9

SPATIAL MODELS

"Design is a question of substance, not just form."
—Adriano Olivetti

In Chapter 7, annual counts were used to create rate models, and in Chapter 8, lifetime maximum winds were used to create intensity models. In this chapter, we show you how to use cyclone track data together with climate field data to create spatial models.

Spatial models make use of location information in data. Geographic coordinates locate the hurricane's center on the surface of the earth and wind speed provides an attribute. Spatial models make use of location separate from attributes. Given a common spatial framework, these models can accommodate climate data including indices (e.g., North Atlantic Oscillation) and fields (e.g., sea-surface temperature).

9.1 TRACK HEXAGONS

Here, we show you how to create a spatial framework for combining hurricane data with climate data. The method tessellates the basin with hexagons and populates them with local cyclone and climate information (Elsner et al., 2012).

9.1.1 Spatial Points Data Frame

In Chapter 5, you learned how to create a spatial data frame using functions from the **sp** package (Bivand et al., 2008). Let us review. Here you are interested in wind speeds along the entire track for all tropical storms and hurricanes during the 2005 North Atlantic season. You begin by creating a data frame from the *best.use.RData* file, where you subset on year and wind speed and convert the speed to meters per second.

```
> load("best.use.RData")
> W.df = subset(best.use, Yr==2005 & WmaxS >= 34
+    & Type=="*")
> W.df$WmaxS = W.df$WmaxS * .5144
```

The asterisk for `Type` indicates a tropical cyclone as opposed to a tropical wave or extratropical cyclone. The number of rows in your data frame is the total number of cyclone hours (3,010), and you save this by typing

```
> ch = nrow(W.df)
```

Next, assign the `lon` and `lat` columns as spatial coordinates using the `coordinates` function (**sp**). Finally, make a copy of your data frame, keeping only the spatial coordinates and the wind speed columns.

```
> require(sp)
> W.sdf = W.df[c("lon", "lat", "WmaxS")]
> coordinates(W.sdf) = c("lon", "lat")
> str(W.sdf, max.level=2, strict.width="cut")
Formal class 'SpatialPointsDataFrame' [package "sp"..
  ..@ data       :'data.frame': 3010 obs. of  1 var..
  ..@ coords.nrs : int [1:2] 1 2
  ..@ coords     : num [1:3010, 1:2] -83.9 -83.9 -8..
  .. ..- attr(*, "dimnames")=List of 2
  ..@ bbox       : num [1:2, 1:2] -100 11 -12.4 44.2
  .. ..- attr(*, "dimnames")=List of 2
  ..@ proj4string:Formal class 'CRS' [package "sp"]..
```

The result is a spatial points data frame with five slots. The data slot is a data frame and contains the attributes (here only wind speed). The coordinate (`coord`) slot contains the longitude and latitude columns from the original data frame and the coordinate numbers (here two spatial dimensions) are given in the `coords.nrs` slot. The bounding box (`bbox`) slot is a matrix containing the maximal extent of the hurricane positions as defined by the lower left and upper right longitude/latitude coordinates.

A summary of the information in your spatial points data frame is obtained by typing

```
> summary(W.sdf)
Object of class SpatialPointsDataFrame
Coordinates:
     min   max
lon -100 -12.4
lat   11  44.2
Is projected: NA
proj4string : [NA]
Number of points: 3010
```

```
Data attributes:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   17.5    23.7    29.5    33.3    38.0    80.2
```

Again, there are 3,010 cyclone hours. The projection (`proj4string`) slot contains an `NA` character indicating that it has not yet been specified.

You give your spatial data frame a coordinate reference system using the `CRS` function. Here you specify a geographic reference (intersections of parallels and meridians) as a character string in an object called `ll_crs`, then use the `proj4string` function to generate a CRS object and assign it to your spatial data frame.

```
> ll = "+proj=longlat +datum=WGS84"
> proj4string(W.sdf) = CRS(ll)
```

Check this slot by typing

```
> slot(W.sdf, "proj4string")
CRS arguments:
 +proj=longlat +datum=WGS84 +ellps=WGS84
+towgs84=0,0,0
```

Next, you transform the geographic CRS into a Lambert conformal conic (LCC) planar projection using the parallels 30 and 60°N and a center longitude of 60°W. First save the reference system as a CRS object and then use the `spTransform` function from the **rgdal** package.

```
> lcc = "+proj=lcc +lat_1=60 +lat_2=30 +lon_0=-60"
> require(rgdal)
> W.sdf = spTransform(W.sdf, CRS(lcc))
> bbox(W.sdf)
        min     max
lon -3999983 3987430
lat  1493807 5521312
```

The coordinates are now planar rather than geographical, although the coordinate names remain the same (`lon` and `lat`). Projection of the hurricane locations to a plane makes it easy to perform distance calculations.

9.1.2 Hexagon Tessellation

Next, you construct a hexagon tessellation of the cyclone tracks. This is done by first situating points representing a staggered grid and and then drawing hexagon boundaries around each grid point. You create the grid points inside the bounding box defined by your spatial points data frame using the `spsample` function. Here you specify the number of points, but the actual number will vary depending on the bounding box. You expand the bounding box by multiplying each coordinate value by 20 percent and use an offset vector to fix the position of the grid point, over the tracks.

```
> hpt = spsample(W.sdf, type="hexagonal", n=250,
+   bb=bbox(W.sdf) * 1.2, offset=c(1, -1))
```

The expansion and offset values require a bit of trial and error. The methods used in spatial sampling function assume that the geometry has planar coordinates, so your spatial data object should not have geographic coordinates.

Next, call the function to convert the points to hexagons. The more points, the smaller the area of each hexagon.

```
> hpg = HexPoints2SpatialPolygons(hpt)
```

The number of polygons generated and the area of each polygon is obtained by typing

```
> np = length(hpg@polygons)
> area = hpg@polygons[[1]]@area
> np; area
[1] 228
[1] 2.14e+11
```

This results in 228 equal-area hexagons. The length unit is meters. To convert the area from square meters to square kilometers, multiply by 10^{-6}. Thus the area of each hexagon is approximately 213,961 km^2.

9.1.3 Overlays

With your hexagons and cyclone locations having the same projection, you now obtain the maximum intensity and number of observations per hexagon. The function over combines points (or grids) and polygons by performing point-in-polygon operations on all point–polygon combinations. First you obtain a vector containing the hexagon identification number for each hourly cyclone observation by typing

```
> hexid = over(x=W.sdf, y=hpg)
```

The length of the vector is the number of hourly observations. Not all hexagons have cyclones, so you subset your spatial polygons object keeping only those that do.

```
> hpg = hpg[unique(hexid)]
```

Next, you create a data frame with a single column listing the maximum wind speed over the set of all cyclone observations in each hexagon.

```
> int = over(x=hpg, y=W.sdf, fn=max)
> colnames(int) = c("WmaxS")
> head(int)
```

```
      WmaxS
ID60   76.3
ID80   65.7
ID99   65.9
ID118  74.8
ID137  31.9
ID96   23.1
```

The result is a data frame with a single column representing the maximum wind speed value over all cyclone observations in each hexagon. The row names are the hexagon numbers prefixed with `ID`. Finally, you combine the spatial polygons with the maximum per hexagon wind speeds to create a spatial polygon data frame.

```
> hspdf = SpatialPolygonsDataFrame(hpg, int,
+   match.ID = TRUE)
```

You plot the hourly storm locations together with an overlay of your hexagons by typing

```
> plot(hspdf)
> plot(W.sdf, pch=20, cex=.3, add=TRUE)
```

Some hexagons contain many cyclone observations while others contain only a few. This difference might be important in modeling intensity, so you add total cyclone hours as a second attribute. First, replace the data in the spatial points data frame with an index of ones.

```
> W.sdf@data = data.frame(num=rep(1, ch))
```

Then, perform an overlay of the hexagons on the cyclone locations with the function argument (`fn`) set to `sum`). Check that the sum of the counts over all grids equals the total number of cyclone hours.

```
> co = over(x=hspdf, y=W.sdf, fn=sum)
> sum(co) == ch
[1] TRUE
```

Finally, add the counts to the spatial polygon data frame and list the first six rows of the data frame corresponding to the first six hexagons. Here, the rows of `co` correspond to those in the data slot of `hspdf` so there is no need to match the polygon IDs.

```
> hspdf$count = co[, 1]
> head(slot(hspdf, "data"))
      WmaxS count
ID60   76.3    95
ID80   65.7    50
ID99   65.9   104
ID118  74.8    50
```

```
ID137  31.9    13
ID96   23.1    45
```

9.1.4 Maps

You now have a spatial polygon data frame with each polygon as an equal-area hexagon on an LCC projection and two attributes (maximum wind speed and cyclone count) in the data slot. Choropleth maps of cyclone attributes are created using the `spplot` method introduced in Chapter 5.

Before mapping, you create a couple of lists that are used by the `sp.layout` argument. One specifies the set of hourly locations from the spatial points data frame and assigns values to plot arguments.

```
> l1 = list("sp.points", W.sdf, pch=20, col="gray",
+   cex=.3)
```

Another specifies the coastlines as a spatial lines object. Some additional work is needed.

Next require the **maps** and **maptools** packages. The first package contains the `map` function to generate an object of country borders and the second contains the `map2SpatialLines` conversion function. The conversion is made to geographic coordinates, which are then transformed to the LCC projection of the earlier spatial objects. You set geographic coordinate limits on the map domain to limit the amount of conversion and projection calculations.

```
> require(maps)
> require(maptools)
> cl = map("world", xlim=c(-120, 20),
+   ylim=c(-10, 70), plot=FALSE)
> clp = map2SpatialLines(cl, proj4string=CRS(ll))
> clp = spTransform(clp, CRS(lcc))
> l2 = list("sp.lines", clp, col="gray")
```

Depicting attribute levels on a map is done using a color ramp. A color ramp creates a character vector of color hex codes. The number of colors is specified by the argument in the color ramp function. Color ramp functions are available in the **colorRamps** package (Keitt, 2009). Acquire the package and assign 20 colors to the vector `cr` using the blue-to-yellow color ramp.

```
> require(colorRamps)
> cr = blue2yellow(20)
```

If the number of colors is less than the number of levels, the colors get recycled. A map of cyclone frequency is made by typing

```
> spplot(hspdf, "count", col="white",
+    col.regions=blue2yellow(20),
+     sp.layout=list(l1, l2),
+     colorkey=list(space="bottom"),
+     sub="Cyclone Hours")
```

Similarly, a map of the highest hurricane intensity is made by typing

```
> spplot(hspdf, "WmaxS", col="white",
+     col.regions=blue2red(20),
+     sp.layout=list(l2),
+     colorkey=list(space="bottom"),
+     sub="Highest Intensity (m/s)")
```

The results are shown in Figure 9.1. Areas along the southeastern coastline of the United States have the greatest number of cyclone hours while the region from the western Caribbean into the Gulf of Mexico has the highest hurricane intensities.

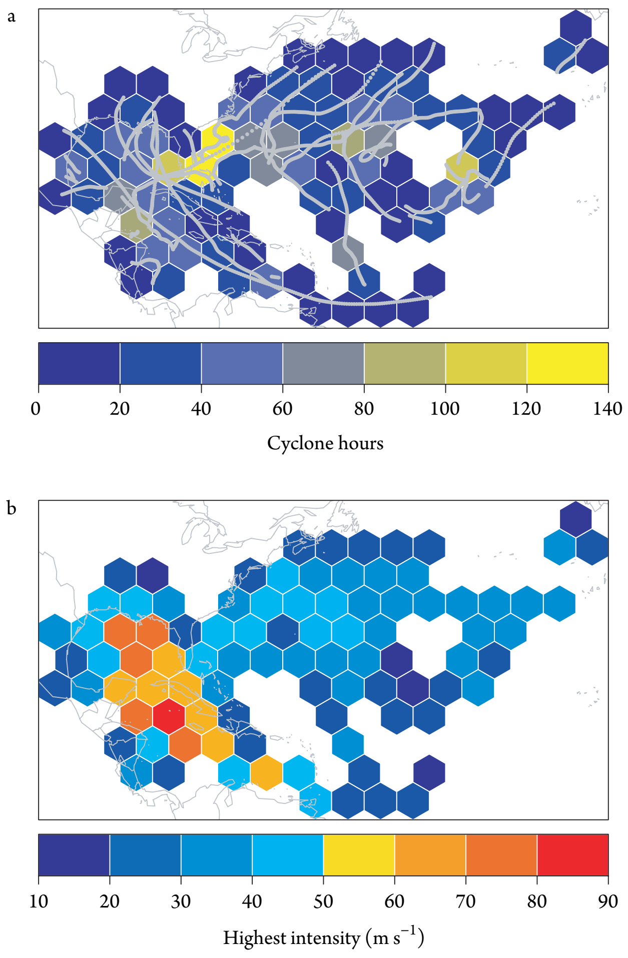

Figure 9.1 Cyclone frequency and intensity. (a) Hours and (b) highest intensity.

9.2 SST DATA

The effort needed to create track grids pays off nicely when you add regional climate data. Here you use sea-surface temperature (SST) from July 2005, as an example. July values indicate conditions occurring before the active part of the North Atlantic hurricane season.

In Chapter 6, you used functions in the **ncdf** package along with some additional code to extract a data frame of SST consisting of monthly values at the intersections of parallels and meridians at 2° intervals. We return to these data here. Input them by typing

```
> sst = read.table("sstJuly2005.txt", header=TRUE)
> head(sst)
   SST  lon lat
1 24.1 -100   0
2 24.0  -98   0
3 23.8  -96   0
4 23.5  -94   0
5 23.4  -92   0
6 23.6  -90   0
```

The data are listed in the column labeled `SST`. You treat these values as point data because they have longitudes and latitudes although they are regional averages.

At 25°N latitude, the 2° spacing of the SST values covers an area of approximately the same size as the hexagon grids used in the previous section. The SST locations are converted to planar coordinates using the same LCC projection after the data frame is converted to a spatial data frame. Recall that you need to first assign a projection string to the `proj4string` slot.

```
> coordinates(sst) = c("lon", "lat")
> proj4string(sst) = CRS(ll)
> sst = spTransform(sst, CRS(lcc))
```

To examine the SST data as spatial points, type

```
> spplot(sst, "SST", col.regions=rev(heat.colors(20)))
```

This produces a plot of the spatial points data frame, where the SST attribute is specified as a character string in the second argument.

Your interest is the average SST within each hexagon. Again, use the `over` function to combine your SST values with the track polygons setting the argument `fn` to `mean`.

```
> ssta = over(x=hspdf, y=sst, fn=mean)
```

The result is a data frame with a single column representing the average over all SST values in each hexagon. The row names are the hexagon numbers prefixed with `ID`; however, hexagons completely over land have missing SST values.

Next, you add the average SSTs as an attribute to spatial polygon data frame and then remove hexagons with missing SST values.

```
> hspdf$sst = ssta$SST
> hspdf = hspdf[!is.na(hspdf$sst),]
> str(slot(hspdf, "data"))
'data.frame':   88 obs. of  3 variables:
 $ WmaxS: num  76.3 65.7 65.9 74.8 23.1 ...
 $ count: num  95 50 104 50 45 58 56 26 40 24 ...
 $ sst  : num  29.2 29.4 29.8 30 29.5 ...
```

Finally, you create the July SST map corresponding to where the cyclones occurred during the season by typing

```
> spplot(hspdf, "sst", col="white",
+    col.regions=blue2red(20),
+    sp.layout=list(l1, l2),
+    colorkey=list(space="bottom"),
+    sub="Sea Surface Temperature (C)")
```

Results are shown in Figure 9.2. Ocean temperatures exceed 26°C over a wide swath of the North Atlantic extending into the Caribbean Sea and Gulf of Mexico. Coldest waters are noted off the coast of Nova Scotia and Newfoundland.

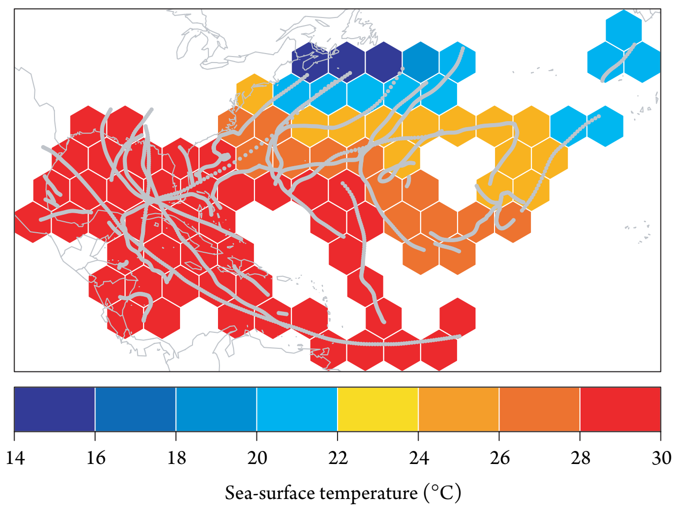

Figure 9.2 Sea-surface temperature in the cyclone hexagons.

9.3 SST AND INTENSITY

Analyzing and modeling your SST and cyclone data are easy with your spatial polygon data frame. For instance, the average maximum wind speed in regions where the SST exceeds 28°C is obtained by typing

```
> mean(hspdf$WmaxS[hspdf$sst > 28])
[1] 45.1
```

Here you treat your spatial data frame as you would a regular data frame. Continuing, you create side-by-side box plots of wind speed conditional on the collocated values of SST (see Chapter 5) by typing

```
> boxplot(hspdf$WmaxS~hspdf$sst > 28)
```

Spatial information allows you to analyze relationships on a map. Figure 9.3 shows the hexagons colored by groups defined by a two-way table of cyclone intensity and SST using above and below median values. The median SST and intensity values calculated from your data are 28.2°C and 33.9 m s^{-1}, respectively. Red hexagons indicate regions of high intensity and relatively high ocean temperature and blue hexagons indicate regions of low intensity and relatively low ocean temperature. More interesting are regions of mismatch. Magenta hexagons show low intensity coupled with relatively high ocean temperature indicating "underperforming" cyclones (cyclones weaker than the thermodynamic potential of the environment). By contrast, cyan hexagons show high intensity coupled with relatively low temperature indicating "overperforming" cyclones (cyclones stronger than the thermodynamic potential of the environment).

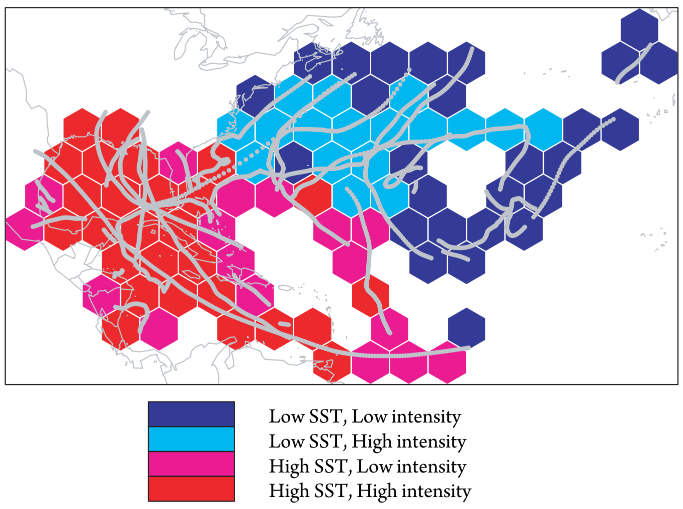
🟦 Low SST, Low intensity
🟦 Low SST, High intensity
🟪 High SST, Low intensity
🟥 High SST, High intensity

Figure 9.3 SST and cyclone intensity. Groups are based on median values.

Maps provide insight into the relationship between hurricanes and climate not accessible with basin-level analyses. Next, we show you how to model these spatial data. We begin with a look at spatial correlation and then use a spatial regression model to quantify the regional variation in intensity as a function of SST.

9.4 SPATIAL AUTOCORRELATION

Hurricane intensities in neighboring hexagons will tend to be more similar than intensities in hexagons farther away. Spatial autocorrelation quantifies the degree of similarity across geographic space. It is more complicated than temporal autocorrelation because map space has two dimensions and multiple directions.

9.4.1 Moran's I

A measure of spatial autocorrelation is Moran's I (Moran, 1950), defined as

$$I = \frac{m}{s}\frac{y^T W y}{y^T y} \tag{9.1}$$

where m is the number of hexagons, y is the vector of values within each hexagon (e.g., cyclone intensities) where the values are deviations from the overall spatial mean, W is a weights matrix, s is the sum over all the weights, and the subscript T indicates the transpose operator.

Values of Moran's I range from -1 to $+1$ with a value of zero indicating a pattern with no spatial autocorrelation. Although not widely used in climate studies, de Beurs and Henebry (2008) use it to identify spatially coherent eco-regions and biomes related to the North Atlantic Oscillation (NAO).

To compute Moran's I, you need a weights matrix. The weights matrix is square with the number of rows and columns equal to the number of hexagons. Here the weight in row i, column j, of the matrix is assigned a zero unless hexagon i is contiguous with hexagon j. The **spdep** package (Bivand et al., 2011a) has functions for creating weights based on contiguity (and distance) neighbors. The process requires two steps.

First, you use the `poly2nb` function on your spatial polygon data frame to create a contiguity-based neighborhood list object.

```
> require(spdep)
> hexnb = poly2nb(hspdf)
```

A summary of the neighborhood list is obtained by typing

```
> hexnb
Neighbour list object:
Number of regions: 88
Number of nonzero links: 372
Percentage nonzero weights: 4.8
Average number of links: 4.23
```

The list is ordered by hexagon number, starting with the southwest-most hexagon. It has five neighbors: hexagon numbers 2 and 21. Hexagon numbers increase to the east and north. A hexagon has at most six contiguous neighbors. Hexagons at the borders have fewer neighbors. A graph of the hexagon connectivity, here defined by the first-order contiguity, is made by typing

```
> plot(hexnb, coordinates(hspdf))
> plot(hspdf, add=TRUE)
```

A summary method applied to the neighborhood list (`summary(hexnb)`) reveals the average number of neighbors and the distribution of connectivity among the hexagons.

You turn the neighborhood list object into a `listw` object using the `nb2listw` function that duplicates the neighborhood list and adds the weights. The `style` argument determines the weighting scheme. With the argument value set to `W`, the weights are the inverse of the number of neighbors. For instance, the six neighbors of a fully connected hexagon each get a weight of 1/6.

```
> wts = nb2listw(hexnb, style="W")
> summary(wts)
```

Now you are ready to compute the value of Moran's *I*. This is done using the `moran` function. The first argument is the per hexagon maximum intensity followed by the name of the `listw` object. Also needed is the number of hexagons and the global sum of the weights, which is obtained using the `Szero` function.

```
> m = length(hspdf$WmaxS)
> s = Szero(wts)
> moran(hspdf$WmaxS, wts, n=m, S0=s)
$I
[1] 0.564

$K
[1] 2.87
```

The function returns Moran's *I* and the sample kurtosis.[1] The value of 0.56 indicates fairly high spatial autocorrelation in cyclone intensity. This is expected, since the strength of a hurricane as it moves across the hexagons often does not vary by much and because stronger hurricanes tend to occur at lower latitudes.

9.4.2 Spatial Lag Variable

Insight into the Moran's *I* statistic is obtained by noting that it is the slope coefficient in a regression model of Wy on y, where Wy is the spatial lag variable (see Eq. 9.1).

[1] Kurtosis is a measure of the peakedness of the distribution. A normal distribution has a kurtosis of 3.

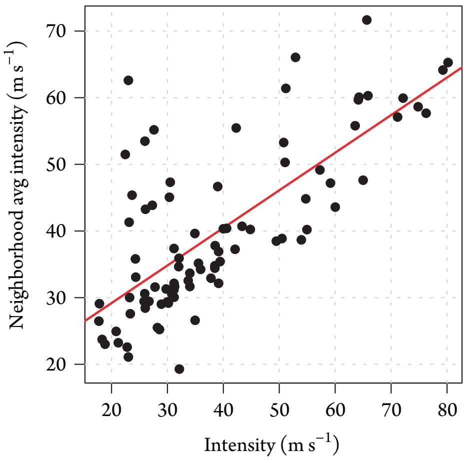

Figure 9.4 Moran's scatter plot for cyclone intensity.

Let `y` be the set of intensities in each hexagon, then you create a spatial lag intensity variable using the `lag.listw` function.

```
> y = hspdf$WmaxS
> Wy = lag.listw(wts, y)
```

Thus, for each intensity value in the vector object `y`, there is a corresponding value in the vector object `Wy`, representing the mean intensity over the neighboring hexagons. The neighborhood average does not include the value in `y`, so for a completely connected hexagon, the average is taken over the adjoining six neighboring values.

A scatter plot of the neighborhood average intensity versus the intensity in each hexagon shows the spatial autocorrelation relationship. The slope of a least-squares regression line through the points is the value of Moran's *I*. Use the following code below to create the scatter plot shown in Figure 9.4.

```
> par(las=1, pty="s")
> plot(y, Wy, pch=20, xlab="Intensity (m/s)",
+   ylab="Neighborhood Avg Intensity (m/s)")
> abline(lm(Wy ~ y), lwd=2, col="red")
```

The scatter plot (Moran's scatter plot) indicates a high level of spatial autocorrelation. High-intensity hexagons tend to be surrounded, on average, by hexagons with high intensity and vice versa as evidenced by the upward slope of the regression line.

9.4.3 Statistical Significance

The expected value of Moran's *I* under the hypothesis of no spatial autocorrelation is

$$E(I) = \frac{-1}{m-1} \tag{9.2}$$

where *m* is the number of hexagons. This allows you to test the significance of your sample Moran's *I*. The test is available in the `moran.test` function. The first argument is the vector of intensities and the second is the spatial weights matrix in the weights list form.

```
> moran.test(y, wts)
    Moran's I test under randomisation

data:  y
weights: wts

Moran I statistic standard deviate = 7.53,
p-value = 2.548e-14
alternative hypothesis: greater
sample estimates:
Moran I statistic       Expectation
          0.56410          -0.01149
         Variance
          0.00584
```

The output shows the standard deviate computed by taking the difference between the estimated *I* and its expected value under the null hypothesis of no autocorrelation. The difference is divided by the square root of the difference between the variance of *I* and the square of the mean of *I*. The *p*-value is the chance of observing a standard deviate this large or larger assuming that there is no spatial autocorrelation (the null hypothesis). The *p*-value is extremely small leading to conclude that there is significant autocorrelation. The output also gives the value of Moran's *I* along with its expected value and variance.

By default, the variance of *I* is computed by randomizing the intensities across the hexagons. If the intensities have a normal distribution then a direct computation of the variance of I is made by adding the argument `random=FALSE`. Moran's *I* and the corresponding significance test are sensitive to the definition of neighbors and to the neighborhood weights, so conclusions should be stated as conditional on your definition of neighborhoods.

9.5 SPATIAL REGRESSION MODELS

Spatial regression models make use of spatial autocorrelation. If significant autocorrelation exists, spatial regression models have parameters that are more stable and statistical tests that are more reliable than nonspatial alternatives. For instance, confidence intervals on a regression slope will have the proper coverage probabilities and prediction errors will be smaller. Autocorrelation is included in a regression model by adding a spatial lag-variable or by including a spatially correlated error term (Anselin et al., 2006).

Spatial autocorrelation can also enter a model by having the relationship between the response and the explanatory variable vary across the domain. This is called geographically weighted regression (GWR) (Brunsdon et al., 1998, Fotheringham et al., 2000). GWR allows you to see where an explanatory variable contributes strongly to the relationship and where it contributes weakly. It is similar to a local linear regression.

For example, you compare a standard linear regression model of intensity on SST with a local linear regression of the same relationship using the `loess.smooth` function by typing

```
> x = hspdf$sst
> par(las=1, pty="s")
> plot(x, y, pch=20, xlab="SST (C)",
+   ylab="Intensity (m/s)")
> abline(lm(y ~ x), lwd=2)
> lines(loess.smooth(x, y, span=.85), lwd=2, col="red")
```

With the local regression, the relationship between intensity and SST changes for different values of SST (Fig. 9.5). For low values of SST, the relationship has a gentle slope and with high values of SST the relationship has a steeper slope. By contrast, the "global" linear regression results in a single moderate slope across all values of SST.

Fitting is done locally in the domain of SST. That is, for a point *s* along the SST axis, the fit is made using points in a neighborhood of *s* weighted by their distance to *s*. The size of the neighborhood is controlled by the `span` argument. With `span=.85`, the neighborhood includes 85 percent of the SST values.

With GWR, this localization is done in geographic space. For example, a regression of cyclone intensity on SST is performed using paired values of intensity and SST values across the 88 hexagons. For each hexagon, the weight associated with a paired

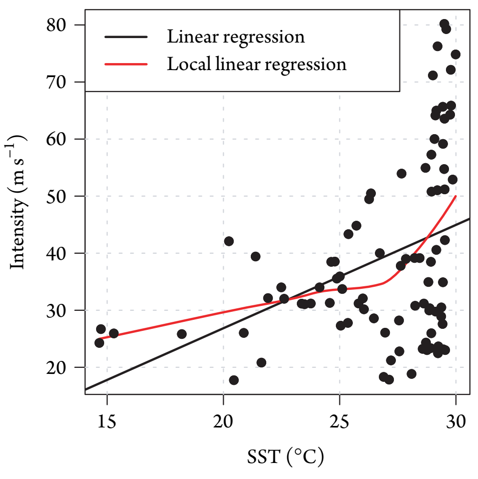

Figure 9.5 Linear and local linear regressions of cyclone intensity on SST.

value in another hexagon is inversely proportional to physical distance between the two hexagons. In this way, the relationship between intensity and SST is localized.

9.5.1 Linear Regression

The standard regression model consists of a vector y of response values and a matrix X containing the set of explanatory variables plus a row vector of 1s. The relationship is given by

$$y = X\beta + \varepsilon \tag{9.3}$$

where β is a vector of regression coefficients and $\varepsilon \sim N(0, \sigma^2 I)$ is a vector of independent and identically distributed residuals with variance σ^2. The maximum-likelihood estimate of β is given by

$$\hat{\beta} = (X^T X)^{-1} X^T y. \tag{9.4}$$

You begin with a linear regression of cyclone intensity on SST across the set of hexagons. Although your interest is the relationship between intensity and SST, you know that intensity within the hexagon will likely also depend on the number of cyclone hours. On average, a hexagon with a large number of hurricane hours will have a higher intensity. Thus, your regression model includes SST and cyclone hours as explanatory variables, in which case the SST coefficient from the regression is an estimate of the effect of SST on intensity after accounting for cyclone frequency.

You use the `lm` function to create a linear regression model. The `summary` method is used to obtain statistical information about the model.

```
> model = lm(WmaxS ~ sst + count, data=hspdf)
> summary(model)
Call:
lm(formula = WmaxS ~ sst + count, data = hspdf)

Residuals:
   Min     1Q Median     3Q    Max
-25.27 -10.03  -1.54   7.32  36.41

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -0.9402    11.3521   -0.08   0.9342
sst           1.1981     0.4390    2.73   0.0077
count         0.2343     0.0547    4.28  4.9e-05

Residual standard error: 13.6 on 85 degrees of freedom
Multiple R-squared: 0.305,  Adjusted R-squared: 0.289
F-statistic: 18.7 on 2 and 85 DF,  p-value: 1.89e-07
```

The formula is specified as `WmaxS ~ sst + count` to indicate that the mean `WmaxS` is related to `sst` and `count`. Results show that SST and cyclone hours are important in statistically explaining cyclone intensity across the set of hexagons. The parameter on the SST variable is interpreted to mean that for every 1°C increase in sea-surface temperature, cyclone intensity goes up by 1.2 ms^{-1} after accounting for cyclone hours. The model explains 30.5 percent of the variation in intensity. These results represent the average relationship over the basin.

9.5.2 Geographically Weighted Regression

With GWR the SST parameter is replaced by a vector of parameters one for each hexagon. The relationship between the response vector and the explanatory variables is given by

$$y = X\beta(g) + \varepsilon \tag{9.5}$$

where g is a vector of geographic locations—here the set of hexagons with cyclone intensities—and

$$\hat{\beta}(g) = (X^T WX)^{-1} X^T Wy \tag{9.6}$$

where W is a weights matrix given by

$$W = \exp(-D^2/h^2) \tag{9.7}$$

where D is a matrix of pairwise distances between the hexagons and h is the bandwidth. The elements of the weights matrix, w_{ij}, are proportional to the influence hexagons j have on hexagons i. Weights are determined by an inverse-distance function (kernel) so that values in nearby hexagons have greater influence on the local relationship between x and y compared with values in hexagons farther away.

The bandwidth controls the amount of smoothing. It is chosen as a trade-off between variance and bias. A bandwidth too narrow (steep gradients on the kernel) results in large variations in the parameter estimates (large variance). A bandwidth too wide leads to a large bias as the parameter estimates are influenced by processes that do not represent local conditions.

Functions for selecting a bandwidth and running GWR are available in the **spgwr** package (Bivand et al., 2011b). First, require the package and select the bandwidth using the `gwr.sel` function. The first argument is the model formula and the second is the data frame. The data frame can be a spatial points or spatial polygon data frame.

```
> require(spgwr)
> bw = gwr.sel(WmaxS ~ sst + count, data=hspdf)
> bw * .001
```

The procedure is an iterative optimization with improvements made based on previous values of the bandwidth. Values for the bandwidth and the cross-validation score are printed. After several iterations, no improvement to the score occurs. The bandwidth has dimensions of length representing a spatial distance. The units of the LCC projection in the spatial polygon data frame is meters, so to convert the bandwidth to kilometers, you multiply by 10^{-3}.

GWR is performed using the `gwr` function. The first two arguments are the same as in the function to select the bandwidth. The value of the bandwidth is supplied with the bandwidth argument.

```
> model = gwr(WmaxS ~ sst + count, data=hspdf,
+   bandwidth=bw)
```

The output is saved in an object of class `gwr`. To print a brief summary of the output, type

```
> model
Call:
gwr(formula = WmaxS ~ sst + count, data = hspdf,
   bandwidth = bw)
Kernel function: gwr.Gauss
Fixed bandwidth: 644341
Summary of GWR coefficient estimates:
                  Min.   1st Qu.    Median   3rd Qu.
X.Intercept. -8.12e+02 -3.21e+02 -1.20e+01  2.86e+01
sst          -3.59e+00 -1.79e-02  1.46e+00  1.21e+01
count         9.71e-03  1.45e-01  1.90e-01  2.71e-01
                  Max. Global
X.Intercept.  1.20e+02  -0.94
sst           2.88e+01   1.20
count         5.83e-01   0.23
```

The output repeats the function call, which includes the form of the model, the kernel function (here Gaussian), and the bandwidth (units of meters). The output also includes a summary of the regression parameter values across the hexagons. In general, intensity is positively related to SST, but the minimum parameter value indicates that in at least one hexagon the relationship is negative. The units on this regression parameter are meter per second per degree celsius. The summary also includes the parameter values from a standard regression under the column heading `Global`. These values are the same as those output previously using the `lm` function.

The object `model$SDF` inherits the spatial polygon data frame class from the model object along with the corresponding map projection. A summary method on this object provides additional information about the GWR.

```
> summary(model$SDF)
Object of class SpatialPolygonsDataFrame
Coordinates:
       min     max
x -4356736 4341671
y  1124111 6002653
Is projected: TRUE
proj4string :
[+proj=lcc +lat_1=60 +lat_2=30 +lon_0=-60
+ellps=WGS84]
Data attributes:
     sum.w        X.Intercept.          sst
 Min.   : 2.72   Min.   :-811.9   Min.   :-3.5920
 1st Qu.: 6.14   1st Qu.:-320.7   1st Qu.:-0.0179
 Median : 7.88   Median : -12.0   Median : 1.4619
 Mean   : 7.72   Mean   :-143.4   Mean   : 6.0174
 3rd Qu.: 9.55   3rd Qu.:  28.6   3rd Qu.:12.1371
 Max.   :11.11   Max.   : 119.5   Max.   :28.7864
     count             gwr.e              pred
 Min.   :0.00971   Min.   :-27.1580   Min.   :17.2
 1st Qu.:0.14454   1st Qu.: -3.5292   1st Qu.:29.9
 Median :0.19029   Median : -0.0526   Median :36.2
 Mean   :0.22379   Mean   : -0.1788   Mean   :39.2
 3rd Qu.:0.27065   3rd Qu.:  4.2381   3rd Qu.:46.3
 Max.   :0.58289   Max.   : 21.3694   Max.   :74.0
    localR2
 Min.   :0.237
 1st Qu.:0.464
 Median :0.543
 Mean   :0.566
 3rd Qu.:0.668
 Max.   :0.966
```

Here, the coordinate bounding box is given along with the projection information. Data attributes are stored in the data slot. The attributes include the model parameters (including the intercept term), the sum of the weights, predicted values, prediction errors, and R-squared values. A histogram of the R-squared values is made by typing

```
> hist(model$SDF$localR2, main="",
+   xlab="Local R-Squared Values")
```

The R-squared values are centered near a value 0.5, but there are three hexagons with a value above 0.9 (Fig. 9.6).

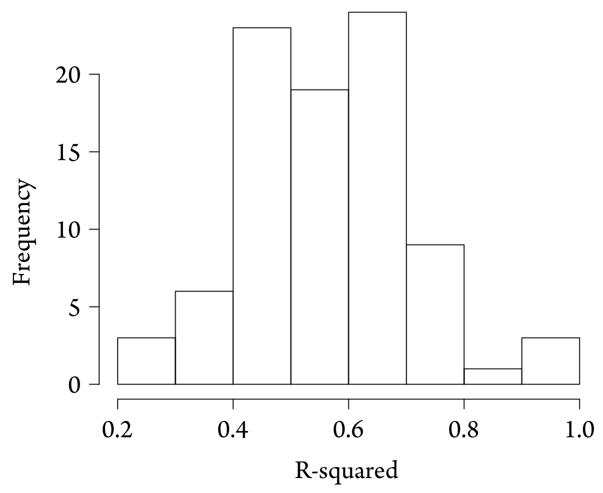

Figure 9.6 R-squared values from a GWR of intensity on SST and cyclone hours.

Additional insight is obtained by mapping the results. For instance, to make a choropleth map of the SST parameter values, type

```
> spplot(model$SDF, "sst", col="white",
+   col.regions=blue2red(10), at=seq(-25, 25, 5),
+   sp.layout=list(l2), colorkey=list(space="bottom"),
+    sub="SST Effect on Intensity (m/s per C)")
```

The results are shown in Figure 9.7. The marginal influence of SST on cyclone intensity is shown along with corresponding *t* values. The *t* value is calculated as the ratio of the parameter value to its standard error. Large values of $|t|$ indicate a statistically-significant relationship. Standard errors are available by specifying the argument `hatmatrix=TRUE` in the `gwr` function.

Hexagons are colored according to the SST parameter. The SST parameter (coefficient) represents a local "trend" of intensity as a function of SST holding cyclone hours constant. Hexagons with positive coefficients, indicating a direct relationship between cyclone strength and ocean warmth in meter per second per degree celsius are displayed using red hues and those with negative coefficients are shown with blue hues. The divergent color ramp `blue2red` creates the colors.

Hexagons with the largest positive parameters (greater than 5 m $s^{-1}/^{\circ}$C) are found over the Caribbean Sea extending into the southeast Gulf of Mexico and east of the Lesser Antilles. Coefficients above zero extend over much of the Gulf of Mexico northeastward into the southwestern Atlantic. A region of coefficients less than zero is noted over the central North Atlantic extending from the middle Atlantic coast of the United States eastward across Bermuda and into the central Atlantic. This map quantifies the grouped results shown in Figure 9.3.

Local statistical significance is estimated by dividing the parameter value by its standard error. The ratio, called the *t* value, is described by a *t* distribution under the null hypothesis of a zero-value (see Chapter 3). Regions of high *t* values (absolute value

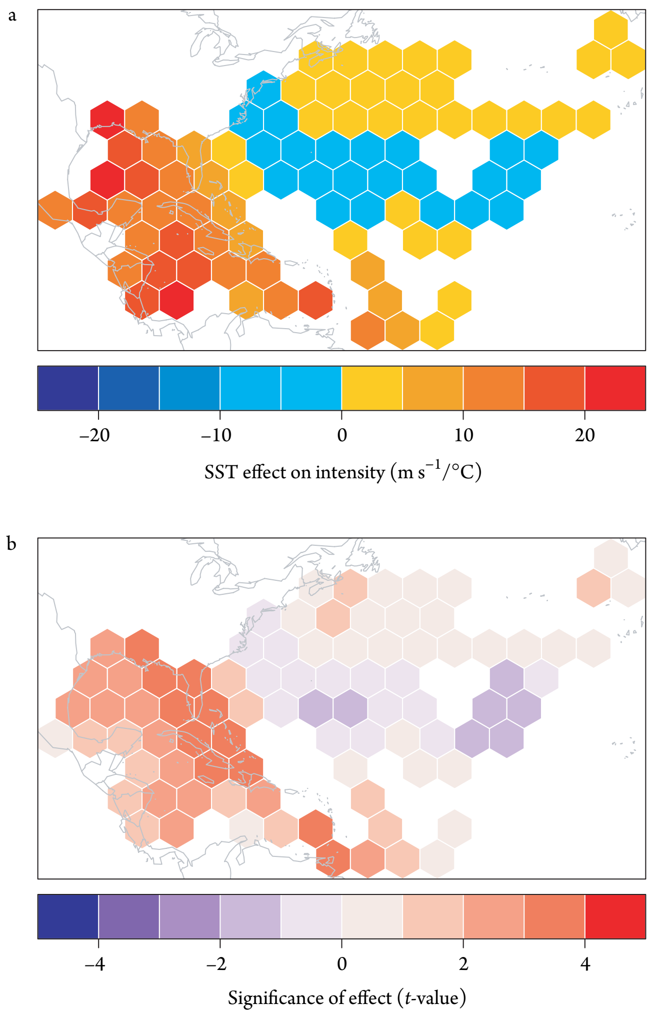
a
−20
−10
0
10
20
SST effect on intensity (m s^{-1}/°C)
b
−4
−2
0
2
4
Significance of effect (*t*-value)

Figure 9.7 Effect of SST on cyclone intensity. (a) SST coefficient and (b) *t* value.

greater than 2) denote areas of statistical significance and generally correspond with regions of large upward trends including most of the Caribbean sea and the eastern Gulf of Mexico. Regions with negative trends over the central Atlantic extending to the coastline fail to show significance. To some degree, these results depend on the size of your hexagons. One strategy is to rerun the GWR model with larger and smaller hexagons. In this way, you can check how much the results change and whether the changes influence your conclusions.

9.5.3 Model Fit

To assess model fit, you examine the residuals. Here, a residual is the difference between observed and predicted intensity in each hexagon. Residuals above zero

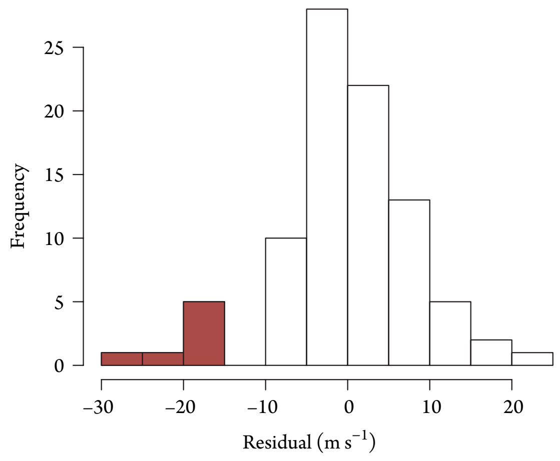

Figure 9.8 Histogram of GWR residuals. Residuals less than -10 m s^{-1} are brown.

indicate that the model underpredicts the intensity and residuals below zero indicate that the model overpredicts intensity. Residuals are saved in `model$SDF$gwr.e`. A histogram of the residuals (Fig. 9.8) shows the values are centered on zero. The left tail indicates that the model overpredicts intensity in some hexagons. Use the code below to check yourself.

```
> rsd = model$SDF$gwr.e
> hist(rsd, main="", xlab="Residual (m/s)")
> hist(rsd[rsd <= -10], add=TRUE, col="brown")
```

An interesting question concerns the location of these overpredictions. This is answered by coloring the hexagons having the largest negative residuals. First create a new vector for the spatial polygon data frame consisting of 1s indicating largest negative residuals and 0s indicating otherwise. You then use the `spplot` method and restrict the color region argument to `"brown"` and `"white"` and turn off the color key.

```
> model$SDF$op = as.integer(rsd <= (-10))
> spplot(model$SDF, "op", col="white",
+   col.regions=c("white", "brown"),
+   sp.layout=list(l2), colorkey=FALSE)
```

Results are shown in Figure 9.9. In general, the largest overpredictions occur at low latitudes in hexagons near land. This makes sense; in these regions, although SST values are high, cyclone intensities are limited by other environmental factors associated with land like drier air and greater friction. The map suggests a way the model can be improved. In this regard, a factor indicating the presence or absence of land or a covariate indicating the proportion of hexagon covered by land would be a reasonable thing to try.

Other types of spatial regression models fit this hexagon framework. For instance, if interest is cyclone counts, then a Poisson spatial model (Jagger et al., 2002) can be

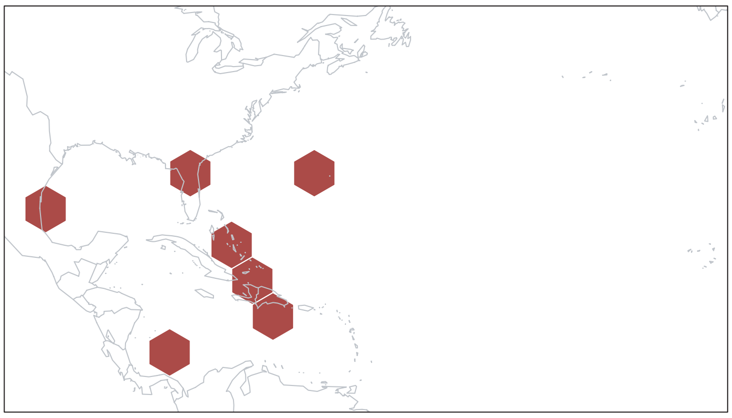
Figure 9.9 Regions of largest negative residuals.

used. And if interest is prediction rather than explanation a model that includes spatial autocorrelation through a correlated error term or a spatial, lag variable is possible. In Chapter 12, we show how to use this hexagon framework to construct a space-time model for hurricane occurrence.

9.6 SPATIAL INTERPOLATION

Spatial data often need to be interpolated. Cyclone rainfall is a good example. You know how much it rained at locations having rain gauges but not everywhere. Spatial interpolation uses the rainfall collected at the gauge sites to estimate rainfall at arbitrary locations. Estimates on a regular grid are used to create contour maps.

This can be done in various ways. Here we show how to do it statistically. Statistical interpolation is preferable because it includes uncertainty estimates. The procedure is called "kriging," after the name of a mining engineer (Matheron, 1963).

A kriged estimate (prediction) of some variable z at a given location is a weighted average of the z values over the entire domain where the weights are proportional to the spatial correlation. The estimates are optimal in the sense that they minimize the variance between the observed and interpolated values. In short, kriging involves estimating and modeling the spatial autocorrelation and then using the model together with the observations to interpolate valves at arbitrary locations.

Here you work through an example using rainfall from tropical cyclone Fay in 2008. Fay formed near the Dominican Republic as a tropical wave, passed over the island of Hispaniola and Cuba before making landfall on the Florida Keys. Fay then crossed the Florida peninsula and moved westward across portions of the Florida panhandle producing heavy rains in parts of the state.

9.6.1 Preliminaries

Here you use a spatial interpolation model to generate a continuous isohyet surface of rainfall totals from Fay. The data are in *FayRain.txt* and are a compilation of reports from the U.S. NOAA/NWS official weather sites and coop sites. The coop sites are the Community Collaborative Rain, Hail and Snow Network (CoCoRaHS), a community-based, high-density precipitation network made up of volunteers who take measurements of precipitation in their backyards. The data were obtained from NOAA/NCEP/HPC and from the Florida Climate Center.

You make use of functions in the **gstat** package (Pebesma, 2004). Require the package and read in the data by typing

```
> require(gstat)
> FR = read.table("FayRain.txt", header=TRUE)
> names(FR)
[1] "lon" "lat" "tpi" "tpm"
```

The data frame contains 803 rain guage sites. Longitude and latitude coordinates of the sites are given in the first two columns and total rainfall in inches and millimeters are given in the second two columns.

Create a spatial points data frame by specifying columns that contain the spatial coordinates. Then assign a geographic coordinate system and convert the rainfall from millimeters to centimeters.

```
> coordinates(FR) = c("lon", "lat")
> ll2 = "+proj=longlat +datum=NAD83"
> proj4string(FR) = CRS(ll2)
> FR$tpm = FR$tpm/10
```

A summary of the observed rainfall totals in centimeters is given by typing

```
> summary(FR$tpm)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    0.0     8.3    15.8    17.6    24.4    60.2
```

The median value is 15.8 cm and the maximum is 60.2 cm.

Next, import the Florida shapefile containing the county polygons as used in Chapter 5. You use the `readShapeSpatial` function from the **maptools** package to accomplish this task (see Chapter 5).

```
> require(maptools)
> FLpoly = readShapeSpatial("FL/FL",
+   proj4string=CRS(ll2))
```

Next, create a character string specifying the tags for a planar projection and transform the geographic coordinates of the site locations and map polygons to the projected coordinates. Here, you use Albers equal-area projection with true scale at latitudes 23 and 30°N and include a tag to specify meters as the unit of distance.

```
> require(rgdal)
> aea = "+proj=aea +lat_1=23 +lat_2=30 +lat_0=26,
+   +lon_0=-83 +units=m"
```

```
> FR = spTransform(FR, CRS(aea))
> FLpoly = spTransform(FLpoly, CRS(aea))
```

A map of the rain guage sites and storm totals with the state boundaries is made by typing

```
> l3 = list("sp.polygons", FLpoly, lwd=.3,
+   first=FALSE)
> spplot(FR, "tpm", sp.layout=l3)
```

Two areas of extreme rainfall are noted: one running north–south along the east coast and another one over the north. Rain gauges are clustered in urban areas.

9.6.2 Sample Variogram

Rainfall is an example of geostatistical data. In principle, it can be measured anywhere, but typically you have values at a sample of sites. The pattern of sites is not of much interest as it is a consequence of constraints (convenience, opportunity, economics, etc.) unrelated to the phenomenon. Instead, your interest centers on inference about how much rainfall across the region. This is done using kriging—the cornerstone of geostatistics (Cressie [1993]).

Kriging requires you to model the spatial autocorrelation with a variogram. The sample variogram $\hat{\gamma}(h)$ is given as

$$\hat{\gamma}(h) = \frac{1}{2N(h)} \sum_{i,j}^{N(h)} (z_i - z_j)^2 \tag{9.8}$$

where $N(h)$ is the number of distinct pairs of observation sites a lag distance h apart, and z_i and z_j are the rainfall totals at gauge sites i and j. Note that h is an approximate distance used with a lag tolerance δh. The assumption is that the rainfall field is stationary. This means that the relationship between rainfall at two locations depends only on the relative positions of the sites and not on where the sites are located. This relative position refers to distance and orientation.

You further assume that the variance of rainfall between sites depends only on their lag distance and not on their orientation relative to one another (isotropy assumption). Smaller variances are expected between nearby sites compared with variances between more distant sites. The variogram is the inverse of the spatial correlation function (correlogram). You expect larger correlation in rainfall amounts between sites that are nearby and smaller correlation between sites farther apart. This means, for instance, that if gauge, has a large rain total, nearby sites will also tend to have large totals (small difference).

By definition, $\gamma(h)$ is the semivariogram and $2\gamma(h)$ is the variogram. However, for conciseness, $\gamma(h)$ is often referred to as the variogram. The variogram is a plot of the semivariance as a function of lag distance. Since your rainfall values have units of centimeters, the units of the semivariance are square centimeter.

The sample variogram is computed using the `variogram` function. The first argument is the model formula specifying the rainfall column from the data frame and the second argument is the data frame name. Here `~1` in the model formula indicates your assumption of no trend in the rainfall field. Trends can be included by specifying coordinate names (e.g., `~lon+lat`). Recall that although you specified a planar projection, the coordinate names are not changed from the original geographic latitude and longitude.

You compute the sample variogram for Fay's rainfall and save it by typing

```
> v = variogram(tpm ~ 1, data=FR)
```

To see the variogram values as a function of lag distance, use the plot method on the variogram object.

```
> plot(v)
```

The result is shown in Figure 9.10. Values start low (50 cm^2) at short lag distances, then increase to over 200 cm^2 at lag distances of about 200 km (Fig. 9.10). The zero-lag semivariance is called the "nugget" and the semivariance level where the variogram values no longer increase is called the "sill." The lag distance to the sill is called the "range." These three parameters (nugget, sill, and range) are used to model the variogram.

Semivariances are calculated using all pairs of rainfall observations within a lag distance (plus a lag tolerance). The number of pairs (indicated below each point on the plot) varies with lag distance. There are more pairs at short range.

You check on the assumption of isotropy by plotting sample variograms at specified azimuths. For example, separate variograms are estimated for four directions (0, 45, 90, and 135°) using the argument `alpha` in the `variogram` function, where 0° is

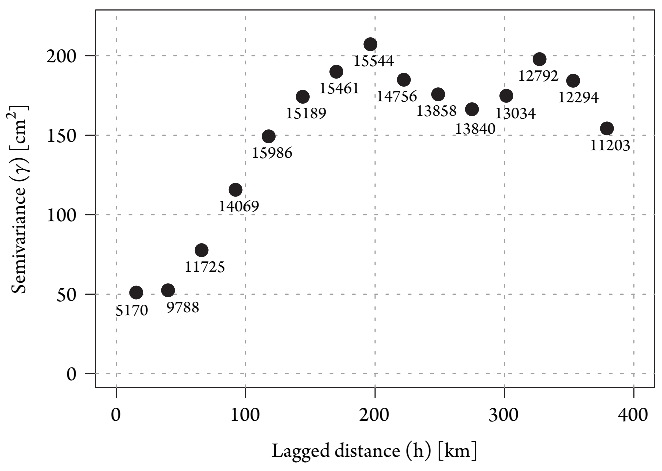

Figure 9.10 Empirical variogram of storm-total rainfall from tropical cyclone Fay.

north–south. These directional variograms restrict observational pairs to those having similar relative orientation within an angle tolerance (think of a pie slice). If the directional variograms look similar to each other, then the assumption of isotropy is valid.

9.6.3 Variogram Model

Next you fit a model to the sample variogram. The variogram model is a mathematical relationship defining the semivariance as a function of lag distance. There are several choices for model type (functions). To see the selection type `show.vgms()`. The nugget is shown as an open circle. The Gaussian function ("Gau") appears reasonable since it increases slowly at short lag distances then more rapidly at larger distances before leveling off at the sill.

You save the function and initial parameter values in a variogram model object by typing

```
> vmi = vgm(model="Gau", psill=150, range=200 * 1000,
+   nugget=50)
```

The `psill` argument is the partial sill as the difference between the sill and the nugget. You get eyeball estimates of the parameter values from your sample variogram.

You then use the `fit.variogram` function to fit the model to the sample variogram. Specifically, given the Gaussian function and initial eyeball estimated parameter values, a weighted least-squares method improves these estimates. Note that ordinary least-squares is not appropriate in this case as the semivariances are correlated across the lag distances and the precision on the estimates varies depending on the number of site pairs for a given lag.

```
> v.fit = fit.variogram(v, vmi)
> v.fit
  model psill  range
1   Nug  46.8      0
2   Gau 157.1 128666
```

The result is a variogram model with a nugget of 46.8 cm^2, a partial sill of 157.1 cm^2, and a range of 128.7 km. You plot the variogram model on top of the sample variogram by typing

```
> plot(v, v.fit)
```

Let r be the range, c the partial sill, and c_o the nugget, then the equation defining the curve over the set of lag distances h in Figure 9.11 is

$$\gamma(h) = c\left(1 - \exp\left(-\frac{h^2}{r^2}\right)\right) + c_o \tag{9.9}$$

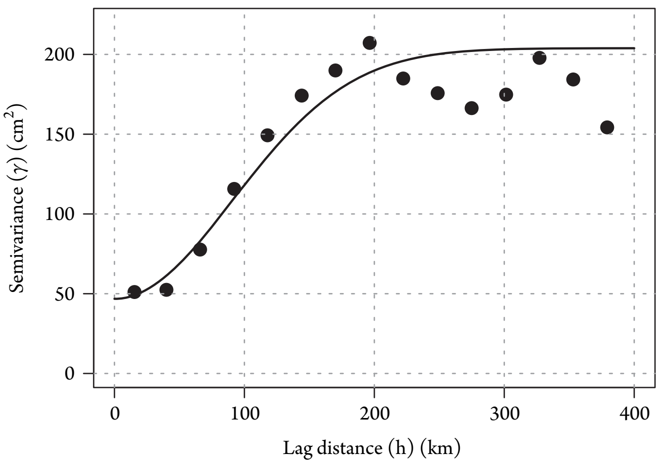

Figure 9.11 Gaussian variogram model for Fay's rainfall across Florida.

There are a variety of variogram functions. You can try the spherical function by replacing the `model="Gau"` with `model="Sph"` in the earlier `vgm` function.

```
> vmi2 = vgm(model="Sph", psill=150,
+    range=200 * 1000, nugget=50)
> v2.fit = fit.variogram(v, vmi2)
> plot(v, v2.fit)
```

The **geoR** package contains the `eyefit` function that can make choosing a function easier; however, the interpolated values are not overly sensitive, assuming a reasonable choice.

9.6.4 Kriging

The final step is to use the variogram model together with the rainfall values at the gauge sites to create an interpolated surface. The process is called kriging. As Edzer Pebesma notes, "krige" is to "kriging" as "predict" is to "prediction." Here, you use ordinary kriging as there are no spatial trends in the rainfall. Universal kriging is used when trends are present.

Interpolation is done using the `krige` function. The first argument is the model specification and the second is the data. Two other arguments are needed. One is the variogram model using the argument name `model` and the other is a set of locations, identifying where the interpolations are to be made. This is specified with the argument name `newdata`.

Here you interpolate first to locations (point kriging) on a regular grid and then to the county polygons (block kriging). To create a grid of locations within the boundary of Florida, type

```
> grd = spsample(FLpoly, n=5000, type="regular")
```

You specify the number of locations using the argument `n`. The actual number will be slightly different because of the irregular boundary. To view the locations simply, use the plot method on the grid object. More options for plotting are available if the spatial points object is converted to a spatial pixels object. You do this by typing:

```
> grd = as(grd, "SpatialPixels")
```

First use the `krige` function to interpolate (predict) the observed rainfall at the grid locations. For a given location, the interpolation is a weighted average of the rainfall across the entire region where the weights are determined by the variogram model.

```
> ipl = krige(tpm ~ 1, FR, newdata=grd, model=v.fit)
[using ordinary kriging]
```

The function recognizes the type of kriging being performed. Inverse distance-weighted interpolation is performed if the variogram model is left out. The function will not work if there are multiple values at a given location.

The object (`ipl`) inherits the spatial pixels object specified in the `newdata` argument, but extends it to a spatial pixels data frame by adding a data slot. The data slot is a data frame with two variables. The first `var1.pred` is the interpolated rainfall and the second `var1.var` is the prediction variance.

You plot the interpolated field using the `spplot` method. You specify an inverted topographic color ramp to highlight in blue regions with the highest rain totals.

```
> spplot(ipl, "var1.pred", col.regions=
+   rev(topo.colors(20)), sp.layout=l3)
```

The map (see Fig. 9.12) makes it easy to see that parts of east central and north Florida were deluged by Fay.

You use block kriging to estimate rainfall amounts within each county. The county-wide rainfall average is relevant for water resource managers. Block kriging produces a smoothed estimate of this area average, which will differ from a simple arithmetic average over all sites within the county.

You use the same function to interpolate but specify the spatial polygons rather than the spatial grid as the new data.

```
> ipl2 = krige(tpm ~ 1, FR, newdata=FLpoly,
+   model=v.fit)
[using ordinary kriging]
```

You then map the results using the same plot arguments as before.

```
> spplot(ipl2, "var1.pred", col.regions=
+   rev(topo.colors(20)))
```

Rainfall maps using point and block kriging are shown in Figure 9.12. The overall pattern of rainfall from Fay featuring the largest amounts along the central east coast and over the eastern panhandle is similar on both maps.

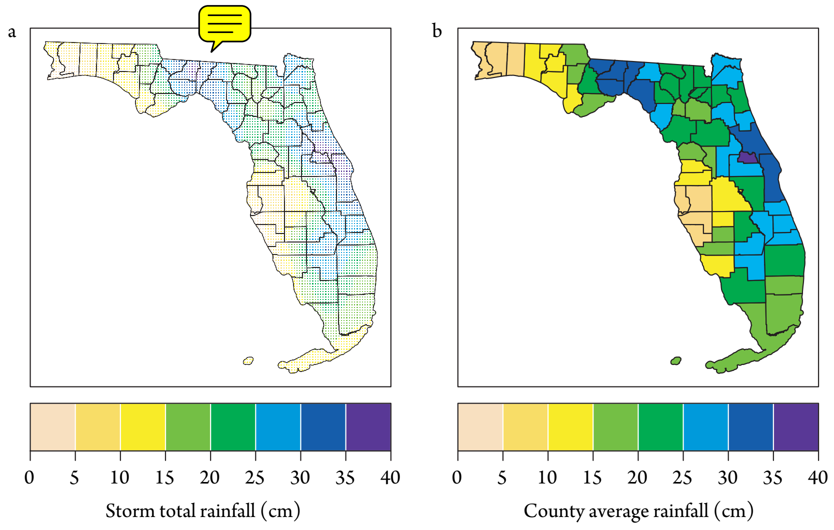

Figure 9.12 Rainfall from tropical cyclone Fay. (a) Point and (b) block kriging.

You compute the arithmetic average of county-wide rainfall, again using the `over` function by typing

```
> ipl3 = over(x=FLpoly, y=FR, fn=mean)
```

The function returns a data frame of the average rainfall in each county. The statewide mean of the kriged estimates is 20.8 cm, which compares with a statewide mean of the arithmetic averages of 20.9 cm. The correlation between the two estimates across the 67 counties is 0.87. The variogram model reduces the standard deviation of the kriged estimate (7.77 cm) relative to the standard deviation of the simple averages (9.93 cm) because of the smoothing.

9.6.5 Uncertainty

An advantage of kriging as a method of spatial interpolation is the accompanying uncertainty estimates. The prediction variances are listed in a column in the spatial data frame saved from your application of the `krige` function. Variances are smaller in regions with a greater number of rainfall gauges. Prediction variances are also smaller with block kriging as much of the variability within the county gets average out. To compare the distribution characteristics of the prediction variances for the point and block kriging of the rainfall observations, type

```
> round(summary(ipl$var1.var), 1)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   47.4    48.8    49.6    50.6    50.8   179.0
> round(summary(ipl2$var1.var), 1)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    0.5     1.4     2.0     2.4     2.8     8.5
```

The median prediction variance for your point kriging is 49.6 cm^2, which is close to the value of the nugget. By contrast, the median prediction variance for your block kriging is a much smaller 1.9 cm^2.

Simulations exploit this uncertainty and provide synthetic data for use in deterministic models. A rainfall–runoff model, for instance, can be run with simulated rainfall fields providing a realistic representation of the variation in the amount of runoff resulting from the uncertainty in the rainfall field (Bivand et al., 2008).

Conditional simulation, where the simulated field (realization) is generated given the data and the variogram model, is done using the same `krige` function by adding the argument `nsim` to specify the number of simulations. For a large number of simulations, it may be necessary to limit the number of neighbors in the kriging. This is

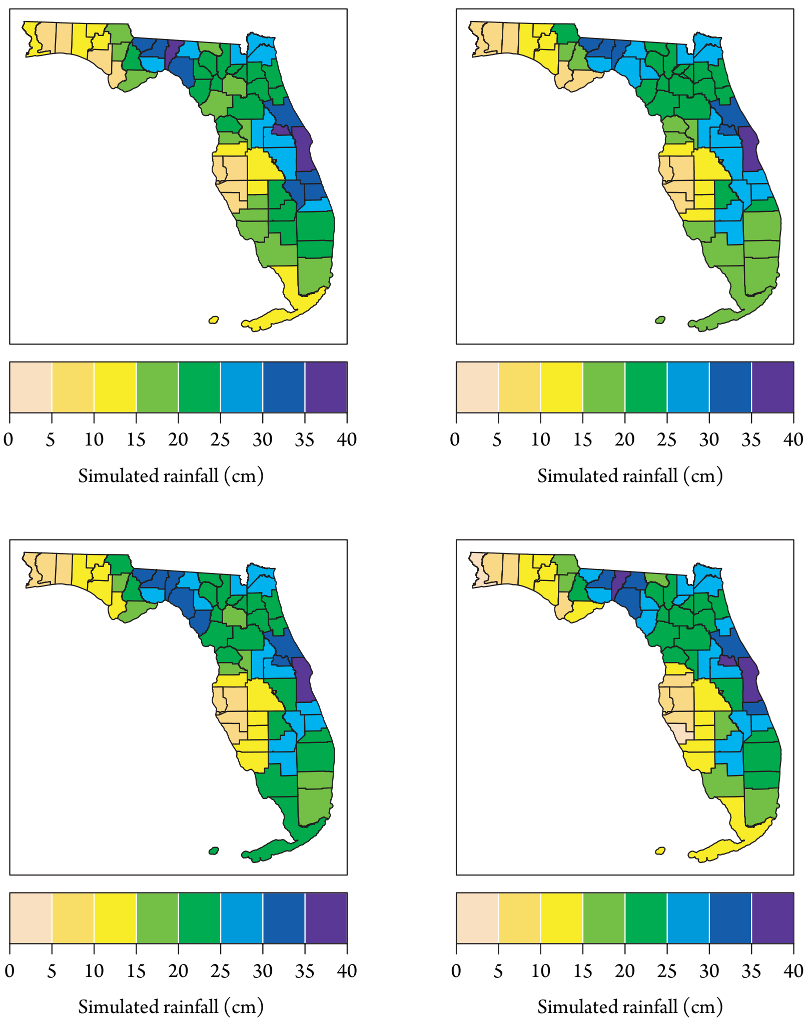

Figure 9.13 Rainfall simulations from tropical cyclone Fay.

done using the `nmax` argument. For a given location, the weights assigned to observations far away are very small, so it is efficient to limit how many are used in the simulation.

As an example, here you generate four realizations of the county-level storm total rainfall for Fay and limit your neighborhood to 50 of the closest observation sites. Note that it may take a few minutes to finish processing this function.

```
> ipl.sim = krige(tpm ~ 1, FR, newdata=FLpoly,
+   model=v.fit, nsim=4, nmax=50)
```

Maps of the four realizations are shown in Figure 9.13. Simulations are conditional on the observed rainfall and the variogram model using block kriging on the counties. Note that the overall pattern of rainfall remains the same, but there are differences especially in counties with relatively few observations and where the rainfall gradients are steep. The functionality is limited to Gaussian simulation, but the **RandomFields** package (Schlather, 2011) provides additional simulation algorithms.

In summary, kriging is a statistical method for spatial data interpolation involving three steps. First, you estimate a sample variogram that describes the spatial autocorrelation structure of your observations. This step includes checking for trends and isotropy. Second, you determine a variogram model using the method of weighted least squares. This step involves a somewhat subjective choice of the model, but the interpolation is not too sensitive to this choice assuming that it is reasonable. Third, you use the variogram model together with your observations to interpolate values at specified locations, on a grid, or over an area. Finally, simulation methods generate synthetic data for expressing and propagating uncertainty associated with the spatial variability in the data and with model specification. Kriging is a nuanced process, but with practice, it can be an important tool in your toolbox.

This chapter demonstrated ways to analyze and model hurricane data spatially. We began by showing how to combine hurricane track data with climate data. We then showed how to analyze the data including estimating the spatial autocorrelation. This was followed by a demonstration of how to use GWR to model hurricane intensity across space. Quantifying and mapping the changing relationship between the response variable and covariates can lead to new insights about how hurricanes operate across space. This research area is wide open. Finally, we showed how to perform a spatial interpolation using a prediction method from geostatistics. In the next chapter, we will look at some ways to analyze and model time series data.