Analysis of Spatial Point Patterns
Computing using Splus and splanes

Philip M. Dixon
Department of Statistics
Iowa State University
2 June 2003
pdixon@iastate.edu

Data sets are available at http://www.public.iastate.edu/~pdixon/spatial

Computing options

Stand-alone programs to estimate K(t)
written by various folks in various languages
do a google search to see what's currently available

EXCEL:
folks have written K(t) functions,
dge correction minimal

SAS:
Barry Moser has a K(t) macro,
www.stat.lsu.edu/faculty/moser/spatial/spatial.html

Splus:
Various add-on packages available, most are public domain
spatialStat: K(t), simulation of various processes
spatial: Venables and Ripley book, K(t), some modelling
splanes: K(t), bivariate K(t), kernel estimation, simulation,
space-time, www.maths.lancs.ac.uk/~rawlings/splanes
spatstat: K(t) even for non-convex or 'holey' regions
fits very general set of models incl. non-Poisson process

R: public domain version of Splus, easy to load packages
splanes and spatstat.
Rspat: (forthcoming) combination of splanes and spatstat.

Use whatever gets the job done fastest.
for me, because of inertia, that's splanes
if starting over, I might choose spatstat

We'll use Splus and splanes for the hands-on.
Reading point and boundary information
To enable the functions in splancs:

```r
library(splancs)
```

You must do this once every session.
The basic structure is a points object. This contains locations of events or locations of points on the boundary.
To read in the duck nest data:

```r
nests <- read.table("nest.txt", header=T)
dim(nests)
names(nests)
nests[1:10,]
nest.pnts <- as.points(nests)
nest.pnts <- as.points(nests[,1], nests[,2])
npts(nest.pnts)
nest.poly <- as.points(c(0,30,30,0), c(0,0,20,20))
# corners of region are (0,0) (30,0) (30,20) and (0,20)
```

Plotting polygons and points, other utilities

```r
par(pty='s')
polymap(nest.poly)
pointmap(nest.pnts, add=T)
pointmap(nest.pnts, pch='o')
```

There are also options for color (col=) and linetypes (lty=). These can be used in any plotting command.
splancs includes various utility functions:
  comparing points to polygons: inset, pip
  adding or deleting points interactively: addpoints, delpoints
  calculating polygon area or overall density: area, pdenes
Estimating K function, plotting

```r
defs <- seq(0.25,10,0.25)
defs.k <- khat(nest.pts,nest.poly,defs)

plot(defs,defs.k,type='l')

l <- function(k,s) {sqrt(k/pi)-s}

plot(defs, l(defs.k,defs), xlab='Distance',ylab='L(t)'
title('Duck nests')

# can get cute with the graphics using multiple figures:
par(mfrow=c(1,2))
par(pty='s')
points(nest.poly)
points(nest.pts,add=T)

par(pty='r')
plot(defs, l(defs.k,defs), xlab='Distance',
ylab='L(t)',type='l'
title('Duck nests')

# go back to one figure per page
par(mfrow=c(1,1))
```

---

Simulations

`spatstat` provides a default envelope function, `Kest`, but it only returns the largest and smallest of R simulations. I prefer to get the quantiles, so let’s do simulation “by hand”. There are 159 locations in the data, so each simulated data set has 159 observations.

```r
nsim <- 99
kall <- matrix(0,nrow=length(defs),ncol=nsim)
for (i in 1:nsim) {
  kall[,i] <- khat(as.points(runif(159,0,30),runif(159,0,20))
}

# each column of kall is the Khat for a random data set
# compute quantiles for each distance, i.e. row (index i)
k.q <- apply(kall,1,quantile,c(0.025,0.05,0.95,0.975))

# each row is now one quantile, t() will transpose the matrix
# plot obs and simulated K fn's
matplot(defs,cbind(defs.k,t(k.q)),type='l')

# or convert everything to L(t) and
# use lty= to make the line types informative
matplot(defs, l(apply(kall,1,k.q)),nest.s),
  type='l',lty=c(1,2,3,3,2))
```
Your turn

Try to repeat the calculations on a new data set. I suggest the cypress data set, or feel free to use your own.

How would you describe the spatial pattern of the cypress trees?

Does the K function analysis confirm your visual impression of the pattern?

Fitting a non-CSR process using the K function

This requires a bit more programming in Splus. nlminn() is a general purpose minimization function in Splus. nlminn() is not available in R. You need to use optim() instead.

Also, non-linear minimization is large part art. Lots of things can go wrong. I found it easier to estimate 1/p instead of p, the density of cluster parents. You need to provide starting values. Reasonable estimates are helpful. I plot curves for some trial values and see what looks close.

# define the K function for a Neyman-Scott process
Kclust <- function(t, v, invp) {
  pi*v*x + (1-exp(-t*v*0.25/v))*invp
}

Kclust <- function(t,v,invp) {Kclust(t,v,invp)}

# plot the l function, then try some values
plot(nes1, l(nest.k, nest.s))
lines(nes1, lclust(nest.s,3,5))
lines(nes1, lclust(nest.s,3,20))
lines(nes1, lclust(nest.s,5,20))

# define the closeness function
Klof <- function(theta, Kobs, t) {
  sum((Kobs-(0.25) - Kclust(t, theta[1], theta[2])*(0.25))^2)
}

# then fit to first 20 distances (through 5 m)
# the $parameters component contains the estimates
nsfit <- nlminn(start=c(3,20), objective= Klof,
               lower=0, Kobs=nest.k[1:20], t=nest.s[1:20])

names(nsfit)
nsfit$parameters
Estimating and Mapping intensity

\texttt{splancs} provides:
- \texttt{kernel2d()} 2D (i.e. spatial) smooth
- \texttt{nest2d()} estimate mse for specific bandwidth
- \texttt{kernel3d()} 3D (i.e. space time) smooth
- no \texttt{mse}! Have to see what "looks good"
- \texttt{kernel()} ratio of 2 kernel smooths (e.g. for cases and controls)

\texttt{# start by estimating mse for various bandwidths}
\texttt{nest.mse <- mse2d(nest.pnts,nest.poly,20,10)}
\texttt{plot(nest.mse,h,nest.mse$mse)}

\texttt{nest.lambda <- kernel2d(nest.pnts,nest.poly,3.5, nx=60, ny=40)}

\texttt{par(pty='s')}
\texttt{polymap(nest.poly)}
\texttt{image(nest.lambda,add=T)}
\texttt{pointmap(nest.pnts,add=T)}
\texttt{contour(nest.lambda,add=T)}

---

Modeling intensity: counts in boxes

This requires two steps:
- converting location data to counts in boxes
- fitting a Poisson regression model to the counts

\texttt{bin.do} on my web site is an Splus/R function to bin locations.
- it defines the bin() function, output is a matrix
- columns 1 and 2 are xy of the center, 3 is the count
- the grid=argument is the size of the box

\texttt{# calculate the counts}
\texttt{nest.bin1 <- bin(nest.pnts, nest.poly, grid=1)}
\texttt{nest.bin1[1:10,]}

\texttt{# calculate distance from box center to nearest edge}
\texttt{nest.dist <- bounddist(as.points(nest.bin1))}
\texttt{plot(nest.dist,nest.bin1[,3])}

\texttt{# calculate the mean count for each unique distance}
\texttt{tapply(nest.bin1[,3],nest.dist,mean)}

\texttt{# fit the Poisson regression}
\texttt{nest.fit1 <- glm(nest.bin1[,3] ~ nest.dist, family=poisson)}
\texttt{summary(nest.fit1)}

\texttt{# if desired, can add trend components or covariates, e.g.}
\texttt{nest.fit1a <- glm(nest.bin1[,3] ~ nest.dist+nest.bin1$x,nest.bin1$y, family=poisson)}
\texttt{summary(nest.fit1a)}
Modeling intensity: point process likelihood

This is hard. To do maximum likelihood, you need to install the adapt package to do adaptive numerical integration in 2 dimensions. This is easiest in R. Here’s my R code. It takes ca 20 minutes on my slow PC. The spatstat package has maximum pseudolikelihood methods that can quickly fit heterogeneous Poisson models.

# define the log likelihood function.
# beta are the parameters and f is the intensity function
poisslnl <- function(beta, f, x, maxpoly) {
  sum (log( f(x,beta))) -
  adapt(2,c(0,0),maxpoly,func=f,
    beta=beta,minpts=1200,maxpts=1201)$value }

# define the intensity function at any point x for param beta
# bounddist() is a function that returns distance to boundary
1.dist <- function(x,beta) {
  exp(beta[1]+beta[2]*bounddist(x)) }

# find the maximum log likelihood
ests.fit <- optim(c(-2.4,0.4), fn=poisslnl, method="BFGS",
  control=list(fnscale=-1), f=1.dist, x=nest.pts,
  maxpoly=c(30,20))

Your turn

Pick any one of the data sets, or your own.
Determine the appropriate bandwidth, then plot the intensity.

If you want to try to fit a theoretical K function, try fitting a Neyman-Scott process to the cypress data.

Modeling intensity is very data set specific.
Bivariate K functions

The "self" K functions, $K_{11}(t)$ and $K_{22}(t)$ are easy to estimate. Just use khat() on the appropriate subset of the data. k12hat() estimates the cross-K function.

```r
swamp <- read.table('swamp.txt', as.is=T, header=T)
swamp.pnts <- as.points(swamp)
table(swamp$p,p)

# subset the data set to get each species
swamp.td <- swamp.pnts[swamp$p=="TD",]
swamp.ns <- swamp.pnts[swamp$p=="NS",]

swamp.poly <- as.points(c(0,20,20,0),c(0,0,50,50))

#plot both patterns together
polyplot(swamp.poly)
mpoint(swamp.td,swamp.ns,add=T)

swamp.s <- 1:25
k.td <- khat(swamp.td,swamp.poly,swamp.s)
k.ns <- khat(swamp.ns,swamp.poly,swamp.s)
k.tdns <- k12hat(swamp.td,swamp.ns,swamp.poly,swamp.s)

# plot all 3 L functions together
par(pty="m")
matplot(swamp.s,1(cbind(k.td,k.ns,k.tdns),swamp.s),type='l')
abline(h=0)
```

Simulation using random labelling

Random labelling is done with rlabel(), but it is also easy by hand. Can use rlabel() with more than two processes.

```r
nsim <- 99
dall <- matrix(0,nrow=length(swamp.s), ncol=nsim)
dall[1:2all] <- dall

for (i in 1:nsim) {
  r <- rlabel(swamp.td,swamp.ns)
k11 <- khat(r[[1]], swamp.poly,swamp.s)
k22 <- khat(r[[2]], swamp.poly,swamp.s)
k12 <- k12hat(r[[1]], r[[2]], swamp.poly,swamp.s)
dall[i,1] <- k11-k22
dall[i,2] <- k11-k12
dall[i,3] <- k22-k12
}

dq <- apply(dall,1,quantile,c(0.025,0.05,0.95,0.975)
dq <- apply(d2all,1,quantile,c(0.025,0.05,0.95,0.975)

dq <- apply(dall,1,quantile,c(0.025,0.05,0.95,0.975)

dq <- apply(d2all,1,quantile,c(0.025,0.05,0.95,0.975)

# plot the quantiles and observed difference function
matplot(swamp.s,cbind(k.td-k.ns,t(dq)),type='l',
  lty=c(1,2,3,2)); title('Ktd-Kns');
matplot(swamp.s,cbind(k.td-k.tdns,t(dq)),type='l',
  lty=c(1,2,3,2)); title('Ktd-Ktdns');
matplot(swamp.s,cbind(k.ns-k.tdns,t(dq)),type='l',
  lty=c(1,2,3,2)); title('Kns-Ktdns');
Simulation using toroidal shifts

Toroidal shifting to test independence done with rtor.shift. If there are two types, you hold one fixed and shift the other. If you shift the entire pattern for one type, \( K_{11} \) and \( K_{22} \) never change. Only have to look at \( K_{12} \) and compare it to \( \pi \hat{c} \). \( L(i) \) is helpful here.

```r
nsim <- 99
k12all <- matrix(0,nrow=length(swamp.s), ncol=nsim)
for (i in 1:nsim) {
  r <- rtor.shift(swamp.td)
  k12all[,i] <- k12hat(r,swamp.ns, swamp.poly,swamp.s)
}
k12q <- apply(k12all,1,quantile,c(0.025,0.05,0.95,0.975))

# Plot L(i) for observed and quantiles
matplot(swamp.s,l(cbind(k.tdns,t(k12q))), swamp.s,type='l')
```

Your turn

Use the ash yellows data set. D indicates a diseased plant. H indicates a healthy plant.

Are diseased trees more clustered than healthy plants?
Are diseased and healthy plants spatially segregated?
Space-time data: plotting intensity

Mantel tests are possible in Splus/R, but they require additional packages. The vegan package includes dist() to compute distance measures. The ape package includes mantel() to do the randomizations. I would be happy to show you how to do this. Here’s how to plot the space-time intensity, then estimate and plot the space-time K function, using the rabies data as the example. Column 2 is the year, columns 3 and 4 are the x and y coordinates.

The kernel3D function does a 3 dimensional smooth of the intensity. The output is an array with (x,y,time) coordinates. A multiple panel figure is a great way to portray the spatial intensity over time.

```r
rabies <- read.table("rabies.txt",header=T)
rabies.pts <- as.points(rabies[,3],rabies[,4])
rabies.3d <- kernel3d(rabies.pts, rabies[2],
  seq(0,33,0.25),seq(0,33,0.25), 1:7, 3, 1)
par(mfrow=c(3,3))
for (i in 1:7) {image(rabies.3d$v[,i];
  title(paste("Year",i)))}
# following forces the same Z scale for all years
range(rabies.3d$v)
par(mfrow=c(3,3))
for (i in 1:7) {image(rabies.3d$v[,i], zlim=c(0,2))}
```

Additional features of splus

Splus can also:

- calculate and test nearest-neighbor and point-neighbor distances.
- simulate locations in an arbitrary shaped polygon under CSR.
- be used to simulate a cluster process (§5.4 of splus documentation).
- estimate raised incidence models, used in geographical epidemiology. These are models for 2 type processes (case and control). They describe the probability of being a case as a function of distance from a source and other covariates.

Space-time K function

The stkhat() function computes the spatial K(d), temporal K(t), and space-time K(d,t). The steskal() function estimates the standard error of the D values. The stmcst() function does a Monte-Carlo test of space-time independence. The stdiags() function produces a four panel plot of the results.

```r
kst <- stkhat(rabies.pts, rabies[,2], rabies.poly,
  c(1,7) rabies.s, 1:4)
names(kst)

kstse <- steskal(rabies.pts, rabies[,2], rabies.poly,
  c(1,7), rabies.s, 1:4)

kstnc <- stmcst(rabies.pts, rabies[,2], rabies.poly,
  c(1,7), rabies.s, 1:4, nsim=99)

stdiags(rabies.pts, kst, kstse, kstnc)
```