Spatial analysis lab

- data sets etc. from Waller and Gotway, *Applied Spatial Statistics for Public Health Data*
- another intro to spatial regression

```
library(nlme)         ## geostatistical model fitting
library(RandomFields) ## simulating Gaussian random fields
library(spdep)        ## SAR/CAR
library(foreign)      ## for read.dbf
library(spatstat)     ## point processes
library(deldir)       ## Delaunay/Dirichlet/Voronoi
library(ggplot2); theme_set(theme_bw())
```

Simulating spatial data

Being able to simulate spatial data is an important component of learning to understand spatial data, as well as a good testing ground for trying out methods. In order to simulate data, we have to decide on:

- **distribution of sample points** (located on an even grid, or on line transects, or irregularly spaced: irregular samples may be completely spatially random, or follow some specified spatial point process, or be specified according to some real set of map coordinates)
- **trend model**, if any; the simplest case is a linear (x-y) trend, or polynomial, or ...
- spatial distribution and dependence on **covariates**;
- **spatial autocorrelation structure** (we'll use the `RandomFields` package)
- **conditional distribution**, if not Normal.

This is a *lot* of choices to make, but in less artificial situations your choices will be much more constrained by the question you're asking and what you know about your biological system.

Distribution of sample points

Your sample points might be on an evenly spaced grid, e.g.

```
dx <- dy <- 0.5           ## grid spacing
xmax <- ymax <- 10        ## grid max (could also set ymin)
```

```
xvec <- seq(0,xmax,by=dx)
yvec <- seq(0,ymax,by=dy)
ddgrid <- expand.grid(x=xvec,y=yvec)  ## all x*y combinations
plot(y~x,data=ddgrid)                 ## pretty boring!
```

Alternatively, you might want to completely spatially random (CSR) sample points: you can do this by sampling `x` and `y` uniformly over the sample space.

```
npts <- 100
dd <- data.frame(x=runif(npts,max=xmax),
                 y=runif(npts,max=ymax))
plot(y~x,data=dd,xlim=c(0,10),ylim=c(0,10))
```

One last possibility is to simulate the sampling points from a more complex spatial point process. There are many different point process models, some regularly spaced (i.e. "self-avoiding", "underdispersed", or more even than CSR) and some clustered ("contagious", "overdispersed"). We'll try the *Strauss* process, which simulates evenly spaced points: the parameters are `beta` (density of points), `r` (radius of avoidance), and `gamma` (strength of avoidance: `gamma==1` means no avoidance, or a CSR process, while `gamma==0` gives a *hard core* process where points can never fall within distance `r` of each other).

```
mod01 <- list(cif="strauss",       ## specify point process; see ?rStrauss
              par=list(beta=2,     ## intensity (average density)
                       gamma=0.2,  ## strength of avoidance
                       r=0.7),     ## radius of avoidance
              w=c(0,10,0,10))      ## square window; x={0,10}, y={0,10}
mm <- rmh(model=mod01,
          start=list(n.start=200), ## starting number of points
          control=list(nrep=5000),   ## number of stoch iterations
          verbose=FALSE)
dd <- data.frame(x=mm$x,y=mm$y)
plot(y~x,data=dd,xlim=c(0,10),ylim=c(0,10))
```

If you like, experiment with the parameters of the Strauss process ...

advanced/challenge topic If you want to your sampling area to be an *irregular* polygon, e.g. you want randomly spaced points within North America, use the `sp` package to load in polygon data from a shapefile or other map data and use the `spsample` or `makegrid` functions.

Trend model

Now that you've picked your sample points, you can specify a spatial trend (if any). This is probably the easiest part of the problem; for a simple linear trend you just have to decide on slopes in the x and y directions, e.g.

```
tpars <- list(int=1,x=2,y=4)  ## trend parameters
dd <- transform(dd,r0=tpars$int + tpars$x*x+tpars$y*y)

ggplot(dd,aes(x,y,colour=r0))+geom_point(size=4)
```

- Try using `size` as an aesthetic rather than colour, or in addition to colour (you'll have to take out the explicit `size=4` specification in `geom_point`); what do you think makes it easier to see the pattern?
- Try breaking the y axis into intervals (`dd$xcat <- cut_interval(dd$x,5)`) and plotting a lines for each category:

```
ggplot(dd,aes(y,r0,colour=xcat))+geom_line()
```

(*lots* more possibilities here, e.g. use `geom_point()+geom_smooth(method="lm",se=FALSE)` to get linear regression lines for each x-category)

It's not too hard to see that this is regular, but of course it's *much* easier if we are looking at data on a grid:

```
ddgrid <- transform(ddgrid,r0=tpars$int + tpars$x*x+tpars$y*y)
ggplot(ddgrid,aes(x,y,fill=r0))+geom_tile()
```

advanced/challenge topic Interpolating data from an irregular sample is a non-trivial task. Kriging is one possibility (see `krige` in the `sp` package), but that requires you to fit a geostatistical model in the first place. Probably the easiest method is to use `interpp` from the `akima` package; see if you can create an image (`geom_tile()`) plot from the interpolated data.

Covariates

To keep things a tiny bit simpler, I'm going to assume that our covariate doesn't have any spatial pattern. Once we learn (in the next section) how to generate spatially autocorrelated values, we could make our covariates spatially autocorrelated too ...

Assume precipitation is Normally distributed with some parameters that won't make it go negative ...

```
dd$precip <- rnorm(nrow(dd),mean=8,sd=2)
dd$r1 <- dd$r0 + 1.5*dd$precip  ## effect of covariate on response
```

Autocorrelation structure

Here's the interesting bit.

First we set up a spatial autocorrelation model: a Gaussian spatial autocorrelation (i.e. the correlation falls off with distance r as $C(r) = \exp(-(r/s)^2)$) with scale (s) equal to 1.5 and variance 4, along with a nugget variance of 0.5 (you can also add a trend here with `RMtrend`):

```
m <- RMgauss(var=4, scale=1.5)+RMnugget(var=0.5)
ss <- RFsimulate(m,x=dd$x,y=dd$y)  ## ignore message you get here ...
plot(ss)
```

(If you want to use `RFsimulate` for gridded data such as `ddgrid`, you should specify `grid=TRUE` to make it much more efficient.)

Extract the values from `ss` and add them to what we've got so far:

```
dd$r2 <- dd$r1 + ss@data$variable1
```

Conditional distribution

Now suppose we wanted the data to be Poisson distributed rather than Normal. The standard way to do this is to start with values like those we've generated and transform them appropriately so they can be mean values of a Poisson. In particular, we would generally use a *log link* (or an *exponential inverse-link* model): if what we've computed so far is η, we'd use $y \sim \text{Poisson}(\exp(\eta))$. (For binary or binomial response we'd use a logit or probit link.) The only problem here is that the actual range of the data we've generated so far (about 7-70) is a bit ridiculous for exponentiating, so we'll scale it down:

```
dd$r3 <- rpois(nrow(dd),dd$r2/10)
```

The pattern we have now is much more irregular – practically speaking, we've buried our original trend in noise:

```
ggplot(dd,aes(x,y,colour=r3))+geom_point(size=4)
```

We've also added so much noise that the precipitation signal is not particularly clear:

```
ggplot(dd,aes(precip,r3))+geom_point()+
    geom_smooth(method="glm",family=poisson)
```

The `RandomFields` has an (obsolete) `Variogram()` function, which interferes with `nlme`'s function of the same name, so we'll unload the package ...

```
detach("package:RandomFields")
```

Constructing weight matrices

```
library(spdep)
```

```
nyfile <- system.file("etc/misc/nydata.dbf", package="spdep")
nydata0 <- read.dbf(nyfile)  ## from the foreign package
```

Use `head` or `View` (or RStudio's data viewer) to look at the resulting data. Check out `?nydata` for more explanation of the data set.

To use the data we have to specify which columns of the data frame contain the spatial coordinates:

```
nydata <- nydata0  ## make a copy to turn into a 'sp' object
coordinates(nydata) <- c("X", "Y")  ## set coordinates
nycoord <- coordinates(nydata)      ## retrieve coordinates
## or: nycoord <- nydata[,c("X","Y")] *before* setting coordinates(nydata)
```

Take a preliminary look at the data, coding the number of cases by size and the exposure by colour:

```
ggplot(nydata0,aes(x=X,y=Y,size=PROPCAS,colour=PEXPOSURE))+
       geom_point(alpha=0.5)
```

The "neighbour-list" (`nb`) and "weight list" (`listw`) structures are the basic components that we need in order to do basic weight-matrix operations (compute Moran's I, do simultaneous or conditional autoregressions ...)

Optional You can explore the available functions for converting to and from these neighbour-lists and weight-lists, and doing things with them:

```
## use regular expressions! ^='beginning of line', $='end of line'
apropos("(^nb2|2nb$)")       ## convert to or from neighbour-lists
apropos("(^listw2|2listw$)") ## convert to or from weight-lists
methods(class="nb")          ## methods for neighbour-lists
methods(class="listw")       ## methods for weight-lists
```

Construct a Delaunay tesselation graph of nearest neighbors.

```
library(deldir)
nynb <- tri2nb(nycoord)  ## convert X,Y coordinates to a neighbour list
                         ## via Delaunay triangulation
```

(Ignore the warning messages.)

To plot a neighbour-list object, you need to specify the object as well as a set of coordinates for the nodes of the graph:

```
plot(nynb,nycoord)       ## plot the resulting neighbour-graph
```

Try computing a weight matrix based on a distance threshold instead:

```
nymat2 <- as.matrix(dist(nycoord))<20
nymat2[] <- as.numeric(nymat2) ## convert without losing matrix structure
## note 'dist' can also use alternative distance metrics such as 'manhattan'
listw_NY <- mat2listw(nymat2)  ## convert matrix to weight list
plot(listw_NY,nycoord,col=adjustcolor("black",alpha=0.5))
```

You can also use a pre-computed adjacency matrix:

```
nyadjfile <- system.file("etc/misc/nyadjwts.dbf",package="spdep")
nyadjdat <- read.dbf(nyadjfile)
```

(Ignore all the messages generated because R is trying to make the fields unique . . .)

If you try `image(nyadjmat,col=0:1)` you'll see that spatial areas are *not* self-neighbours (the diagonal is blank), but they are arranged in an order so that areas are mostly neighbours with other areas that are close to them in the list (most of the filled-in squares are near the diagonal).

The first column of `nyadjdat` is an ID column; drop it and convert the data frame to a matrix, and check that the IDs derived from `nyadjdat` are the same as the IDs in `nydata`:

```
nyadjmat <- as.matrix(nyadjdat[,-1]) ## first column is an ID variable
ID <- names(nyadjdat)[-1]
## check that area keys and IDs are the same ...
identical(substring(ID, 2, 10), substring(as.character(nydata$AREAKEY), 2, 10))
```

```
## [1] TRUE
```

Convert the matrix to a weight list, and add the weights list to the neighbour list we already set up (`style="B"` is a simple binary coding; there are other options for standardizing the weights by column or by row).

```
nyadjlw <- mat2listw(nyadjmat, ID)
summary(nyadjlw)
listw_NY <- nb2listw(nyadjlw$neighbours, style="B")
plot(listw_NY,nycoord)
```

Now we're (almost) ready to try some spatial analysis! First, take a quick look at the association between exposure and number of cases, without worrying about spatial autocorrelation:

```
ggplot(nydata0,
       aes(x=PEXPOSURE,y=PROPCAS))+
    ylim(0,0.003)+  ## restrict y-axis (omits one value at 0.007)
    geom_point(aes(size=POP8),alpha=0.5)+
    geom_smooth(aes(weight=POP8),method="loess")
```

Looks like a linear regression might be reasonable.

Fit a linear model weighted by population size:

```
lm1 <- lm(PROPCAS~PEXPOSURE,weights=POP8,data=nydata0)
summary(lm1)
```

Finally we can test Moran's I on the residuals – we can (just barely) reject the

```
lm.morantest(lm1,listw_NY)  ## reject null (just ...)
```

```
##
##  Global Moran's I for regression residuals
##
## data:
## model: lm(formula = PROPCAS ~ PEXPOSURE, data = nydata0, weights =
```

```
## POP8)
## weights: listw_NY
##
## Moran I statistic standard deviate = 1.685, p-value = 0.046
## alternative hypothesis: greater
## sample estimates:
## Observed Moran's I        Expectation           Variance
##           0.053705          -0.006257           0.001266
```

See also `?lm.LMtests` and `?moran.test`

What about a graphical examination of residuals?

`fortify(lm1)` produces a data frame that contains the data used in the linear regression, plus useful auxiliary information like the residuals (`.resid`): we need to add the spatial coordinates of the points too:

```
lm1Fort <- data.frame(fortify(lm1),
                      subset(nydata0,select=c(X,Y)))
```

Plot residuals in space with size proportional to magnitude, negative residuals in blue and positive residuals in red:

```
ggplot(lm1Fort,aes(x=X,y=Y))+
    geom_point(aes(size=abs(.resid),
                   colour=(.resid>0)),alpha=0.3)+
    scale_colour_manual(values=c("blue","red"))+
    scale_size(range=c(2,7))  ## make points a little bigger
```

- It's a bit hard to distinguish the clustering of the points themselves from a consideration of whether the residuals are autocorrelated in space ...
- The big residual near (X=-15,Y=40) is Syracuse (`nydata0[which.max(lm1Fort$.resid),]`: we should really do some non-spatial diagnostic plots first to find out what's going on ...

Finally, we can fit a spatial autoregressive model. There are two possibilities in `spdep`, `?lagsarlm` and `?errorsarlm`; to be honest, I don't understand the difference. In order to make the fit work well, we need to scale the variables:

```
summary(lagsarlm(scale(PROPCAS)~scale(PEXPOSURE),data=nydata0,listw=listw_NY))
```

The effect of exposure is still significant (shame on me), but much weaker (≈ 0.01 rather $\approx 10^{-5}$).

Geostatistical

WARNING: there's still quite a bit about this example that I don't understand as well as I'd like to.

A *quick* geostatistical fitting example with the `Wheat2` data set from `nlme` ... described in more detail in Pinheiro and Bates 2000 (Google books link)

Do a basic fit (as suggested by P&B):

```
library(nlme)
g1 <- gls(yield~variety-1,data=Wheat2,method="ML")
```

(Use maximum likelihood instead of restricted maximum likelihood – this is necessary if you want to compare models with different fixed effects (i.e. test hypotheses about fixed effects), and seems to make the stuff below behave better ...

Compute and plot the variogram of the residuals:

```
plot(Variogram(g1,form=~latitude+longitude))
```

The variogram doesn't seem to saturate at all, suggesting the data are *non-stationary* – we ought to try fitting a trend model:

```
g2 <- update(g1,.~.+latitude+longitude)
plot(Variogram(g2,form=~latitude+longitude,maxDist=32))
```

Now it seems to peak around 20. We fit a "spherical" correlation model, putting in our initial guess of a scale=20 and a nugget effect of 0.5:

```
g3 <- update(g2,correlation=corSpher(c(20,0.5),
                form=~latitude+longitude,nugget=TRUE))
## purple = smooth (nonparametric) line, blue = model fit
plot(Variogram(g3),xlim=c(0,35),
     smooth=TRUE,showModel=TRUE,ylim=c(0,2))
```

If we want to see what the variogram looks like *with the geostatistical model taken into account*, we use `resType="normalized"` instead of the default (`"pearson"`, which controls for heteroscedasticity but not autocorrelation).

```
plot(Variogram(g3,resType="normalized"),ylim=c(0,2))
```

This looks flat, which is good ...