point 5: Space time point patterns

Philip Dixon

4/9/2020

Space-time point pattern visualization and analyses

These are illustrated using the Fox rabies data. This is a time sequence of reported cases in one area of Germany. The data document the spread of an epidemic of Fox rabies through the area. The original data given in Andrews and Herzberg, 1985, Data, are monthly from 1963 to 1970. I extracted the data for one month, arbitrarily chosen as April, to focus on the between-year dynamics.

The data in rabiesApr.txt have the location and year for each detection of Fox rabies. Locations are in a local coordinate system, but I believe the units are km. Years are recoded as 1 to 8. The approximate boundary of the study area is in rabiesPoly.csv.

spatstat has a class ppx for space-time point patterns, but methods (at least as of Apr 2020) are limited to printing, plotting, and intensity estimation.

Better packages for space-time analyses are:\ splancs: The first major package (1993) for point pattern analyses. Implements Diggle's K(s,t) based analyses.\ stpp: much newer package only for space-time point pattern analyses. Described in Gabriel et al. 2013. stpp: An R Package for Plotting, Simulating and Analyzing Spatio-Temporal Point Patterns. Journal of Statistical Software 53(2). http://dx.doi.org/10.18637/jss.v053.i02

These notes describe how to repeat the plots and analyses in part 6c of the 2020 lecture notes.

```
library(splancs)

rabies <- read.table('rabiesApr.txt', as.is=T, header=T)
rabies.poly <- read.csv('rabiesPoly.csv', as.is=T)

# Splancs point pattern data set
rabies.pts <- as.points(rabies$x,rabies$y)

# Splancs wants the border stored as a matrix
rabies.poly <- as.matrix(rabies.poly)
```

Splancs does very little data checking (much less than spatstat). Space-time data is stored as matrix of x and y coordinates with a separate vector for the time coordinates. Notice that the sampling window is not specified when the points object is created.

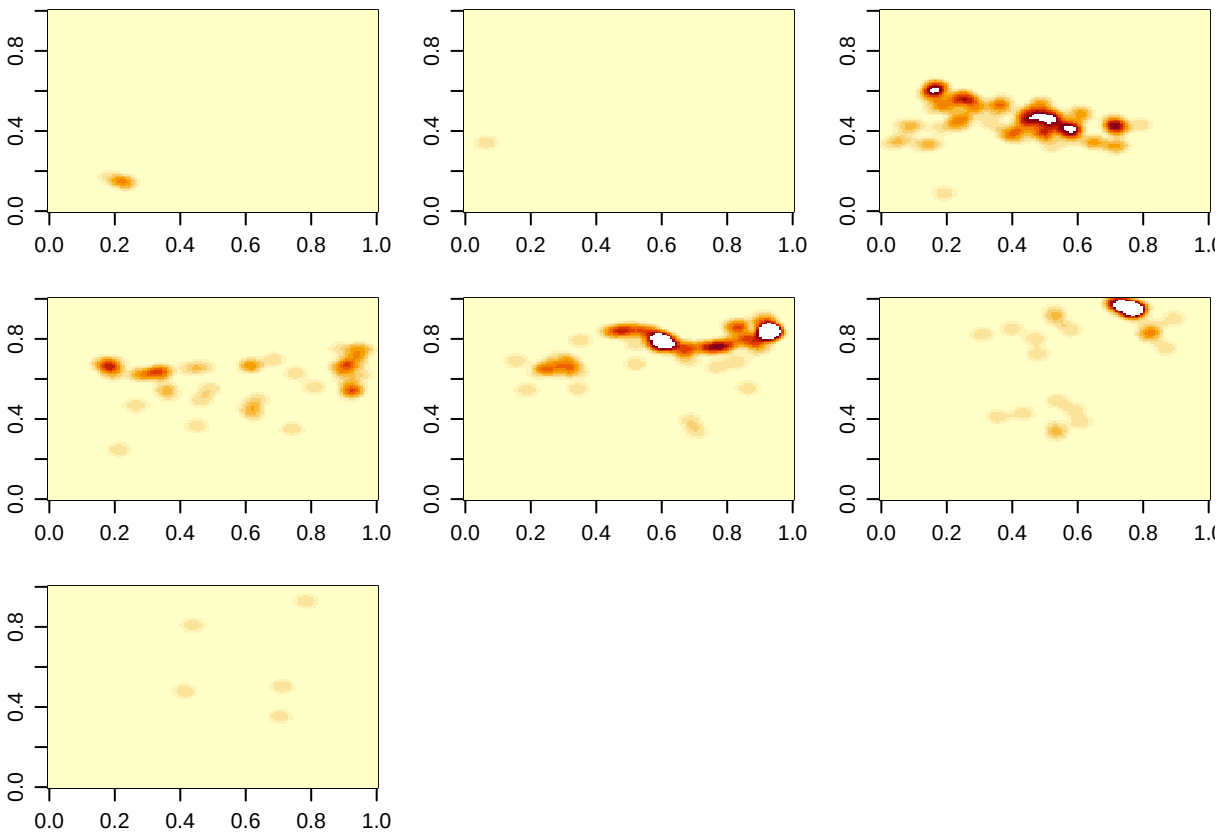
Estimating intensity in space-time

```
rabies.3d <- kernel3d(rabies.pts, rabies$year,
  seq(0,33,0.25), seq(0,33,0.25), 1:7, 2, 1)

par(mfrow=c(3,3), mar=c(3,3,0,0)+0.2, mgp=c(2,0.8,0))
range(rabies.3d$v)
```

```
## [1] 0.000000 4.042489
```

```
for (i in 1:7) {
  image(rabies.3d$v[,,i], zlim=c(0,2)) }
```



kernel3d() computes the 3D kernel smooth. The arguments are:

- the spatial points matrix
- the vector of times for each location
- 3 arguments given the grid for X, for Y, and for Time
- 2 arguments giving bandwidth for space and for time
  - here 2 units in space and 1 unit in time.

The result is a list with information about the grid (xgr, ygr, zgr), the bandwidths (hxy, hz), and the estimated intensity (v). v is a 3 dimensional array. The indices are x, y, and time. The image() plots each spatial array in sequence. zlim sets the bounds for the intensity. Although values go to 4, I clip at 2 so that we can differentiate lower values. You can also plot one spatial location over time by looking at v[i,j,].

Estimating K(s,t)

```
kst <- stkhat(rabies.pts, rabies$year,
  rabies.poly, c(1,7),
  seq(0,10,0.25), 0:4)
```

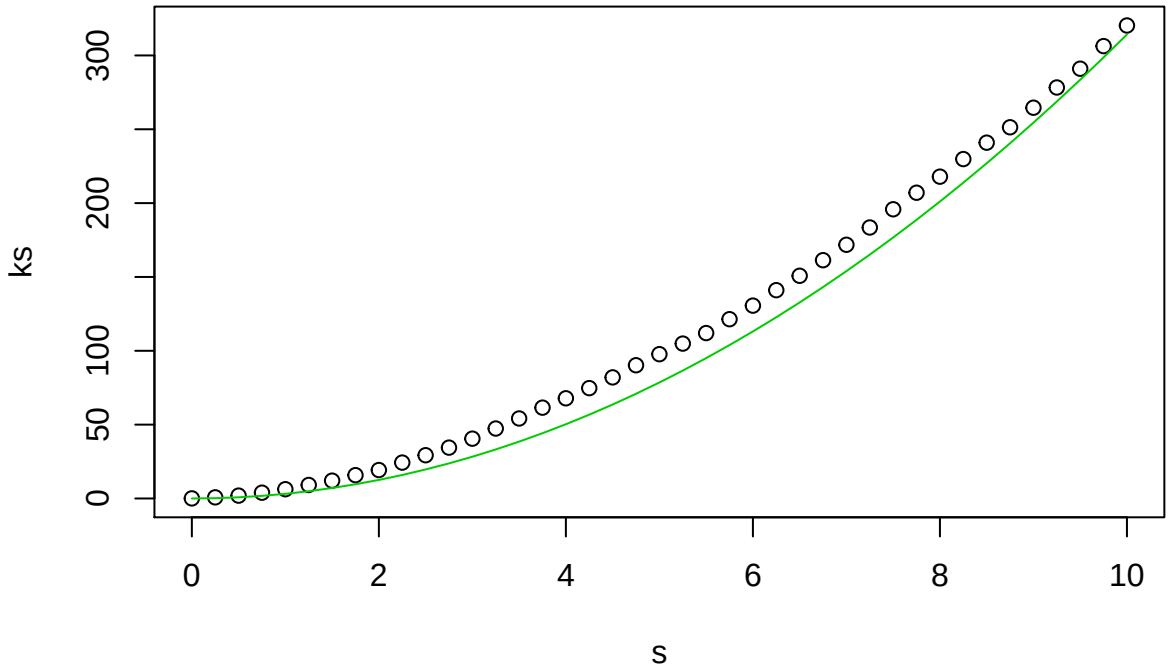
stkhat() estimates K(s,t). Arguments are, in order:

- the spatial locations
- the time stamps for each location
- the spatial boundary
- the temporal boundary
- Distances to evaluate K(s,t) at
- Time separations to evaluate K(s, t) at

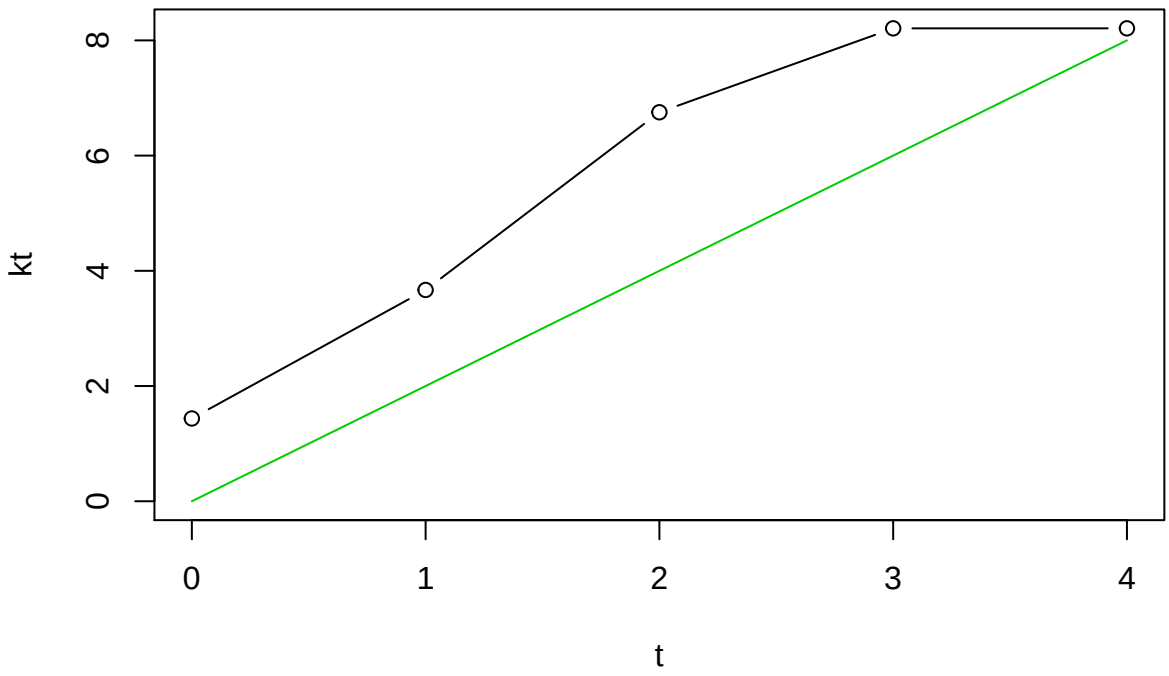
The result is a list with 5 components:

- Values of s (distance)
- Values of t (time separation)
- Estimated K(s), ignoring time. Uses Ripley edge correction
- Estimated K(t), ignoring space. Uses 1D edge correction
- Estimated K(s,t), matrix (space , time) of K(s,t)

```
# Spatial K
with(kst, plot(s, ks, type='b') )
with(kst, lines(s, pi*s^2, col=3) )
```

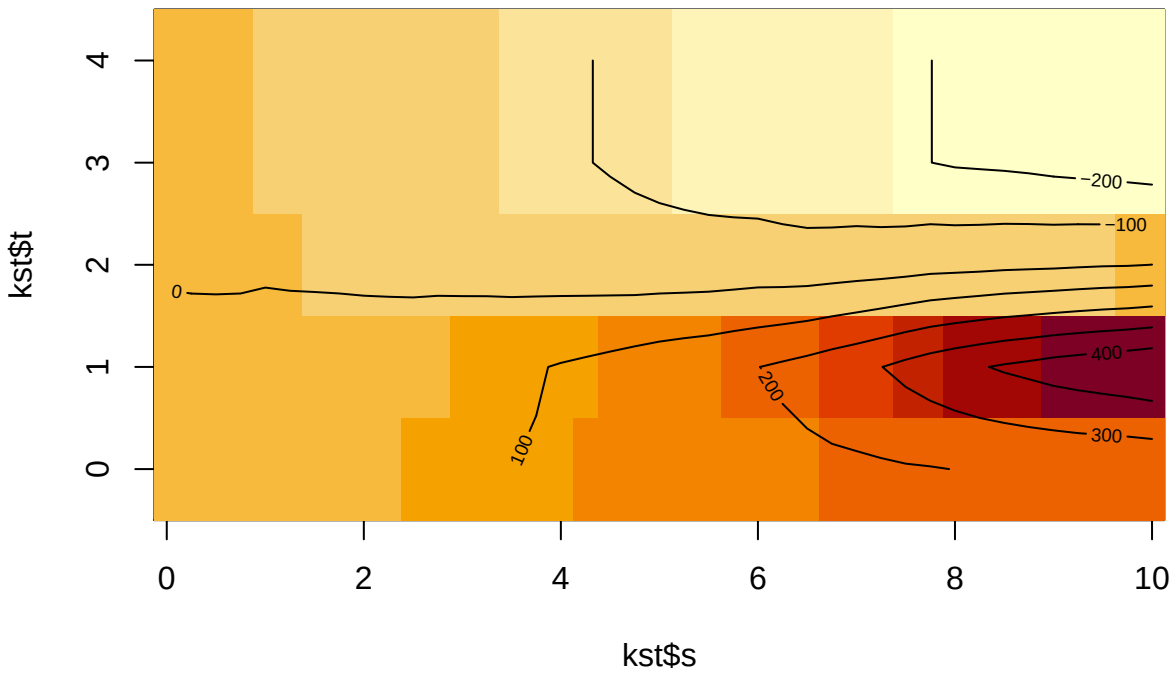



```
# Temporal K
with(kst, plot(t, kt, type='b', ylim=c(0, max(kt))) )
with(kst, lines(t, 2*t, col=3))
```



You can use the components of kst to compute Dst. outer() is the outer product function. This is the product of each element of the first object with each element of the second.

```
Dst <- with(kst, kst - outer(ks, kt))
image(kst$s, kst$t, Dst)
contour(kst$s, kst$t, Dst, add=T)
```



Image() is the base graphics function to draw an image plot. It is hard to add a legend, so I usually add contour lines.

Additional useful information is provided by stsecal() and stmctest() functions. stsecal() computes standard errors for K(s,t). stmctest() does a Monte-Carlo test of CSTR by randomizing times to locations. The summary statistic is the sum of standardized residuals.

```
kstse <- stsecal(rabies.pts, rabies$year,
  rabies.poly, c(1,7),
  seq(0,10,0.25), 0:4)
kstmc <- stmctest(rabies.pts, rabies$year,
  rabies.poly, c(1,7),
  seq(0,10,0.25), 0:4, nsim=99, quiet=T)
c(pvalue = (1+sum(kstmc$t >= kstmc$t0)) /
    (1+length(kstmc$t)) )
```

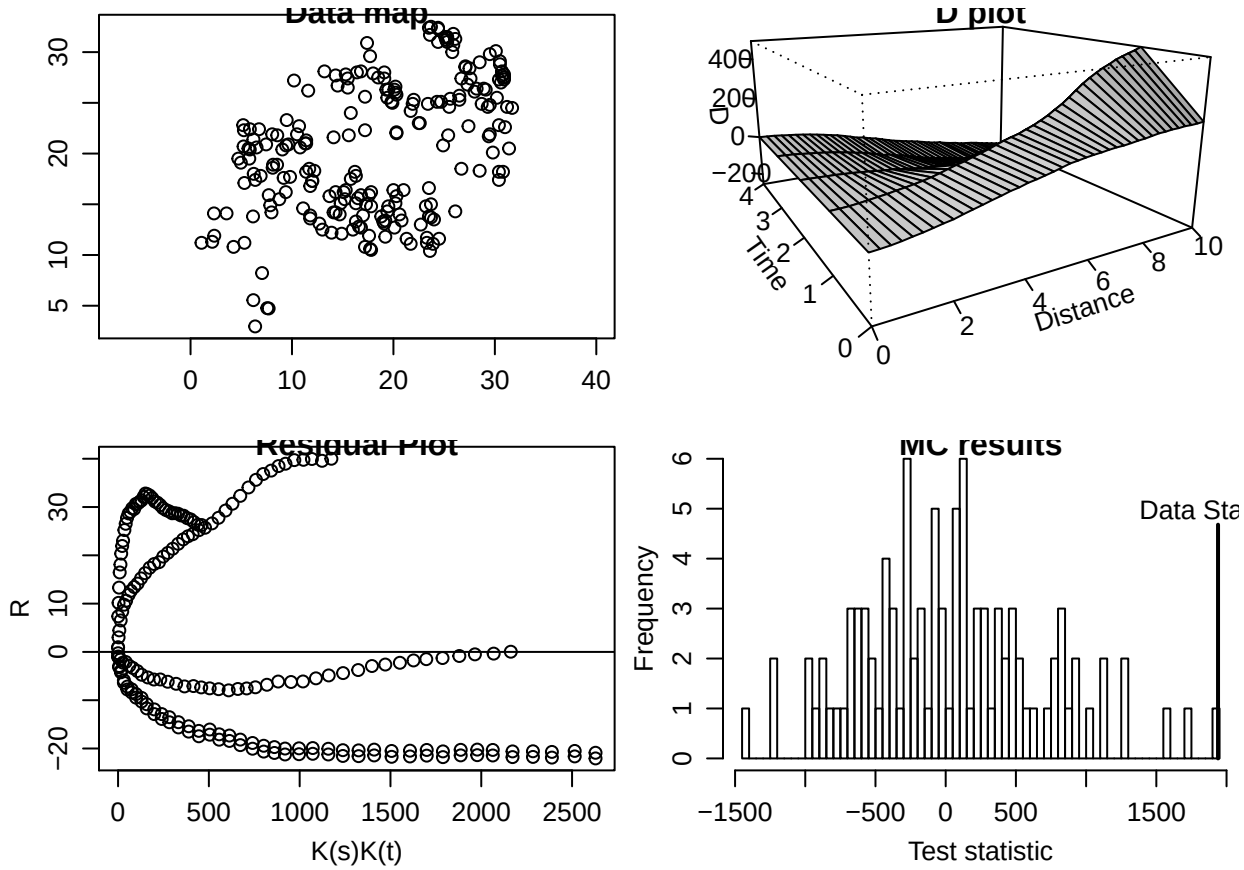
```
## pvalue
##   0.01
```

Both take the same arguments as stkhat(), except that stmctest() also requires the number of simulations. In addition, quiet = T suppresses printing a progress indicator.

stsecal() returns a matrix of the standard error each each estimate of K(s,t). stmctest() returns a list with two components. t0 is the summary test statistic for the observed data. t is a vector (length = nsim) giving the summary test statistics for each randomization. The p-value is computed in the usual way for a sample of randomizations.

stdiagn() draws three or four plots useful for evaluating space-time clustering.

```
par(mar=c(3,3,0,0)+0.2, mgp=c(2,0.8,0) )
stdiagn(rabies.pts, kst, kstse, kstmc)
```



stdiagn() requires three arguments and allows a fourth. In order, these are:

- the points object
- the estimated K functions
- the estimated standard errors
- the fourth argument is the MC test results

You get a 2x2 array of plots: the locations, a perspective plot of Dst(s,t), a plot of standardized residuals for each time. If you provide the MC results, you get a histogram of the test statistics with the observed value marked.