. And, there appear to be areas of the city with similarly increased (or decreased) risk of pediatric TBI among children enrolled in Medicaid. This clustering of similar risk is consistent with our CAR model, which "borrows strength" and information from neighboring ZCTA's.

Next, we take a spatial look at our residuals.

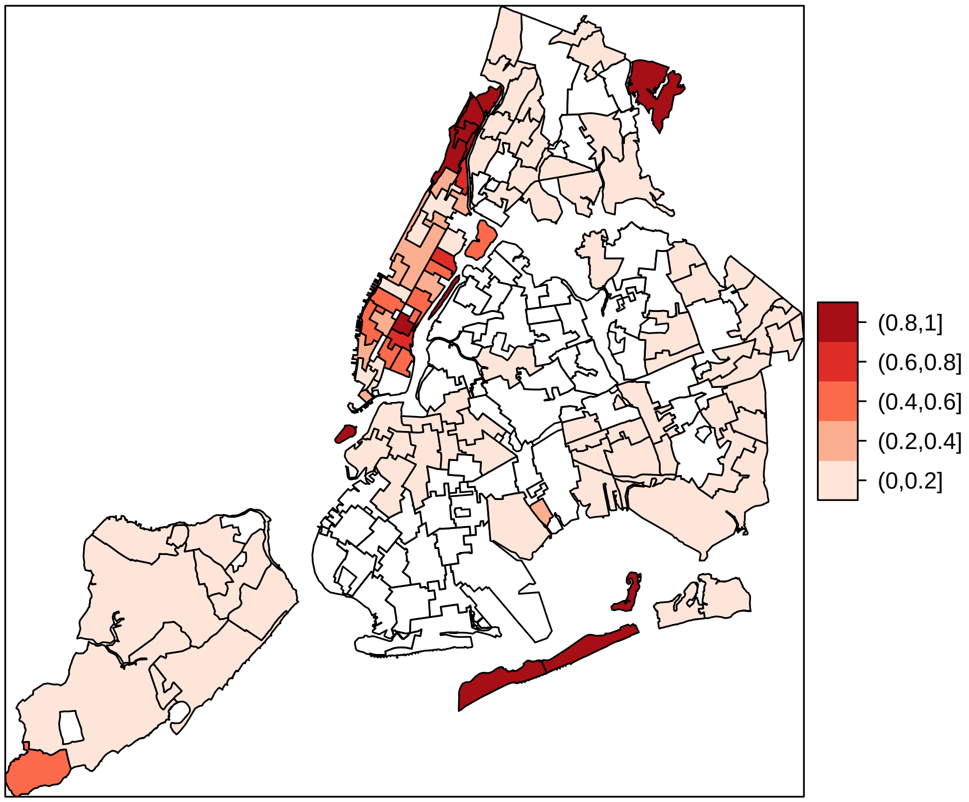
```
> nycZIPS4$r.cat<-cut(tbi.bugs$mean$r, breaks=c(-.7,-.5, 0, 1, 2, 7))
> r.plot1<-spplot(nycZIPS4, "r.cat", col.regions = brewer.pal(5, "Reds"))
> print(r.plot1)
```



As expected and desirable, the residuals appears spatially randomly distributed.

Finally, let's take a look at the probability exceedance map.

```
> nycZIPS4$prob2x.cat<-cut(tbi.bugs$mean$prob2x, breaks=c(0,.2, .4, .6, .8, 1))
> prob2x.plot1<-spplot(nycZIPS4, "prob2x.cat", col.regions = brewer.pal(5, "Reds"))
> print(prob2x.plot1)
```

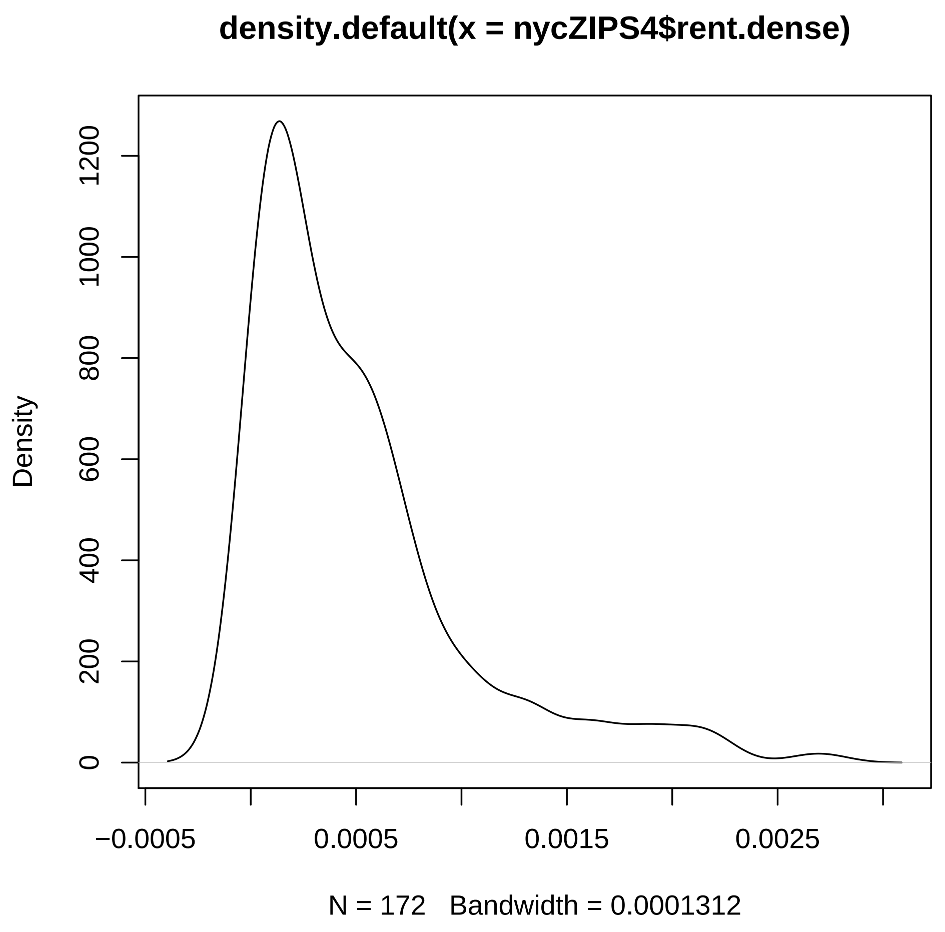


This approach brings into starker relief the clustering we may have appreciated on the map of relative risks. The values represent the probability of a ZCTA having a more than 2 times the risk of traumatic brain injury.

7.7 Regression Coefficients

Up to now, we've modeling and mapping age and gender-adjusted TBI risk estimates. We can, as a next step, look at the role of potential explanatory variables. We'll return to our consideration of rental housing in a ZCTA as a possible explanatory variable. We begin by re-creating and graphing a variable for rental density.

```
> nycZIPS4$rent.dense<-nycZIPS4$house.rent/nycZIPS4$area
> plot(density(nycZIPS4$rent.dense))
```



By just looking at the distribution, it appears this may be a problematic variable for MCMC simulation. To preemptively avoid some of the convergence issues described in the previous chapter about modeling New Orleans Hurricane Katrina data, I will transform this variable by centering and standardizing it.

```
> nycZIPS4$rent.dense2<-(nycZIPS4$rent.dense-mean(nycZIPS4$rent.dense))/sd(nycZIPS4$rent.d
> plot(density(nycZIPS4$rent.dense2))
```

There are a couple of steps to modeling rental density. First, we include it as a variable and coefficient ($b1 * rent.dense2[i]$)in a rewritten model statement. We will also have to add a statement to define a prior distribution on the coefficient and a hyperprior on the precision term for this prior.

```
model
{
for( i in 1 : m ) {
      y[i] ~ dpois(mu[i])
      mu[i] <- e[i] * rr[i]
      log(rr[i]) <- b0 + b1*rent.dense2[i] +v[i] + h[i]  # h spatial (structured) variance
                                        # v is unstructured heterogeneity
r[i]<-(y[i]-mu[i])                      # r is residual
prob2x[i]<-step(rr[i]-2)                # step function to calculate and plot
v[i]~dnorm(0,tau.v)      # normal prior on unstructured heterogenity
   }
b0~dnorm(0,tau.b0)   # normal prior on intercept term
b1~dnorm(0,tau.b1)            # normal prior on rental density term
h[1:m]~car.normal(adj[], weights[], num[], tau.h) #no need weights in model
tau.b0~dgamma(0.001,0.001)      # gamma prior on precision term
```

```
  tau.b1~dgamma(0.001,0.001)
  tau.v~dgamma(0.001,0.001)
      tau.h~dgamma(0.001,0.001)
  }
```

Then, we have to update our R2WinBUGS code by first defining the variables for the *bugs()* call to include the rent.dense2 variable and the initial values for the prior for the b1 coefficient.

```
> m<-nrow(nycZIPS4) # for loop number
> d<-c(list(y = nycZIPS4$observe, e = nycZIPS4$expect, rent.dense2=nycZIPS4$rent.dense2, m=
> inits1<-list(b0 = 1, b1 = 1, v = rep(0, m), h = rep(0, m), tau.v = 1, tau.h = 1, tau.b0 =
> inits2<-list(b0 = 10,  b1 = 10,v = rep(1, m), h = rep(1, m), tau.v = 0.1, tau.h = 0.1, ta

> save(tbinb, tbiWBweights, d, inits1, inits2, file = "/Users/charlie/Dropbox/spatialEpiHow
```

Then, to include b1 as a variable to monitor in the output.

```
  tbi.bugs3 <- bugs(
  data = c(d, tbiWBweights),
  inits = list(inits1, inits2),
  parameters.to.save = c("rr", "r", "v", "h", "prob2x", "b0", "b1"),
  model.file = "Z:\\Dropbox\\spatialEpiHowTo\\spatialEpiBayes\\model_rent.txt",
  n.chains = 2, n.iter = 40000, codaPkg=TRUE, debug=TRUE,
  bugs.directory = "C:\\WinBUGS14",
  )
  tbi.coda3<-read.bugs(tbi.bugs3)
  save(tbi.bugs3, tbi.coda3,
  file="Z:\\Dropbox\\spatialEpiHowTo\\spatialEpiBayes\\tbirent.RData")
```

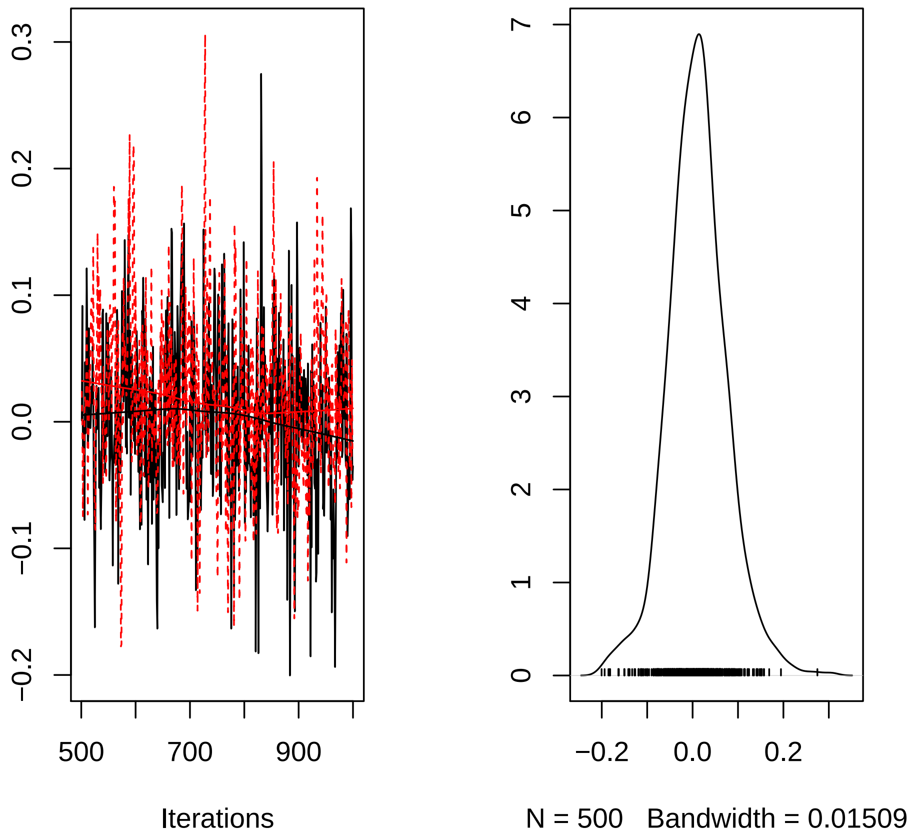
After running the model in R on a virtual Windows machine, I return to my Mac and load the results for analysis.

```
> load("/Users/charlie/Dropbox/spatialEpiHowTo/spatialEpiBayes/tbirent.RData")
```

We will, in the interests of time, restrict our attention to the b1 coefficient for the effect of rental density.

```
> library(coda)
> plot(tbi.coda3[,2])
> summary(tbi.coda3[,2])
```

```
Iterations = 501:1000
Thinning interval = 1
Number of chains = 2
Sample size per chain = 500

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

          Mean             SD       Naive SE Time-series SE
      0.010743       0.065270       0.002064       0.003354

2. Quantiles for each variable:

     2.5%       25%       50%       75%     97.5%
-0.123452 -0.028770  0.009142  0.047192  0.149402
```



We see here that, in contrast to our previous results, that the density of rental units in a ZCTA has little or no effect on the risk of TBI. This may be due to the effects of the CAR term in modeling and accounting for the spatial distribution of TBI risk.

Chapter 8
Additional Topics and Spatial Models in BUGS and JAGS

8.1 The Poisson-gamma Spatial Model (Again)

The material in this chapter is based entirely on a workshop held by Dr. Andrew Lawson at the Medical University of South Carolina in Charleston in March 2014. It presents additional information and explanations about the Poisson-gamma model we have been discussing in the previous two chapters, and introduces other models that might be of interest to epidemiologists. As opposed to previous chapters, which tend to be relatively self-contained and generally allow the code to be reproduced, I here present the code without data and refer you to Dr. Lawson's textbook for further study. [1]

As an overview, the standard Besag-York-Mollie[2] (BYM) spatial analytic model is again:

$$
\begin{aligned}
y_i &\sim Pois(\lambda_i = e_i\theta_i)\\
log(\theta_i) &= \beta_n + \upsilon_i + \nu_i\\
\upsilon &\sim nl(0, \tau_\upsilon)\\
\nu &\sim nl(\bar{\nu_\delta}, \tau_\nu / n_\delta)
\end{aligned}
$$

where,

(1) the y_i counts in area i, are independently identically Poisson distributed and have an expectation in area i of e_i , the expected count, times θ_i , the risk for area i :

$$y_i, iid \sim Pois(e_i\theta_i)$$

(2) a logarithmic transformation ($log(\lambda_i)$) allows a linear, additive model of regression terms (β_n), along with

(3) a spatially *unstructured* random effects component (υ_i) that is i.i.d normally distributed with mean zero ($\sim nl(0, \tau_\nu)$), and

(4) a conditional autoregressive spatially structured component ($\nu \sim nl(\bar{\nu_\delta}, \tau_\nu / n_\delta)$) In the usual formulation, each "neghborhood" consists of adjacent spatial shapes that share a common border. The mean θ_j in neighborhood j is

[1] Or better yet, attend one of his highly recommended workshops...

[2] J. Besag, J. York, A. Mollie (1991). Bayesian image restoration, with two applications in spatial statistics (with discussion). Annals of the Institute of Statistical Mathematics 43(1), 1-59

normally distributed with its parameters defined as μ_j the average of the μ_{ij}'s in the neighborhood and σ_j equal to the σ's of the neighborhood μ_{ij}'s divided by the number (δ_j) of spatial shapes in the neighborhood. [3]

The spatially *unstructured* random effects component (υ_i) is a random effects term that captures normally-distributed or Gaussian random variation around the mean or intercept α_θ. This unstructured heterogeneity represents, essentially, âĂĬnoiseâĂİ that arises from the data that we canâĂŹt capture with our covariates. It is important, and it contributes to the risk estimate, but it is essentially random and varies normally around zero. A model that consists solely of a random effects term is sometimes referred to as a "frailty" or susceptibility model, because it can be thought of as describing the individual (or area) level susceptibility or frailty to disease given some overall or global susceptibility. It doesn't take covariates, space or geography into account. The addition of a spatially-structured term, ν to a random effect model is sometimes referred to as a convolution model.

A useful and commonly used prior distribution for θ in the setting of spatial analyses is the gamma (Γ) distribution:

$$\theta \sim \Gamma(\alpha, \beta)$$

8.1.1 A Bit More About Poisson-gamma Spatial Models

As we have seen, Bayesian spatial analyses often use a poisson-gamma formulation. One might reasonably ask, *why not just gamma-gamma?* There is a problematic association between the mean and variability in a gamma-gamma model. Recall,

$\theta \sim \Gamma(a,b)$, $\mu = a/b, Vvar = a/b^2$

To model estimates we need to set the b parameter to a constant C, so that $\mu = C \cdot a/C$, and $var = \mu/C^2$. But, this links the variance to the mean, and variability depending on mean (i.e. heteroskedasticity) is a bad model. INLA doesn't even include it as an option.

On a related note, one might ask *why not a negative binomial?* Negative binomials models include an extra parameter to account for overdispersion and tend to be popular in epidemiologic research. Once does not commonly see them in Bayesian spatial models because the unstructured heterogeneity term υ in the basic frailty or susceptibility model should account for any overdispersion. You can, though, fit a negative binomial model (and it is available as a likelihood in INLA) if you are interested in characterizing the over dispersion.

Poisson models used in epidemiologic analyses often include a population **offset variable**, and the exponentiated coefficients for risk factors are interpreted as incidence density ratios. Poisson spatial models similarly often include an offset variable, though the motivation is somewhat different. In spatial Poisson models, a simple population offset is perhaps the most basic approach. Often, one will see a offset variables that "bake in" important characteristics like gender and age as part of a standardized incidence ratio (SIR). Those characteristics will be "held constant" or *fixed* in the model. It can be considered a modulating effect on the model, rather than assessments of possible associations.

Generally, Poisson models are used when there is no finite or logical restriction on the number of trials, such as rare outcomes in large populations. If there are a limited number of trials a binomial model is indicated, and if the outcome is binary a Bernoulli or logistic model is appropriate. If you are working with binary data at the regional level e.g. a spatial lattice of binary outcomes, (the binary outcome is at the regional level, i.e. region has the outcome or not), a *pseudo-likelihood for binary data*is a very good approximation. It is sometimes referred to as an {auto-logistic model, and is essentially a logistic regression conditioned on the sum of the outcome in the neighbors. It can be a better fit in some cases than CAR models.

[3] The spatially structured component CAR is sometimes described as a Gaussian process $\lambda \sim Nl(W, \tau_\lambda)$ where W represents the matrix of neighbors that defines the neighborhood structure, and the conditional distribution of each λ_i , given all the other λ_i is normal with μ = the average λ of it's neighbors and a precision (τ_λ). See Banerjee, 2004

One could capture *geographic trend* in spatial data, for example increasing rates of disease along a geographic axis like more malaria the further south you go, by including covariates for x and y coordinates of centroids of spatial areas, e.g. $log(\theta_i) = \alpha_i + \beta_1 * x_{coord} + \beta_2 * y_{coord} + v_i + u_i$. Also of note, in a convolution model the υ and v terms are only weakly identified, and not entirely comparable. This has implications for the interpretation of calculations for the proportion of variance accounted for by v, which cannot, strictly speaking be interpreted as a variance partition coefficient.

The concept of **conditional independence** is important to Bayesian hierarchical spatial modeling. In defining the hierarchies, you can include the complex correlations and non-independent relationships that characterize spatial data, and still have a relatively simple model at the data level. If you specify the components of the hierarchy well, then the data y_i are essentially an independent manifestation of a random process. Programs likes BUGS return the *marginal* posterior distribution. The entire posterior remains high dimensional and correlated.

8.1.2 A Note About Partitioning Heterogeneity

This business of partitioning variance or heterogeneity into unstructured and structured components can be a bit tricky. Recall simple model $y = a + bx + e$, where e is an error or residual term. We are partitioning out parts of that error. Unstructured heterogeneity υ represents, "noise" that arises from the data that we don't capture with our covariates. It is important, *and it contributes to the risk estimate*, but it is essentially random and varies around zero. We also assume that there is some portion of that overall heterogeneity accounted for by place or geography and *also contributes to risk*, and can be captured by the spatial conditional autoregression v term. This is why both υ and v are included at part of the linear additive model as contributing to overall risk. When we map the CAR term, we are essentially mapping spatial risk, which can be interpreted as spatial clustering or similarity of that part of risk. The remaining error or residual term is interpreted as is usually is, and is not included in model statement.

It is important to note that because it represents risk, *the CAR term can account for or attenuate some of the effect of covariates*. You can expect some effect on the standard errors for the coefficients, but the point estimates themselves may be affected. If inference on the covariates is of primary importance, a recommended approach is to compare covariate models with and without the structured spatial term on DIC. If there is no improvement by including the CAR term, leave it out. If the model with CAR is a better fit keep it in, but interpret the covariate coefficients cautiously.

8.1.3 Example Model: Nested Aggregation Oral Cancer

As an example of running a Poisson Bayesian hierarchical model of nested data in BUGS or JAGS, consider an example of oral cancer counts in the 159 counties in 18 public health districts of the US state of Georgia. The public health districts "inherit" characteristics of the counties that make them up, which are known as *contextual effects*. Effects at county level *scale up* to the districts they make up. So, for example, 5 counties with 5 cases each equal 25 cases at the district level. The districts could be analyzed on their own, but it often informative to include the county effects. The following data represent observed and state standardized expected counts of oral cancer.

```
(PH_district_model_georgia.odc)
# data

list(nsumph
= 88, num = c(3, 4, 5, 4, 6, 7, 4, 8, 6, 8,
5, 6, 5, 5, 4, 2, 2, 4
),
adj = c(
10, 5, 2,
6, 5, 4, 1,
13, 12, 9, 8, 4,
8, 6, 3, 2,
10, 9, 8, 6, 2, 1,
10, 9, 8, 7, 5, 4, 2,
10, 9, 8, 6,
12, 10, 9, 7, 6, 5, 4, 3,
10, 8, 7, 6, 5, 3,
14, 12, 9, 8, 7, 6, 5, 1,
18, 15, 14, 13, 12,
14, 13, 11, 10, 8, 3,
18, 17, 12, 11, 3,
16, 15, 12, 11, 10,
18, 16, 14, 11,
15, 14,
18, 13,
17, 15, 13, 11
),eph=c(26.7313582387354,17.8569786812963,19.000833240556,24.8322642703346,
35.6928573604747,38.1880967220473,12.4232592469938,40.2840771546551,
31.6839976247491,33.1291602142528,6.65287755667638,23.232040109711,
20.3645029329909,16.4453885396069,10.949025543461,16.922717476408,
15.6506686820585,23.9598964049923),
yph=c(24,17,13,22,24,38,8,38,24,48,5,27,30,17,8,29,18,24) )


# model (y=data, mu=mu, th=theta)

model{
for( i in 1:18){
yph[i]~dpois(muph[i])
log(muph[i])<-log(eph[i])+log(thph[i])
thph[i]<-exp(aph0+uph[i]+vph[i]) # exponentiate to get relative risk
vph[i]~dnorm(0,tau.vph)
}

for (k in 1 :nsumph){weiph[k]<-1}
uph[1:18]~car.normal(adj[],weiph[],num[],tauph)
tau.vph<-1/pow(sdvph,2)
tauph<-1/pow(sdph,2)
aph0~dflat()
sdvph~dunif(0,100)
sdph~dunif(0,100)
}

)

#inits
list( uph=c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),vph=c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),sdph=7.5,
aph0=-1.5
)
```

8.2 Spatial Survival Models

8.3 Spatial Survival Models

Classically, survival analysis involves the study of some endpoint that has a duration associated with it, e.g. the time of diagnosis to the time of death using cancer registry data. In the spatial setting, there are contextual effects that a person "inherits" from the region in which he or she resides.

As in any survival analysis, the random variable is t_i, the time to the outcome event for the i^{th} individual, and $f(t)$ is a probability distribution of those times. Choices for $f(t)$ include Weibull, exponential, Gamma, and log normal, though one most often sees Weibull models in spatial survival analyses.

The Weibull distribution has 2 parameters ($\sim (\nu, \lambda)$) and is flexible enough to resemble most any form. It can be used for proportional hazards modeling, though it can be a little odd in that while it is easy to calculate the median, it is somewhat difficult to calculate the mean.

A Weibull model often assumes λ is a linear function of covariates ($\lambda = e^{z'\beta}$),and can be specified rather simply in WinBUGS with a λ for each individual,

```
t[i]~dweib(nu,lambda[i])
lambda[i]<-exp(beta0+v[i])
```

where the median survival time is calculated as log of 2 ($[0.693e^{z'\beta}]^{1/\nu}$),

```
median[i] <- pow((log(2) /lambda[i]), 1/nu)
```

and the survival function as 1 minus the Failure function ($1 - f(t)$) which is equal to $e^{-\lambda_i t_i^{\nu}}$

```
log(surv[i])<--lambda[i]*pow(t[i],nu)
```

```
# example spatial survival model code
model{
for(i in 1: 10){
t[i]~dweib(nu,lambda[i])
lambda[i]<-exp(beta0+v[i])
v[i]~dnorm(0,tauv)
median[i] <- pow((log(2) /lambda[i]), 1/nu)
log(surv[i])<--lambda[i]*pow(t[i],nu)
}
beta0~dnorm(0,0.0001)
nu~dgamma(1.0,0.0001)
tauv<-pow(sdv,-2)
sdv~dunif(0,10)
}
```

8.3.1 Accounting for Censoring

Survival data are usually censored. This is most commonly right-censoring. where participants drop out for reasons unrelated to the outcome of interest, or the study ends. We don't know exact time of the event outcome for some individuals, only that it happens in some time "window". It is necessary to identify censored observations in the data in some way.

In WinBUGS, one approach is to create separate vectors of censored and uncensored outcome times. For the uncensored vector of outcome times, indicate censored times with NA. For the censored vector of times, substitute zero for the uncensored observations, e.g.

```
t=c(23, 12, 18, NA, NA, 38, 29, 30, NA...)
tcen=c(c(0,0,0,40, 40, 0,0,0,40, ...))
```

This works because NA represents missing data, and WinBUGS automatically imputes missing data from the predictive distribution. This is an ideal way to deal with censored times. *But*, because WinBUGS will impute values *outside the appropriate censoring time*, we have to truncate the imputed times. In WinBUGS, the *indictor function: I()* is used to truncate a variable. The following code will return imputed censored values so they have to be greater than the vector of truncated values.

```
t[i]~dweib(nu,lambda[i])I(tcen[i],)
```

Anything that is imputed in WinBUGS, needs to be initialized, but we only need inits for the missing values, so we substitute NA for the *uncensored* observation.[4]

```
#inits
list(t=c(NA, NA, 7.0 7.0, NA, NA, NA, ... ))
```

```
# example weibull with censoring
model{
for(i in 1: 10){
t[i]~dweib(nu,lambda[i])I(tcen[i],)
lambda[i]<-exp(beta0+v[i])
v[i]~dnorm(0,tauv)
median[i] <- pow((log(2) /lambda[i]), 1/nu)
log(surv[i])<--lambda[i]*pow(t[i],nu)
}
beta0~dnorm(0,0.0001)
nu~dgamma(1.0,0.0001)
tauv<-pow(sdv,-2)
sdv~dunif(0,10)
}
```

8.3.2 Adding a Spatial Component to Survival Models

To incorporate location information into a survival model we will need to use nested indexing, by assigning a label for a contextual unit to an individual:

```
W[label[i]]

lambda[i]<-exp(beta0+v[i]+W[label[i]])
```

where W is a spatial random effect

```
# example weibull full model

model{
for(i in 1: 100){
t[i]~dweib(nu,lambda[i])I(tcen[i],)
lambda[i]<-exp(beta0+v[i]+W[label[i]]) # label vector assigns county to individual, links the two
v[i]~dnorm(0,tauv) # individual level random
median[i] <- pow((log(2) /lambda[i]), 1/nu)
log(surv[i])<--lambda[i]*pow(t[i],nu)
}
beta0~dnorm(0,0.0001)
nu~dgamma(1.0,0.0001)
tauv<-pow(sdv,-2)
sdv~dunif(0,10)
W[1:159]~car.normal(adj[],weights[],num[],tauU) # county level
for(k in 1:sumNumNeigh)
{weights[k]<-1}
tauU<-1/(sigU*sigU)
sigU~dunif(0,10)
}
```

The individual-level data now informs about the county much more than simple counts would.

[4] As you can imagine, this process can become *very* tedious for large data sets...

8.4 Point-Process Models in BUGS and JAGS

Point-process or case-event data (as opposed to aggregated or lattice data) are informative for putative sources of health outcomes at small spatial scales and for cluster alarms. But, they can be more difficult to obtain because of confidentiality concerns, and can be analytically challenging, starting with the choice of an appropriate study window, i.e. the area to be mapped and analyzed. The *a posteriori* choice of an area to study can result in the so-called *Texas sharpshooter problem*, where you select an areas because of a pre-conceived notion of a problem and (perhaps more importantly) *don't* looking at other areas.

Data consist of *m* events represented as s_i points of x and y coordinates. You can never really get an *expected* count at the same level as the cases or outcomes of interest. One suggestion is to select a *control* disease[5] for which you can get a spatial distribution at the same scale as the outcome, but that is not affected by the exposure under study. The assumption is the control disease acts like an expected rate, but this approach in general, and the choice of control disease in particular, can be quite controversial. The *n* control events are added to the vector of cases for a single vector of cases and controls.

Point process models anayze the location of the events directly with integrations, assuming an intensity function that has two components: a background effect and a risk effect. These integrations are not available in standard packages, though there are approximations (e.g. Berman-Turner) that can be useful. Conditional logistic models, also known as *logistic spatial models* are an alternative to integrations for point process data. The approach involves setting up a vector of 1's for cases and 0's for controls, then asking the question: What is the probability that *y* is a 1 or a 0? It is essentially a logistic regression, and specified much the same way. By condition on the labels one avoids the problematic aspects of spatial location. Models can include covariates like distance from putative sources as well as individual-level variables.

Abasic model could be: $e^{\beta_0+\beta_1*distance_i+\beta_2*age_i}$ If distance important, one would expect β_1 to be negative (and β_2 to be positive if age-related...). This is an individual-level model, so a random effects term would be individual, which is simply a random-intercept multi-level model. We can't apply a simple conditional autoregression (CAR) term because there are no regions, and hence no neighbors, but there is a way to set up a kind of CAR model by choosing "natural neighbors" via either tessellation or Delauney triangulation.[6]

Tessellation or tilings creates areas or "tiles" around points, so that for any point, you are closer to that point than to any other point. This is also known as Dirichlet or Voronoi tessellation. It can be accomplished in with most any set of points using the R package *DELDIR*. Delauny triangulation connects points to form a "jewel" of the tessellation. Tessellations must be "suspended" with fake points at the edges where there are no actual points, but Delauny triangulations are self-contained. There can be some difference in the number of neighbors depending on which approach you take

8.4.1 Example Model: Laryngeal Cancer in NW England

Laryngeal Cancer near incinerators in NW England in 1970's was modeled by Diggle using lung cancer as a control disease.[7]

```
#logistic case-control model
# x,y and ind as inputs.
# dis is distance
#distance only model with RE and W CAR
model
{
```

[5] A *geographical control*, not a matched case-control as is commonly used in epidemiology.

[6] One could specify a full multivariate normal model because we have the distances between people, but would result in a problematically large matrix similar to that seen in integration approaches.

[7] As one might imagine, this choice of a control outcome was controversial

```
for (i in 1:N){
ind[i]~dbern(p[i])
f[i]<-(1+exp(-gam1*dis[i]))*exp(gam0+gam2*age[i]+v[i]+W[i])
        logit(p[i])<-log(f[i])

v[i]~dnorm(0,tauv)
res[i]<-(ind[i]-p[i])/sqrt(p[i]*(1-p[i]))
x1[i]<-x[i]
y1[i]<-y[i]
}
 for(k in 1:sumNumNeigh){wei[k]<-1}
            W[1:N] ~ car.normal(adj[],wei[],num[],tauW)
tauW<-pow(sdW,-2)
sdW~dunif(0,10)
gam0~dnorm(0,0.001)
gam1~dnorm(0,0.001)
gam2~dnorm(0,0.001)
tauv<-1/pow(sdv,2)
sdv~dunif(0,100)
}
```

A couple of observations about this model. The *car.normal* specification is based on Delauny triangulation. The *gam1* coefficient for distance is taken out of the main linear term and has a 1 added to it because at distances of zero it blows up the model. Also note, it is a negative coefficient, so a *positive* coefficient indicates an effect... The *v[i]* term is the uncorrelated or "frailty" term. [8]

8.5 Zero-Inflated Regression Models

Some data contain large numbers of zero counts. This could be structural, e.g. areas with no probability of outcome (no one lives there) or it be could due to a low probability of disease on those areas. For example, if one looked at a histogram of sudden infant death syndrome (SIDS) counts in Georgia 1994 to 2005, in any given year the distribution is extremely skewed toward zero, with some years demonstrating multiple modes (a sea of zeros and isolated clusters of counts typical of sparse data). These kinds of data are hard to fit with conventional distributions. They are highly over dispersed (e.g. mean down around 1, variance greater than 3) with multiple modes. A Bayesian hierarchical specification is sufficient if you add terms to capture clustering. But, if you want to directly address the issue of zeros *zero-inlfated* Poisson, binomial and negative binomial (ZIP, ZIB, ZINB) models may be appropriate. regression often do zero inflation, e.g. **zero-inflated poisson**(/ binomial / negative-binomial) (ZIP, ZIB, ZINB)

The basic approach is to first model the probability that the data, y_i, is zero with probability (p_i). Then, if it is not zero (probability $1 - p_i$), model as $Pois(e_i\theta_i)$. You can place a non-informative $Beta(1,1)$ (uniform...) prior on p_i, or model p_i as a logistic regression with set of linear predictors, after which you model θ. Another option is a sparse Poisson convolution model (SPC), which applies a *factor*, the number 2 if $y_i > 0$ and the number 1 otherwise, which scales the observations with positive counts to make them bigger than the zero counts. In the following example code of an SPC model of Georgia asthma data, *a2[z1[i]]* is the factored effect. It is also an example of nested indexing.

```
model{
for (i in 1:N){
z1[i]<-z[i]+1
asthmaD[i]~dpois(mu[i])
mu[i]<-expect[i]*theta[i]
log(theta[i])<-a0+a1*pop[i]/10000+a2[z1[i]]+v1[i]+u[i]
v1[i]~dnorm(0,tauv1)
log(L[i])<-asthmaD[i]*log(mu[i])-mu[i]-logfact(asthmaD[i])
CPinv[i]<-exp(-log(L[i]))
```

[8] As an aside, it occurs to me that since these are binary data, one could run this as an autologistic model with pseudolikelihood (see elsewhere in these note...)

```
#cpo[i]<-pow(CPinv[i],-1)
#Lcpo[i]<-log(cpo[i])
}
#MargL<-sum(Lcpo[])
u[1:N]~car.normal(adj[],weights[],num[],tauU)
for(k in 1:sumNumNeigh)
{weights[k]<-1}
tauU<-1/(sigU*sigU)
sigU~dunif(0,10)
tauv1<-pow(sigV1,-2)
sigV1~dunif(0.0,10)
tau2<-pow(sig2,-2)
sig2~dunif(0.0,10)
a0~dflat()
a1~dnorm(0,0.001)
a2[1]~dnorm(0,tau2)
a2[2]~dnorm(0,tau2)
}
```

A ZIP model is more complex, because it is essentially a mixture model or two different models. We need to write our own likelihood, and need to use the "zeros trick" to get WinBUGS to use that likelihood.

```
  zeros[i]~dpois(gamma[i])
```

p_i is given a beta(0.5,0.5) prior, which is called a Jeffries prior, that is almost, but not quite uniform (it bends up at the very end). We can calculate a DIC because it is a mixture model, so instead use a CPO (recall close to 1 is good fit, close to 0 is poor fit). Because CPO is calculated for each area, we can map them, with lighter (lower value) areas have worse fits.

```
model{
C <- 10000
for(i in 1:N){
zeros[i] <- 0
gamma[i] <- -log(L[i]) + C
mu[i]<-expect[i]*theta[i]
prat[i]<-p[i]/(1-p[i])
log(L[i])<-z[i]*log(prat[i]+exp(-mu[i]))+
(1-z[i])*(asthmaD[i]*log(mu[i])-mu[i])-log(1+prat[i])
-(1-z[i])*logfact(asthmaD[i])
zeros[i]~dpois(gamma[i])
log(theta[i])<-a0+a1*pop[i]/10000+v1[i]+u[i]
v1[i]~dnorm(0,tauv1)
p[i]~dbeta(1,1)
CPinv[i]<-exp(-log(L[i]))
cpo[i]<-pow(CPinv[i],-1)
Lcpo[i]<-log(cpo[i])
}
MargL<-sum(Lcpo[])

u[1:N]~car.normal(adj[],weights[],num[],tauU)
for(k in 1:sumNumNeigh)
{weights[k]<-1}
tauU<-1/(sigU*sigU)
sigU~dunif(0,10)
tauv1<-1/(sigV1*sigV1)
sigV1~dunif(0.005,10)

a0~dflat()
a1~dnorm(0,0.001)
}
```

Generally SPC models don't work as well as ZIP models, but are worth a try. Several ZIP models are also available in INLA.

```
http://www.r-inla.org/models/likelihoods
```

Chapter 9
Brief Introduction to Spatiotemporal Modeling in R with INLA

This brief introduction and overview of spatiotemporal modeling using Integrated Nested Laplace Approximations (INLA) is intended to supplement the material on creating the data sets and conducting the analyses for the paper "Small-Area Spatiotemporal Analysis of Pedestrian and Bicyclist Injuries in New York City 2001-2010". It is based primarily on textbooks and workshops by Andrew Lawson of the Medical University of South Carolina, articles and analyses by Marta Blangiardo of Imperial Medical College in London and and colleagues, papers by Birgit Schroedle and Leonhard Held of the University of Zurich, and documents by Havard Rue of the Norwegian University of Science and Technology, who is the author of INLA. If you are interested in a fuller introduction to spatial epidemiology in R (similarly stolen from folks smarter than I), please see this document.

9.1 Space-Time Models

We have already seen a fair amount about spatial models. In epidemiology, we are often interested in both the spatial and *temporal* aspects of disease data, e.g. disease location and date of diagnosis, or (perhaps more commonly) disease counts in small areas in a fixed time period. A frequently used approach to spatial disease modeling in small areas dates to work by Besag (1991) which was extended by Bernardinelli (1995) to include a linear term for space-time interaction, and by Knorr-Held (2000) to include a non-parametric spatio-temporal time trend.[1]

Space-time (ST) data can be thought of as multivariate or correlated observations of disease counts within fixed spatial and temporal units that evolve over time, with both space and time are subscripted in the analysis. One might, for example, have 10 years of monthly SMR's at the county level for a state. We could (if we wanted) look at separable models of spatial and temporal terms, or look at the interaction between space and time. For *i*= 1...M small areas, and j = 1...J time periods the disease outcome, y_{ij}, is characterized by an expected counts, e_{ij}, and a risk, $\theta)ij$. The simplest overall average for the expected counts is $e_{ij} = p_{ij} \cdot \Sigma\Sigma / \Sigma\Sigma p_{ij}$, where p_{ij} is the population of the ij^{th} unit.

A basic retrospective model assumes a Poisson distributed outcome in an infinite population with a small disease probability:

$$y_{ij} \sim Pois(e_{ij}, \theta_{ij})$$
$$log(\theta_{ij}) = \alpha_o + S_i + T_j + ST_{ij}$$

where, S = spatial term(s), T= temporal terms(s), ST = space-time interaction

[1] See "A primer on disease mapping and ecological regression using INLA" by Birgit Schroedle and Leonhard Held for a nice description of the evolution of spatiotemporal disease modeling.

In broad overview, possible random effects specifications for the temporal term include a linear time trend (β_t), a random time effect (γ_j), a random walk (γ_{1j}), a first-order autoregression (γ_{2j}), or an interaction between space and time (ψ_{ij}). Some commonly seen specification are:

$$\begin{aligned}
log(\theta_{ij}) &= \alpha_0 + \upsilon_i + \nu_i + \beta_t \\
log(\theta_{ij}) &= \alpha_0 + \upsilon_i + \nu_i + \gamma_j \\
log(\theta_{ij}) &= \alpha_0 + \upsilon_i + \nu_i + \gamma_{1j} + \gamma_{2j} \\
log(\theta_{ij}) &= \alpha_0 + \upsilon_i + \nu_i + \gamma_{2j} + \psi_{ij} \\
log(\theta_{ij}) &= \alpha_0 + \upsilon_i + \nu_i + \gamma_{1j} + \gamma_{2j} + \psi_{ij}
\end{aligned}$$

Which can separated into simple models vs. interaction models.

9.1.1 Simple (Separable) Models

- $log(\theta_{ij}) = \alpha_0 + \upsilon_i + \nu_i + \beta_t$
  - simple separable model, spatial components (BYM) plus *linear* time trend
- $log(\theta_{ij}) = \alpha_0 + \upsilon_i + \nu_i + \gamma_j$
  - simple trend model, spatial component + *random* time effect
  - the γ_j random time effect can be a
    - random walk: $\gamma_j \sim N(\gamma_{j-1}, \tau_\gamma^{-1})$
    - or an $AR_1 : \gamma_j \sim N(\lambda\gamma_{j-1}, tau_\gamma^{-1})$
- $log(\theta_{ij}) = \alpha_0 + \upsilon_i + \nu_i + \gamma_{1j} + \gamma_{2j}$
  - simple model with two random time effects

9.1.2 Interaction Models

- $log(\theta_{ij}) = \alpha_0 + \upsilon_i + \nu_i + \gamma_j + \psi_{ij}$
  - interaction with single random time effect
  - Knorr-Held (2000) suggested dependent priors, but, two simple possible priors are:
    - uncorrelated prior for interaction term: $\psi_{ij} \sim N(0, \tau_\psi)$
    - random walk prior for interaction term: $psi_{ij} \sim N(\psi_{i,j-1}, \tau_\psi)$
- $log(\theta_{ij}) = \alpha_0 + \upsilon_i + \nu_i + \gamma_{1j} + \gamma_{2j} + \psi_{ij}$
  - interaction with two random time effects

In general, the best fitting DICs are seen with the interaction models. The space-time interaction term is essentially a random-effect added to the linear model in the way the unstructured heterogeneity term and spatially-structured conditional autoregression (CAR) heterogeneity terms are added to spatial convolution models. It can be viewed as a kind of residual effect after the unstructured, spatially structured and time effects are modeled. The interaction then is the heterogeneity you didn't "pick up" with the other terms, or (perhaps more accurately) the *additional effect* beyond that which is already picked up with the other space and time terms. In infectious disease models, space-time interaction may represent sporadic outbreaks, like short-term clusters.

9.2 About INLA

Integrated Nested Laplace Approximations (INLA) are a less computationally expensive alternative to MCMC to estimate the integral of a posterior distribution.[2] Commonly used MCMC software like BUGS and JAGS use a computational approach to sample from the posterior distribution of parameters. The INLA approach returns similarly accurate approximations to posterior marginals in significantly less time. Laplace approximations have been known for some time, but only recently developed sufficiently to be accurate enough for inclusion in statistical software and practical application. [3]

Rasmus Baath has a wonderfully succinct and clear explanation of Laplace approximations. In a nutshell, Laplace approximations "work" because most posterior distributions have the largest proportion of their probability "heaped up" in the center and "trail off" symmetrically on either side, i.e. they "look" normal. The Laplace approximation uses the mode as the mean, and calculates the derivative in the area of the mode to approximate the variance.[4] As you might expect, the closer the posterior is to Gaussian, the better the approximation. Fortunately, this is usually the case. When it is not log and logit transformations can be helpful.

Because it doesn't rely on sampling and computations, INLA returns results very quickly, often much faster than MCMC in BUGS or JAGS.[5] It is also useful for large and high dimensional datasets that can stall or freeze up up in BUGS or JAGS, for random effects models, and for complex spatial and spatiotemporal modeling. Also, INLA was written to be seamlessly integrated into R code, and this ease of use has made it an increasingly popular alternative to MCMC for Bayesian spatial modelin.

INLA documentation lags somewhat behind its popularity, but r-inla.org is very helpful website, with numerous examples and useful links, such as lists of likelihoods that can be specified in INLA. [6] Because there is no sampling in INLA, deviance is determined using the likelihood. [7]

You will need to install INLA from from source as described on the r-inla site

```
source("http://www.math.ntnu.no/inla/givemeINLA.R")
```

Once installed, INLA is accessed through an R session. A call to INLA consists of two parts: a formula statement, followed by a model fitting statement.

9.2.1 The INLA Formula Statement

A formula statement for a spatial model looks like this:

[2] Most of this material on the application of INLA to spatial epidemiology comes from Dr. Andrew Lawson's workshop and textbook on the topic.

[3] See Rue, 2009 Approximate Bayesian inference for latent Gaussian models by using integrated nested Laplace approximations

[4] Recall from calculus that the derivative is the instantaneous rate of change. The Laplace approximation makes use of the inverse of the Hessian matrix for the curve to arrive at the result for the variance. The Hessian matrix is a square matrix of partial derivatives for a multivariate function that can be used to describe a local curvature.

[5] INLA is not the only game in town. See for example, Hoffman and Gelman. The no-U-turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. INLA, though, is currently the most mature package in terms of spatial analytic capabilities.

[6] For spatial models, you may notice many of the precisions specified as gamma distributed. This has fallen out of favor in BUGS and JAGS because of computational difficulties, but that is not an issue in INLA.

[7] Recall the deviance information criterion (DIC) is a relative measure. Smaller is better, and results can be negative. As opposed to the number of effective parameters (pD), where smaller is also better, but results can't be negative.

```
form1<-obs ~ 1 +f(region, model="iid")
```

where ,

- *obs* is the disease count or outcome from your dataset
- *1* forces an intercept onto the model
- and, *f()* is a function to specify the spatial region, and how it should be modeled
  - In this statement, spatial region is modeled as "iid" which is Gaussian zero uncorrelated heterogeneity or a random effects term

The *f()* function is a powerful feature of INLA, and is quite flexible. It can be used to invoke most any spatial model, such as geographically weighted regressions (different explanatory associations for outcomes in different regions), or local conditional autoregression models. You can add *f()* functions to each other to build up models. So, for example, you can add a conditional autoregression to a model by adding an *f()* statement with *model = "besag"* to a "iid" uncorrelated heterogeneity *f()*, to create a convolution model. The same model can be specified in more than one way. For example, you can also specify a convolution model directly with a single *f(model="bym")* statement. The approaches differ only in how some elements of the INLA output are displayed.

9.2.2 The INLA Model Statment

An INLA model-fitting statement looks like this:

```
results1<-inla(form1, family="poisson",
data=myDat, control.compute=list(dic=T, cpo=T, graph=T),
E=exp)
```

where,

- *form1* refers to and invokes the previously defined formula
- *family*= specifies the likelihood
- *data*= specifies the data
- *control.compute* = specifies options like DIC and CPO.
  - You can request a DIC for each spatial area with *local dic*, where lighter colors indicate smaller, better fits.
  - **CPO** is the conditional predictive ordinate, a cross validation tool, that predicts an area value using all the data except that area, and compares that value to the actual value. Values range from 0 (poor) to 1 (perfect)
- and, *E* = specifies the offset variable (required for a Poisson likelihood)

You will need some additional tools and utilities to set up a neighborhood adjacency matrix for use with conditional autoregression models and to map results. The R tools *maptools::readShapePoly()* will read shapefiles into R, and *spdep::poly2nb()* followed by *INLA::nb2INLA()* are used to create the adjacency matrix neighbor structures for use with a CAR model. To map results, you can use *sp::spplot()* or *ggplot2* (with some tweaks...)

As an alternative for uncomplicated map and data structures, a function written by Andrew Lawson's colleague, Bo Cai, called "fillmap" can be used to create simple thematic maps. The function is available in Appendix C of Dr. Lawson's textbook on Bayesian Disease Modeling The *fillmap()* function is sourced, and can then be used with any polygon object to fill a thematic map. The basic arguments are:

```
source("~/fillmap.R")
fillmap(obj, "title", varToPlot, n.col=)
```

As a quick example of the use of *fillmap()*, first source the *fillmap()* function, read in a map file (here using *spdep::readSplus* to read a map exported from geobugs) and plot a normally simulated set of data.

```
> #source fillmap function
> source("/Users/charlie/Dropbox/charleston14/charleston14Files/BDMIfiles/BDMI\ participan
> # quick demo use of fillmap
> geobugsSC<-readSplus("/Users/charlie/Dropbox/charleston14/charleston14Files/ABDMfiles/pa
> plot(geobugsSC)
> x<-rnorm(46,1,1)
> fillmap(geobugsSC,"normal simulation",x,n.col=6)
```

9.3 Example: Poisson Model of Suicides in London Boroughs

Blangiardo, et al present an INLA tutorial using suicides counts in the 32 boroughs of London from 1989 to 1983. [8] The data consist of observed suicide counts at the borough level, the expected counts (the result of standardizing the borough rates, by applying London-wide rates to borough populations), a measure of economic deprivation, and a social fragmentation index.

The first basic model is based on a Poisson-distributed variable of counts for each borough $y_i \sim Pois(\lambda_i)$, where the rate in a borough, λ_i, is a risk, θ_i, times the expected count, E_i, for that borough, and the linear predictor for risk on the log scale, $log(\theta_i) = \alpha + \upsilon$, is a linear additive combination of the intercept or *average* city-wide risk on the log scale (α) and a term for spatially *unstructured* heterogeneity, υ.

In WinBUGS or JAGS a model with only unstructured heterogeneity looks like this:

```
model{
for (i in 1: 32){
y[i]~dpois(lambda[i])
log(lambda[i])<-log(expected[i])+log(theta[i])
log(theta[i])<-a0+u[i]
u[i]~dnorm(0,tau.u)}
a0~dnorm(0,tau.0)
sd0~dunif(0,100)

sdu~dunif(0,100)
tau.0<-1/(sd0*sd0)

tau.u<-1/(sdv*sdv)
}
```

We will fit this in INLA.

Load the necessary packages, download the London borough map shapefile, and read it into R.

```
> library(INLA)
> library(maptools)
> library(spdep)
> london<-readShapePoly("~/LDNSuicides.shp")
> str(london)
> plot(london)
> names(london)
> london$NAME
```

The next two chunks of code involve data manipulations. First, read in vectors of data for observed suicide counts (y), expected counts (E), economic deprivation (x1), and social fragmentation (x2). These data vectors are arranged by an alphabetically sorted list of London borough names. Sort the names of the boroughs in the London map object to match the data order and combine the data with this list of alphabetically sorted borough names into a data frame.

[8] See also Marta Blangiardo, Michela Cameletti, Gianluca Baio, Havard Rue. Spatial and spatio-temporal models with R-INLA. Spatial and Spatio-temporal Epidemiology. 4 (2013): 33âĂŞ49, and Congdon P. Bayesian statistical modelling. John Wiley and Sons Ltd; 2007.

```
> y=c(75,145,99,168,152,173,152,169,130,117,124,119,134,90,
+     98,89,128,145,130,69,246,166,95,135,98,97,202,75,100,
+     100,153,194)
> E=c(80.7,169.8,123.2,139.5,169.1,107.2,179.8,160.4,147.5,
+     116.8,102.8,91.8,119.6,114.8,131.1,136.1,116.6,98.5,
+     88.8,79.8,144.9,134.7,98.9,118.6,130.6,96.1,127.1,97.7,
+     88.5,121.4,156.8,114)
> x1=c(0.87,-0.96,-0.84,0.13,-1.19,0.35,-0.84,-0.18,-0.39,0.74,
+      1.93,0.24,0.59,-1.15,-0.8,-0.59,-0.12,1.43,-0.04,-1.24,1,
+      0.53,-0.75,1.36,-0.93,-1.24,1.68,-1.04,2.38,0.03,-0.2,0.14)
> x2=c(-1.02,-0.33,-1.43,-0.1,-0.98,1.77,-0.73,-0.2,-0.96,-0.58,
+      0.36,1.48,0.46,-0.93,-1.62,-0.96,-0.48,0.81,2.41,-0.4,
+      0.71,-0.05,-0.33,-0.47,-0.92,0.06,0.22,-0.73,0.1,-0.59,
+      0.7,2.28)
> names<- sort(london$NAME) # sort vector of borough names
> suicideDat <- data.frame(NAME=names, y=y, E=E, x1=x1, x2=x2) # create dataframe
```

Get the data back into the same order as the map so they line up for plotting. Copy the map to an object called "boroughs", extract the data slot from this map object to a dataframe called "boroughsDat" using the *attr()* function. Create an index called "lineUp" and use it to reorder the suicide data. Then add a sequential ID number to the suicide data set.

```
> boroughs <- london # copy map object
> boroughsDat <- attr(boroughs, "data") # extract data from map
> lineUp <- match(boroughsDat$NAME,suicideDat$NAME) # index map data to dataframe on boroug
> suicideDat <- suicideDat[lineUp,] # sort dataframe by index names from map
> ID<-seq(1,32) # create region ID variable for modeling
> suicideDat <- cbind(ID,suicideDat)
```

9.3.1 Uncorrelated Heterogeneity (Frailty) Model

Run the model in INLA. [9]

```
> formulaUH <- y ~ f(ID, model = "iid") # specify model
> resultUH <- inla(formulaUH,family="poisson", # run inla
+ data=suicideDat, control.compute=list(dic=TRUE,cpo=TRUE),E=E)
```

A summary of the results returns some general information about the run, along with the results for the fixed effects part of the model (α), as well as the DIC. More precise estimates of the fixed effects results are stored in the "summary.fixed" list object in the results object, *resultUH*. Exponentiate the fixed effects to see that based on these data there is on average a 4.6% suicide rate across all London boroughs, with a 95% probability interval ranging from -5.5% to 14.5%.[10]

In this first model, we are interested in the random effect term. Those results are stored in the "region" part of "summary.random" in the named list object created from the INLA run. Extract and examine the mean random effect term for each county. Recall, this is the random variation, on the log scale, around the mean or intercept value of the number of deaths due to congenital birth defects in a county. A plot of the density distribution for the random effect term appears reasonably (for such a small number of observations) normally distributed.

[9] You may want to compare the speed of this run with WinBUGS or JAGS and consider the implications for data much larger than 32 observations...

[10] If this strikes you a high, it did me too. And it is. In fact, the most recent population-based rates for the high-risk middle-age male group across all London is about 0.3%. Clearly , this spatial model is most useful for relative rather than absolute differences.

```
> summary(resultUH)
> exp(resultUH$summary.fixed)
> names(resultUH) # want "summary.random" for UH effects
> resultUH$summary.random # want the mean, which is 2nd column
> RE1<-resultUH$summary.random$ID[,2]
> plot(density(RE1)) # plot density random effects
```

Next, map the UH results. It takes a bit of care to make sure the results are matched to the correct geographic regions. The *fillmap()* function is a quick and easy approach to simple, consecutive data, but in this case the default approach returns the wrong results. Rather than adjust the results to fit *fillmap()*, I find the *attr()* function to be the most reliable tool to manipulate modeling results into the correct order and format for mapping.

First, add the random effect results to the *suicideDat* dataframe and create *cuts* to plot categories of outcomes. Then use the unique names to merge the data frame to the *boroughsDat* data frame that was extracted from the map object and replace the *data* slot of the map object with the merged datafame.

```
> source("/Users/charlie/Dropbox/charleston14/charleston14Files/ABDMfiles/participant_file
> fillmap(london, "UH effect", RE1*100, n.col=6) # map
> # add random effects results to dataframe
> suicideDat$UH<-RE1
> #categorize the random effects results
> summary(suicideDat$UH)
> cuts <- c(-0.396100, -0.184800, -0.035180,  0.131400,  0.450200)
> suicideDat$UH.cut<-cut(suicideDat$UH,breaks=cuts,include.lowest=TRUE)
> #merge suicide and map dataframes and save into data slot of map
> attr(london,"data") <- merge(boroughsDat,suicideDat,by="NAME")
> # map
> spplot(london, "UH.cut", col.regions=gray(3.5:0.5/4),main="")
>
>
```

Looking at the uncorrelated heterogeneity is useful from a purely spatial perspective, but as epidemiologists we will invariably be most interested in extracting and characterizing the risk estimates from the INLA results. The result list object "summary.fitted.values" is the exponentiated risk $\theta_i = \alpha_i + \upsilon_i$. A more nuanced result is the exponentiated υ_i output which is the risk represented by the uncorrelated heterogeneity, which can be interpreted as the normally distributed geographic risk. This is obtained by applying the INLA function *inla.emarginal()* on the marginal results for the random effect. [11] In a convolution model with a CAR term, this process will return the spatially correlated or structured risk, $\zeta = \upsilon_i + \nu_i$.

Exceedance probabilities for an area are the posterior probabilities that the spatial risk is greater than 1 ($Pr[\zeta_i > 1|y]$). They are obtained by applying the INLA function *inla.pmarginal()* to the marginals for the random effect. [12]

The exceedance probability of spatial risk $\upsilon + \nu$ greater than 1 is more readily accessible from the model results on the un-exponentiated scale, so we set the threshold to 0 ("a=0"). To obtain exceedance of a greater threshold, change that value, e.g. risk of 2 on the un-exponentiated scale is 0.6931472.[13]

```
> #risk (theta = alpha + upsilon)
> UHrisk <- resultUH$summary.fitted.values[,1]
> UHrisk
```

[11] For this simple frailty or random effects model of uncorrelated hetergeneity, if you exponentiate the random effects result from the previous section you should get the same result as that from *inla.marginal()* ($exp(RE1)$ vs. $unlist(UHzeta)$). And you do. Basically. There are some slight differences. I am not sure why.

[12] Exceedance probabilities have been proposed as a Bayesian approach to hotspot identification. They can be very useful, though they can be sensitive to model specification. See this commentary

[13] There is a function unrelated to INLA that automates exceedance probability calculations. It is called *excProb*, is available in the "geostatsp" package, and requires the package "pracma". To use it, extract the list of two-column matrices with columns named x and y containing the posterior distributions of random effects, as produced by INLA (e.g. inlaMargs<-resultCAR$marginals.random$ID.area[1:1000]) and set the threshold to what you want, e.g. excProb(inlaMargs, threshold=2)

```
> #spatial risk (zeta = upsilon only)
> UHmarginals <- resultUH$marginals.random$ID[1:32] # extract UH term (1 to number of area
> UHzeta <- lapply(UHmarginals,function(x)inla.emarginal(exp,x)) # exponentiate
> # exceedence probability
> a=0
> UHexceedence<-lapply(resultUH$marginals.random$ID[1:32], function(X){
+   1-inla.pmarginal(a, X)
+ })
```

Add these results to the dataframe, categorize, save the dataframe to the map data slot and map.

```
> spatResults <- data.frame(NAME=suicideDat$NAME, risk=UHrisk,    geoRisk=unlist(UHzeta),
+                              exceed=unlist(UHexceedence))
> RR.cut<-c(0.6, 0.9, 1.0, 1.1,  1.8)
> geoRisk.cut<- c(0.6, 0.9, 1.0, 1.1,  1.8)
> exceed.cut<- c(0,0.2,0.5,0.8,1)
> spatResults$risk.cut<-cut(spatResults$risk,breaks=RR.cut,include.lowest=TRUE)
> spatResults$geoRisk.cut<-cut(spatResults$geoRisk,breaks=geoRisk.cut,include.lowest=TRUE)
> spatResults$exceed.cut<-cut(spatResults$exceed,breaks=exceed.cut,include.lowest=TRUE)
> attr(london,"data")<-merge(boroughsDat,spatResults,by="NAME")
> spplot(london, c("risk.cut","geoRisk.cut"), col.regions=gray(3.5:0.5/4),main="")
> spplot(london, c("exceed.cut"), col.regions=gray(3.5:0.5/4),main="")
```

We see that the patterns are fairly similar across all three outcomes.

9.3.2 *Conditional Autoregression (Convolution) Model with Covariates*

As mentioned above, there are two ways to specify a convolution model in INLA. You can add a CAR term to a random effect component:[14]

```
f(region, model = "iid") + f(region2,model="besag",graph="adj_matrix.txt")
```

or, you can specify a convolution model with a single *f()* function:[15]

```
f(region,model="bym",graph="adj_matrix.txt")
```

The only difference in the results is how the output is structured and hence extracted for evaluation and display. In the formulation with two *f()* statements, there is a separate summary result for each part of the model, one stored in summary.random$region the other in summary.random$region2. (We need to create a second, identical geographic indexing variable called region2.) In the single *f(model="bym")* statement formulation, the results for each component of the model are concatenated in a single summary.random$region result, with the UH terms listed first, followed by the CAR terms. We extract the terms we want by indexing. In either case we must provide an adjacency matrix, which we create by first using the *spdep:::poly2nb* tool to create the matrix, and then the *INLA:nb2INLA* tool to format the matrix for INLA.

```
> temp <- poly2nb(london)
> nb2INLA("LDN.graph", temp)
> LDN.adj <- paste(getwd(),"/LDN.graph",sep="")
```

In this example we'll run a convolution model and include the covariates for economic deprivation and social fragmentation. The model is

[14] "besag" refers to Julian Besag, who developed the CAR model of spatially correlated heterogeneity in 1974.

[15] "bym" refers to Besag, York and Mollie. To whom this model formulation is attributed.

$y_i = \alpha + \beta_1 x_{1i} + \beta_2 x_{2i} + \upsilon_i + \nu_i$

Run the model with a single *f(model="bym")* statement.

```
> formulaCONVcov <- y ~ 1+ f(ID, model="bym", graph=LDN.adj) + x1 + x2
> resultCONVcov <- inla(formulaCONVcov,family="poisson",
+ data=suicideDat, control.compute=list(dic=TRUE,cpo=TRUE),E=E)
```

Review the results. Looking at the fixed effects, the intercept, or average risk across all boroughs, is about 6%. The exponentiated risks for economic deprivation and social fragmentation are interpreted as incidence density ratios or risks. For each 1 unit increase in economic deprivation, there is an approximate 9.5% risk in suicide. Each 1 unit increase in social deprivation is associated with an approximate 19.7% increase in suicide risk.

```
> summary(resultCONVcov) #DIC 259 vs 270 for UH alone
> exp(resultCONVcov$summary.fixed) # mean 6%, econ 9.5%, social 19.7%
> names(resultCONVcov) # "summary.random" stores UH and CAR effects
> resultCONVcov$summary.random # want the mean, which is 2nd column
> re2<-resultCONVcov$summary.random$ID[1:32,2]# random effects
> plot(density(re2)) # plot density random effects
> car1<-resultCONVcov$summary.random$ID[33:64,2] # correlated spatial (car)
> plot(density(car1))
```

Calculate overall risk ($\theta = \alpha + \upsilon + \nu$), spatial risk ($\zeta = \upsilon + \nu$) and spatial exceedance ($\Pr[\zeta_i > 1]$). Note that the spatial risk (ζ) is now interpreted as the *residual relative risk for each area (compared to the whole of London) after economic deprivation and social fragmentation are taken into account.*

```
> #risk (theta = alpha + upsilon + nu)
> CONVcovrisk <- resultCONVcov$summary.fitted.values[,1]
> CONVcovrisk
> #spatial risk (zeta = upsilon + nu)
> CONVcovmarginals <- resultCONVcov$marginals.random$ID[1:32]
> CONVcovzeta <- lapply(CONVcovmarginals,function(x)inla.emarginal(exp,x)) # exponentiate
> # exceedence probability
> a=0
> CONVcovexceedence<-lapply(resultCONVcov$marginals.random$ID[1:32], function(X){
+   1-inla.pmarginal(a, X)
+ })
>
```

Mapping these results demonstrates the spatial character of suicide risk more clearly than with the simple UH model.

```
> spatResults2 <- data.frame(NAME=suicideDat$NAME, risk=CONVcovrisk,    geoRisk=unlist(CONV
+                                 exceed=unlist(CONVcovexceedence))
> RR.cut<-c(0.6, 0.9, 1.0, 1.1,  1.8)
> geoRisk.cut<- c(0.6, 0.9, 1.0, 1.1,  1.8)
> exceed.cut<- c(0,0.2,0.5,0.8,1)
> spatResults2$risk.cut<-cut(spatResults2$risk,breaks=RR.cut,include.lowest=TRUE)
> spatResults2$geoRisk.cut<-cut(spatResults2$geoRisk,breaks=geoRisk.cut,include.lowest=TRUE
> spatResults2$exceed.cut<-cut(spatResults2$exceed,breaks=exceed.cut,include.lowest=TRUE)
> attr(london,"data")<-merge(boroughsDat,spatResults2,by="NAME")
> spplot(london, c("risk.cut","geoRisk.cut"), col.regions=gray(3.5:0.5/4),main="")
> spplot(london, c("exceed.cut"), col.regions=gray(3.5:0.5/4),main="")
>
```

From these results, you can calculate the proportion of variance explained by spatially structured (CAR) component [16] The process involves creating an empty matrix with rows equal to the number of regions, and 1000 columns into which we extract 1000 observations from the marginal distribution of the CAR term (v), which is used to calculate the empirical variance. Similarly, values are extracted for the random effects (UH) term and the variance calculated. Finally, sum the two variances, and calculate the proportion of total heterogeneity accounted for by the spatially structured heterogeneity. In this case, spatially structured variance accounts for about 74% of the total variance.

```
> mat.marg <- matrix(NA, nrow=32, ncol=1000)
> m<-resultCONVcov$marginals.random$ID
> for (i in 1:32){
+   u<-m[[32+i]]
+   s<-inla.rmarginal(1000, u)
+   mat.marg[i,]<-s}
> var.RRspatial<-mean(apply(mat.marg, 2, sd))^2
> var.RRhet<-inla.emarginal(function(x) 1/x,
+                             resultCONVcov$marginals.hyper$"Precision for ID (iid component
> var.RRspatial/(var.RRspatial+var.RRhet)
```

9.4 Example: Logit Model of Cumbria Foot and Mouth Disease

INLA can be used for other model specifications. Here we look at counts of foot and mouth disease in 138 Cumbrian parishes in northwest England in 2001. The data are disease counts and an estimate of the population over a single time period. We will use the following model:

$$\begin{aligned} y_i &\sim bin(n_i, p_i) \\ logit(p_i) &= \alpha_0 + v_i \\ \alpha_0 &\sim N(0, \tau_0^{-1}) \\ v_i &\sim N(0, \tau_v^{-1}) \end{aligned}$$

In the following code, we set up the data as a list object, read in a map file and use *fillmap()* to quickly map the crude rates. We then set up and run a binomial model in INLA.

```
> # set up FMD data
>
> FMDframe<-list(n=138,
+ count=c(1,0,1,0,2,3,1,0,1,0,3,2,0,3,1,6,1,10,2,1,8,4,4,1,1,
+ 3,9,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,
+ 0,0,0,1,1,1,0,0,0,0,0,0,2,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,4,0,0),pop=c( 26,19,28,42,55,30,45,20,35,73,35,5
> FMDmap<-readSplus("~/FMD_splusMAP.txt")
> plot(FMDmap)
> ### map of crude rates with fillmap
>
> rate<-FMDframe$count/FMDframe$pop
> fillmap(FMDmap,"crude rate map",rate,n.col=6)
```

[16] Note. Andrew Lawson cautions against measures like this because CAR and UH are incompletely identified. Even Blangiardo, from whom this code is adapted, says the UH and CAR variances are not directly comparable, though the result can give an indication of the relative contribution of each.

```
> # set up area indicators for map (NB need to be indexed to map units...)
> ind<-seq(1:138)
> # set up and call inla
> formula1<-count~1+f(ind,model="iid")
> res1<-inla(formula1,family="binomial",data=FMDframe,Ntrials=FMDframe$pop,
+ control.compute=list(dic=TRUE))
>
```

Assess the results by first looking first at a summary, then at the UH or unstructured spatial random effects and then at a map of residuals. The unstructured random effect term picks up some spatial structure since it is the only spatial term on the right side of the equation. Map a measure of the local DIC, as well as the probability of infection based on the linear predictor results in the logistic formula:

$$\frac{e^{\beta_0+\beta_1 x1 \ldots \beta_i xi}}{1+e^{\beta_0+\beta_1 x1 \ldots \beta_i xi}}$$

```
> summary(res1)
> # look at uncorrelated heterogeneity
> UH<-res1$summary.random$ind[,2]
> fillmap(FMDmap,"posterior mean UH component",UH,n.col=4)
> # calculate and map residuals
> fit<-res1$summary.fitted.values$mean
> resid<-FMDframe$count-fit
> fillmap(FMDmap,"estimated crude residuals",resid,n.col=4)
> # map local dic
> LOCdic<-res1$dic$local.dic
> fillmap(FMDmap,"local DIC",LOCdic,n.col=5)
> # calculate and map risk estimates
> # grab the linear predictors, then calculate inverse logit to
> # get probability vector
> linPred<-res1$summary.linear.predictor$mean
> prob<-exp(linPred)/(1+exp(linPred))
> fillmap(FMDmap,"posterior mean infection probability",prob,n.col=4)
>
```

9.5 Space-Time Models With INLA

9.5.1 Example: Spatiotemporal Model of Respiratory Cancer in Ohio Counties

As an example of spatiotemporal modeling in INLA we look at respiratory cancer in Ohio counties. 21 years of data are available. We will analyze 10 years of data, from 1979 ti 1988.

The first task is setting up the data. The data are in a list object called *ohioDat* and consist of observed (y) and expected (e) respiratory cancers over 10 years for 88 counties in Ohio. The variable m stores the number of counties (88); the variable T is the number of years (10), the variable t is the sequence of years from 1 to 10.

Next manipulate the data object to get it from the "wide" format into the "long" format of one space-time observation per row required for analysis in INLA. [17]

[17] There are a number of ways to do this, e.g with *base::reshape* or *reshape2::melt*. Here we use an approach described by Dr. Lawson.

```
> load("/Users/charlie/Dropbox/srtsAnalysis/srtsNotes/srtsINLA/ohioDat.RData")
> # convert data to long form
> yL<-rep(0,880)
> eL<-rep(0,880)
> T <- 10
> for (i in 1:88){
+ for (j in 1:10){
+ k<-j+T*(i-1)
+ yL[k]<-ohioDat$y[i,j]
+ eL[k]<-ohioDat$e[i,j]
+ }}
>  # set up variables for year (simple trend 1-10 repeated), region, then copy repeat regi
>  # (for use in second f() function), then set up space-time interaction term (ind2) seq
>  # (incrementing over each observation)
> year<-rep(1:10,len=880)
> region<-rep(1:88,each=10)
> region2<-region
> ind2<-rep(1:880) # this is the space-time interaction term
> # inla requires a data frame
> ohioDatL<-data.frame(year,region,yL,eL,region2,ind2)
```

The next step is to read in the map file and create the adjacency matrix.[18]

```
> library(maptools)
> polyOHIO<-readSplus("~/Ohio_splusMAP.txt")
> plot(polyOHIO)
> source("~/fillmap.R")
> x<-rnorm(88,1,1)
> fillmap(polyOHIO, "Ohio Cancer", x, 6)
> library(spdep)
> library(INLA)
> adjpoly<-poly2nb(polyOHIO)
> nb2INLA("OHIO_map",adjpoly)
> # OHIO.adj <- paste(getwd(),"OHIO.adj",sep="")
```

In the following code we run and save a number of potential models. These models are various combinations of unstructured heterogeneity or spatial random effect (UH), spatially structured heterogeneity (SH) or CAR models, linear time trend (just adding the sequential time variable), unstructured time (iid distributed time variable), autoregressive time (random walk) and space time interactions. We compare them on DIC.

```
> #### spatial only UH, poisson likelihood
> formula1<-yL~1+f(region,model="iid")
> result1<-inla(formula1,family="poisson",data=ohioDatL,
+ E=eL,control.compute=list(dic=TRUE,cpo=TRUE))
> summary(result1)
> result1$dic$dic # 6823
> # random effect (UH)
> result1$summary.random$region[,2]
> #risk (theta = alpha + upsilon) value for each space-time observation
> result1$summary.fitted.values[,1]
> #exponentiated fixed effect
> exp(result1$summary.fixed) # avg -3%
> #### add  random effect (UH) and correlated spatial (CH or CAR), i.e. convolution model
> formula2<-yL~1+f(region,model="iid")+f(region2,model="besag",graph="OHIO_map")
```

[18] Note that though I am not doing it here, a good place to start any time-space analysis is to simply look at a series of maps for each year...

```
> result2<-inla(formula2,family="poisson",data=ohioDatL,
+ E=eL,control.compute=list(dic=TRUE,cpo=TRUE))
> result2$dic$dic # 6824 (no improvement)
> ### spatial (u and v) + time trend (simply add year as covariate)
> formula3<-yL~1+f(region,model="iid")+f(region2,model="besag",graph="OHIO_map")+year
> result3<-inla(formula3,family="poisson",data=ohioDatL,
+ E=eL,control.compute=list(dic=TRUE,cpo=TRUE))
> result3$dic$dic # 6777 (improved over UH only)
> exp(result3$summary.fixed) # avg 3.7% (intercept) ~1.3% decrease each year
> result1$summary.fitted.values[,1]
> result3$summary.random$region[,2] # UH result
> result3$summary.random$region2[,2] # CAR result
> ###  uncorrelated (UH or random effect) + correlated (CH or CAR) spatial + uncorrelated t
> formula4<-yL~1+f(region,model="iid")+f(region2,model="besag",graph="OHIO_map")+f(year,mod
> result4<-inla(formula4,family="poisson",data=ohioDatL,
+ E=eL,control.compute=list(dic=TRUE,cpo=TRUE))
> result4$dic$dic # 6598 (random effect time better than linear time)
> ### uncorrelated (UH or random effect) + correlated (CH or CAR) spatial + random walk ti
> formula5<-yL~1+f(region,model="iid")+f(region2,model="besag",graph="OHIO_map")+f(year,mod
> result5<-inla(formula5,family="poisson",data=ohioDatL,
+ E=eL,control.compute=list(dic=TRUE,cpo=TRUE))
> result5$dic$dic # 6598 (correlated time same as random effect time)
> result5$summary.random$region[,2] # UH
> result5$summary.random$region2[,2] # CAR
> result5$summary.random$year[,2] # time effect
> ###  uncorrelated (UH or random effect) + correlated (CH or CAR) spatial + random walk t
> year2<-year # need second time variable for model
> formula6<-yL~1+f(region,model="iid")+f(region2,model="besag",graph="OHIO_map")+f(year,mod
> result6<-inla(formula6,family="poisson",data=ohioDatL,
+ E=eL,control.compute=list(dic=TRUE,cpo=TRUE))
> result6$dic$dic # 6598  (having random plus correlated time effects same as having one o
> ### now adding an interaction term to the UH random effect model with random walk time te
> formula7<-yL~1+f(region,model="iid")+f(year,model="rw1")+f(ind2,model="iid")
> result7<-inla(formula7,family="poisson",data=ohioDatL,
+ E=eL,control.compute=list(dic=TRUE,cpo=TRUE))
> result7$dic$dic # 6033 (adding interaction meaningfully improves the model)
> ###  convolution model with interaction term:  UH +CH + year RW1 + INT IID
> formula8<-yL~1+f(region,model="iid")+f(region2,model="besag",graph="OHIO_map")+f(year,mod
> result8<-inla(formula8,family="poisson",data=ohioDatL,
+ E=eL,control.compute=list(dic=TRUE,cpo=TRUE))
> result8$dic$dic # 6033 (no improvement over previous random effect)
```

Model 7 appears to have the best combination of DIC and parsimony. This model consisted of a spatial random effect or unstructured heterogeneity term, an autoregressive temporal term and a space-time interaction term.

Explore some results. Save the spatial random effect, autoregressive temporal term and the interaction results to objects. Map the unstructured heterogeneity term and plot it's density. It appears random and centered around zero. Then plot the temporal heterogeneity terms by year. There appears to be an overall negative trend with the final year perhaps something of an outlier. A plot omitting year 10 with a line for the simple linear regression of temporal heterogeneity on time perhaps illustrates the trend more clearly.

Finally, set up a matrix of 88 rows counties and 10 columns for time and populate it with the interaction effect results. Plot the first two years of results.

As noted above, the space-time interaction is a random-effect term can be viewed as a kind of residual effect after the unstructured, spatially structured and time effects are modeled. It is the additional effect beyond that which is already picked up with the other space and time terms. In this case, the areas with elevated values represent sporadic outbreaks, like short-term clusters, in years one and two.

```
>  UH<-result7$summary.random$region[,2]
> yearR<-result7$summary.random$year[,2]
> STint<-result7$summary.random$ind2[,2]
> fillmap(polyOHIO,"UH",UH*100,n.col=6)
> plot(density(UH))
> time<-seq(1:10)
> plot(time,yearR)
> lines(time,yearR)
> yearR[-10]
> plot(time[-10],yearR[-10])
> abline(lm(yearR[-10]~time[-10]))
> STest<-matrix(0,nrow=88,ncol=10)
> for(k in 1:880){
+ i=ceiling(k/10)
+ j=k-10*(i-1)
+ STest[i,j]<-STint[k]}
> ST1<-STest[,1]
> ST2<-STest[,2]
> fillmap(polyOHIO,"ST interaction yr 1",ST1,n.col=6)
> x11()
> fillmap(polyOHIO,"ST interaction yr 2",ST2,n.col=6)
```

Chapter 10
Introduction to Spatial Point Pattern Analysis

10.1 About Spatial Point Data

As promised, here is a brief introduction to point process data. [1] As their name implies, point processes are discrete events that can be located with more or less precision in space (and in time, actually). They are usually assumed to follow a probability distribution, and are often described as Poisson processes. The points may be accompanied by additional information (sometimes called "marks"). Note the these marks differ from covariates. You can think of them as "part" of the point. So, for example, a mark might be whether a point represents a case or a control.

Collections of spatial points may be characterized by their "intensity" or density per unit area. Intensity may be homogenous (uniform) or it may vary (in-homogenous or non-uniform.) Much like time series data (which are points, too, only in time) spatial points are not independent, and points near each other are like each other, or perhaps even affect each other.

Spatial points may be associated with covariates, and analyses might involve assessing the role of covariates in determining intensity (e.g. does disease incidence depend on elevation above sea level?), or controlling for covariate effects when assessing interaction between points (e.g. is there clustering of disease outcomes after controlling for elevation?).

10.1.1 Analytic Approaches to Point Data

The analysis of point data falls under three (very) general headings: summary statistics, assessments of randomness, and modeling. Summary statistics are things like nearest neighbor analyses. Assessments of randomness usually involve determining if a set of points can be reasonably accepted as a uniform Poisson process ("completely at random"). By random, we mean, the *number* of points, as well as their *location* in 2-dimensional space is random. We refer to a *realization* of a point process, or the actaul data before us. Note that the data are essentially unordered (in time). If time is (ahem) of the essence, or a critical part of the data, then it is not a classic spatial point process, and may require other analytic approaches.

Finally, we can fit models to point pattern data and use the models for prediction and/or explanation. We can use the model to simulate patterns. Think of the point process as the result of a set of "rules" that produced the points precisely where we found them. A point process model is an approximate of those "rules" which we can then use to produce (simulated) points. We can then compare our simulation to the actual points.

[1] In the words of Clint Eastwood, "A man's got to know his limitations." Point pattern analyses are one of my limitations. As I mentioned very early on, I usually defer to folks who have the necessary expertise to understand and deal with this fascinating, but technically complex, topic. I include this material for completeness sake. In addition to Roger Bivand's "ASDAR", I highly recommend this very nice and thorough description of point pattern analysis by Adrian Baddely, the author of the spatstat packaage, available here As for my notes, *Caveat emptor*

10.1.1.1 Windows

We assume that the data extend beyond the boundaries of what is presented to us. What we see is referred to as a "sampling window" of unordered spatial points. This idea of "windows" is actually quite important. First, the size of the window may affect the density and appearance of the points within it. Also, the location of the window (if it is an in-homogenous process) may also impact the density of the points. Second, in the same way we as epidemiologists know we need to compare groups (e.g. exposed and unexposed) we need comparison windows to assess the role of covariates.

10.2 R Tools for Spatial Point Pattern Analysis

In addition to the *sp* package which has SpatialPoints objects, the R packages *spatstat* and *splancs* have a number of useful tools for spatial pattern analysis. Of the two, the *spatstat* package is the most comprehensive. It utilizes *ppp* objects to store spatial points. The *maptools* package has functions to convert SpatialPoint objects to ppp objects and vice versa.

10.2.1 spatstat

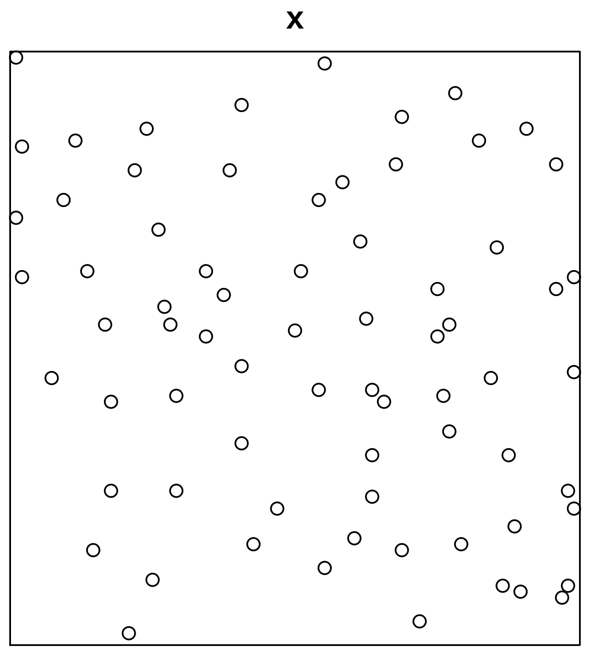
Let's take a quick look at a spatstat. We begin by loading the "swedishpines" data set which containts the position of 71 trees in a Swedish forest. We will plot it, print the basic information about it, get some summary statistics, and then plot it's intensity with both a so-called "kernal estimate" and as a contour plot.[2]

```
> library(spatstat)
> data(swedishpines)
> x<-swedishpines
> summary(x)
```

```
Planar point pattern: 71 points
Average intensity 0.0074 points per square unit (one unit = 0.1 metres)

Window: rectangle = [0, 96]x[0, 100]units
Window area =  9600 square units
Unit of length: 0.1 metres
```

```
> plot(x)
```

[2] The parameter 10 in the density plot is a smoothing factor. It is based on an estimate of the standard deviation of the "Gaussian smoothing kernal". The value (based on the default scale of the data) is 10 decimeters or 1 meter. We'll discuss this a bit more later on. For now, feel free to play around with it.



```
> plot(density(x, 10))
> contour(density(x,10),axes= FALSE)
```

In this next code we calculate and plot a spatial point statistic called *Ripley's K function*. This is one of those summary tests of whether the data are randomly distributed. We'll discuss in more detail later. For now, let's just run through it quickly.

```
> K<- Kest(x)
> plot(K)
```

```
       lty col    key             label
iso      1   1    iso   hat(K)[iso](r)
trans    2   2  trans hat(K)[trans](r)
border   3   3 border  hat(K)[bord](r)
theo     4   4   theo       K[pois](r)
                                               meaning
iso    Ripley isotropic correction estimate of K(r)
trans        translation-corrected estimate of K(r)
border            border-corrected estimate of K(r)
theo                       theoretical Poisson K(r)
```

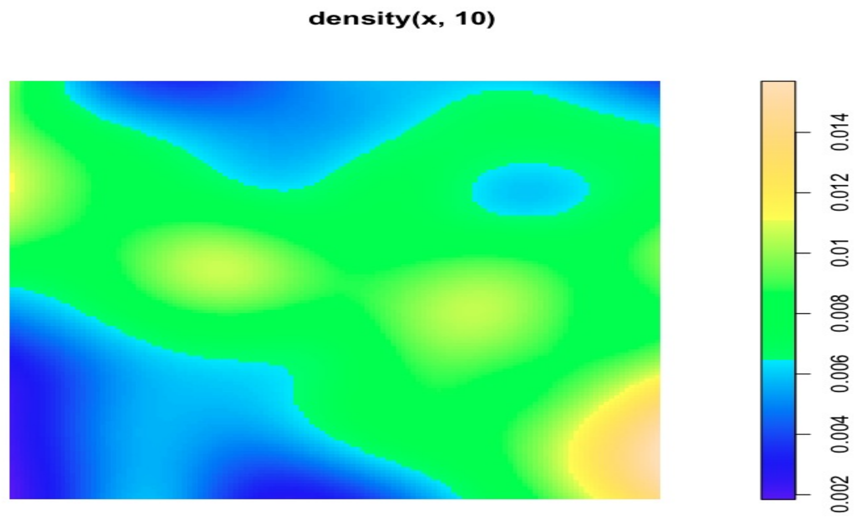


Fig. 10.1 caption

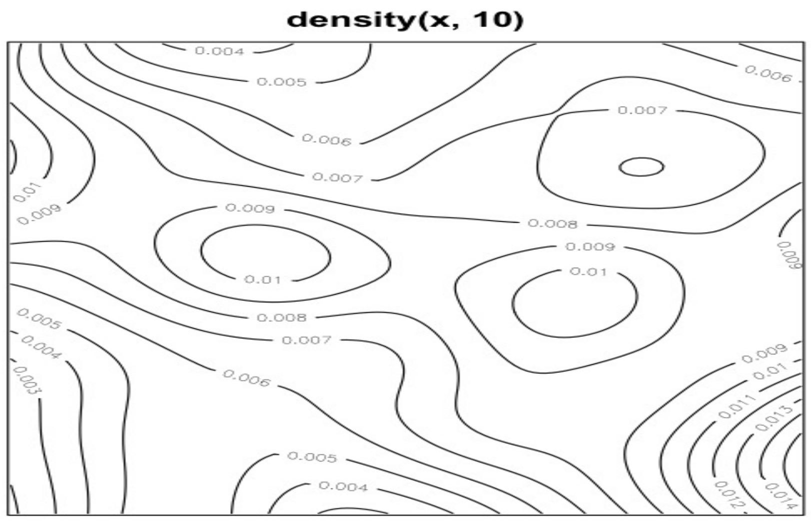
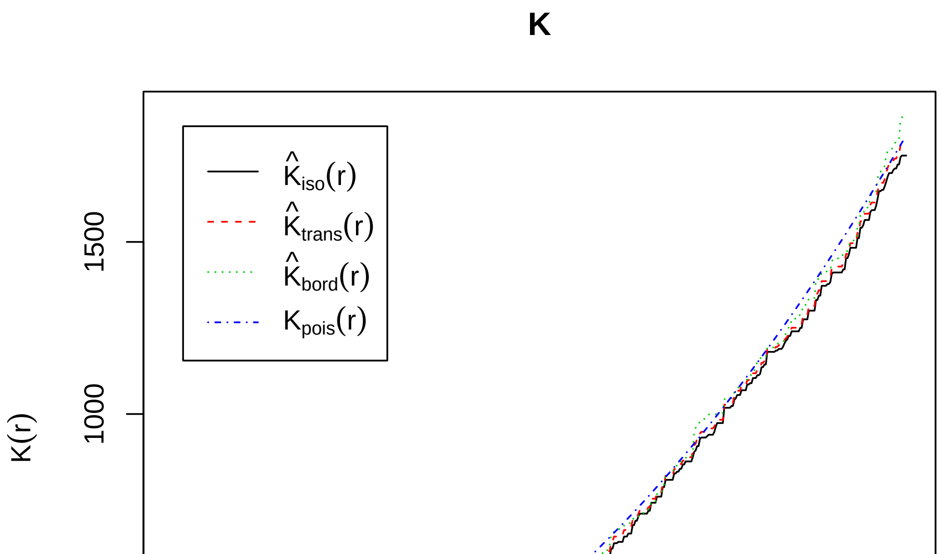


Fig. 10.2 caption



The solid lines are the (observed) K function. The dashed blue line is the expected value assuming the points are completely at random. We see that the observed data appears to deviate from the expected, indicating that the data are not randomly distributed. We can test this observation with a (Monte Carlo) simulated values test based on "envelopes" of the K function.

```
> E<-envelope(x, Kest, nsim=39)
```

```
Generating 39 simulations of CSR  ...
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30,
31, 32, 33, 34, 35, 36, 37, 38, 39.

Done.
```

```
> plot(E)
```

```
     lty col  key      label                                          meaning
obs    1   1  obs  K[obs](r)           observed value of K(r) for data pattern
theo   2   2 theo K[theo](r)                 theoretical value of K(r) for CSR
hi     1   8   hi   K[hi](r) upper pointwise envelope of K(r) from simulations
lo     1   8   lo   K[lo](r) lower pointwise envelope of K(r) from simulations
```

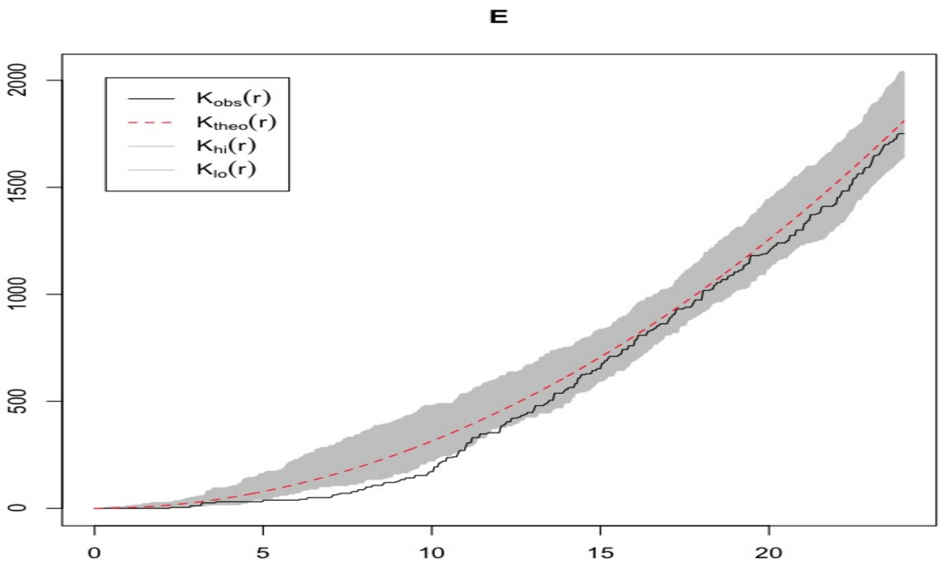
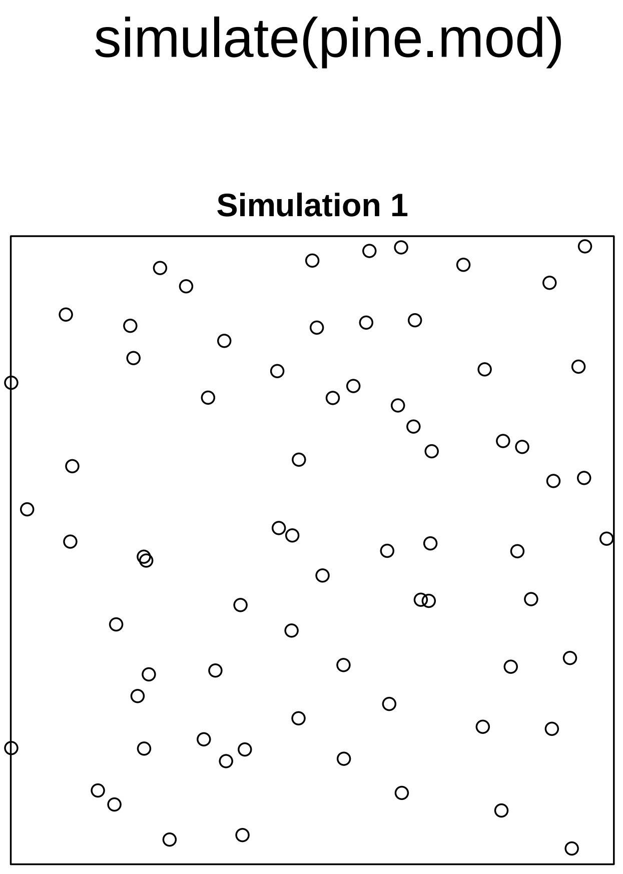


Fig. 10.3 caption

The *ppm* ("point process model") function in spatstat can be used to model point data. (It's like *lm* or *glm* in the R base statistics package.) Here we fit a model to the swedishpine data. (Don't worry right now about what the parameters mean for now.) We then plot a simulation of the fitted model, and use the *envelope()* function to perform a goodness-of-fit test.

```
> pine.mod<-ppm(x,~1,Strauss(9))
> plot(simulate(pine.mod))
```



```
> plot(envelope(pine.mod, Kest, nsim=39))
```

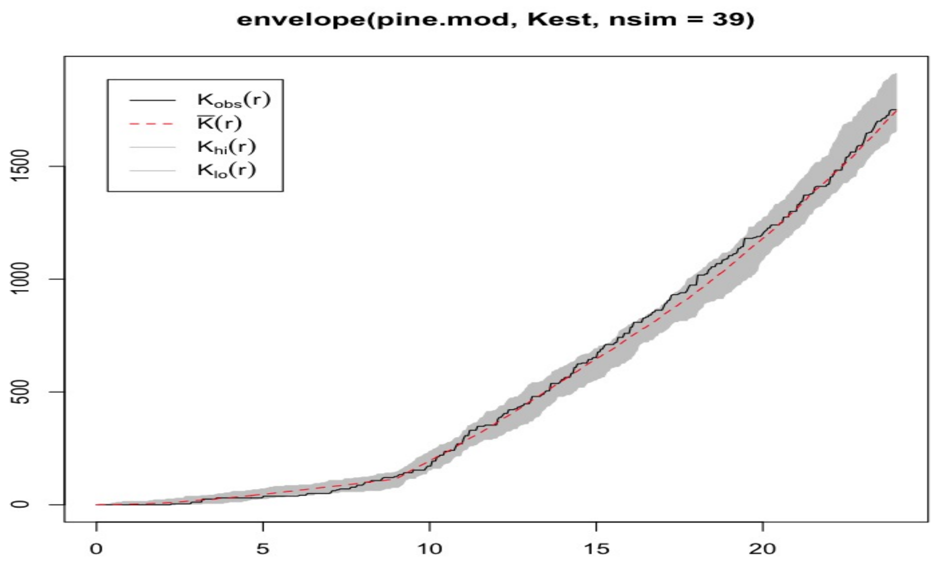


Fig. 10.4 caption

The plot indicates pretty good agreement between the data and the model.

10.2.2 Marked Points

As I mentioned, spatial points may come with additional inherent information called "marks". Let's look at this kind of data. We'll load another spatstat data set, this time the "lansing" data set which consists of different kinds of trees.

```
> data(lansing)
> lansing
```

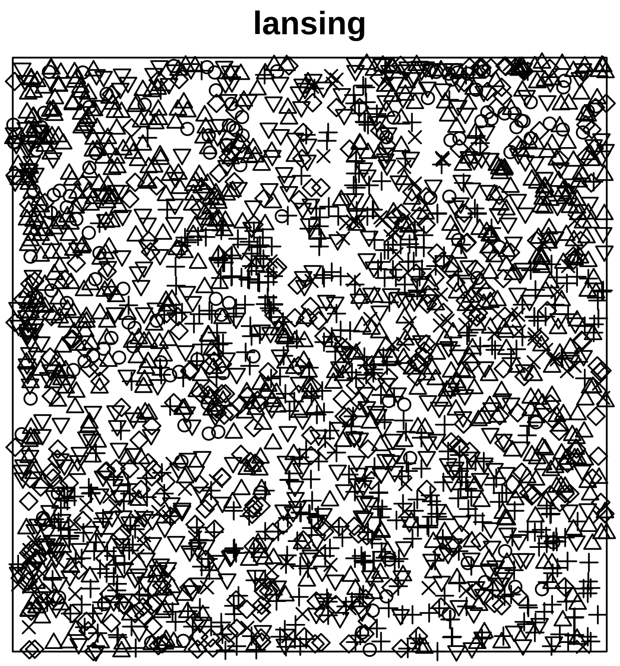
```
marked planar point pattern: 2251 points
multitype, with levels = blackoak        hickory        maple        misc        redoak
window: rectangle = [0, 1] x [0, 1] units (one unit = 924 feet)
```

```
> summary(lansing)
```

```
Marked planar point pattern: 2251 points
Average intensity 2250 points per square unit (one unit = 924 feet)

*Pattern contains duplicated points*
Multitype:
         frequency proportion intensity
blackoak       135     0.0600       135
hickory        703     0.3120       703
maple          514     0.2280       514
misc           105     0.0466       105
redoak         346     0.1540       346
whiteoak       448     0.1990       448

Window: rectangle = [0, 1]x[0, 1]units
Window area =  1 square unit
Unit of length: 924 feet
```

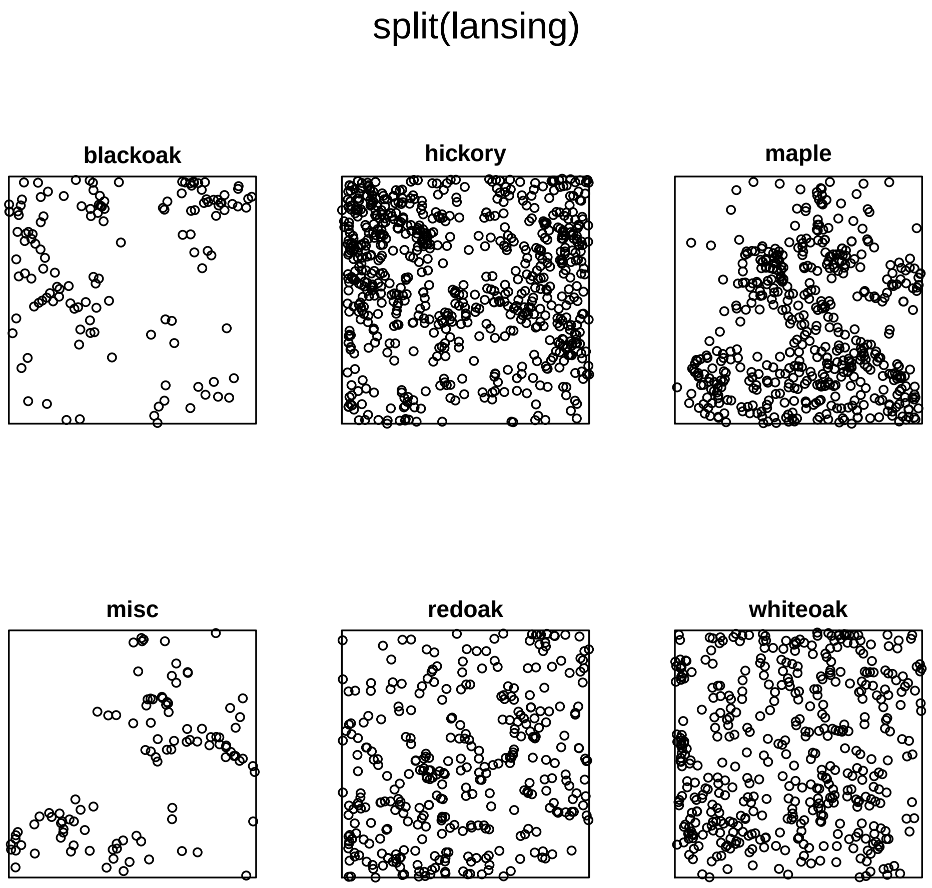
```
> plot(lansing)
```

```
blackoak  hickory    maple     misc   redoak whiteoak
       1        2        3        4        5        6
```



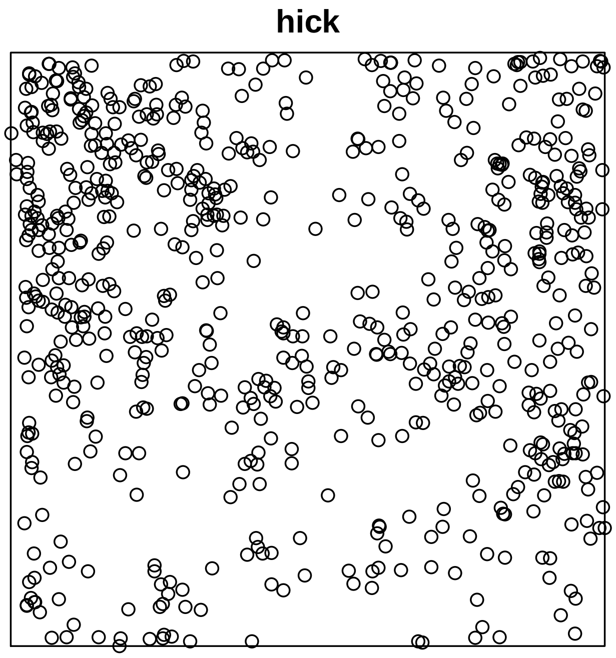
We see that there is a different symbol for each type of tree. The *split()* function allows us to stratify by the plot by type.

```
> plot(split(lansing))
```



We can also look at one of the types individually.

```
> hick<-split(lansing)$hickory
> plot(hick)
```



10.3 Reading in and Plotting with a Spatial Point Data Set

Much of the initial work with spatial points can be accomplished with with the *sp* package and a *spatialPointsDataFrame* object. Afterward, you can convert the *sp* objects *ppp* object for analysis in *spatstat*.

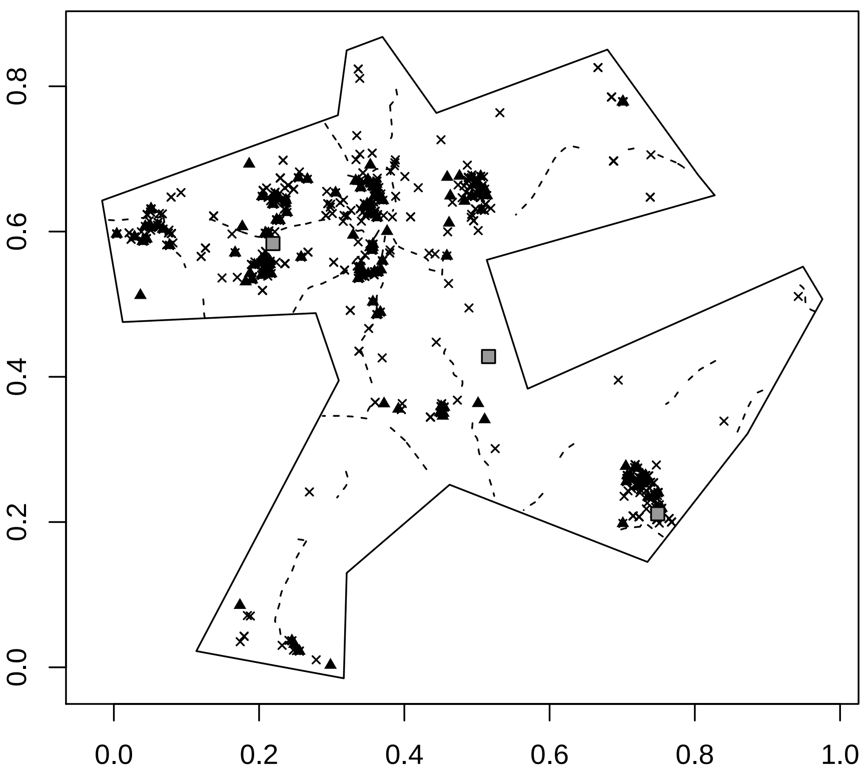
As an example, let's look at a case-control asthma data set from North Derbyshire UK created by Peter J Diggle. [3] The exposure of interest is proximity to main roads and industrial pollution sources. The data are in shapefile format and contain files for cases and controls, boundaries, pollution sources, and roads. The files were downloaded from Dr. Diggle's website and saved. The following code uses the *readOGR()* in *rgdal* to create sp objects.

```
> install.packages("rgdal")
> library(rgdal)
> spasthma <- readOGR("/Users/charlie/Dropbox/asdarHowTo/asdarExamples/Ch7_pointSource/ast
> spbdry <- readOGR("/Users/charlie/Dropbox/asdarHowTo/asdarExamples/Ch7_pointSource/asthma
> spsrc <- readOGR("/Users/charlie/Dropbox/asdarHowTo/asdarExamples/Ch7_pointSource/asthma,
> sproads <- readOGR("/Users/charlie/Dropbox/asdarHowTo/asdarExamples/Ch7_pointSource/asthn
```

Now, let's plot the data:

[3] Dr. Diggle is one of the founding fathers of spatial point data analysis. His books are short, elegant and highly recommended.

```
> library(sp)
> plot(spbdry, axes=TRUE)
> plot(sproads, add=TRUE, lty=2)
> plot(spasthma, add=TRUE, pch=c(4,17)[(spasthma$Asthma == "case") + 1], cex=c(0.6, 0.75)[
> plot(spsrc, pch=22, add=TRUE, cex=1.2, bg="grey60")
```



10.4 Analyzing Point Process Data

As mentioned, our two main issues of interest are: the distribution of events in space and the existence of possible interactions between them.

10.4.0.1 First-order properties and spatial randomness

First-order spatial properties measure the *distribution* of events in the study region. They do not, though, tell us anything *about* any relationship. We can measure the statistical *density* of a spatial process in some region, which can be thought of as similar to a univariate density. Or, we can measure the so-called spatial *intensity* of spatial points, which is normalized by an expected count, and can be thought of as a kind of "rate".

The probability model most often applied to spatial processes is the Poisson distribution. Spatial analysts distinguish between homogeneous and inhomogeneous Poisson processes with the distinction being that in a homogenous Poisson process, there is a single rate or intensity λ, while in an inhomogeneous process the intensity varies across regions.

In a homogeneous Poisson process, the spatial intensity is independently distributed and the presence of one point has no effect on the probability of the presence of a nearby point. A simple unbiased estimator of λ is n/A or the number of events in an area or region. A homogeneous Poisson process is, essentially, complete spatial randomness.

Complete spatial randomness is not likely to arise in any process that's of interest to epidemiologists. By contrast, an *inhomogeneous* Poisson process can be though of as a generalization of a homogeneous process. The intensity is represented by a generic function for λ, called the *kernel* that allows it to vary spatially.This spatial variation can be diverse, with events appearing more likely in some areas than others.

For completeness sake, we should distinguish between kernel *density* estimates (the risk or probability of events in an area) and kernel **intensity** estimates (number or rate of events in an area). Densities should integrate to 1, intensities integrate to the overall mean number of events in an area.

There are a number of ways of estimating the kernal function. One proposed by Diggle is

$$\lambda(x) = \frac{1}{h^2}\sum_{i=1}^{n} \kappa \frac{||x - x_i||}{h} / q(||x||) \tag{10.1}$$

Where $q(||x||)$ is a border correction term and κ is the kernel function. The term h is called the "bandwidth". It can be thought of as a weighting or smoothing factor. We met just such a smoothing factor in the density plot of the Swedish pines in the introduction to this chapter. Small values of h result in spiky estimates and large values of h result in smoother functions. You can think of it in terms of physically mapping something. Say you use clay discs to represent events on your tabletop map. If you make those clay discs small, you will get spikey "peaks" as events pile up on each other. If you make the discs wider, you will get less pronounced more undulating valleys.

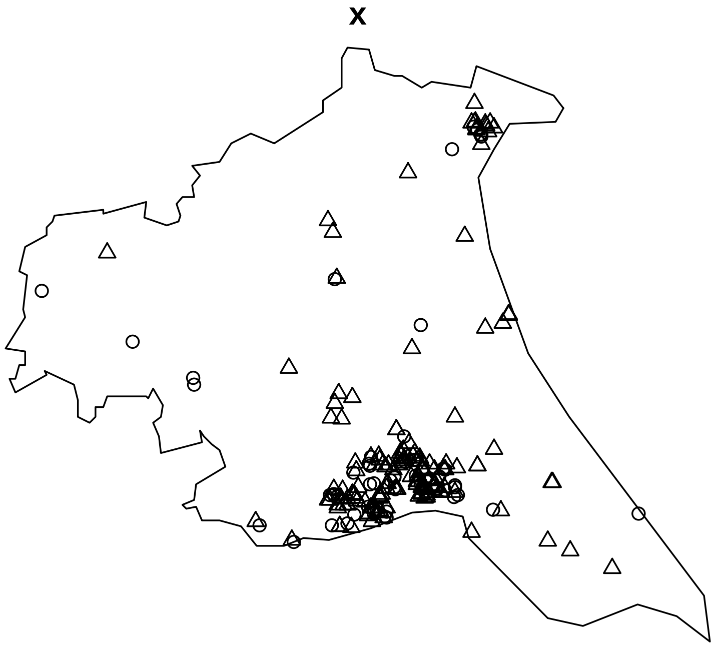
We can demonstrate this with a couple of perspective graphs of childhood leukemia data. We load the data, and take a quick look at it.

```
> library(spatstat)
> data(humberside)
> x<-humberside
> summary(x)
```

```
Marked planar point pattern: 203 points
Average intensity 0.000993 points per square unit (one unit = 100 metres)

*Pattern contains duplicated points*
Multitype:
        frequency proportion intensity
case           62      0.305  0.000303
control       141      0.695  0.000690

Window: polygonal boundary
single connected closed polygon with 102 vertices
enclosing rectangle: [4690, 5411]x[4150, 4758]units
Window area =  204487 square units
Unit of length: 100 metres
```

```
> plot(x)
```

```
   case control
      1       2
```



Next we compare a bandwidth of 10 to one of 20.

```
> x1<-density(x,10)
> persp(x1)
> x2<-density(x,20)
> persp(x2)
```

Clearly, bandwidth plays an important role in our graphical impression of these data. The issue becomes even more crucial when we compare cases to controls, again with a bandwidth of 10 vs. one of 20.

```
> plot(density(split(x),10))
> plot(density(split(x),20))
```

In fact, different kernel functions will produce similar estimates for equivalent bandwidths, but the same kernel with different bandwidths may produce different results. Despite its importance, there is, unfortunately, no strong statistical guidance on the choice of bandwidth. One approach is to try a few based on the results of using the mean square error in the data. The *mse2d()* function in the *splancs* can help you do this.

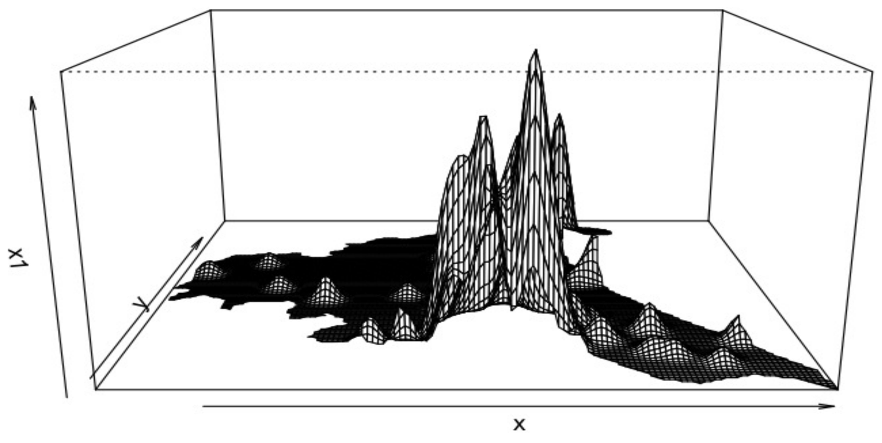


Fig. 10.5 caption

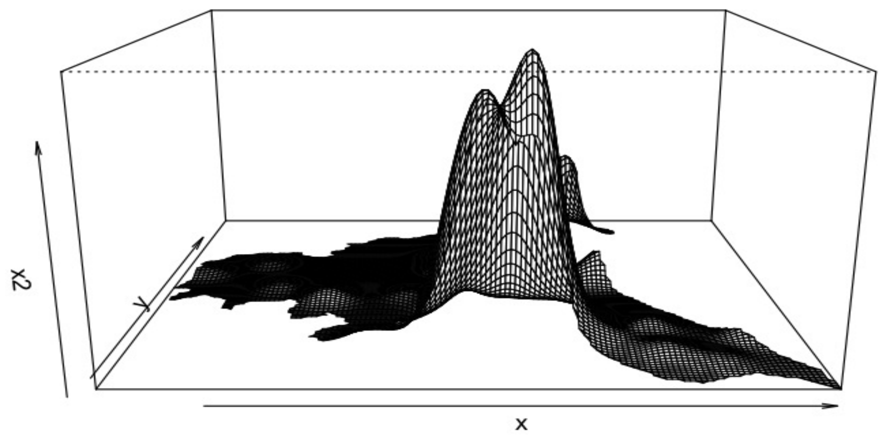


Fig. 10.6 caption

10.4.0.2 Assessing Spatial Randomness

You can begin by just looking at or "eyeballing" the data. Does there appear to be clustering or regular (non-random) patterning? If so, we can test our subjective assessment formally. We generally do this with "nearest neighbor" distances.

The '*G function* tests the role of distance to the nearest neighbor by summarizing the distribution of these distances. For each point, we calculate the distance to the nearest neighbor. We then count the number of points that are less than or equal to a set of distances, *r* and divide each of these by the total number of points:

$$G(r) = \frac{d_i : d_i \leq r, \forall i}{n} \tag{10.2}$$

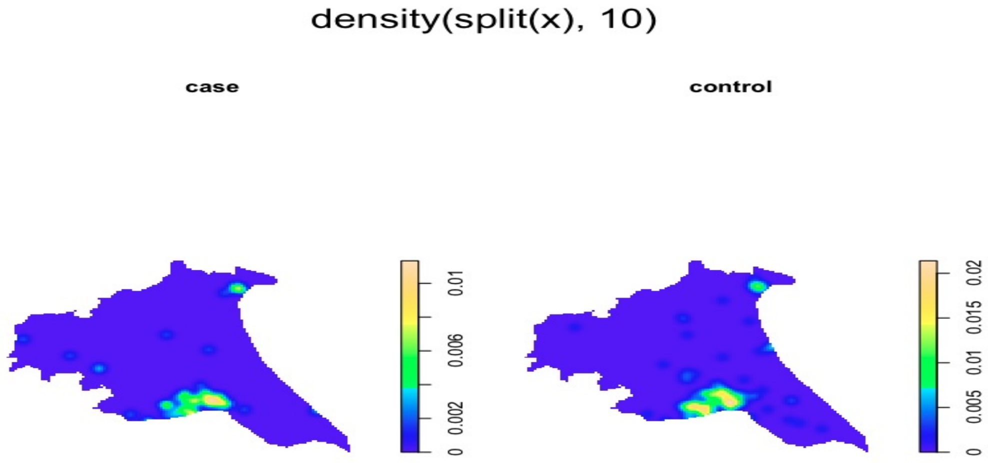


Fig. 10.7 caption

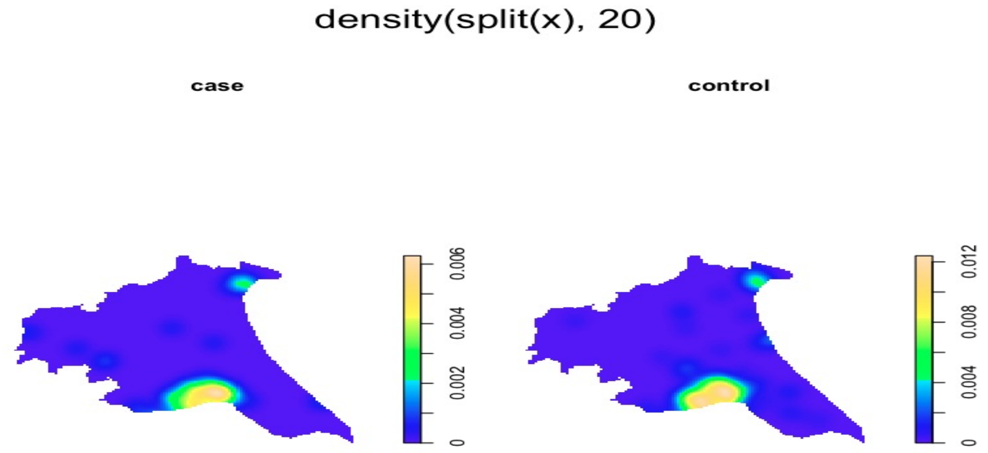


Fig. 10.8 caption

We plot this against the expected number under the assumption of complete spatial randomness:

$$G(r) = 1 - e^{\lambda \pi r^2} \tag{10.3}$$

If there is no clustering, we should get a straight diagonal line [4]. Bowing of the line above the envelope indicates clustering. Bowing below the envelope, indicates a regular pattern. We can get 95% *envelopes* (think confidence intervals...) from MCMC samples by using the *envelope()* function. We can pass the data to the *Gest()* function, or we can invoke "Gest" as an option for the *envelope()* function. The G function is a first-order spatial function, that tells us nothing *about* the relationship between the points, just that there is one. Another first order function available in spatstat, the *F function*, is much like the G function, but is based on sampled points and tends to return smoother plots.

[4] Like a probability plot in simple statistics

10.4.1 Second-order properties

Second-order spatial statistical properties involve information on the relationship or interaction between points, such as the probability of two points being in the same vicinity.

The *K function* measures the number of events found up to a given distance form any particular event point. It is defined as:

$$K(s) = \lambda^{-1} E(N_0(s)) \tag{10.4}$$

where E is the expectation, and N the number of events up to distance s

an unbiased estimate is:

$$K(s) = (n(n1))^1 |A| \sum w_{ij}^1 ij |\{x_j : d(x_i, x_j) \leq s\}| \tag{10.5}$$

wherew_{ij} are weights equal to the proportion of the area inside the region A of the circle centred at x_i and radius $d(xi, xj)$ is the the distance between x_i and x_j.

The K function for an homogenous Poisson process is $K(s) = \pi s^2$. Values higher than πs^2. indicate clustering. Smaller values indicate a regular pattern. We can get a K function value from the spatstat package. If you recall, we invoked the Kest function as part of our call to he *envelope()* function in the introduction. Here is an example of the syntax where we use an sp object "on the fly".

```
>  Kfunct <- envelope(as(spObject, "ppp"), fun = Kest,
+ r = r, nrank = 2, nsim = 99)
```

10.5 Point Processes Analysis of Epidemiological Data

The distribution of an epidemiological point process reflects the underlying distribution of the population and those factors related to the disease outcomes which depend on the subjects, e.g. age, gender. Of critical importance is the distribution of the population at risk. In spatial analysis, this is usually accomplished through selection of controls, which are most often a random sample of the population at risk or cases of another, presumably unrelated, disease.

Keep in mind is that spatial point analysis addresses a *continuous* distribution of events in a spatial plane. As epidemiologists, we may be more used to working with categorical data. The relative risk estimate is the ratio of the *intensity* of the cases to controls $\rho(x) = \lambda_1(x)/\lambda_0(x)$. Such analyses are often conducted on the log scale, so under the null, $\rho(x) = 0$ The intensity of controls may be zero at some points, so you'll see kernel smoothing techniques with an (some) appropriate choice of bandwidth to avoid 0 values.

As we have seen, significance testing may be accomplished with Monte Carlo methods. this involved re-labeling cases as controls k times, assuming it should make no difference if there is no increased or decreased risk, calculating the relative risk based on these iterations and comparing the results of the simulated relative risk to the observed relative risk.

10.5.1 Example: Spatial Case Control Study of Asthma

We return to the Diggle asthma data we read in and mapped above to calculate the risk ratio for asthma given proximity to roads and industrial pollutant sites.

The general approach will be:

1. Use *Sobj_SpatialGrid()* in maptools to create a spatial grid that will hold the points
2. Divide point locations between cases and controls.
3. Use the cv.R, a function written by Roger Bivand, to help choose a bandwidth
4. Use *spkernel2d()* from the splancs package to calculate the intensity (λ) of cases and controls
5. Construct a SpatialGridDataFrame object to hold the case and control points and coerce that object to a SpatialPixelsDataFrame dropping any missing cells.
6. Calculate the relative risk. [5]
7. Calculate a p value for the relative risk, using Monte Carlo methods.
8. Plot the risk ratio kernels.

This first bit of code creates a grid from the spdry boundary object, defines the grid as a spatial object "slot", and assigns or "unpacks" the coordinates for the study area. We'll the resulting object to calculate the risk ratio, and define cases and controls based on the spasthma attribute table columns.

```
> library(maptools)
> sG <- Sobj_SpatialGrid(spbdry, maxDim=50)$SG
> gt <- slot(sG, "grid")
> pbdry <- slot(slot(slot(spbdry, "polygons")[[1]], "Polygons")[[1]], "coords")
```

This next bit of code defines and enumerates the cases and controls.

```
> cases<-spasthma[spasthma$Asthma=="case",]
> ncases<-nrow(cases)
> controls<-spasthma[spasthma$Asthma=="control",]
> ncontrols<-nrow(controls)
```

Our next task is to define a bandwidth. Dr. Bivand has written R code called cv.R that implements an approach suggested by Kelsall and Diggle[6] that requires additional libraries and packages. The method returns a recommended bandwidth of 0.275, but Bivand feels subjectively that the value is too high given the scale of the data and opts instead for a bandwidth of 0.125. I am going to print, but not run this code.

```
> #SELECT BANDWIDTH
> library(spatialkernel)
> library(splancs)
> install.packages("cubature")
> library(cubature)
> # Bivand .R function to choose bandwidth
> source("/Users/charlie/Dropbox/asdarHowTo/asdarExamples/Ch7_pointSource/cv.R")
> h<-seq(0.20, .4, by=.025)
> cvres<-cvkernratio(rbind(coordinates(cases), coordinates(controls)), ncases, ncontrols,
> plot(h, cvres[[2]])
> bwasthma <- .125 #.275
```

We next use *spkernel2d()* to calculate the intensity of cases and controls in the study area. This procedure returns a lot of missing values for grid cells outside the study area. We construct a SpatialGridDataFrame object and coerce it to a SpatialPixelDataFrame and drop the missing cells. We then use the SpatialPixelDataFrame object to calculate the risk ratio.

```
> library(splancs)
> bwasthma <- .125
> #Calculate Intensity of Cases and Controls
> kcases<-spkernel2d(cases, pbdry, h0=bwasthma, gt)
```

[5] See the end of Chapter 7 in ASDAR to see a binary regression approach to spatial point analysis that controls for confounders

[6] see ASDAR page 174

```
> kcontrols<-spkernel2d(controls, pbdry, h0=bwasthma, gt)
> #Remove Cells With Missing Values
> df0 <- data.frame(kcases=kcases, kcontrols=kcontrols)
> splancs_ge_24 <- packageDescription("splancs")$Version > "2.01-23"
> if (!splancs_ge_24) idxna <- complete.cases(df0)
> spkratio0 <- SpatialGridDataFrame(gt, data=df0)
> spkratio <- as(spkratio0, "SpatialPixelsDataFrame")
> #Calculate the Risk Ratio
> spkratio$kratio <- spkratio$kcases/spkratio$kcontrols
> #set non-finite values from division by zero to missing
> is.na(spkratio$kratio) <- !is.finite(spkratio$kratio)
> #transform to log scale, print out and summarize results
> spkratio$logratio <- log(spkratio$kratio)-log(ncases/ncontrols)
> spkratio[1:5,]
```

```
Object of class SpatialPixelsDataFrame
Object of class SpatialPixels
Grid topology:
  cellcentre.offset   cellsize cells.dim
x      -0.006287757 0.01983898        50
y      -0.005192132 0.01983898        45
SpatialPoints:
             x        y
[1,] 0.3309749 0.847884
[2,] 0.3508139 0.847884
[3,] 0.3706529 0.847884
[4,] 0.3309749 0.828045
[5,] 0.3508139 0.828045
Coordinate Reference System (CRS) arguments: NA

Data summary:
     kcases         kcontrols         kratio           logratio
 Min.   : 1.898   Min.   :167.8   Min.   :0.01114   Min.   :-2.887
 1st Qu.: 2.009   1st Qu.:170.4   1st Qu.:0.01197   1st Qu.:-2.815
 Median : 2.199   Median :174.7   Median :0.01259   Median :-2.765
 Mean   : 7.944   Mean   :197.8   Mean   :0.03539   Mean   :-2.109
 3rd Qu.:16.230   3rd Qu.:230.7   3rd Qu.:0.07036   3rd Qu.:-1.044
 Max.   :17.384   Max.   :245.1   Max.   :0.07091   Max.   :-1.036
```

```
> summary(spkratio$kcontrols)
```

```
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
   8.352  447.100 1627.000 2299.000 3743.000 8818.000
```

Now we assess the significance of this finding.

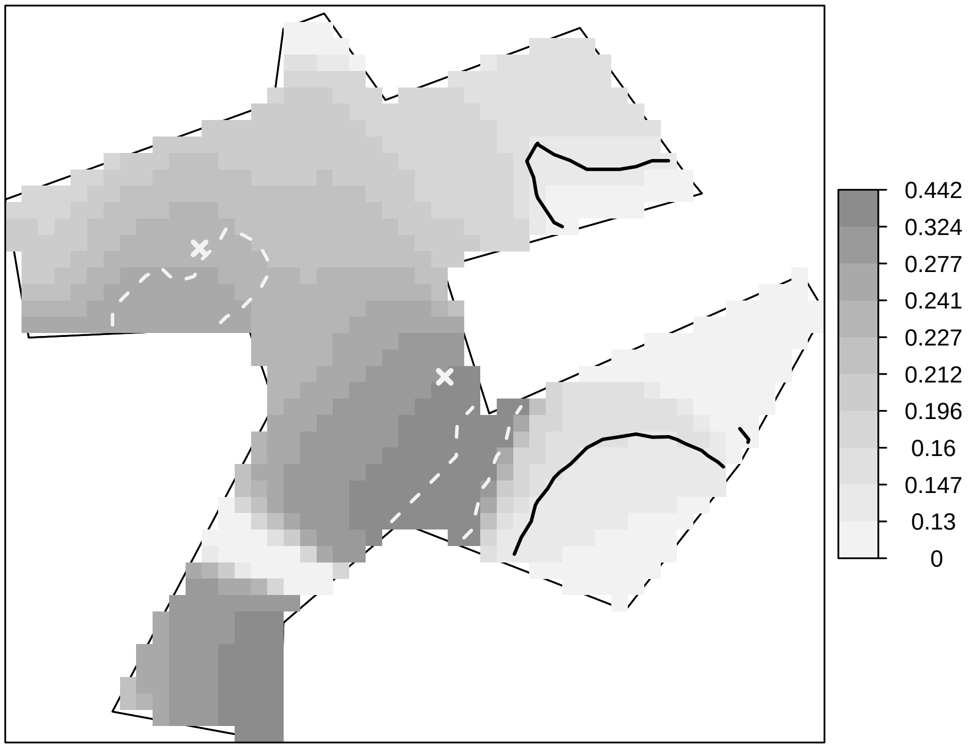
```
> #overlay boundary file on grid file
> if (splancs_ge_24) {
+   idxinbdry <- overlay(sG, spbdry)
+   idxna <- !is.na(idxinbdry)
+ }
> #create probability map
> set.seed(1234)
> niter <- 99
> ratio <- rep(NA, niter)
> pvaluemap <- rep(0, sum(idxna))
> rlabelratio <- matrix(NA, nrow=niter, ncol=sum(idxna))
> #calculate p values
> set.seed(1)
> for(i in 1:niter)
+ {
+   idxrel <- sample(spasthma$Asthma) == "case"
+   casesrel <- spasthma[idxrel,]
+         controlsrel <- spasthma[!idxrel,]
+
+         kcasesrel <- spkernel2d(casesrel, pbdry, h0=bwasthma, gt)
+         kcontrolsrel <- spkernel2d(controlsrel, pbdry, h0=bwasthma, gt)
+         kratiorel <- kcasesrel[idxna]/kcontrolsrel[idxna]
+         is.na(kratiorel) <- !is.finite(kratiorel)
+         rlabelratio[i,] <- kratiorel
+
+         pvaluemap <- pvaluemap + (spkratio$kratio < kratiorel)
+ }
> # calculate p value
> idxna2 <- apply(rlabelratio, 2, function(x) all(is.finite(x)))
> rhomean <- apply(rlabelratio[, idxna2], 2, mean)
> c <- prod(slot(gt, "cellsize"))
> ratiorho <- c*sum((spkratio$kratio[idxna2]-ncases/ncontrols)^2)
> ratio <- c*apply(rlabelratio[,idxna2], 1,
+  function(X, rho0 ){sum((X-rho0)^2)}, rho0=ncases/ncontrols
+ )
> pvaluerho <- (sum(ratio > ratiorho)+1)/(niter+1)
> pvaluerho
```

```
[1] 0.69
```

The p-value indicates any association between location and asthma is due to chance.

Finally, we map our results.

```
> # create kernel intensity map
> spkratio$pvaluemap <- (pvaluemap+1)/(niter+1)
> imgpvalue <- as.image.SpatialGridDataFrame(spkratio["pvaluemap"])
> clpvalue <- contourLines(imgpvalue, levels=c(0,.05, .95, 1))
> cl <- ContourLines2SLDF(clpvalue)
> cl05 <- cl[cl$level == "0.05",]
> xzx <- slot(slot(cl05, "lines")[[1]], "Lines")
> cl05a <- SpatialLines(list(Lines(xzx, ID="0.05")))
> lyt05 <- list("sp.lines", cl05a, lwd=2, lty=2, col="grey95")
> lyt95 <- list("sp.lines", cl[cl$level == "0.95",], lwd=2, lty=1)
> lytb <- list("sp.polygons", spbdry)
> lytp <- list("sp.points", spsrc, cex=0.9, pch=4, col="grey95", lwd=3)
> brks <- quantile(spkratio$kratio[spkratio$kratio>0], seq(0,1,1/10), na.rm=TRUE)
> brks[1] <- 0
> lbrks <- formatC(brks, 3, 6, "g", " ")
> cols <- colorRampPalette(grey.colors(5, 0.95, 0.55, 2.2))(length(brks)-1)
> colorkey<-list(labels=lbrks,
+   at=(0:10)/10, height=.5)
> print(spplot(spkratio, "kratio",
+    col.regions=cols,
+    do.log=TRUE,
+    colorkey=colorkey,
+     at=c(0, brks[-c(1,11)], max(spkratio$kratio, na.rm=TRUE)),
+     sp.layout=list(lyt05, lyt95, lytb, lytp)
+ ))
>
```



10.6 A Note About Cluster Detection

Cluster detection is an often difficult and complex task. Here, I give the merest introduction. I won't work through any vignettes, but I will provide sample code that you might adapt to your needs.

Cluster analysis aims to detect and locate areas where the risk of disease is unusually high. The package *DCluster* uses models and bootstrap approaches to compute the significance of observed values. The general approach involves resampling the observed number of cases in each area and re-computing the value of the test statistic for each of the simulated data sets. A p-value is computed by ranking the observed value of the test statistic vs. values from the simulations.

The default assumption is that O_i is Poisson distributed with mean θE_i, conditioned on the total number of cases, so that the distribution of $(\theta_1 \ldots . \theta_n)$ is Multinomial. Available options include non-parametric, bootstrap, Poisson (not conditioned on the observed total), and a Negative Binomial distribution.

10.6.1 Global Tests

The first step is a global chi square for heterogeneity of SMRs:

```
chtest <- achisq.test(Observed ~ offset(log(Expected)), as(nc, "data.frame"), "multinom", 999)
chtest
```

To help choose the correct model (such as a negative binomial), we next test for over dispersion:

```
pwtest <- pottwhitt.test(Observed ~ offset(log(Expected)), + as(nc, "data.frame"), "multinom", 999)
```

We can then apply Moran's I statistic to our set of SMRs to assess the spatial distribution of the population at risk:

```
col.W <- nb2listw(ncCR85, zero.policy = TRUE)
moranI.test(Observed ~ offset(log(Expected)), as(nc, "data.frame"),
     "negbin", 999, listw = col.W, n = length(ncCR85), S0 = Szero(col.W))
```

10.6.2 Detection of the Location of a Cluster: Scan Statistics

Once clustering is established, scan statistics are used to determine its location. Scan statistics can be conceptualized as a moving window that covers only a few areas each time and for which a test of clustering is carried out locally repeating procedure throughout study area. Scan methods differ in the way windows are defined, how it is moved over the area, and how the local test of clustering is done.

In my limited experience of using scan statistics, I've generally used Martin Kuldorf's SatScan program. This is not an R tool, bur rather a self-contained program. If your work calls on you to conduct cluster analysis, it is probably the first program to which you should turn.

Kulldorff's Statistic is based on a variably-sized window. It considers only the most likely cluster around a given region. The over-all relative risk in the regions inside the window is compared to that of the regions outside the window. The null hypothesis of no difference is resolved with a likelihood ratio test. An advantage of this approach is that the most likely cluster window is that with the highest value of the likelihood ratio test. Also, there is no need to correct the p-value because the simulations for different centroids are independent. If there is more than one likely cluster, that with the lowest p-value is considered the primary cluster. Secondary clusters, are those that do not overlap primary clusters.

Chapter 11

Appendix: How to Use GRASS

11.1 Introduction

Back in chapter 2, I introduced GRASS as an open-source GIS. I very briefly described how you can connect to this powerful program through R. I want to spend a little more time here giving you a better sense of how GRASS works. [1] This material is taken from "Open Source GIS: A GRASS Approach" by Markus Neteler and Helena Mitasova, available from Springer Texts. These notes are somewhat sketchy at present, but I hope to add to it in the future, so you can have a fuller reference should you need one.

11.2 GRASS Basics

11.2.1 Files

In GRASS-speak:

- a LOCATION is a subdirectory of your /grassdata directory that contains maps and data for a particular project
- MAPSETS are subdirectories of a location that contain files for an individual team member [2]
- PERMANENT is a special default mapset that is created for every location. It contains default mapsets, boundaries and coordinate system definitions. The permanent mapset is available to all users, but it can't be overwritten or manipulated. You would have to copy it into your personal mapset to change it.
- WIND is the file that contains the current boundary coordinates. DEFAULT_WIND is the default boundary file found in the permanent mapset.

11.2.2 Commands

In GRASS, it seems, GUI's come and go, but the command line interface is forever. So we will concentrate on working from syntax.

GRASS commands invoke *modules* and are prefaced by a letter keyed to a suite of commands:

[1] There is a (purportedly) more -friendly version called QGIS (Quantum GIS) but I haven't played with it yet.

[2] This is clearly unimportant if you work alone on your own machine, but serves as a way to organize sets of files for a particular project

- d.* - display something
- v.* - vector commands
- r.* - raster commands
- db.* - work with databases
- g.* - general and file-related commands
- r3.* - work with 3D rasters
- m.* - miscellaneous commands

The general form of a GRASS command is:

> *module name* (e.g. d.rast to display a raster map) *flags* (e.g. -c to display attribute column information for a database) *parameters* (like input or output file names) *options* (e.g. –o, –q, –v for overwrite, quiet and verbose)

Flags and parameters are white-space delimited, so don't put spaces after things like equal signs, e.g

```
map=soils   NOT  map = soils
```

You can abbreviate parameter names as long as they are distinguishable, e.g. out instead of output.

Some immediately useful GRASS commands:

- g.manual - displays the GRASS user manual
- g.manual index - display the user manual index
- g.manual d.rast - display the user manual help pages on the d.rast command
- help - after a command returns a list of available options, e.g *d.vect help*
- g.list rast / g.list vect - list all the available raster or vector maps in a location
- r.info / v.info - get information on a raster or vector file, add the -c flag to get attribute table column information

11.2.3 Running GRASS from GNU emacs

You can run GRASS from a emacs terminal session [3] by first typing *M-x shell* to start the terminal, switching to your home directory with *cd home* (where "home" is the name of your directory) and then invoking the application e.g. */Applications/GRASS-6.4.app/Contents/MacOS/grass.sh*

You can also run GRASS from within ess, but it takes a little work around because there is no dedicated GRASS mode in ess. Instead, you can take advantage of the ess R mode. [4]

Start emacs as you normally would. Split the screen into two panels with C-2. Open or write a .R file in one panel (C-f or C-w). Open R in the other panel (M-x R). Use R's system() command to invoke GRASS, e.g. *system("/Applications/GRASS-6.4.app/Contents/MacOS/grass.sh")*. Now you can sent lines of GRASS code from within emacs/ess using *C-c n* to submit the current line of code.

You can also start R from within this GRASS session by simply typing R at the GRASS prompt. You can then use Roger Bivand's wonderful spgrass6 library to bounce data and map files back and forth between R and GRASS. [5]

Some things don't work well with this set up. C-c r which in ess usually sends a selected region of code to R will, instead freeze up the system. If you make this mistake, use C-g to quit and unfreeze. Some GRASS modules, notable

[3] You can, of course, just work directly in the terminal.

[4] This trick comes from Adam Wilson

[5] Yes. You are running R within GRASS which is already running within R. Seems to work. Go figure

v.digit, used to digitize maps needs a mouse/cursor-based interface. Sweave doesn't like the spaces in GRASS command and will issue vaguely ominous fatal error messages when you try to weave a document with GRASS commands in it. You can get around this last, bit using verbatim environments for GRASS commands in your .Snw documents. And you have to exit out of GRASS before Sweave will run at all.

Otherwise, this approach seems to work surprisingly well.

11.2.4 displaying maps

Let's display some of the North Carolina tutorial maps.

We begin by opening emacs and using C-w to write a new Sweave file that ends with a .Snw extension. Split the screen with C-2 and open up an R session with M-x R. Now, in R we uses a system command (using C-n to submit command lines) to start up GRASS: [6]

```
> system("/Applications/GRASS-6.4.app/Contents/MacOS/grass.sh")
```

The GRASS python GUI should start up. Choose the nc_spm_08 location and the user1 mapset. You should now see a GRASS prompt in your R window. Try the following syntax: [7]

```
  g.region rast=elevation -p  # get some information about the elevation raster map
  d.mon x0 # open a monitor
  d.erase # prepare to open a monitor by erasing any existing maps from previous sessions
  d.rast elevation # display the elevation map as a 2D raster
  g.manual d.his # get some info on the d.his module
  d.his h=elevation i=elevation_shade
  d.vect streams col=blue #display vector map of streams
  d.vect roadsmajor #display major roads
  d.vect overpasses icon=extra/bridge size=20 fcol=blue
  d.mon x1 #open another monitor
  d.rast geology_30m #display geology map in the new monitor
  d.mon select=x0 #switch back to the other monitor
  d.rast aspect
  d.mon select=x1 #switch to the geology monitor
  d.barscale at=0,0 #add a barscale to the geology map
```

For a bit of fun, let's open up GRASS's 3D viewer:

```
  nviz elevation vect=streams,roadsmajor point=overpasses
```

Play around with the image by using your mouse to manipulate the green handle on the left side of the screen. Use your mouse with the file command to quit out of the nviz viewer.

Exit GRASS by typing “exit” at the prompt.

11.2.5 File Management

Do not manipulate (copy, rename etc...) files outside of GRASS. There are a lot of associated files and you will break the links among them. Use the g.* set of commands for general file management.

[6] More generally, you will open and run GRASS from a command prompt in a terminal session, but I use GNU emacs as a way to integrate R and latex in Sweave, and this approach works for me

[7] If you are working from the Sweave version of this document, I had to use verbatim for the GRASS commands I was running in R. Because they use spaces to delimit, which R doesn't, they return an error in the Sweave process. replace the beginverbatim with «»= and the endverbatim with @ and the commands should work

Some useful file management commands:

- g.copy - e.g. *g.copy vect=railroads@PERMANENT,myrailroads* will copy the railroad map from the permanent mapset to your current user mapset
- g.rename - e.g. *g.rename vect=myrailroads,railnetwork*
- g.remove - e.g. *g.remove vect=myrailroads*
- g.mapset - to add a new mapset, e.g *g.mapset -c mapset=user2*
- g.mapsets - to add a mapset to your search path, e.g. *g.mapsets addmapset=user2*
- g.proj -p - to get current coordinate system parameters

11.3 Setting Coordinates

Geospatial files have to be defined in terms of their coordinate systems, map projections and geodetics or map datums, of which there are various types.

11.3.1 Defining a new location by importing a georeferenced shape file

The easiest way to define a new location correctly is to use the “Location wizard” under “Define new location” on the opening panel to import a shape (.shp) file which contains geographic definitions. You can then use “Create mapset” on the panel to create a mapset within the new location.

GRASS uses the GDAL/OGR Geospatial Data Abstraction Library (referred to as GDAL, and pronounced *“goodal”* to import and convert geospatial files.

You can also import a shape file and create a new location while GRASS is running by clicking *File - Import vector data*

You can also import raster files with *r.in.gdal* and export raster files with *r.out.gdal*. the vector equivalents are v.in.gdal and v.out.gdal.

11.3.2 Other ways to define a new location

You can define a location by its European Petroleum Survey Group (EPSG) code if you know it or with projection values by entering latitude and longitude, datum etc... directly. Both of these options are (I believe) available under the New location wizard on the opening panel. See Neteler pp 40-44. You can also use the projection values approach to define a non-georeferenced map using x,y data.

11.3.3 Transforming coordinate systems

GRASS uses the PROJ4 command *cs2cs* to reproject map files. For example:

```
cs2cs -v +proj=latlong +to +proj=utm +zone=17 +ellps=WGS84 \ +datum=WGS84 +units=m}
```

11.3.3.1 Transforming coordinate systems in R

In R, you can use spTransform() in the rgdal package with EPSG codes to transform coordinate systems:

```
library(rgdal)

data<-read.table ("mydata.txt", h=T, sep=",") #your original dataset
coordinates(data)<- X + Y # where X and Y stand for the name of your lat/lon columns
proj4string(data)<-CRS("+init=epsg:4326") #if your coordinates are in WGS84
data.proj<-spTransfrom(data, CRS("+init=epsg:the.correct.epsg.number.of.your.projection")
```

In another approach, [8] if you have a data frame named with latitude and longitude variables, you can convert it to a spatial data frame with the following R code. First, you use the *coordinates()* function on the latitude and longitude variables. Then you specify the coordinate system, replacing the text in quotes with the proj4 string for your data.

```
coordinates(dataFrame) <- ~ longitude + latitude
proj4string(dataFrame) <- CRS("+proj=latlong +datum=NAD83")
```

11.3.4 The State Plane Coordinate System (SPCS)

This coordinate system divides the US and Puerto Rico into about 120 numbered sections, referred to as zones. Each zone has a code that defines its projection parameters. It is used extensively by government agencies in the US and you are likely to come across it.

SPCS uses three projections, depending on the geography of the state: the Lambert Conformal Conic for states that are longer east-west, Transverse Mercator for states that are longer north-south, and the Oblique Mercator for the panhandle of Alaska which is angled.

The zones were originally based on a set of geodetics or control points known as the North American Datum of 1927 (NAD27). These were updated in 1983 to line them up with satellite data, creating NAD83. In general, ZONE refers to NAD27 and FIPSZone refers to NAD83.

You can find a link to information on zones at: where you will find the following information about our very own New York metropolitan area: [9]

```
NEW YORK LONG ISLAND ZONE FIPSZONE: 3104 ADSZONE: 4976 UTM ZONE: 18
BRONX, KINGS, NASSAU, NEW YORK, QUEENS, RICHMOND, SUFFOLK
```

There is a helpful online discussion on transforming x,y coordinates to SPCS which recommends this site to get the EPSG codes for SPCS zones.

Speaking of EPSG codes for SPCS, searching for “New York” returns a number of possibilities. First, EPSG 32118 (to which Roger Bivand refers to in his text) has units in meters and is not helpful with NYC government map files which have units in feet. So first off, search for an EPSG that specified units in feet. Even then, there are a couple of options.

You will see references to HARN (High Accuracy Reference Network) versions ("a statewide or regional upgrade in accuracy of NAD 83 coordinates using Global Positioning System (GPS) observations", and to the NSRS2007 version (another update to "Resolve inconsistencies between each stateâĂŹs statewide HARN adjustments").

The two main choices are epsg: 2263 and the HARN version epsg: 2908.For my work with New York City data, I eventually chose EPSG:2908: NAD83(HARN) / New York Long Island (ftUS). I decided against the NSRS version because much of data is pre-2007.

[8] Thanks to Jonathan Bearak

[9] According to Matt Roe at the New York City Department of Transportation, the NYC uses SPCS zone 18.

11.3.5 Regions and resolutions

GRASS allows you to set the part of a map with which you want to work (region) and the level of detail you want to see (resolution). Here are some commands to help in that regard:

- region information commands
  - g.region -pec returns the current region, spatial extent and resolution
  - g.proj -p returns the current measurement units
- setting regions
  - g.region n=228000 s=215500 w=632000 e=640000 -p defines the coordinates for northern, southern, eastern and western edges
  - g.region n=s+500 e=w+500 -p creates a small 500m by 500m subregion in south-west corner of current region
  - g.region rast=elevation -p adjust the curent region according to the raster map elevation
- saving a region
  - g.region res=15 save=myregion -p

11.4 Vector Data

11.4.1 Vector File Structure

Vector-related files consist of:

- a category index file that links the vector object IDs to the attribute table rows
- a geometry file of coordinates of graphic elements
- a topology that file describes the spatial relationships between the map's graphic elements

GRASS takes a topological approach to vector maps; adjacent points and lines are stored once and shared (vs. non topological or spaghetti data). Level 1 of GRASS vectors is usually non- topological.

11.4.1.1 Vector DBMS

Vector graphical elements are linked to a descriptive (DBMS) table by category number (attribute ID, usually the âĂIJcatâĂİ integer column). You print or maintain the database with *v.category*. Vector objects can be linked to more tha 1 table, each of which represents a vector data âĂIJlayerâĂİ. Layers are listed or maintained using *v.db.connect*. Use *db.** commands, which operate on the vector map, for basic SQL.

11.4.2 Importing vector maps

Use *v.in.ogr*, eg:

```
# get the coordinates of the study region
g.region swwake_10m -p
# import the NC state soil data associations from a SHAPE file
# clipped to the current region (W,S,E,N)
v.in.ogr gslnc.shp out=soils_nc \
spatial=630000,215000,645000,228500
# check the imported data, there will be only two associations
v.info soils_nc
v.info -c soils_nc
d.vect soils_nc
```

You can generate a vector area map of a rectangle defining the current region, which you can then use to clip data:

```
v.in.region out=myregion
d.vect myregion
```

SHAPE files that were not generated in a topological GIS may need to be cleaned up with snap parameters, v.clean, or v.digit tool. ASCII format vectors can be imported using *v.in.ascii*. You can use *v.in.garmin* or *v.in.gpsbabel* to import GPS data. You can import data directly from web sources using Open Geospatial Consortium's Web Feature Service (OGC WFS) compliant Web servers with *v.in.wfs*

11.4.2.1 Exporting vector data

Use *v.out.ogr* to export vector data, and *v.out.ascii* to export a GRASS vector map to GRASS ASCII format.

11.5 Viewing vector maps

Here, we set the region to the swwake_10m map (and use -p to print the parameters of this region), start up a monitor, erase any left over maps on that monitor (good practice), and then display the streams, streets and overpasses. We then do some zooming with d.zoom, and

```
system("/Applications/GRASS-6.4.app/Contents/MacOS/grass.sh")
g.region region=swwake_10m -p
d.mon x0
d.erase
d.vect streams col=blue
d.vect streets_wake
# display symbols
d.vect overpasses icon=extra/bridge size=15 fcol=red

d.zoom #to zoom

d.mon x1
# display areas with centroids (sized 2 pixels), -c for random colors
d.vect -c soils_wake size=2
# display areas without borders and centroids
d.vect -c soils_wake type=area
```

Note that you can do the same things with the python gui interface "GRASS GIS Layer Manager" and the "GRASS GIS Map Display" which both open up when you start GRASS. The Layer Manager icon that looks like a "V" or a broken line, opens up an interactive window to select vector maps to display. Then there are icons on the Map Display window (like the magnifying glass with the plus sign for zooming) that allows you to interact with the map. [10]

In this next snippet of code, we display the wakefield census tracts and use the display parameter (disp) to view both the geometry (shape) and information from the attribute table (attr) with the additional parameters specifying which column of the attribute table we want to display, the color of the display, and its size.

```
d.mon x2
d.vect -c census_wake2000 disp=shape,attr attrcol=FIPSSTCO lcol=black lsiz=11
d.vect help # get a list of available options for d.vect
```

A neat little trick to see all (or selected) maps in a single monitor window is to use d.slide.show with the -v flag (to display vector, rather than raster maps):

```
d.slide.show -v mapsets=user1
```

11.5.1 Intervals and Charts

Use d.vect.thematic to display vector attributes by number of intervals. Here, we use a graduated color scheme for number of households in a census block, and graduated icon sizes for school capacity data. GRASS will also create the associated legends for the intervals.

```
g.region swwake_30m -p
d.mon x0
d.erase
d.vect.thematic -l censusblk_swwake column=HOUSEHOLDS nint=6 color=yellow-cyan
d.vect.thematic -l schools_wake column=CAPACITYTO type=point size=10 nint=6 themetype=graduated_points
```

Use d.vect.chart to display charts based on vector attribute data. Here, we display monthly precipitation data:

```
# set the region and display DEM, roads and lakes
g.region swwake_10m -p
d.erase
d.rast elevation
d.vect roadsmajor
d.vect lakes type=area fcol=cyan col=cyan
# display chart
d.vect.chart -c precip_30ynormals ctype=bar size=80 scale=0.6 column=jan,feb,mar,apr,may,jun,jul,aug,sep
```

11.6 Vector Attribute Tables

Use v.db.select to see contents of a vector map table, and v.info with the -c flag to get info on the columns:

```
system("/Applications/GRASS-6.4.app/Contents/MacOS/grass.sh")
v.db.select roadsmajor
v.info -c roadsmajor
```

Here are a couple of other ways of getting information about tables associated with a vector map:

```
# show table connection(s) of a map
v.db.connect -p schools_wake
```

[10] Look at the bottom of the Layer Manager window as you select maps and choose parameter options. You will see the d.vect syntax for what you are doing

```
# show attribute column names and types of a map
v.info -c schools_wake
# show available tables in current mapset
db.tables -p
# describe details of a table
db.describe mysoils
# describe it in shortened form
db.describe -c mysoils
```

Next,let's query a vector table using the default GRASS DBF driver. First, we display all the schools. Then we display just schools that meet certain criteria.

Note that the text-string matching ('Raleigh" and 'E') is single-quoted. Multiple queries, though, must be contained in double quotes.

```
g.region vect=schools_wake -p
d.mon x0
d.erase
# show all schools in Wake County
d.vect schools_wake col=red icon=basic/circle siz=5
d.erase
# show a subset of all elementary schools in Raleigh
d.vect schools_wake where="ADDRCITY=Raleigh and GLEVEL=E"
```

11.6.1 Setting up a SQLite connection

The default GRASS DBF driver is kind of limited, and most folks recommend using something like SQLite. The best approach is to create a new mapset with sqlite as the dbms. Then, import maps into that mapset and they will automatically be converted to sqlite.

```
# use g.mapset to create a new mapset
g.mapset -c mapset=user2
# add the mapset to search path
g.mapsets addmapset=user2
# define the sqlite as the dbms for the new mapset
db.connect driver=sqlite database="/Users/charlie/grassdata/nc_spm_08/user2/sqlite.db"
# test the connection
db.connect -p
# copy map from PERMANENT, this converts table to SQLite
g.copy vect=roadsmajor,myroadsmajor
# show attribute connection
v.db.connect -p myroadsmajor
# show SQLite tables that can be modified (in current MAPSET)
db.tables -p
```

Note that we called the sqlite dbms for this mapset, simply, 'sqlite.db'. In general, this approach will work.

11.6.1.1 Converting a csv file to sqlite

There are two approaches to importing a csv file without geometry. Easiest is to use *db.in.ogr*

```
db.in.ogr "/Users/charlie/Desktop/wake_soil_groups.csv" out=wake_soil_groups
```

Since we are importing it into a mapset (user2) that is set up as sqlite, it will be imported as a sqlite database.

Alternatively, we could use ogr2ogr to convert the csv table to sqlite directly, and then add it as a table to the sqlite.db:

```
ogr2ogr -update -f SQLite "/Users/charlie/grassdata/nc_spm_08/user2/sqlite.db" "/Users/charlie/Desktop/w
```

11.6.2 Table maintenance commands

- to add a new column, use v.db.addcol
- to remove a column from a table, use v.db.dropcol
- to rename a column, use v.db.renamecol
- to add a new table to a vector map, use v.db.addtable
- to delete an entire table, run v.db.droptable (you have to activate the -f flag to really remove the table connected to the map)
- to update a column, use v.db.update
- to join two tables, use v.db.join

11.6.3 Reclassifying Vector Maps

```
system("/Applications/GRASS-6.4.app/Contents/MacOS/grass.sh")
g.copy vect=geonames_NC,mygeonames # copy the counties of points of interest map from permanent into cur
# use SQL query to select by attribute
v.db.select mygeonames where="POPULATION<>0 and FEATURECOD=ADM2"
```

Write an ASCII “rules” file of the following and save it as countypop.cls

```
cat 1
WHERE FEATURECOD=âĂŹADM2âĂŹ AND POPULATION=0
cat 2
WHERE FEATURECOD=âĂŹADM2âĂŹ AND POPULATION>0 AND POPULATION<1000
cat 3
WHERE FEATURECOD=âĂŹADM2âĂŹ AND POPULATION>=1000 AND POPULATION<10000
cat 4
WHERE FEATURECOD=âĂŹADM2âĂŹ AND POPULATION>=10000 AND POPULATION<100000
cat 5
WHERE FEATURECOD=âĂŹADM2âĂŹ AND POPULATION>=100000 AND POPULATION<500000
cat 6
WHERE FEATURECOD=âĂŹADM2âĂŹ AND POPULATION>=500000
```

```
v.reclass mygeonames rules="/Users/charlie/Dropbox/grassHowTo/countypop.cls" out=mygeonames_reclass
v.category mygeonames_reclass op=report
# add new attribute table to the new map
# generate single column to store text label for each category number (ID)
v.db.addtable mygeonames_reclass col="popclass varchar(50)"
# insert values into the table
v.db.update mygeonames_reclass col=popclass value="unk" where="cat=1"
v.db.update mygeonames_reclass col=popclass value="very low" where="cat=2"
v.db.update mygeonames_reclass col=popclass value="low" where="cat=3"
v.db.update mygeonames_reclass col=popclass value="medium" where="cat=4"
v.db.update mygeonames_reclass col=popclass value="high" where="cat=5"
v.db.update mygeonames_reclass col=popclass value="very high" where="cat=6"

# verify and display
v.db.select mygeonames_reclass
v.info mygeonames_reclass
d.mon x0
d.erase
d.vect nc_state type=area
d.vect -c mygeonames_reclass where="popclass<>unknown"
```

Chapter 12
Appendix: USCensus2000 R Suite of Packages

To calculate population-based rates, you will often need to use data from the US Census. There are a number of ways to do this (see, for instance, the description on using American Fact Finder in the chapter on models)

You can, though, make use of the convenient suite of R US Census tools developed by Zack W Almquist, at the department of Sociology at UC Irvine called UScensus2000. As the name specifies, these data are from the 2000 census. These notes are based on Dr. Almquist's online introduction to using the package

12.1 UScensus2000 R suite of packages

The first step is to obtain and download the packages.

```
> install.packages("UScensus2000", dependencies=T)
> install.packages("UScensus2000add", dependencies=T)
> library(UScensus2000)
> install.blk("osx")
```

Note that the last step, *install.blk()* is a convenience wrapper that installs all the data. This is a big download (about 2 gigs), and the installation takes nearly an hour.

After downloading and installing, load some data. I loaded the New York State census-tract level data. There are files for the block, block group and "cdp" (census defined place name) levels.

```
> library(UScensus2000)

     US Census 2000 Tract Level Shapefiles and
                    Additional Demographic Data
Version      0.03 created on      2009-11-11.
copyright (c) 2009, Zack W. Almquist, University of California-Irvine
For citation information, type citation("UScensus2000tract").
Type help(package="UScensus2000tract") to get started.
     US Census 2000 Block Group Shapefiles and
                    Additional Demographic Data
Version      0.03 created on      2009-11-11.
copyright (c) 2009, Zack W. Almquist, University of California-Irvine
For citation information, type citation("UScensus2000blkgrp").
Type help(package="UScensus2000blkgrp") to get started.
     US Census 2000 Designated Places Shapefiles and
                    Additional Demographic Data
Version      0.03 created on      2009-11-11.
```

```
copyright (c) 2009, Zack W. Almquist, University of California-Irvine
For citation information, type citation("UScensus2000cdp").
Type help(package="UScensus2000cdp") to get started.

UScensus2000: US Census 2000 Suite of R Packages
Version 1.01 created on 2010-10-05
copyright (c) 2011, Zack W. Almquist, University of California-Irvine
Type help(package="UScensus2000tract") to get started.

For citation information, type citation("UScensus2000tract").
```

```
> data(new_york.tract)
> summary(as(new_york.tract,"SpatialPolygons"))
```

```
Object of class SpatialPolygons
Coordinates:
         min       max
r1 -79.76215 -71.85621
r2  40.49610  45.01585
Is projected: FALSE
proj4string :
[+proj=longlat +datum=NAD83 +ellps=GRS80 +towgs84=0,0,0]
```

```
> names(new_york.tract)
```

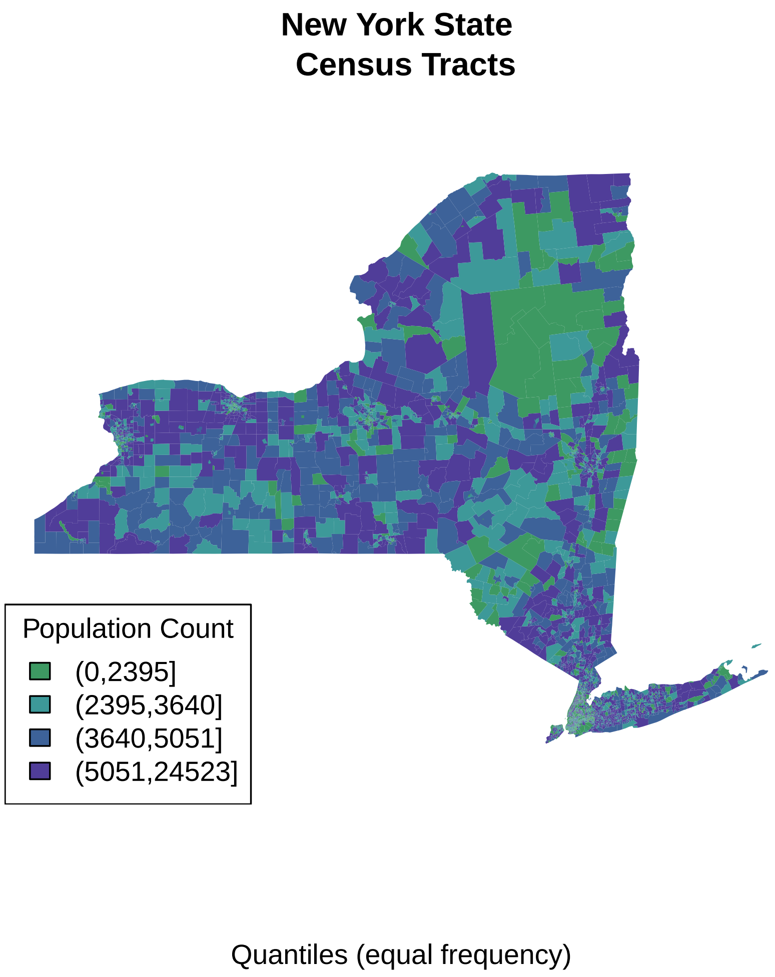
```
 [1] "state"          "county"         "tract"          "pop2000"
 [5] "white"          "black"          "ameri.es"       "asian"
 [9] "hawn.pi"        "other"          "mult.race"      "hispanic"
[13] "not.hispanic.t" "nh.white"       "nh.black"       "nh.ameri.es"
[17] "nh.asian"       "nh.hawn.pi"     "nh.other"       "hispanic.t"
[21] "h.white"        "h.black"        "h.american.es"  "h.asian"
[25] "h.hawn.pi"      "h.other"        "males"          "females"
[29] "age.under5"     "age.5.17"       "age.18.21"      "age.22.29"
[33] "age.30.39"      "age.40.49"      "age.50.64"      "age.65.up"
[37] "med.age"        "med.age.m"      "med.age.f"      "households"
[41] "ave.hh.sz"      "hsehld.1.m"     "hsehld.1.f"     "marhh.chd"
[45] "marhh.no.c"     "mhh.child"      "fhh.child"      "hh.units"
[49] "hh.urban"       "hh.rural"       "hh.occupied"    "hh.vacant"
[53] "hh.owner"       "hh.renter"      "hh.1person"     "hh.2person"
[57] "hh.3person"     "hh.4person"     "hh.5person"     "hh.6person"
[61] "hh.7person"     "hh.nh.white.1p" "hh.nh.white.2p" "hh.nh.white.3p"
[65] "hh.nh.white.4p" "hh.nh.white.5p" "hh.nh.white.6p" "hh.nh.white.7p"
[69] "hh.hisp.1p"     "hh.hisp.2p"     "hh.hisp.3p"     "hh.hisp.4p"
[73] "hh.hisp.5p"     "hh.hisp.6p"     "hh.hisp.7p"     "hh.black.1p"
[77] "hh.black.2p"    "hh.black.3p"    "hh.black.4p"    "hh.black.5p"
[81] "hh.black.6p"    "hh.black.7p"    "hh.asian.1p"    "hh.asian.2p"
[85] "hh.asian.3p"    "hh.asian.4p"    "hh.asian.5p"    "hh.asian.6p"
[89] "hh.asian.7p"
```

A help file is available to get information about the variables in the state data set.

```
> help(new_york.tract)
```

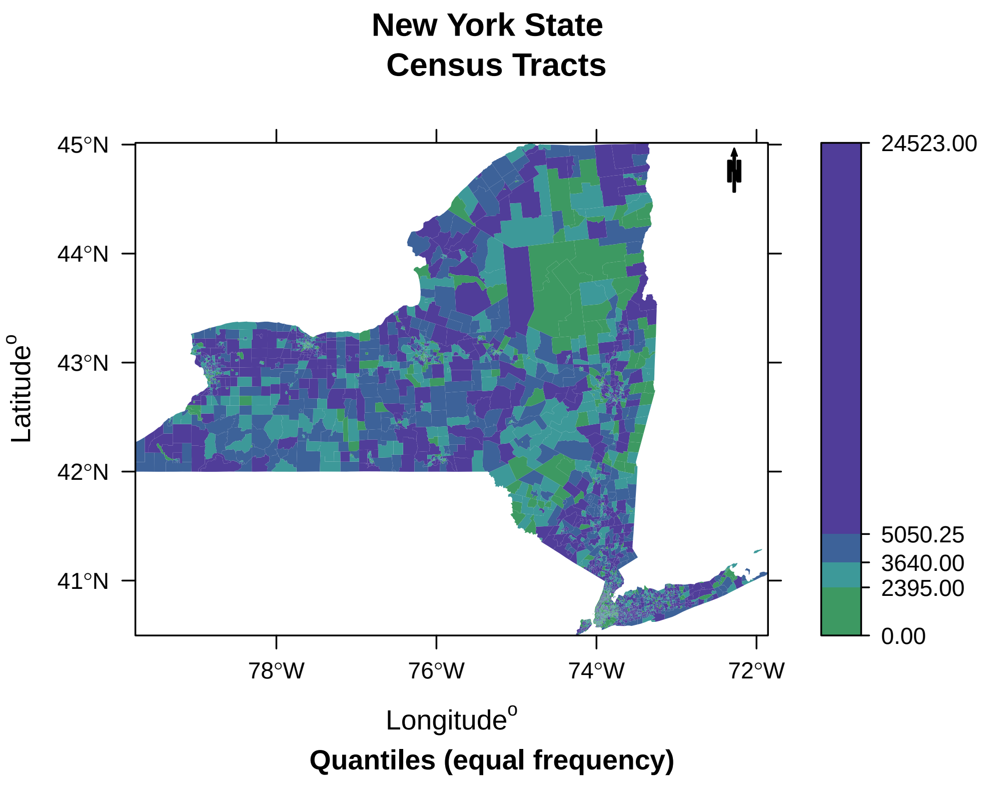
The data set is based on SpatialPolygonsDataframe objects, from which data frames can be extracted. The *choropleth()* function conveniently invokes *plot()*

```
> choropleth(new_york.tract, main="New York State \n Census Tracts", border="transparent")
```



There is also an "spplot" option available.

```
> print(choropleth(new_york.tract, main="New York State \n Census Tracts", border="transpa
```



There are a couple of functions available to extract shapefiles and data at various geographic levels. The *county()* function, will return data for a county at various levels, here at the census tract level.

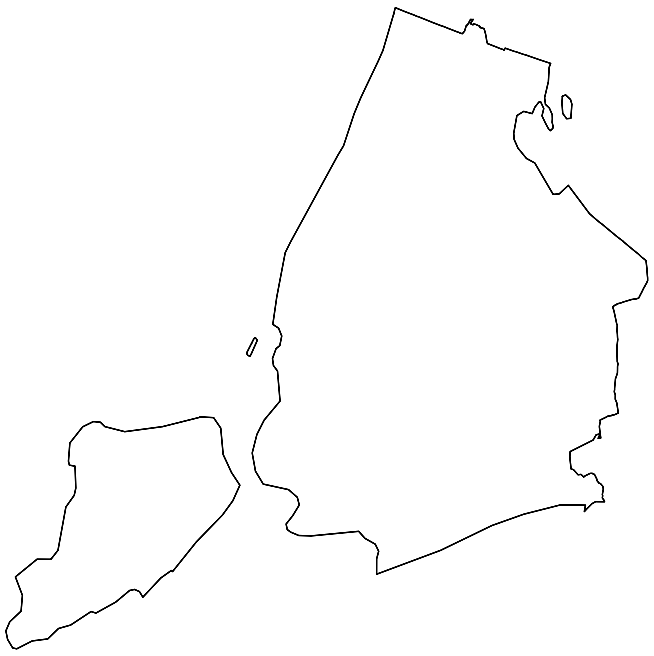
```
> brooklyn<-county(name="kings", state="ny", level="tract")
> plot(brooklyn)
> choropleth(brooklyn)
```

You can also request data using place names. To get a list of available names, load and review an alphabetical listing of the "cdp" place names in the data files.

```
> data(new_york.cdp)
> head(new_york.cdp$name[order(new_york.cdp$name)])
```

In the New York State data set, New York City is listed as "New York". The plot for a place name, though, only returns the boundary file.

```
> nyc<-city(name="New York", state="ny")
> plot(nyc)
```



To get sub-levels, like census tracts, you will need to use the *poly.clipper()* function.

```
> nyc.tract<-poly.clipper(name="New York", state="ny", level="tract")
> plot(nyc.tract)
```

12.2 census tract files for spatial analysis

For the actual spatial analyses, you will to create a data frame of population variables at the census tract level. To do this, use the *demographics()* function in "USCensus2000".

As a first step, you might list the names of the variables available at the tract level. Again, the help file for new_york.tract is, well, helpful. you can directly list the names. Here, in the interests of space, I'll list just the first few.

```
> head(names(new_york.tract))
```

From these, select variables you think might be helpful and informative, and save them as a vector of names.

```
> vars <- c("pop2000", "white", "black", "asian", "hispanic", "males", "females",
+ "age.under5", "age.5.17", "age.18.21", "age.22.29", "age.30.39", "age.40.49",
+ "age.50.64", "age.65.up", "households", "ave.hh.sz", "fhh.child", "hh.units",
+ "hh.urban", "hh.occupied", "hh.vacant", "hh.owner", "hh.renter")
```

There is (at least to my knowledge) no way to directly extract census-tract level rows of these variables for geographies like a city. So, for New York City, I had to first create a file for the state, and then subset to the counties in New York City.

This first chunk of code creates the state file, and then converts the resulting matrix into a dataframe.

```
> nys.pop<-demographics(dem=vars, state="ny", level="tract")
> nys.pops<-data.frame(nys.pop)
```

The data frame is labeled by state fips codes. Here, I manipulate these codes to create a county fip code for each census tract.

```
> codes<-labels(nys.pop)
> county.fips<-substr(codes,3,5)
> nys.pops$county.fip<-county.fips
```

Next, I create a logical index of codes that match counties in New York City (New York State is 36, the bronx 005, kings (brooklyn) 047, new york (manhattan) 061, queens 36081, and richmond (staten island) 085)

```
> nyc.counties<-nys.pops$county.fip=="005" | nys.pops$county.fip=="047" | nys.pops$county..
> nyc.pop<-nys.pops[nyc.counties,]
```

And now you have a very nice R data object based on geography and the US Census.