Spatial Analysis Techniques in R

Lesson 2: Point Pattern Analysis in R

2.1 Introduction

In this lesson we first revise some basic material and then go on to look at how the R environment can be used to analyze mapped patterns (dot/pin maps) of point objects/'events' using the spatstat package.

2.2 Maps as outcomes of spatial processes

When undertaking spatial statistical analysis with any type of mapped data, the key concept is that mapped patterns are regarded as *realizations* of an imagined or real *p*rocess that has some mechanism that creates spatial order by way of first- or second-order effects and leads to a non-random distribution of whatever it is we are interested in.

Task 2.1: Revision - Maps and spatial processes

Read Chapter 4 of O'Sullivan and Unwin *Geographic Information Analysis* (Second Edition), pages 93 -119. (Chapter 3 of the First Edition is more-or-less the same).

In Brunsdon and Comber's *An Introduction to R for Spatial Analysis and Mapping* the relevant sections are 6.1 to 6.6, pages 173-199. Their treatment is slightly more advanced and in addition has a useful section on a visualization technique called *hexagonal bining*.

All the material in this Lesson is also covered at even more advanced level in Chapter 7 *Spatial Point Pattern Analysis*, pages 155-172, of Bivand *et al*. (2008) *Applied Spatial Data Analysis with R* (NY: Springer). This text contains all you need to know and is an excellent, if terse and advanced, over-view of the field.

On the use of spatstat I would struggle to improve on Adrian Baddeley's own two-day course lecture notes, *Analyzing spatial point patterns in R*, a 12Mb download available at: www.csiro.au/resources/pf16h.html (checked 27-11-15)

2.3 Elementary point pattern description

Task 2.2: Describing a mapped point pattern

Read Chapter 5 of O'Sullivan and Unwin *Geographic Information Analysis* (Second Edition), Sections 5.1 and 5.2 pages 121-139, noting how point patterns can be described using 'centrography', density and distances. It is useful also to go forward and read Sections 6.8 and 6.9, pages 177-182, to see how an approach based on areas and neighborhoods can also be used.

(This is Chapter 4, Section 4.1 – 4.4, pages 77-95 of the First Edition, together with materials from pages 37-45 and 126-131)

2.4 Elementary point pattern tests against CSR

As the readings should have shown, the most often used benchmark process against which point patterns are evaluated is one that is stationary and has no first or second order effects. We call it complete spatial randomness (CSR). The expected statistics for this can sometimes be derived analytically (although seldom in a fully closed form) or can be simulated by using a random number generator to select (x, y) co-ordinate pairs for each event in the pattern. Statistical theory as such only comes into play when we compare the observed measures against what is expected if the pattern is CSR.

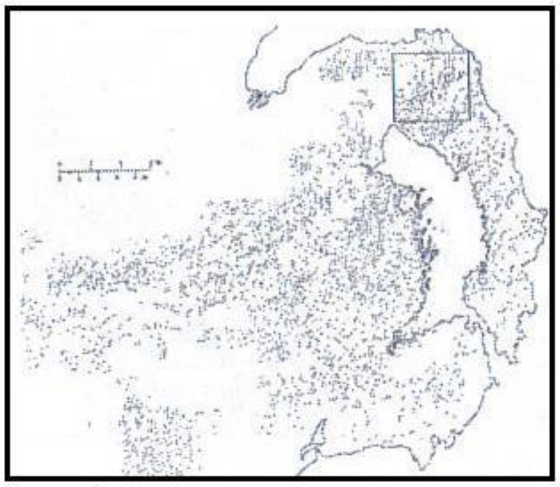
Task 2.3: Assessing point patterns statistically

Now read Sections 5.3 to 5.5, pages 139-154 of O'Sullivan and Unwin (2010). It will be useful to concentrate on the sections that deal with nearest neighbor distances and the $K(d)$ function (also known as Ripley's K). In view of our worked example in the next section of this Lesson, you might also read the 'cautionary tale' page 144-145 with some care. If you have a copy of the First Edition the same materials are in Section 4.5 pages 95-112.

2.5 A worked example using spatstat

One of the very first academic studies I was involved with was published as Smalley and Unwin (1968). In it, Ian Smalley and I suggested a mechanism by which the low streamlined conical hills called *drumlins* created by ice-sheet glaciation could have been formed. When they occur, as over much of Northern Ireland, drumlins seem to be in *swarms* creating what is called a *basket of eggs* landscape. The detail of our idea doesn't now matter, and has been superseded by other ideas, but our theory gave us no reason to think that within a drumlin swarm their distribution should be other than random, and we tested this prediction using data for some drumlins in the Vale of Eden (England) using the simple Clark and Evans nearest neighbor statistic (Clark and Evans, 1954).

A drumlin: ice went from right to left



Drumlins in Co Down, NI (Hill 1973) with the section of the swarm used indicated

I guess the most unusual feature of our work was that the hypothesis to be tested required that map pattern of events would be indistinguishable from CSR, whereas most geographical work has tended to be more interested in proving the alternatives of a more regular than random (dispersed) or more clustered than random (aggregated) pattern. The original paper stimulated a lot of similar work, of which the most recent is Clark (2010). With the benefit of over forty years of hindsight (hindsight is of course the most exact of the sciences), the original analysis was seriously flawed both in terms of its understanding of sub-glacial processes and its spatial statistics. However, data for a small section of the area in Ireland mapped by Hill (1973) digitized by Upton and Fingleton (1985) have become part of the common currency of spatial analysis and will be used in our simple worked example using the spatstat package.

Task 2.4: Are drumlins randomly distributed?

The Upton and Fingleton data are available as drumlins_in_unit_square.txt. Copy them into a convenient place on your machine and the use them to replicate the analysis that follows on your own machine. On my machine the absolute path to them was C:\\Documents and Settings\\David Un win\\Desktop\\R-results\\Lesson2\\drumlins_in_unit_square.txt. You should of course substitute your own path for this.

a) Getting the data

```
> library(spatstat)
>#  read in the data using an absolute reference and remembering the double solidus/backslash
```

```
> drumlin <- read.table("C:\\Users\\David\\Desktop\\drumlins_in_unit_square.txt", header=TRUE)
> drumlin
        X        Y
1   0.29846 0.966154
2   0.23154 0.943846
3   0.03462 0.930000
etc
232 0.88000 0.003077
> nrow(drumlin)
[1] 232
> summary(drumlin)
       X                Y
 Min.   :0.0000   Min.   :0.0000
 1st Qu.:0.2990   1st Qu.:0.1927
 Median :0.4988   Median :0.4400
 Mean   :0.5118   Mean   :0.4518
 3rd Qu.:0.7339   3rd Qu.:0.6933
 Max.   :1.0000   Max.   :0.9992
```

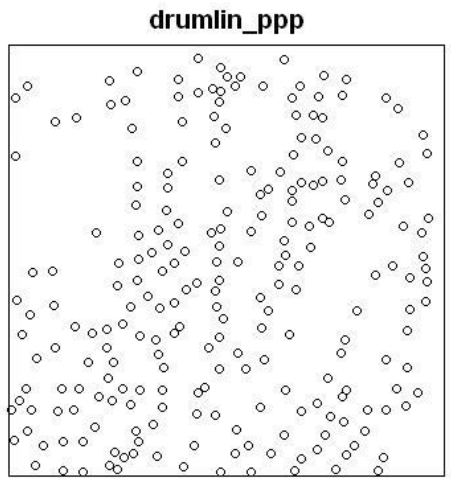
On the data file used I have deliberately scaled the coordinates along *X*- and *Y*-axes into the 'unit square'. Doing this with a point pattern can sometimes help comparisons since the distances, densities, and related functions become comparable between distributions. It isn't essential. The downside is that we sometimes have to convert back into real distance units.

Next, we define the bounding box for these data:

```
> xlim <- c(0.0, 1.0)
> ylim<- c(0.0, 1.0)
> attach(drumlin)
```

Next is the key command which creates a ppp object that spatstat can understand …

```
> drumlin_ppp <- ppp(X, Y, xlim, ylim)
> plot (drumlin_ppp)
```



At this point, and if you haven't already done this, you can save this picture using >file>save in the RGUI. Switching between an on-screen plot and the R-console is done using >window in the R-GUI. I usually save all the pictures in a folder on my desktop. Very frequently I will enhance these plots using some other software such as Microsoft *Paint* ™. (You can get the same picture using just plot(drumlin)).

b) Changing the bounding box

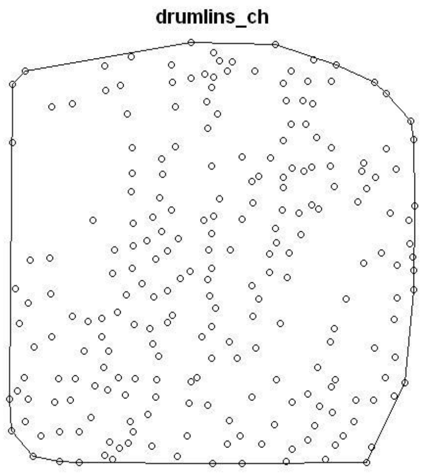
A major issue with almost all point pattern analysis is that the bounding box used affects the expected values under CSR and hence the decisions made about the pattern of events enclosed within it. The next section uses the original bounding box together with two other options. The first, ch, is based on the convex hull of the events and the second, rr, uses Ripley and Rasson's (1977) maximum likelihood estimate of the spatial domain from which the point events came. The estimator Ripley and Rasson developed is a rescaled copy of the convex hull, centered at the centroid of the convex hull with scaling factor *1/√{1 - frac m n}* where *n* is the number of data points and *m* the number of vertices of the convex hull.

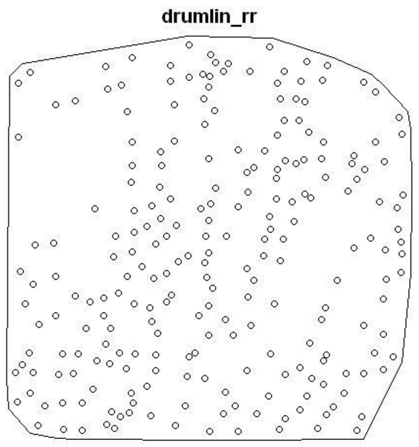
```
> ch <- convexhull.xy(drumlin_ppp)
> rr <- ripras(drumlin_ppp)
```

Next we create ppp objects for each of these alternative bounding boxes:

```
> drumlin_ch <- ppp(drumlin_ppp$x,drumlin_ppp$y, window = ch)
> drumlin_rr <- ppp(drumlin_ppp$x,drumlin_ppp$y, window = rr)
```

These alternative bounding polygons look like:





In the absence of a defined 'natural edge' to the distribution, I'd almost always use the Ripley & Rasson version.

c) The effect on a simple test for CSR

As highlighted on pages 137-139 of the course text, the difference in the bounding box used can matter, although in this case we have a sufficiently large number of points to mean that it doesn't amount to very much. First, we will examine probably the simplest measure used in point pattern work, which is the Clark and Evans (1954*) nearest neighbor index* of aggregation *R*.

This crude measure of clustering or ordering of a point pattern is the ratio of the observed mean nearest neighbor distance in the pattern to that expected for a Poisson point process (CSR) of the same overall intensity. A value *R>1* suggests ordering/dispersion, while *R<1* suggests clustering/aggregation. However, without correction for edge effects, the value of *R* will always be positively biased. Edge effects arise because, for points close to the edge of the window, the true nearest neighbors may actually lie outside the window. Hence observed nearest neighbor distances tend to be larger than the true nearest neighbor distances.

Three corrections are available in spatstat by specifying them as arguments when clarkevans is called. For rectangular windows an empirical edge correction established by Donnelly adjusts the value of the observed mean nearest neighbor downwards. Second, a simple approach uses a guard region or buffer zone around the set of events. The observed mean nearest neighbor distance is re-defined by averaging only over those points that fall inside the sub-window clipregion. Third, the Cumulative Distribution Function method, cdf, estimates the neighbor distance distribution function *G(r)* of the point process and applies a standard edge correction. The mean of the distribution is calculated from that estimate. If no arguments are specified we get all three, the 'naïve' (no correction, using the supplied bounding box), 'Donnelly' and the 'cdf':

```
> # nearest neighbor tests for comparison
> clarkevans(drumlin_ppp)
```

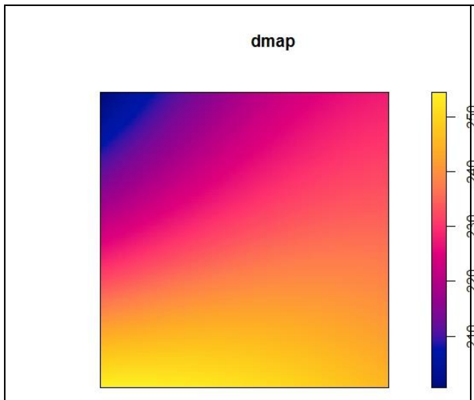
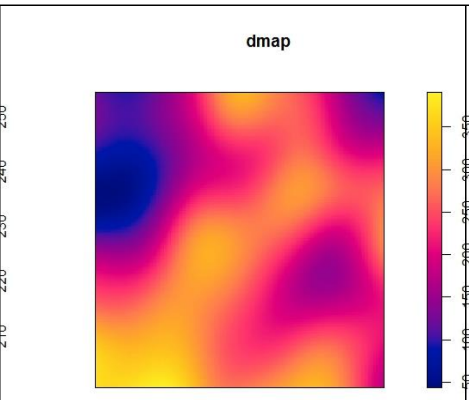
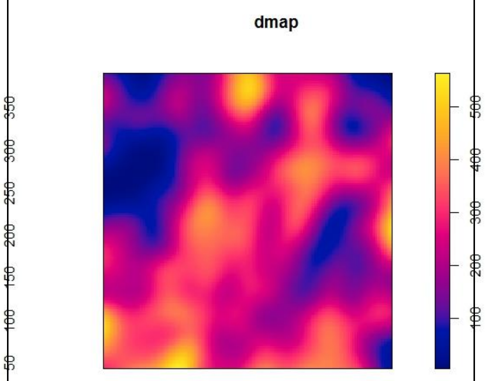
```
   naive Donnelly      cdf
1.249917 1.215380 1.233599
> clarkevans(drumlin_ch)
   naive Donnelly      cdf
1.295184       NA 1.272305
> clarkevans(drumlin_rr)
   naive Donnelly      cdf
1.238626       NA 1.215134
```

For these data it seems safe to conclude that *R* is around 1.24 -1.25, on the uniform side of the scale. Note in passing how use of the convex hull inflates things: given that by definition it puts a lot of events onto the outer edge of the pattern the convex hull shouldn't be used in testing.

d) Visualization using KDE

The various edge corrections don't make much difference to any KDE plot, but changing the band width specified by the sigma argument most certainly does. All we do now is to generate kernel density plots for three different bandwidths, noting that by default spatstat uses a Gaussian function so that the band width argument specifies the standard deviation ('spread') to be employed:

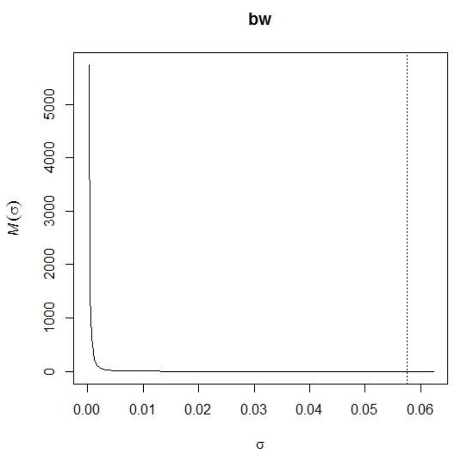
```
> dmap <- density(drumlin_ppp, 0.5, at="pixels")
> plot(dmap)
> dmap <- density(drumlin_ppp, 0.1, at="pixels")
> plot(dmap)
> dmap <- density(drumlin_ppp, 0.05, at="pixels")
> plot(dmap)
```

		
Bandwidth =0.5 gives a very smooth estimate	**Bandwidth = 0.1 roughens it up at bit …**	**Bandwidth = 0.05 is too low?**

Note that dmap is an example of an object class called image. It is worth experimenting with different values for the band width. As you can see, low values, in this case 0.05 distance units, produce a very 'spiky' estimate of the density function whereas values >0.1 seem to over-smooth it.

One option with KDE that has been explored a great deal is the find a numerical criterion that can be used to decide on an optimal bandwidth for a given pattern. Diggle (1985) suggests minimizing the mean square error of the kernel smoothing estimate when the underlying process is assumed to be a stationary Cox process, and this same approach has been used by others as an exploratory technique for guidance on bandwidth selection. In spatstat this is implemented as bw.diggle :

```
> bw=bw.diggle(drumlin_ppp)
> plot(bw)
```



```
> bw2<-as.numeric(bw)
> bw2
[1] 0.05748532
```

This more-or-less confirms what inspection of the sequence of plot indicated. I am not myself sure that the approach improves greatly on basic visualization, or use of some adaptive technique (q.v.).

e) Distance functions *G*(*d*), *F*(*d*), *K*(*d*) and *L*(*d*) and their expected values under CSR

Nowadays, basic exploratory analysis of a mapped point pattern seldom uses the single number approach as in Section (c) , but instead looks at one or other of a series of estimates of *distance* *functions* called *G*(*d*), *F*(*d*) and 'Ripley's' *K*(*d*), comparing their observed to the expected values under CSR. There is also a simple transformation of Ripley's *K*(*d*) function called *L*(*d*) that we will also use in the remainder of this section.

Task 2.4 Monte Carlo testing

Go back to the course text, O'Sullivan and Unwin (2010 and re-read pages 145-152 on these distance functions and how Monte Carlo methods can be used to construct a rough significance test. Next use > help(spatstat) to open an internet link to the spatstat documentation and then search for and read what it says about envelope, runifpoint , expression, and plot.envelope.

In Brunsdon and Comber (2015) this is covered in Section 6.5 *Second-order analysis of point patterns*, pages 184-195.

In spatstat, the simulation envelopes of any of the summary distance functions are computed using the envelope object, which is a complex tool that has a large number of possible arguments not all of which will be needed for every run. The basic call has the general form:

```
env <- envelope( Y, fun,  …)
```

In which Y is an object of class ppp (i.e. something spatstat can handle) fun specifies the function and the ellipsis … indicate extra arguments that are passed to fun including, for example, nsim the number of simulations, followed by a series of other arguments to control further detail as shown in the table below:

Argument	**Function**	**Comment**
`Y`	Object containing the point pattern data	e.g. drumlin_rr
`fun`	Function that computes the statistics required	e.g. Lest. Typically fun is one of the functions Kest, Gest, Fest, Jest, pcf, Kcross, Kdot, Gcross, Gdot, Jcross, Jdot, Kmulti, Gmulti, Jmulti or Kinhom. It can also be a user-supplied function.
`nsim`	Number of patterns to be simulated	e.g. 99, 999 or any other number to taste
`simulate`	OPTIONAL. If simulate is an expression in the R language it will be evaluated nsim times	There are various ways of generating this.
`verbose`	Logical flag. If set = TRUE will report progress during the simulations	Usually set FALSE
`clipdata`	Logical flag to control whether or not the point pattern data should be clipped to the same window as the simulated patterns	For most normal work this should be left at a default of TRUE
`Start, control`	OPTIONAL control on the simulation algorithm used	Not often changed

`transform`	OPTIONAL transformation to be applied to the function value before the envelopes are computed	
`global`	Logical flag indicating whether or not the envelopes should be pointwise (global=FALSE) or simultaneous (global=TRUE)	Not often changed, but care needed here when interpreting the results (see the spatstat documentation)
`ginterval`	OPTIONAL and only relevant of global=TRUE. A vector of length 2 specifying the interval for r values	As above
`savefuns`	Logical flag to indicate whether or not to save simulated function values	Why would you?
`savepatterns`	Logical flag to indicate whether or not to save simulated patterns	Maybe if you are writing a text book? More sensibly if you want to use the same patterns with a different function.
`nsim2`	Number of extra simulated point patterns to be generated if it is necessary to use simulation to estimate the theoretical mean of the summary function. Only relevant when global=TRUE and the simulations are not based on CSR	Not often needed
`VARIANCE`	Logical flag. If set TRUE critical envelopes will be calculated as sample mean plus or minus nSD time the sample standard deviation	Not often changed
`nSD`	As above, if VARIANCE = TRUE	As above
`yname`	Character string to be used as name of the point pattern Y when outputting results	Useful?
`maxnerr`	Maximum number of rejected patterns. If fun yields an error when applied to a simulated point pattern (for example, because the pattern is empty and fun requires at least one point), the pattern will be rejected and a new random point pattern will be generated. If this happens more than maxnerr times, the algorithm will give up.	Not often needed

A call using all the arguments might look like:

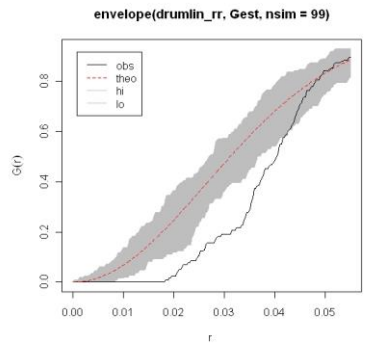
```
envelope(Y, fun=Kest, nsim=99, nrank=1, simulate=NULL, verbose=TRUE, clipdata=TRUE,
transform=NULL,global=FALSE,ginterval=NULL,   savefuns=FALSE, savepatterns=FALSE,
nsim2=nsim, VARIANCE=FALSE, nSD=2, Yname=NULL, maxnerr=nsim)
```

Mercifully we seldom need all this flexibility. For the most basic use, if you have a point pattern and you want to test against CSR, typing a command similar to that below will plot the estimated distance function required (Gest, Fest, Kest or Lest) and a default simulation envelope using the 'hi' and 'lo' values obtained:

```
># calculate and plot the estimated G function and its simulation envelope for the ppp object drumlin_rr
with 99 simulations
> plot(envelope(drumlin_rr, Gest, nsim=99))
Generating 99 simulations of CSR ...
1, 2, 3,  … 99
Done.
    lty col  key   label                              meaning
```

```
obs    1   1  obs  obs(r)           observed value of G(r) for data pattern
theo   2   2 theo theo(r)                theoretical value of G(r) for CSR
hi     1   8   hi   hi(r) upper pointwise envelope of G(r) from simulations
lo     1   8   lo   lo(r) lower pointwise envelope of G(r) from simulations
```

This produces the graphic:



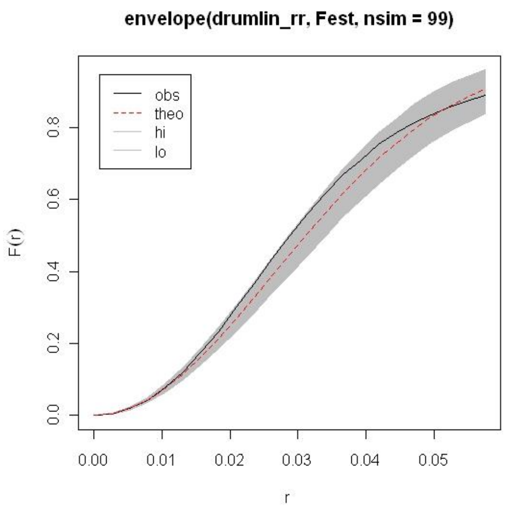
Clearly, the simulation envelopes of what is expected under CSR can be used to assess the goodness-of-fit of this specific point process model to the point pattern we observe. If we have no reason to reject CSR, then we'd expect the observed curve to lie close to the theoretical expectation (the red dotted line) and well inside the grey-shaded simulation envelope. Where the curve of the observed values (the black solid line) lies outside the simulation envelope, above or below, we have grounds for thinking that the process model simulated is not appropriate.

Upper and lower critical envelopes can be computed in one of two ways The default is *pointwise*, in which for each value of the distance argument r the generated list of nsim values is sorted and the m-th lowest and highest values are taken, as determined by the argument nrank. Thus if the default nrank = 1, it is the highest and lowest values that are taken. These pointwise envelopes are not strict confidence bands for the true value of the function. Rather, they specify the critical points for a Monte Carlo test (Ripley, 1981) constructed by choosing a fixed value of *r*, and rejecting the null hypothesis if the observed function value lies outside the envelope at, *and only at*, that value. In practice inspection of the observed graph of the chosen function and the upper and lower limits given by the simulation envelope will usually be all that is required.

In this case, recall that *G*(*d*) is the cumulative frequency distribution of the nearest neighbor distances between all events in the pattern. Given that we have 232 drumlin 'events' this curve is fairly smooth and well-estimated. The shape of the observed values for the pattern tells us that these distances increase quite rapidly between 0.02 and 0.04 distance units and that over this range the observed values lie well outside the simulation envelope.

We can do the same thing to estimate the *F*(*d*) function, which is similar but which uses distances between events in the pattern and any number of chosen 'locations' in the same area. In the literature the word *location* is used to indicate any arbitrary point in the same spatial area as the pattern of *events*.

```
> plot(envelope(drumlin_rr,Fest,nsim=99))
Generating 99 simulations of CSR ...
1, 2, 3, …
Done.
    lty col  key   label                          meaning
obs    1   1  obs  obs(r)        observed value of F(r) for data pattern
theo   2   2 theo theo(r)             theoretical value of F(r) for CSR
hi     1   8   hi   hi(r) upper pointwise envelope of F(r) from simulations
lo     1   8   lo   lo(r) lower pointwise envelope of F(r) from simulations
```



Here, things are less clear and on the evidence of this function we'd be disinclined to reject the CSR mechanism as a possible process.

In using envelope you will often be happy to leave what is required to spatstat, but if you want to change the rules of the game, or access the graphed values for further use the arguments passed to simulate are very useful. If simulate is supplied, it determines how the simulated point patterns are generated. Simulate can be an R expression that typically passes arguments to a random number generator. The expression will be evaluated nsim times to yield nsim point patterns for each of which the selected function will be evaluated. For example if we specify simulate = expression (runifpoint(100)) then each simulated pattern consists of exactly 100 independent uniform random points.

Here is an example in which we estimate Ripley's *K*(*d*), which, by almost common consent, is the most useful initial exploratory approach to point pattern analysis:

```
># First get the number of events in the pattern from the PPP object and check that it is correct
> n <- drumlin_rr$n
> n
[1] 232
># now specify the way we want to simulate
> way <- expression(runifpoint(n, win=rr))
```

It is worth having a look at this. We are using expression to create an object called way that in turn uses runifpoint, to generate a random point pattern containing n independent random points within the Ripley& Rasson window defined by the object rr. A typical use might be:

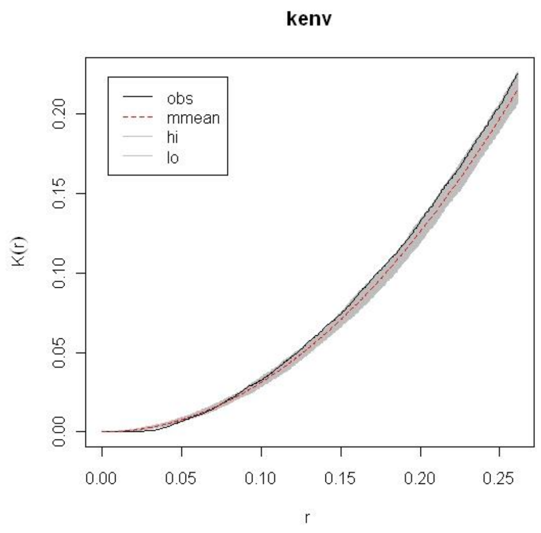
```
runifpoint(n, win=owin(c(0,1),c(0,1)), giveup=1000)
```

The arguments used are n, the number of points to be generated, win, the window in which to simulate the pattern, which must be an object of class owin, or something that can be made into one such as in the example above, and giveup, the number of attempts in the rejection method after which the algorithm should stop trying to generate new points.

The expression in the object way is then passed as an argument to envelope but, instead of plotting this immediately, we create an object kenv that contains a data frame with all the values needed to enable a plot to be drawn and, if need be, further work on the values it contains. The columns of the data table created are reported as obs, mmean, hi and lo with rows being distance (r) values chosen by default. The argument verbose=FALSE switches off reporting as the simulations are performed.

```
> kenv <- envelope(drumlin_rr, Kest, nsim=99,simulate=way, verbose=FALSE)
> plot (kenv)
      lty col   key   label                                    meaning
obs     1   1   obs  obs(r)          observed value of K(r) for data pattern
mmean   2   2 mmean mean(r)             sample mean of K(r) from simulations
hi      1   8    hi   hi(r) upper pointwise envelope of K(r) from simulations
lo      1   8    lo   lo(r) lower pointwise envelope of K(r) from simulations
```

The result looks like:



Again, things are less clear and on the evidence of this plot we'd be disinclined to reject the CSR process.

It is useful to transform these *K*(*d*) values into the function *L*(*d*) using the transformation:

$$L(d) = \sqrt{K(d)/\pi} - d$$

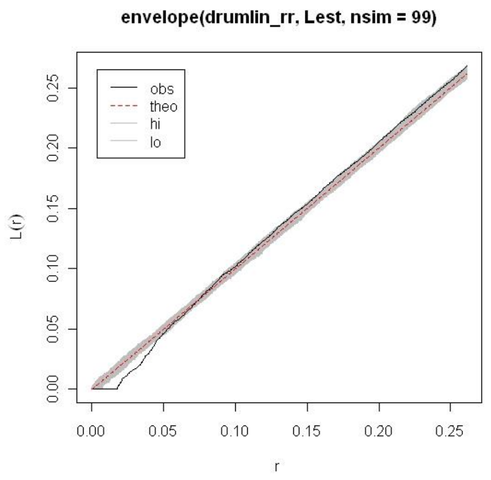
This arises because the theoretically expected value at distance *d* is:

$$E(K(d)) = \lambda \pi d / \lambda = \pi d^2$$

It follows that for all distances *L*(*d*) under the assumption of CSR will be zero. In fact, spatstat doesn't subtract the distance *d* from its estimate of *L*(*d*), with the result that plots of *L*(*d*) against *d* have the theoretically expected values along the diagonal instead of the horizontal straight line used in the course text. I prefer the later way, incidentally.

```
>plot(envelope(drumlin_rr, Lest, nsim=99))
```

Generates the plot:



In terms of the drumlin distribution problem this is probably the most useful of all, showing that at very short range the observed values drop 'significantly' (q.v.) below those expected, and hence indicates a tendency towards regularity at very short range. This has been interpreted (by others) as grounds for suggesting that a spatial inhibition process is operating whereby the presence of an event makes it less likely for another to be close to it. An obvious possibility is that drumlins aren't really point objects, but have a volume and a plan area such that they cannot be closer to each other than some minimum distance. And that's about as far as we can get with the problem using these standard exploratory tools.

It is worth noting that use of the *K*(*d*) and *L*(*d*) functions at distances that are a substantial fraction of the mapped area will always generate edge effect problems because the large circles used close to the edges of the distribution will have a similarly substantial fraction of their area outside the mapped area. Various corrections have been suggested (see pages 137-139 of O'Sullivan and Unwin, 2010).

f) A note on significance testing these functions

The distance function approach does not really give us the usual test of 'significance', and because we are interested in the departure of the observed curve from CSR at a *series* of distances, *d*, we are in a multiple hypothesis testing situation. If there is a need for a single number whose probability under the hypothesis of CSR is required, two approaches have been implemented in spatstat and both use measures of the difference between the observed curve and the theoretical expectation.

1. The maximum absolute difference (MAD) test developed by Ripley takes the largest absolute difference between the curves and so is defined as:

   $MAD = max_d | \hat{K}(d) - K_{csr}(d)|$

   For *K*(*d*), it is implemented as:

   ```
   > drum_mad<-mad.test(drumlin_ppp,Kest)
   Generating 99 simulations of CSR  ...
   1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
   16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30,
   31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45,
   46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60,
   61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75,
   76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90,
   91, 92, 93, 94, 95, 96, 97, 98,  99.
   Done.
   > drum_mad

           Maximum absolute deviation test of CSR
           Monte Carlo test based on 99 simulations
           Summary function: K(r)
           Reference function: sample mean
           Interval of distance values: [0, 0.25]

   data:  drumlin_ppp
   mad = 0.0104, rank = 1, p-value = 0.01
   ```

   The same approach can be taken for *F*(*d*), *L*(*d*) and *G*(*d*)

2. The Diggle-Cressie-Loosmore-Ford (dclf) Test

Is much the same but instead uses the sum of the squares of the differences over the entire range between the observed value and the mean of the simulations, that is:

$$u_i = \sum_{d_{k=d_{\min}}}^{d_{max}} \left[\hat{K}_i(d_k) - \overline{K}_i(d_k)\right] ** 2\ dk$$

The second term in the brackets is the average value of K(d) over all the simulations. This is implemented in the same way:

```
drum_dclf<-dclf.test(drumlin_ppp, Kest)
Generating 99 simulations of CSR  ...
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30,
31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45,
46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60,
61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75,
76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90,
91, 92, 93, 94, 95, 96, 97, 98, 99.
Done.
> drum_dclf
      Diggle-Cressie-Loosmore-Ford test of CSR
      Monte Carlo test based on 99 simulations
      Summary function: K(r)
      Reference function: sample mean
      Interval of distance values: [0, 0.25]
data:  drumlin_ppp
u = 0, rank = 1, p-value = 0.01
```

2.5 Where and why?

Task 2.5: Practical point pattern analysis

Now read Chapter 6, *Practical Point Pattern Analysis*, Sections 6.1 – 6.7, pages 157-177 of O'Sullivan and Unwin (2010), which is a substantial update of a Chapter 5 with the same title of the First Edition. In the First edition, Section 4.6 pages 108 – 112 deal with the critiques due to Peter Gould and David Harvey that form Section 6.1 of the Second Edition.

There are very few studies in the geographical literature that tackle the problem of non-homogeneity. One that does was written by former *statistics.com* student Geoff Phelps as: Phelps, G (2007) *Analysis of a Spatial Point Pattern: Examining the Damage to Pavement and Pipes in Santa Clara Valley resulting from the Loma Prieta Earthquake*. (51 pages)

It can be downloaded from http://pubs.usgs.gov/of/2007/1442/, and is well-worth reading (checked 22-10.14).

Bivand *et al*. (2008) give a much more detailed account of these same issues, see Section 7.4 pages 163-172 and parts of Section 7.5. Brunsdon and Comber (2015) don't go into it at all.

a) Inhomogeneity and what we can do about It

It should be obvious that very many spatial point processes are not homogeneous Poisson (HPP) and, when they are not, differentiation between alternatives based on the analysis of an observed map pattern can be almost impossible. An HPP is a process in which all events are independently distributed in the region, which implies also that an HPP is stationary and isotropic. Stationarity means that the process intensity is everywhere the same and the second-order intensity depends only on the distances between points regardless of their relative positions. It follows that HPP is the formal definition of what we have thus far called Complete Spatial Randomness (CSR). If you remain unconvinced, do the first part of this week's assignment and return to this point.

There are two general causes of inhomogeneity, usually called *first* and *second* order variation. First order variation is where variations across the study region in some covariate (such as soil type, population, etc.) mean that the observed pattern does not have a spatially uniform intensity but is described by an *inhomogeneous Poisson distribution* and the process is 'IPP'. Figure 6.1, page 164 of the course text, provides an illustration, showing that in practice if all that we have is the map pattern of events, realizations of the IPP process can be very hard to distinguish from HPP unless the variation in intensity is very marked. Of course, we may well have some very clear and obvious covariate, such as the distribution of children at risk in the cancer cluster example that forms the basis of Sections 6.4 – 6.7 of the course text that it would be plain silly to ignore. Second order variation arises when there are interactions between events, for example when the location of one event, such as a case of an infectious disease, makes it more likely that other events will cluster around it. Alternatively, it might be that once we have an event, other events are less likely to occur close to it as a result of, for example, competition for resources such as light and nutrients, or even simple packing constraints of the sort that may well explain the distribution of our drumlins in the example used in Section 2.5.

If all that we have is a map pattern of events, then any spatial variation in intensity can be estimated using KDE. However, as we saw in Section 2.5(d) above, these estimates very much depend on the bandwidth chosen for this process (i.e. the area surrounding each location at which the density is evaluated) and to a much lesser extent on the nature of the kernel used (e.g. quartic, Gaussian or whatever). The following quotation from *The Hitchhiker's Guide to the Galaxy* illustrates the difficulty inherent in the idea of a spatial density very well:

> *Area: The area of the Universe is infinite.*
> *Population: None. Although you might see people from time to time, they are most likely products of your imagination. Simple mathematics tells us that the population of the Universe must be zero. Why? Well given that the volume of the universe is infinite there must be an infinite number of worlds. But not all of them are populated; therefore only a finite number are. Any finite number divided by infinity is zero, therefore the average population of the Universe is zero, and so the total population must be zero.*

What follows is based heavily on Baddeley (2008, pages 75 *et seq*.) and it uses the tropical trees data set (bei) that are bundled with the spatstat package. First we load these data and summarize them:

```
> data(bei)
> summary(bei)
Planar point pattern: 3604 points
Average intensity 0.00721 points per square metre
Window: rectangle = [0, 1000]x[0, 500]metres
Window area =  5e+05 square metres
Unit of length: 1 metre
```

Overall, our estimate of the point process intensity, the observed density, is 0.00721 events per square meter. This is easily confirmed since the rectangular window has 3604 events in an area 1000 by 500m, and 3604/(1000x500) = 0.00721. The question we must ask is whether or not this intensity varies markedly over the mapped region. A very simple way to explore variations in the estimated intensity is to divide the region into a regular pattern of quadrats (see the course text, pages 127-130) and to count how many events fall into each. In spatstat there is a single command quadratcount with obvious arguments to do this as:

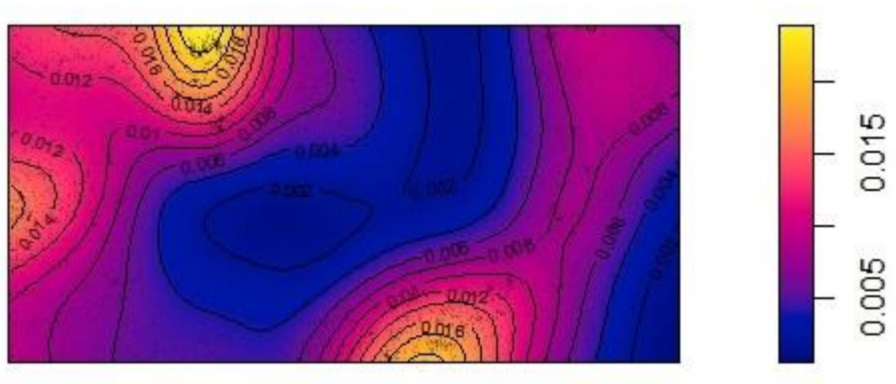
```
># create the object q_map as the density/intensity counts for a 3 by 3 grid of rectangular quadrats
spanning the region
> q_map <- quadratcount(bei, nx=3, ny=3)
> q_map

        x

 y            [0,333] (333,667] (667,1e+03]
  (333,500]     945       235        373
  (167,333]     471        69        274
   [0,167]      365       498        374
```

Although by itself this shows considerable variation even at this very coarse grain of analysis, an obvious improvement is to assess the extent of the inhomogeneity using KDE as an exploratory device:

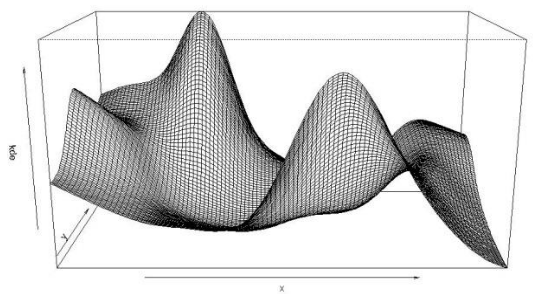
```
>/# plot the points first using a cross as symbol
> plot(bei, pch="+")
>/# now create the kde estimates using a bandwidth sigma of 65 distance units
> kde_65 <- density(bei, sigma=65.0)
>#plot it and then add dots for the points
> plot(kde_65)
> plot(bei,add=TRUE, pch=".")
>/# just to show we can, visualize the kde surface as a contour-type map and perspective diagram
> contour(kde_65, add=TRUE)
> persp(kde_65) #plot a perspective wire frame as well
```

These commands will generate the picture shown below:

(a) Tropical Trees KDE estimates



(b) Wire-frame display of the densities



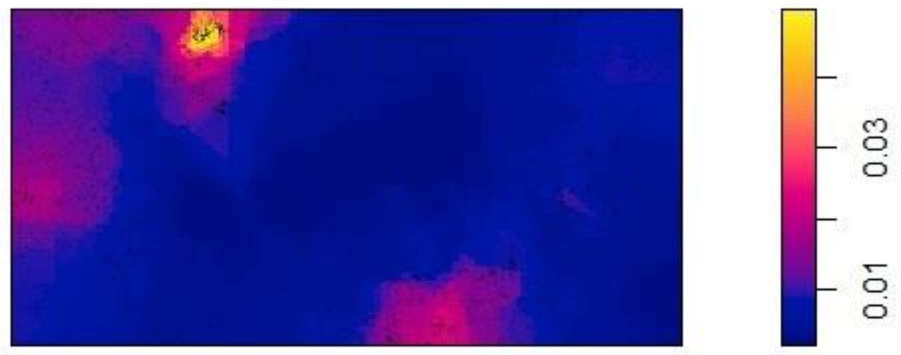
The contour-type map shows that we have a variation of more than an order of magnitude, from <0.002 to >0.02 events/m^2.

Another issue with any bandwidth choice in KDE is that if the pattern is inhomogeneous with wide variations in event density, a wide bandwidth that is appropriate for data sparse areas will miss possibly important detail in parts of the region where the data are plentiful. An obvious stratagem is to develop some method whereby the bandwidth used can be varied in relationship to the density of events. In introducing the idea, fellow s*tatistics.com* instructor, Chris Brunsdon (1995) varied the bandwidth as a function of the local density of points and this is implemented in the R package sparr. Sparr stands for *Spatial Relative Risk* and the package provides functions to estimate fixed and adaptive kernel density estimates using the density ratio method where we have case/control data as noted on pages 170-171 of O'Sullivan and Unwin (2010).

However, spatstat uses an alternative and rather ingenious approach developed by Ogata *et al.* (2003) in which a small fraction, given by the argument, f , of the point data is used to compute a Dirichlet tessellation of the region. Then for each tile of the tessellation the remaining proportion (1-f) of data points that fall in the tile are counted as an estimate of the density. To avoid dependence on a single tessellation, the process can be repeated nrep times, as in the sequence:

```
> kde_adapt <- adaptive.density(bei, f=0.01,nrep=10)
> plot(kde_adapt)
> plot(bei, add=TRUE, pch=".")
```

The object returned from adaptive.density is a spatially continuous pixel image of the image class im. Over-plotted with a dot map of the events it looks like:



Task 2.6 Exploring inhomogeneity in a point pattern

Your assignment has a second task in which I ask you to examine the inhomogeneity in a different mapped pattern. It is probably a good idea to do it now.

b) Correcting for Inhomogeneity using a covariate

Section 6.5 of the course text looks at KDE as an exploratory technique to detect and quantify inhomogeneity in a mapped pattern of events, but in addition suggests that computation of spatial rates of incidence is one way to handle the same issue. An obvious approach, illustrated by our Sellafield childhood cancer example, is to estimate rates of occurrence over small subsets of the entire region, relative to some defined covariate available for these same small regions. A *covariate* is some spatially extensive variable that might explain the observed mapped pattern of events. If we have such a covariate then it can be used to model the inhomogeneity. Critically it will be observed over

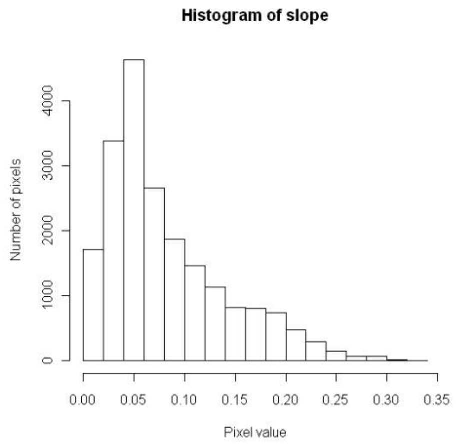
- Different point events (perhaps interpolated onto a continuous surface);
- Some tessellation of the plane such as a census district (as in the childhood cancer case);
- A calculated tessellation (as in our example below).

Almost always it is NOT sufficient to observe the values of covariates at the same locations as the events. In order to investigate the dependence of a point process on a covariate there should be a least some observations of the covariate at other non-data locations. In contrast to a covariate, in the literature a *mark* is defined as any variable (continuous or categorical) observed at the same location as an event and generating what is called a *marked point pattern*. In the Sellafield example the obvious covariate is the number of children in the at risk age bands but the essence of the problem is that these data are available only as aggregates for arbitrary sub regions given by the tessellation used in the most recent population census for the region. Hopefully you can see that the 'now you see it, now you don't' nature of the so-called Sellafield cluster according to how the spatial units are defined is a manifestation of the general modifiable areal unit problem (MAUP).

An alternative is to define the sub-regions over which the rates are estimated on the basis of some hypothesis that actually uses the covariate information. What in Section 6.6 pages 172-173 we call a *focused* approach is an example of this general method. The hypothesized covariate in the Sellafield case is *distance* from an assumed location and the tabulated rates (Table 6.1, page 173) are those for the areas given by the concentric distance bands. The objections, page 173, remain, but, as we illustrate below, this kind of covariate is very easy to assemble in both a GIS and in R.

Baddeley (2008, pages 70-71) illustrates the same approach using the bei data and his *quadrats determined by covariates* approach is followed in the example that follows. With the bei data is also bundled additional information in bei.extra on both the altitude (as bei.extra$elev) and slope gradient (as bei.extra$grad) across the same region. The covariate of interest is the *slope gradient*, which we first extract as the object slope and then use quantile with a vector of probabilities at 0.25 intervals given by the argument probs to create slope_class containing the four quantiles

```
> slope <- bei.extra$grad
> hist (slope)
```



```
> slope_class<- quantile(slope, probs=(0:4)/4)
> slope_class

      0%        25%       50%             75%       100%
```

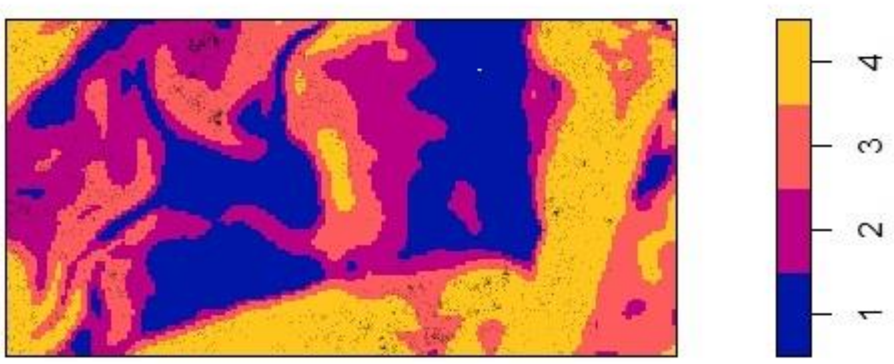
```
0.0008663357  0.0399717900  0.0620298800  0.1129532000  0.3284767000
```

Next we convert these into levels of a factor and put them into slope_cut:

```
> slope_cut <- cut(slope, breaks = slope_class, labels = 1:4)
> slope_cut
factor-valued pixel image
factor levels:
[1] "1" "2" "3" "4"
101 x 201 pixel array (ny, nx)
enclosing rectangle: [-2.5, 1002.5] x [-2.5, 502.5] metres
```

We then use tess to convert into a tessellation with four types of 'tile' that we plot and add the original point event dot map to create a simple picture showing events mapped into the four classes of slope gradient:

```
> picture <- tess(image = slope_cut)
> plot(picture)
> plot(bei,add=TRUE, pch = ".")
```



The colors are horrible, but visual inspection might suggest that the trees are on the steeper slopes and this is confirmed using a simple count:

```
> count_by_slope <- quadratcount(bei, tess = picture)
> count_by_slope
tile
   1    2    3    4
 271  984 1028 1321
```

Since the slope categories were designed so as to have equal area, the density of trees is clearly non-uniform and a chi-square test confirms this. The command quadrat.test uses tiles of the tessellation as the quadrats:

```
> quadrat.test (bei, tess=picture)

	Chi-squared test of CSR using quadrat counts

data:  bei
X-squared = 661.8402, df = 3, p-value < 2.2e-16
```

If you would like to follow the analysis of these data in more detail, Baddeley (2008, pages 79-96) uses them in an extended and clearly laid out example of how next to fit a point process model.

c) Modelling point patterns/processes

The last decade or so has seen advances in modelling spatial point patterns and spatstat contains facilities that help in this work. In particular it contains methods to simulate point patterns from process models that aren't CSR. The table below is taken from the spatstat documentation and illustrates the possibilities. We have already seen runifpoint in action:

`runifpoint`	generate *n* independent uniform random points
`rpoint`	generate *n* independent random points
`rmpoint`	generate *n* independent multitype random points
`rpoispp`	simulate the (in)homogeneous Poisson point process
`rmpoispp`	simulate the (in)homogeneous multitype Poisson point process
`runifdisc`	generate *n* independent uniform random points in disc
`rstrat`	stratified random sample of points
`rsyst`	systematic random sample of points
`rjitter`	apply random displacements to points in a pattern
`rMaternI`	simulate the Matern Model I inhibition process
`rMaternII`	simulate the Matern Model II inhibition process
`rSSI`	simulate Simple Sequential Inhibition process
`rStrauss`	simulate Strauss process (perfect simulation)
`rNeymanScott`	simulate a general Neyman-Scott process
`rMatClust`	simulate the Matern Cluster process
`rThomas`	simulate the Thomas process
`rGaussPoisson`	simulate the Gauss-Poisson cluster process
`rthin`	random thinning
`rcell`	simulate the Baddeley-Silverman cell process
`rmh`	simulate Gibbs point process using Metropolis-Hastings
`simulate.ppm`	simulate Gibbs point process using Metropolis-Hastings
`runifpointOnLines`	generate *n* random points along specified line segments

rpoisppOnLines	generate Poisson random points along specified line segments

These cluster process models (with homogeneous or inhomogeneous intensity) can be fitted by the function kppm. Its result is an object of class kppm. The fitted model can be printed, plotted, predicted, simulated and updated:

plot.kppm	Plot the fitted model
predict.kppm	Compute fitted intensity
update.kppm	Update the model
simulate.kppm	Generate simulated realizations

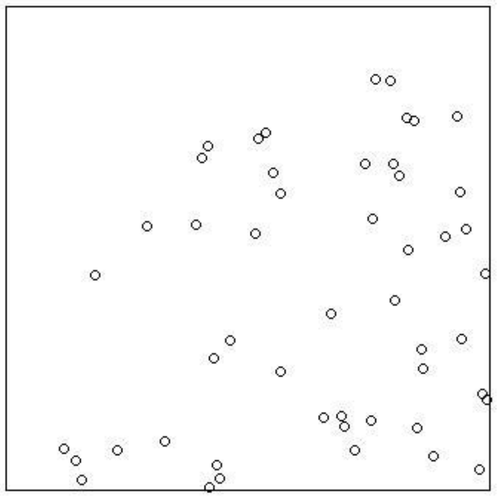
Other model-fitting functions include:

thomas.estK	fit the Thomas process model
thomas.estpcf	fit the Thomas process model
matclust.estK	fit the Matern Cluster process model
matclust.estpcf	fit the Matern Cluster process model
lgcp.estK	fit a log-Gaussian Cox process model
lgcp.estpcf	fit a log-Gaussian Cox process model
mincontrast	low-level algorithm for fitting models by the method of minimum contrast

Whether or not you need to be concerned about these process models really depends on, as we say in UK, *where you are coming from*. If you have sound theoretical reasons for a particular spatial process model, then it makes sense to try to calibrate its parameters from the evidence of some mapped pattern, but, even with an explicit theory, this will often prove tricky.

In part this is a consequence of David O'Sullivan and I (2010, page 160) call *David Harvey's Critique* from way back in the 1960s and using modern likelihood based inference will not always help very much. Sometimes, even a very clear and plausible process model will generate realizations that are impossible to differentiate using any of the methods outlined above, so, if all you have is a mapped pattern, going much beyond a bald and not very useful statement that the pattern is very unlikely to be CSR will be almost impossible. This is also part of *Peter Gould's Critique*, dating from 1970 and outlined on pages 158-159.

We can illustrate this using the tools in spatstat to simulate a very simple model. Suppose we have two generations of events. First we have parent events that are distributed by HPP with a known intensity over our region of interest. Second, each parent is replaced by a second more restricted HPP of offspring events in a small area around it and, third, the results are combined as a mapped point pattern. Ecologists might well recognize this as at least one possible mechanism whereby plants are dispersed and it rejoices in the name of a *Neyman-Scott* process, with characterizing parameters that are the intensity of the dispersal processes and the maximum radius of the offspring clusters. The point pattern below is one realization of this process created using rNeymanScott in spatstat. My challenge would be to you to uncover from the evidence in the mapped pattern not only the fact that this was the generating process but also to estimate its parameters.



Task 2 .7: Now do the final part of the assignment

The third part of the assignment is what I call an 'open-ended' one (Unwin, 1980), designed to allow you to use your initiative in demonstrating your command of the materials in this lesson. Sometime students find this a little disconcerting but I hope you welcome the freedom that it offers. There is no 'right' or 'wrong' here, just a scale of effort and quality of results.

2.6 Conclusion

Hopefully, you will see that although we might have a mapped spatial point pattern as a dot/pin map, and making a classical statement about whether or not it is CSR is (relatively!) straight-forward, further progress in most practical work, at least at the scale of such work by geo-spatial analysts is often much more difficult. It seems to me that the number of 'pure' problems involving point patterns is actually quite small. Given that the class will have some different perspectives on this, perhaps this is something that you would like to debate?

2.7 Data files used in this lesson

- drumlins_in_unit_square.txt : drumlins in County Down, Northern Ireland
- bei and bei.extra: tropical trees and additional covariate information, from spatstat
- nztrees: a few trees in New Zealand from spatstat. Having lived there from time to time, I can assure you that New Zealand has a lot of trees.

2.8 References

Any or all of the following will be of interest:

Baddeley, A. (2008) *Analysing spatial point patterns in R*. CSIRO, Version 3, 199 pages available at www.csiro.au/resources/pf16h.html

Brunsdon, C. (1995) Estimating probability surfaces for geographical point data: an adaptive technique. *Computers and Geociences*, 21: 877-894)

Clark, C.D. (20100 Emergent drumlins and their clones: from till dilatancy to flow instability. *Journal of Glaciology*, 51, 1011-1025.

Clark, P. J. and F.C. Evans (1954) Distance to nearest neighbour as a measure of spatial relationships in populations, *Ecology*, 35: 445-453

Diggle, P. J. (1985) A kernel method for smoothing point process data. *Applied Statistics*, 34: 138-147.

Hill, A.R. (1973) The distribution of drumlins in County Down, Ireland. *Annals, Association of American Geographers*, 63:226-240

Ogata, Y., Katsura, K. and M. Tanemura (2003) Modelling heterogeneous space-time occurrences of earthquakes and its residual analysis. *Applied Statistics*, 52: 499-509.

Ripley, B.D. and J-P. Rasson, (1977) Finding the edge of a Poisson forest. *Journal of Applied Probability*, 14: 483 – 491

Smalley, I.J. and D.J. Unwin (1968) The formation and shape of drumlins and their distribution and orientation in drumlin fields. *Journal of Glaciology*, 7: 377-390

Unwin, D.J. (1980) Make your practicals open-ended, *Journal of Geography in Higher Education*, 4(2), 39-42.

Upton, G. and B. Fingleton (1985) *Spatial Data Analysis by Example: Volume 1: Point Pattern and Quantitative Data* (Chichester: John Wiley & Sons)