Introduction to the R Package Spatstat: Plotting and Exploring Marked Spatial Data

INTRODUCTION:

Having been developed for over 15 years and containing over 1000 commands, spatstat is one of the largest and most diverse packages that R offers to handle spatial point patterns. It can be used to create spatial maps in 1, 2 or 3D, explore spatial clustering or randomness, perform statistical analysis, and evaluate models.

Spatial point data contains (x,y) coordinate locations for each data point. Examples could include water samples along a river (1D), the location of flowers in a meadow (2D), or the location of tree species on the side of a mountain (3D). Each coordinate point can also have multiple values associated with it, called "mark" values. These can be categorical (age class, color, species), or continuous (elevation, height, diameter, parasite intensity, time, pH).

Note: this means that spatstat is used to analyse the distribution and patterns of spatial data over a given region. The mark data itself is not analysed, but instead is used to aid in the description of the spatial spread of the samples. Spatstat is also not designed to deal with fixed and/or repeated measures (weather stations, one animal sampled over a year).

This tutorial will describe how to make a variety of descriptive figures using an example 2D spatial data set containing mark data. Then exploratory statistics will be shown, including functions that define the degree of spatial clustering.

TO BEGIN:

Data needs to be organized in columns, containing (x,y) coordinate values in separate columns to denote location. Each mark value follows after in separate columns (X, Y, Z1, Z2, Z3…). Next, save the file as a CSV.

The example dataset used for this tutorial contains trapping location and parasite load of 2 external parasites and 3 internal parasites for a colony of Columbian ground squirrels (*Urocitellus columbianus)*. The data is formatted as follows:

```
    ID No. sex day Season Time MP ABS  ORD Weight AC Age Rep Lact Obs ...
1 7958   1   2 120   2015 1142  A 106  -82    340  1   9  14    2  BB ...
2 7958   0   2 121   2015 1205  A 104  -86    350  1   9  14    2  BB ...
3 7958   0   2 122   2015 1033  A 104  -86    370  1   9  14    2  BB ...
4 7958   0   2 123   2015  918  A 120  -82    380  1   9  16    2  BB ...
5 7958   0   2 125   2015 1201  A 125 -108    370  1   9  17    2  TH ...
6 7958   0   2 130   2015 1125  A 105  -85    440  1   9  17    2  TH ...
```

1. To begin, start R and read in the data set using *setwd* and *read.csv.* Change the working directory *"C:/Users/Me/DataFiles"* to the location of your file, and change *"mydata.csv"* to the name of your CSV file. The dataset will be named as data. The format of the data can be checked by running *head(data)* .

```
> setwd("C:/Users/Me/DataFiles")
> data <- read.csv("mydata.csv",header=T,stringsAsFactors=F)
> head(data)
```

2. Install and run spatstat using *install.packages* and *library.* Typing in *beginner* will open up an introductory help page. Use the code *help(spatstat)* if you want information or help with spatstat, or view example tutorials in R (called "vignettes"). Alternatively, *help(Functionname)* can be used to look up more information on any function used in this tutorial.

```
> install.packages("spatstat")
> library(spatstat)
> help(spatstat)
> beginner
```

MAKING A POINT PATTERN OBJECT:

1. Now that the data is read into R, we need to figure out the spatial dimensions of the data set. There are multiple ways to do this. Using base R code, we can *sort* x and y into unique values. To do this, replace *columnx* and *columny* with the name of your x and y columns. Make sure the x/y range exceeds the minimum and maximum numbers reported. Alternatively, spatstat contains two functions that can tell you how big to make your windows. *Bounding.box.xy* will calculate the smallest size rectangle that can enclose a set of data. *Convexhull.xy* is similar but will calculate the smallest possible shape.

```
> sort(unique(data$columnx))
> sort(unique(data$columny))
> bounding.box.xy(data$column, data$columny)
> convexhull.xy(data$columnx, data$columny)
```

2. Now use the command *ppp* to make the data into a point pattern object. Point pattern objects are arranged in terms of *(x.coordinates, y.coordinates, x.range, y.range)*. The values for the smallest enclosing rectangle (see above) were [-30, 280] x [-131, -1], so I will make my x range *(-40,290)* and my y range *(-140,0)*. This ppp object will be named *pointdata*.

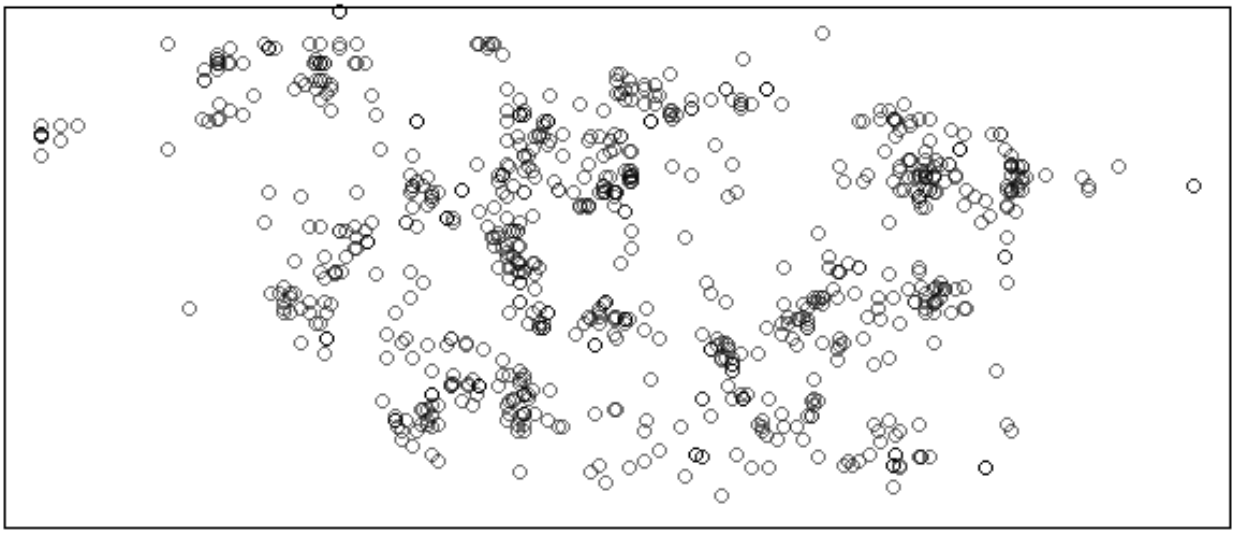
If this returns the warning "data contains duplicate points", the duplicate points can be identified using the *duplicated* function, or they can be removed using the function *unique(pointdata)*.

```
> pointdata<- ppp(data$columnx, data$columny, c(-40,290), c(-140,0))
> pointdata<-unique(pointdata)
> duplicated(pointdata)
```

MAKING PLOTS:

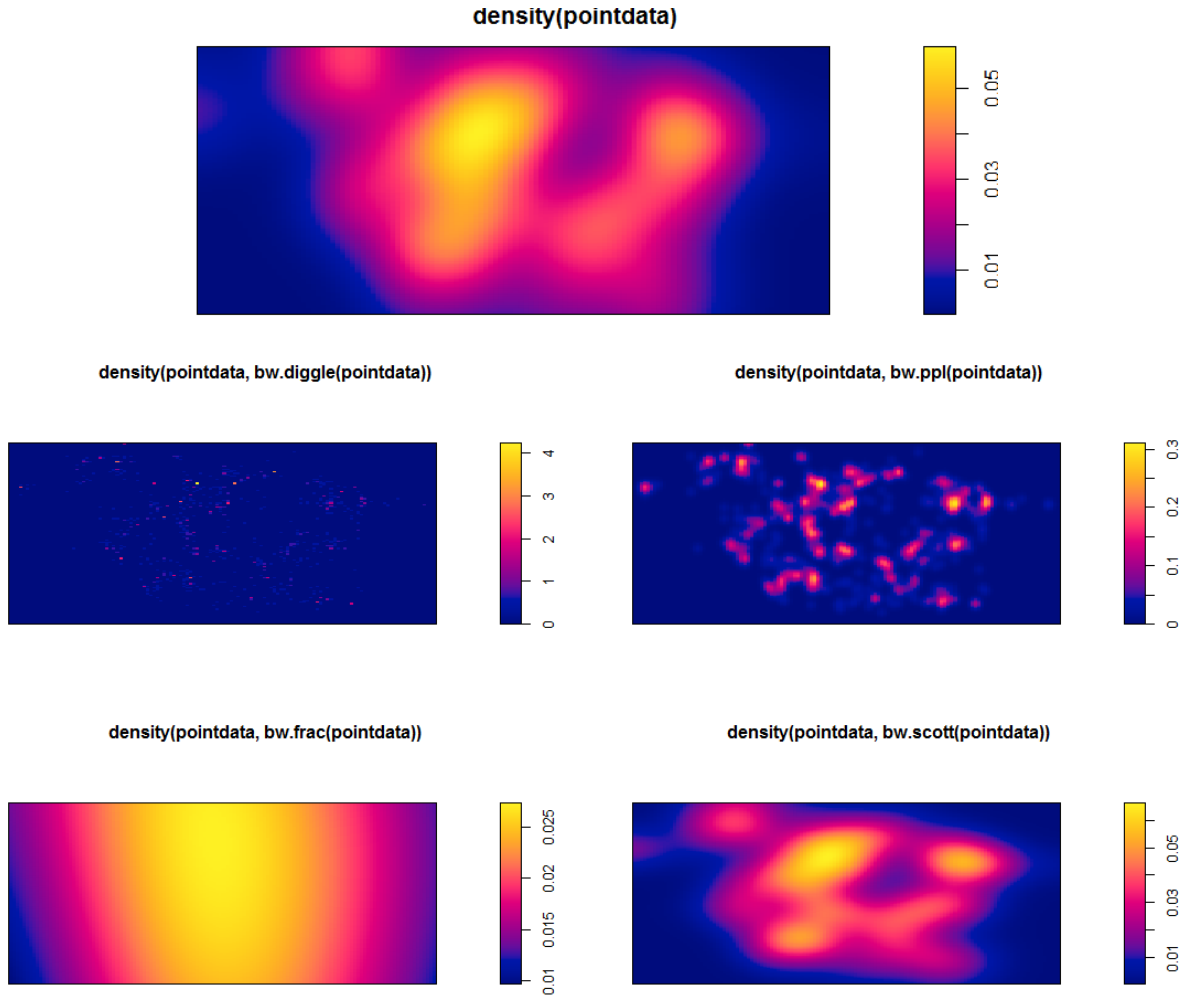
1. The most simple plot is done using *plot(pointdata)*, which shows the arrangement of the (x,y) coordinates in space. The plot default comes with a title, which can be altered by writing the desired title in *main=""*. The title can be removed by inserting a space between the quotation marks. The plots are also by default made with excess white space around the border, which can be removed by using the code *par(mar=rep(0.5, 4)*.

```
> plot(pointdata, main=" ", par(mar=rep(0.5, 4)))
```

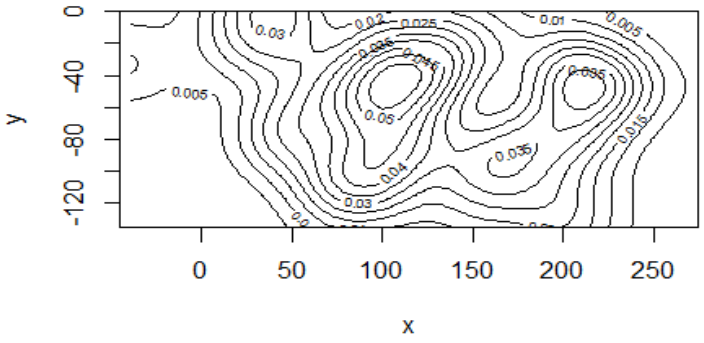


2. The function *density(pointdata)* when plotted will show the density of observations seen over the study area. The smoothing scale can be adjusted through the code *sigma*. If sigma is not specified, the best option will be automatically chosen. If another value for sigma is desired, sigma can be specified manually or calculated with several different functions. *bw.diggle(pointdata)* and *bw.ppl(pointdata)* are standard and commonly used options, *bw.frac(pointdata)* calculates a sigma value using the geometry of the spatial window, and *bw.scott(pointdata)* is useful for estimating gradual trends.

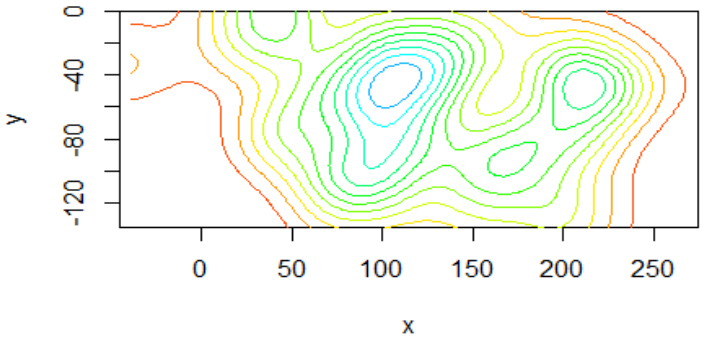
```
> plot(density(pointdata))
> par(mfcol=c(2,2))
> plot(density(pointdata, sigma=bw.diggle(pointdata)), par(mar=rep(2,4)))
> plot(density(pointdata, sigma=bw.frac(pointdata)), par(mar=rep(2,4)))
> plot(density(pointdata, sigma=bw.ppl(pointdata)), par(mar=rep(2,4)))
> plot(density(pointdata, sigma=bw.scott(pointdata)), par(mar=rep(2,4)))
```



3. Contour maps can be made by plotting the function *contour(density(pointdata))*. The function *drawlabels* when TRUE will put numerical values on each of the contour lines, while the function *axes* will label and display a scale on both axes. The lengths of the axes are determined based on a combination of the y-limit and x-limit values as well as the size of the graphic window. Because of this, it is suggested you use a pop-up window to plot these graphs, *windows(5,10)*, and then manually adjust the size of the pop up window until the axes are the correct size.



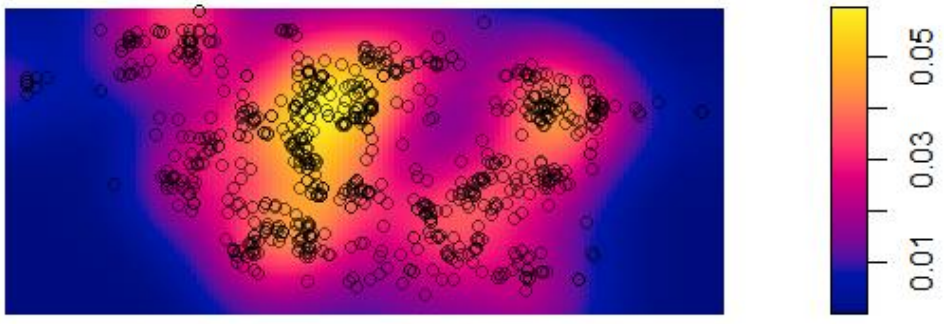
> windows(5,10)
> par(mfcol=c(2,1))
> contour(density(pointdata), drawlabels=TRUE, axes=TRUE, xlim=c(-40,260), ylim=c(-130,-5), main=" ")
> co <- colourmap(rainbow(20), range=c(0,0.1))
> contour(density(pointdata), axes=TRUE, xlim=c(-40,260), ylim=c(-130,-5), col=co, main=" ")



The second contour plot shows how to change the color of the contours using a pre-made color set, *colourmap(rainbow(20))*, with the colour changing per line *range=c(0,0.1))*.

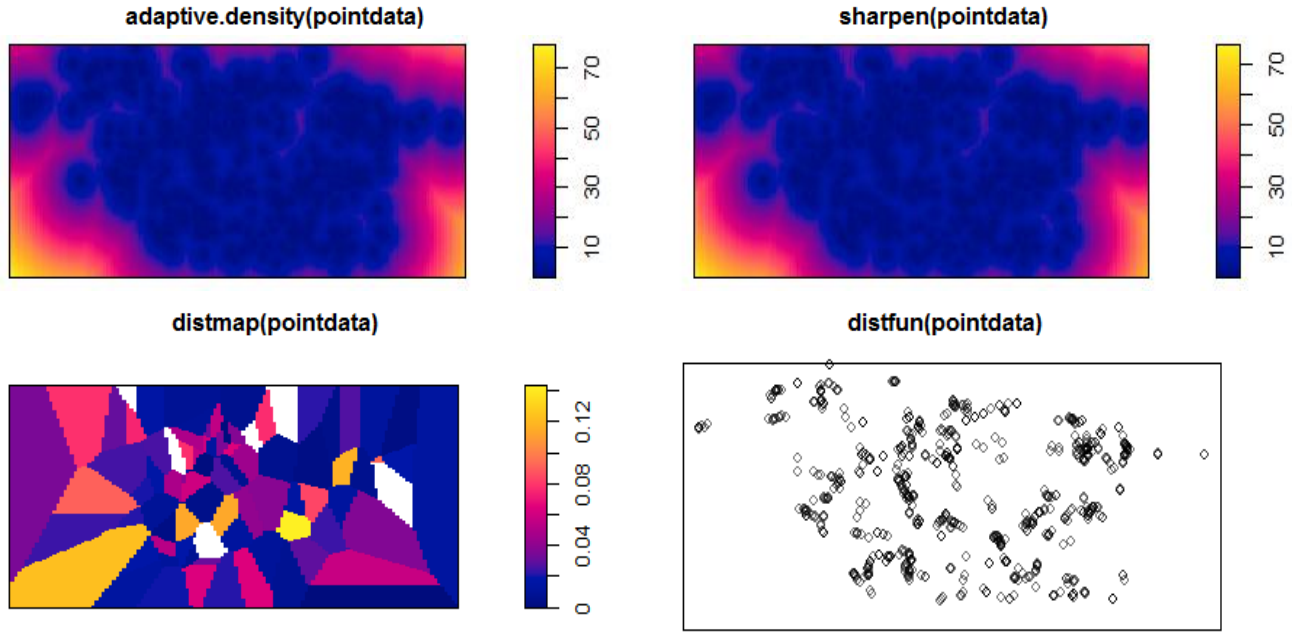
4. Multiple plots can also be *layered* on top of one another. The bottom layer is made first, and the upper layer is made last, as seen in the format *layered(bottom, top)*,

> plot(layered(density(pointdata), pointdata), main= " ", par(mar=rep(2,4)))



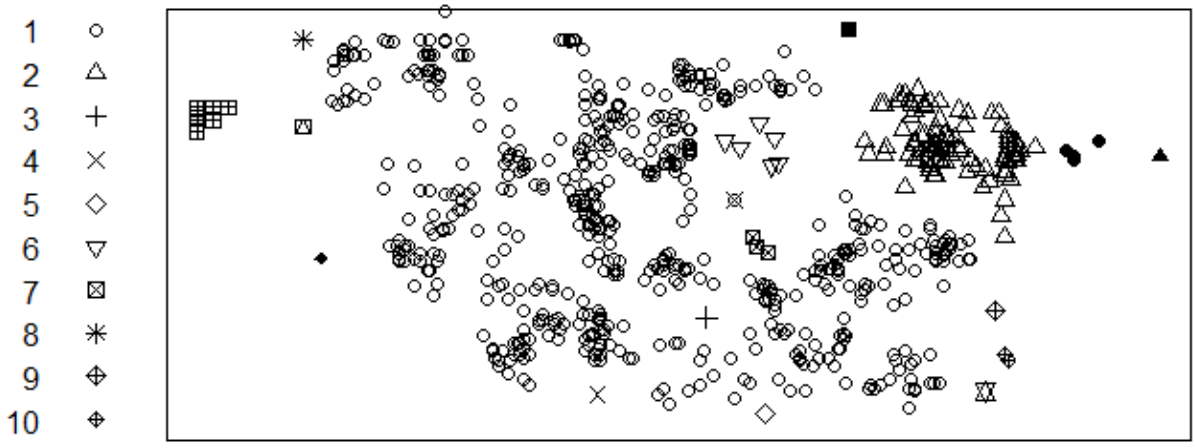
5. The rest of the plots are run similar to *density*. *Distmap* creates a smoothed plot based on the distance from the center pixel of each data point to the nearest data point. *Distfun* is almost an identical function, except it uses the exact location of each data point to calculate distances. Both *adaptive.density* and *sharpen* take sigma values like the *density* function. *Sharpen* is useful for data with dramatic increases in intensity, such as earthquake data.

> plot(distmap(pointdata), par(mar=rep(2,4)))
> plot(distfun(pointdata), par(mar=rep(2,4)))
> plot(adaptive.density(pointdata), sigma=bw.diggle(pointdata), par(mar=rep(2,4)))
> plot(sharpen(pointdata), bw.diggle(pointdata), main="sharpen(pointdata)", par(mar=rep(2,4)))



6. *Connected* will plot the data within r distance of each other, in this case, 10 meters. Each unique cluster of data points is plotted with its own symbol. This is a useful method for seeing clumping in the data.

> plot(connected(pointdata, r=10), main=" ", par(mar=rep(2,4)))



ADDING MARK DATA:

1. Marks can be added to a previously existing ppp object by specifying the column numbers to add from the original dataset. The numbers of the columns can be looked up by using the *colnames* function. The *marks* function and the code *data[,c(20,21,22)]* tells R to add the mark column data included on the 20th, 21st, and 22nd columns (external parasite load) to the previously existing ppp object *pointdata*. Alternatively, a new ppp object can be made by using the *ppp* function including a code specifying mark data by the column name or column number, as seen in *marks=data[,c(20,21,22)]*. Multiple columns of marks can be added to a ppp object, however most analysis and plot functions will treat each mark column as a separate set of data to analyse.

```
> colnames(data)
 [1] "ID"     "No."    "sex"    "day"    "Season" "Time"   "MP"     "ABS"
 [9] "ORD"    "Weight" "AC"     "Age"    "Rep"    "Lact"   "Obs"    "Mother"
[17] "Father" "Ecto"   "Num"    "fleas"  "nits"   "SM"     "other"  "Total"
[25] "Endo"   "NUM"    "Gram"   "eim"    "EIM"    "str"    "STR"    "asc"
[33] "ASC"    "Comm"
> marks(pointdata) <- data[, c(20,21,22)]
> extcomb<-ppp(data[,8], data[,9], c(-40,290), c(-140,0),  marks=data[,c(20,21,22)])
> extflea<- ppp(data[,8], data[,9], c(-40,290), c(-140,0), marks=data$fleas)
```

2. In order to do analysis and plotting with mark data, there must not be any NA values. In the example dataset, coordinates were recorded every time a squirrel was captured while parasite load was not. These missing values will need to be removed, which can be done using *subset*, and specifying which data column you want to subset.

```
> extcomb<-subset(extcomb, data[,20]!="NA" & data[,21]!="NA" & data[,22]!="NA",
drop=FALSE)
> extflea<-subset(extflea, data[,20]!="NA", drop=FALSE)
```

3. To output basic summary statistics of a marked ppp object, use the *summary* function.

```
> summary(extcomb)
Marked planar point pattern:  370 points
Average intensity 0.008008658 points per square unit

*Pattern contains duplicated points*

Coordinates are integers
```

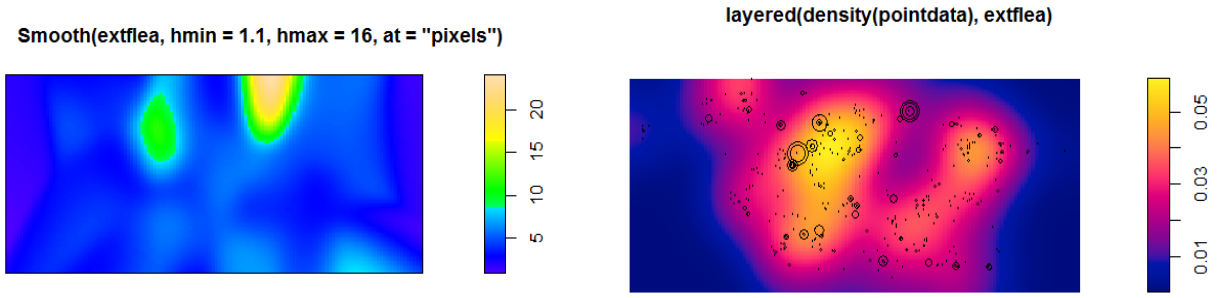
```
i.e. rounded to the nearest unit

Mark variables: fleas, nits, SM
Summary:
     fleas             nits              SM
 Min.   : 0.000   Min.   :  0.00   Min.   :  0.000
 1st Qu.: 0.000   1st Qu.:  0.00   1st Qu.:  0.000
 Median : 1.000   Median :  0.00   Median :  0.000
 Mean   : 4.973   Mean   : 21.45   Mean   :  5.084
 3rd Qu.: 5.000   3rd Qu.: 20.00   3rd Qu.:  3.000
 Max.   :75.000   Max.   :750.00   Max.   :200.000

Window: rectangle = [-40, 290] x [-140, 0] units
Window area = 46200 square units
```

4. To plot a spatial smoothing of mark data, use the *Smooth* function. Occasionally *Smooth* will output the warning "Cross-validation criterion was minimised at right-hand end of interval [1.1, 14]; use arguments hmin, hmax to specify a wider interval". If this happens, enter in the reported values for *hmin* and *hmax* into the *Smooth* function and slowly increase *hmax* until the warning disappears. This happened at *hmax=16*. This function also takes a sigma term similar to the function *density*. Colors can be changed using the *col* command.
   Using the function *plot* for marked ppp objects will create a figure with points that increase in size with numerical value of the marked data if mark data is continuous, or change the shape of the point character if not continuous. This type of figure can be seen layered on top of observation density in the right-hand figure below.

```
> plot(Smooth(extflea, hmin=1.1, hmax=16, at="pixels"), col=topo.colors(256))
> plot(layered(density(pointdata), extflea), par(mar=rep(2,4)))
```



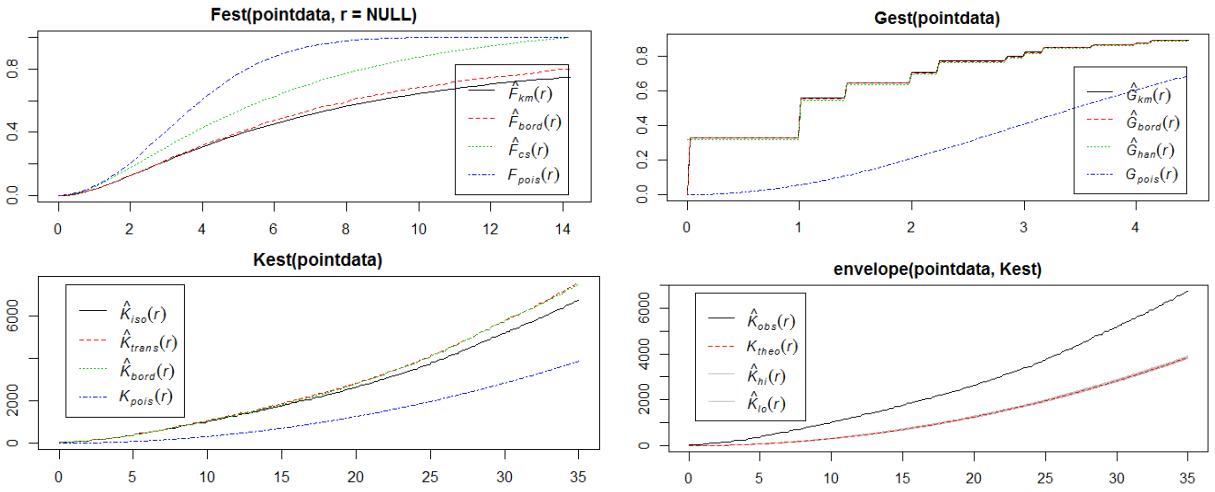
EXPLORATORY STATISTICS:

1. Basic exploratory functions include *intensity*, which reports the average number of points per grid section. *Nndist* will report the distance to the nearest neighbour for each data point, however, multiple neighbours can be specified. For example, *k=1:3* denotes the distances to the 1st, 2nd, 3rd nearest neighbour. *Nnwhich* will report the nearest data point for all data points. *Minnndist* and *maxnndist* will report the smallest and maximum distance seen between all data points. *Closepaircounts* will report the number of neighbours within a specified range (in this case 1 meter) for all data points. *Fardist* will report the farthest distance to the window boundary for all data points, while conversely *bdist.points* will report the nearest distance to the window boundary.

```
> intensity(pointdata)
> nndist(pointdata)
> nndist(pointdata, k=1:3)
> nnwhich(pointdata)
> minnndist(pointdata)
> maxnndist(pointdata)
> closepaircounts(pointdata,1)
> fardist(pointdata)
> bdist.points(pointdata)
```

2. There are plotting functions designed to explore the degree of spatial point clustering. There are different options depending on the type of data to be analysed. *Fest* analyses spatial clustering using a function of empty space. *Gest* uses a function of nearest-neighbour distance. *Jest* combines *Fest* and *Gest* into one function. *Kest* predicts the number of expected random values of a stationary spatial point process. *Kinhom* is a non-stationary alternative to *Kest*, which is useful for data with sharply varying intensities. *Lest* is a square root transformation of *Kest*.
   Estimate data (Ripley isotropic correction, translation-corrected and border-corrected) is compared against a theoretical Poisson trend line. If the estimated curves are above the theoretical Poisson trend line (as seen in all graphs below), the data is clustered. If the estimated curves fall below the theoretical Poisson, the data is random or regular. If the estimated curves differ greatly from each other (as seen in Fest), this means K is hard to estimate for your data as there are possibly too few data points or too irregular a data set. These functions can also be plotted using an *envelope*, which compares a shadowed theoretical curve showing the range for high and low estimates to the observed data. Interpretation is the same as the previous figures.

> plot(Fest(pointdata, r=NULL))
> plot(Gest(pointdata))
> plot(Jest(pointdata, r=NULL))
> plot(Kest(pointdata))
> plot(Lest(pointdata))
> plot(Kinhom(pointdata))
> plot(envelope(pointdata,Kest))



3. Finally, clustering and randomness can be explored using basic statistics. Clarkevans will return an R value using three different methods. Naïve is the most basic method, while cdf and Donnelly both correct for edge effects. If R is <1, it suggests the data is clustered, while an R>1 suggests randomness. Clarkevans.test will test that R is significantly different from 1.

> clarkevans(pointdata)
naive Donnelly cdf
0.4725837 0.4652562 0.4535103
> clarkevans.test(pointdata, correction="cdf")

Clark-Evans test
CDF correction
Monte Carlo test based on 999 simulations of CSR with fixed n

data: pointdata
R = 0.45351, p-value = 0.002
alternative hypothesis: two-sided