Exercise: Spatial Point Pattern Analysis

D G Rossiter
Cornell University

April 18, 2020

Contents

冰冻三尺，非一日之寒
"One day of cold weather is not enough to freeze ice three feet thick!" – Chinese proverb

This tutorial gives an overview of spatial point-pattern analysis. This considers the distribution of one or more sets of points in some bounded region as possibly being the result of some stochastic process which produces a finite number of "events" or "occurrences". Examples are a forest plot with the locations of individual trees, a microscope slide with the locations of individual cell centres, and a municipal boundary with point pollution sources. These may be viewed as pure point-patterns (just the locations) but sometimes attributes are included in the analysis; for example, the size or species of a tree in the forest plot.

After completing this exercise you should be able to:

1. Display spatial point patterns;
2. Analyze single spatial point patterns with the F, G, K and L functions;
3. Compute a kernel density of an inhomogeneous point pattern;
4. Analyze interactions between two point patterns with the K and L functions;
5. Model spatial point patterns as realizations of a spatial data generating process (sDGP) including both trend and interaction components, and evaluate model success.
6. Predict over an areas based on a fitted model.

The theory behind point-pattern analysis is comprehensively presented in the text of Diggle [7]. An accessible and less technical introduction is Boots and Getis [4]. Bivand et al. [3, Ch. 7] present worked examples in the context of R processing. Here we work through some of the main ideas only. Some of the code here is adapted from that chapter. Illian et al. [8] present a computational framework for fitting complex spatial point process models using a recently-developed methodology known as INLA.

Point-pattern analysis is based on theories of point processes. A modern review article is by Møller and Waagepetersen [11].

We will consider two kinds of properties of point patterns:

- **First-order**: considering points as individuals, no interaction, over the whole region. An example is the **spatial density**, which is taken as the indication of the **intensity** of the process that gave rise to the PPA;
- **Second-order**: considering interactions between marked sets of points, e.g., their tendency to cluster or repel. An example is bird nests of two different species in the same area.

In addition, we will attempt (§7) to model the presumed spatial data generating process.

Note: The code in these exercises was tested with Sweave [9, 10] on R version 3.6.3 (2020-02-29), `sp` package Version: 1.4-1, `splancs` package Version: 2.01-40, `lattice` package Version: 0.20-41, and `spatstat` package Version: 1.63-3 running on Mac OS X 10.10.12. So, the text and graphical output you see here was automatically generated and incorporated into LaTeX by running the code through R and its packages. Then the LaTeX document was compiled into the PDF version you are now reading. Your output may be slightly different on different versions and on different platforms.

1 Examining some point patterns

Supplementary reading:

- Bivand et al. [3, §7.2]: R packages relevant for spatial point-pattern analysis.

We use the same examples as Bivand et al. [3, Ch. 7]: location of cell centres in a microscope slide ("Cells"); locations of Japanese black pine saplings ("Japanese"); and locations of saplings of California redwood trees ("Redwood").

Task 1 : Load the Japanese pines example dataset `japanesepines` and summarize it. •

These are examples in the `spatstat` package and are provided in a suitable format, namely as objects of R class `ppp`, a "planar point pattern".

```
> require(spatstat)
> data(japanesepines)
> class(japanesepines)

[1] "ppp"

> str(japanesepines)

List of 5
 $ window    :List of 4
  ..$ type  : chr "rectangle"
  ..$ xrange: num [1:2] 0 1
  ..$ yrange: num [1:2] 0 1
  ..$ units :List of 3
  .. ..$ singular  : chr "metre"
  .. ..$ plural    : chr "metres"
  .. ..$ multiplier: num 5.7
  .. ..- attr(*, "class")= chr "unitname"
  ..- attr(*, "class")= chr "owin"
 $ n         : int 65
 $ x         : num [1:65] 0.09 0.29 0.38 0.39 0.48 0.59 0.65 0.67 0.73 0.79 ...
 $ y         : num [1:65] 0.09 0.02 0.03 0.18 0.03 0.02 0.16 0.13 0.13 0.03 ...
 $ markformat: chr "none"
 - attr(*, "class")= chr "ppp"

> summary(japanesepines)
```

```
Planar point pattern:  65 points
Average intensity 65 points per square unit (one unit = 5.7 metres)

Coordinates are given to 2 decimal places
i.e. rounded to the nearest multiple of 0.01 units
(one unit = 5.7 metres)

Window: rectangle = [0, 1] x [0, 1] units
Window area = 1 square unit
Unit of length: 5.7 metres
```

> **Note:** Notice that the `ppp` class has a structure that, according to the authors of the `spatstat` package, facilitates point-pattern analysis. Of course it has the coordinates (fields `x` and `y`) and the number of points (field `n`), but it also defines a window (field `window`) as a list of four characteristics: the shape, the limiting coordinates, and the units of measure. This field is of class `owin`.

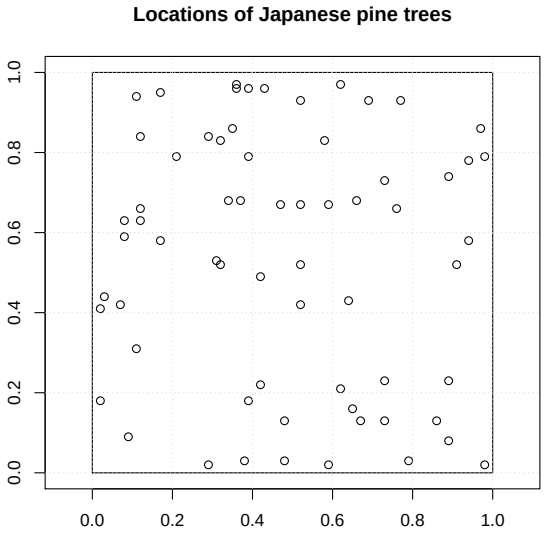
Q1 : *How many trees are represented by this point pattern?* *Jump to A1*
•

Q2 : *What is the area covered by this point pattern?* *Jump to A2* •

Task 2 : Plot the locations of the trees. •

The generic `plot` method specializes to `plot.ppp` for an object of class `ppp`:

```
> plot(japanesepines, main = "Locations of Japanese pine trees",
     axes = T)
> grid()
```



Q3 : *Does this pattern look completely random, clustered, or regular?*
Jump to A3 •

Task 3 : Load the other two example datasets `redwoodfull` and `cells`; view the spatial distribution of all three on one graph. •

```
> data(redwoodfull)
> data(cells)
```

To plot all three patterns together, we could repeat the `plot.ppp` command on each pattern separately; but it is more elegant to make a single data frame with the coordinates of the points, each labelled by its pattern name, and use the `xyplot` function of the `lattice` package. This uses a model formula to specify scatterplots, *conditioned* on a factor; so we need to build a data frame with three columns: the two coördinates and the name of the pattern.

We begin by converting the `ppp` objects to `sp` objects, using the generic conversion method `as`, specialized to convert from `ppp` to `sp` (and vice-versa) by methods loaded with the `maptools` package:

```
> require(sp)
> require(maptools)
> spjpines <- as(japanesepines, "SpatialPoints")
> summary(spjpines)

Object of class SpatialPoints
Coordinates:
     min max
[1,]   0 5.7
[2,]   0 5.7
```

```
Is projected: NA
proj4string : [NA]
Number of points: 65

> spred <- as(redwoodfull, "SpatialPoints")
> summary(spred)

Object of class SpatialPoints
Coordinates:
     min max
[1,]   0   1
[2,]   0   1
Is projected: NA
proj4string : [NA]
Number of points: 195

> spcells <- as(cells, "SpatialPoints")
> summary(spcells)

Object of class SpatialPoints
Coordinates:
     min max
[1,]   0   1
[2,]   0   1
Is projected: NA
proj4string : [NA]
Number of points: 42
```

Q4 : *What has happened to the bounding box, as the object was converted from* `ppp` *to* `sp`*?* *Jump to A4* •

The cells and redwoods are defined on a unit square; we are not told the true size of the bounding box. However, the pines did have a size, namely 5.7 m by 5.7 m[1]. To visualize these patterns on the same scale, we have to convert the `spjpines` `sp` object to the unit square. We do this with the `elide` method of the `maptools` package, which is designed to disguise true coordinates in cases where the location in the real world would compromise privacy.[2] The `scale` argument scales the coördinate with the wider spread to $[0 \ldots 1]$

```
> spjpines1 <- elide(spjpines, scale = TRUE, unitsq = TRUE)
```

We now build a single data frame, using the `data.frame` function, to join the three `SpatialPoints` objects "vertically" into the data frame using the `rbind` "row bind" function. We use the `coordinates` method to extract the coordinates from the `sp` object as two columns:

```
> spall <- data.frame(
                      rbind(coordinates(spjpines1),
```

[1] `?japanesepines`

[2] "Elide" is defined in the OED as "To strike out, suppress, pass over in silence", and in grammar "to omit a vowel, or syllable in pronunciation".

```
                                coordinates(spred),
                                coordinates(spcells)))
```

We now add a third column to the data frame with the `cbind` "column bind" function. This column contains the factor name, which is repeated (using the `rep` function) for the number of rows (using the `nrow` function) of each pattern:

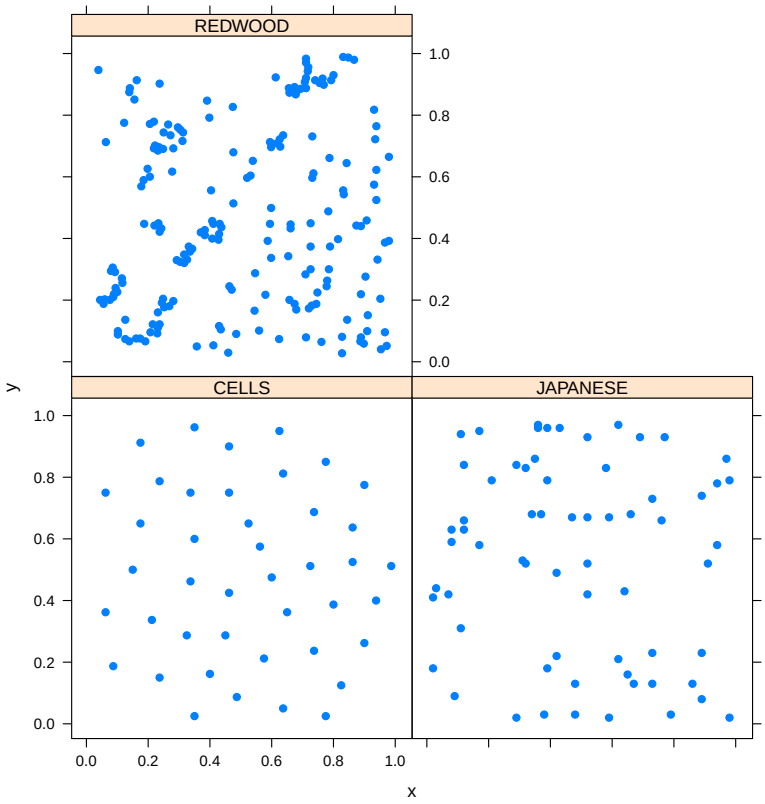
```
> spall <- cbind(spall,
                 c(rep("JAPANESE",nrow(coordinates(spjpines1))),
                   rep("REDWOOD", nrow(coordinates(spred))),
                   rep("CELLS", nrow(coordinates(spcells))))
                 )
```

Finally, we name the three columns of the data frame, using the `names` function:

```
> names(spall) <- c("x", "y", "dsn")
```

The `xyplot` function of the `lattice` package uses the model formula `y ~ x` to specify scatterplots where `y` is the y-axis variable and `x` the x-axis, and used the vertical bar `|` formula operator to specify that the plot is *conditioned* on a factor, written to its right. Here the conditioning factor is the data set name:

```
> require(lattice)
> print(xyplot(y ~ x | dsn, data = spall, pch = 19, aspect = 1))
```



Q5 : *Are there differences in the point patterns? Which pattern(s) look completely random, clustered, or regular?* *Jump to A5* •

2 Assessing complete spatial randomness: the G function

Supplementary reading:

- Bivand et al. [3, §7.3]: Preliminary Analysis of a Point Pattern

A suitable null hypothesis for SPPA is that there is no pattern, i.e., the points are distributed at random; this is known as Complete Spatial Randomness (abbreviation CSR). Assuming CSR we can compute several expected distributions of the points:

- The **G** function: the distribution of the distances from an arbitrary observed **point** to its nearest neighbour; expressed as the *cumulative distribution function* $G(r)$ of the proportion of points that have at least one neighbour within a distance r. Formally:

$$d_i = \min_j \{d_{ij}, \forall j \neq i \in S\}, i = 1, \ldots, n \quad (1)$$

$$\hat{G}(r) = \frac{\{\#d_i : d_i \leq r, \forall i\}}{n} \quad (2)$$

- The **F** function: the distribution of the distances from an arbitrary **location** in the plane to its nearest observation; this is sometimes called the *empty space* function: it measures the average empty space between observed points. This is examined in §5, below.

A **homogeneous Poisson process** (HPP) is a process on the "landscape" by which points (occurrences, events ...) are produced at specific locations, and in which the points are independently and uniformly distributed over a given region. Thus the location of one point does not affect the location of other points; so another point can be anywhere. There are no clusters and no "empty" sub-regions, except by chance.

A homogeneous Poisson process with known *intensity* (a first-order property), conventionally called λ, will produce a CSR pattern, with the statistical properties:

$$G(r) = F(r) = 1 - \exp\{-\lambda\pi r^2\} \tag{3}$$

where r is the distance, and λ is the mean number of points per unit area, in the given distance units; it is also called the *intensity* of the process.

In this section we examine the G function, also known as the *nearest-neighbour-distance distribution function*. It is easily interpretable as the distance one must travel from any observation, to find at least one other observation. It is "short-sighted", since it only considers the nearest neighbour, and so gives no information about behaviour at long distances. It is a **point-related** function, i.e., computed from each point in the pattern. As such it gives no information about empty space; for that see the F function (§5), which is a **location-related** function

---

Task 4 : Compute and display the empirical vs. theoretical G function for the Japanese pines. •

The `Gest` function of the `spatstat` package computes this function on an object of class `ppp`, so we must first convert the class with the `as` method.

```
> G <- Gest(as(spjpines1, "ppp"))
> class(G)

[1] "fv"         "data.frame"

> summary(G)

       r               theo             han              rs
 Min.   :0.0000   Min.   :0.000   Min.   :0.000   Min.   :0.000
 1st Qu.:0.0594   1st Qu.:0.513   1st Qu.:0.519   1st Qu.:0.458
 Median :0.1187   Median :0.944   Median :0.970   Median :0.970
 Mean   :0.1187   Mean   :0.738   Mean   :0.735   Mean   :0.722
 3rd Qu.:0.1781   3rd Qu.:0.998   3rd Qu.:1.000   3rd Qu.:1.000
 Max.   :0.2374   Max.   :1.000   Max.   :1.000   Max.   :1.000
       km            hazard          theohaz
 Min.   :0.000   Min.   :   0.0   Min.   : 0.0
 1st Qu.:0.478   1st Qu.:   0.0   1st Qu.:24.2
 Median :0.958   Median :   0.0   Median :48.5
```

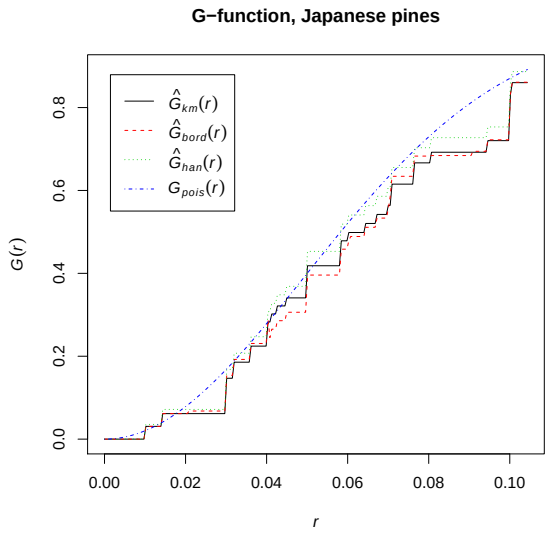
```
Mean   :0.723   Mean   :  13.3   Mean   :48.5
3rd Qu.:1.000   3rd Qu.:   0.0   3rd Qu.:72.7
Max.   :1.000   Max.   :1494.6   Max.   :97.0
```

The returned object has fields for the distance (`r`), the theoretical value of $G(r)$ (`theo`), and four variants of an empirical estimate of G. These differ in how edge effects are accounted for; see `?Gest` for an explanation of the options.

> **Note:** The "edge effect" occurs because any points produced by the spatial process but outside the window can not be observed. If not accounted for there is bias, because points near the edge may well have a nearer neighbour outside the window, but since that is not observed, we can not compute the distance to it. So, we presume there are such unobserved points, and they are "similarly" distributed as the ones we observe; statisticians have proposed various ways to implement this "similarity".

The `plot` method specializes to `plot.fv`, to handle the returned object of class `fv`; this is just a convenient structure for this sort of plot.

```
> plot(G, main = "G-function, Japanese pines")
```

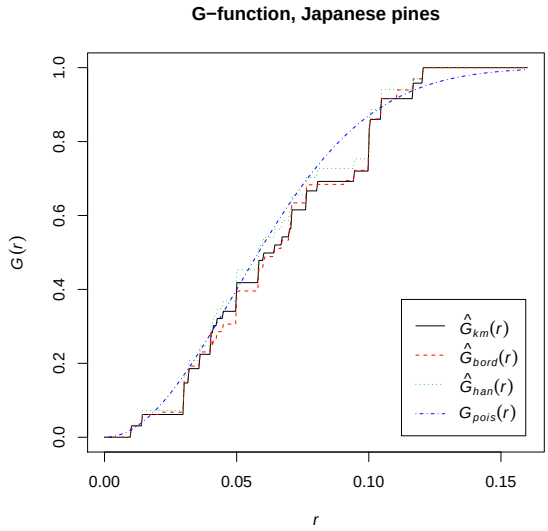


This graph can be interpreted as follows:

- r-axis: the distance away from an arbitrary point;
- $G(r)$-axis: the G function, namely, he proportion of points that have at least one neighbour within a distance r.

The default graph does not quite show the whole empirical function, i.e., where $G(r) = 1$, so we re-draw and specify the radius limits explicitly:

```
> plot(G, xlim = c(0, 0.16), main = "G-function, Japanese pines")
```



Note that at 0.16 the expected value of $G(r)$ is:

```
> 1 - exp(-japanesepines$n * pi * (0.16)^2)

[1] 0.99463
```

This is of course an asymptotic function, so never reaches 1; the empirical function does.

Q6 :

(a) What is the distance across the unit square at which at least 95% of the points have at least one neighbour within that distance, according to the "reduced sample" (`rs`), also called "border", method of edge correction?

(b) What proportion of points have at least one neighbour within 0.05 units?

Jump to A6 •

To answer this questions, we can make a visual estimate from the graph, or look inside the object; field `r` is the radius and `rs` is the reduced-sample estimate of $G(r)$. The selection function `which` picks out the entries meeting the required condition, and the `min` "minimum" function finds the first one in the list. For example, we can find the radius at which at least 95% of the points have at least one point within that threshold

```
> G$r[min(which(G$rs >= 0.95))]

[1] 0.11687

> G$rs[min(which(G$r >= 0.05))]

[1] 0.38776
```

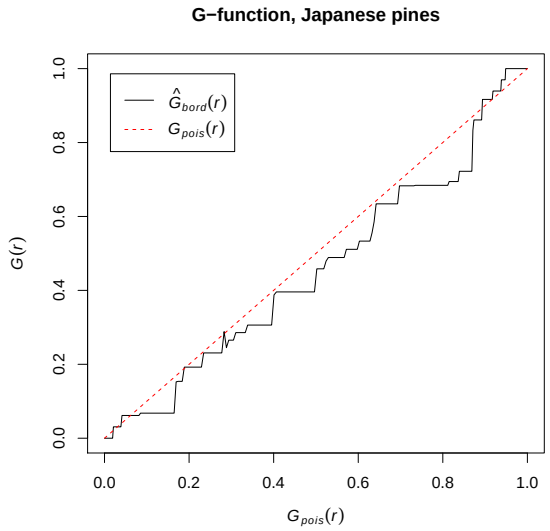
Q7 : *How closely do the four variants of the empirical G function match the*

theoretical G function for this intensity? Note this last is marked $G_{\text{pois}}(r)$ on the graph; the estimates are marked $(\hat{G})_{\text{method}}(r)$. *Jump to A7* •

Another way to look at this is as expected vs. actual, which should be a 1:1 relation.

Task 5 : Plot the actual reduced-sample estimate against the theoretical value of $G(r)$. •

```
> plot(G, cbind(rs,theo) ~ theo,
       main="G-function, Japanese pines")
```



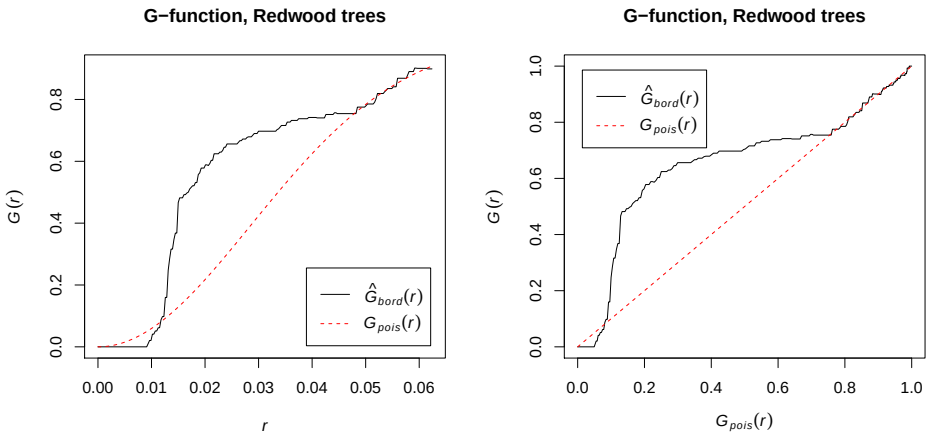
Here both axes are values of $G(r)$; the radius r is not shown.

Q8 : *How closely does the empirical match the theoretical?* *Jump to A8* •

Task 6 : Compute the G function and plot it for the other two datasets. •

First the Redwood trees:

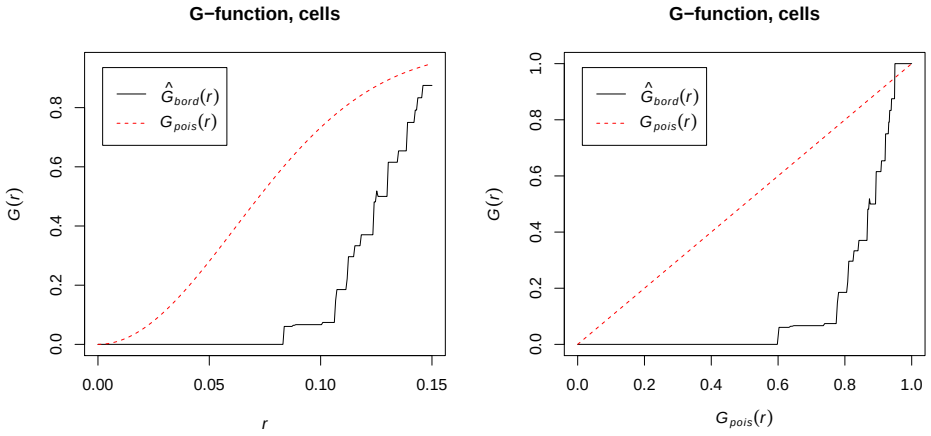
```
> ## 2 : example of clustered pattern?
> G.clust <- Gest(as(spred, "ppp"))
> par(mfrow=c(1,2))
> plot(G.clust, cbind(rs,theo) ~ r,
       main="G-function, Redwood trees")
> plot(G.clust, cbind(rs,theo) ~ theo,
       main="G-function, Redwood trees")
> par(mfrow=c(1,1))
```



Q9 : *Describe and interpret the differences between this G function and that for the Japanese pines.* *Jump to A9* •

Now the cells:

```
> ## 3 : example of dispersed / regular pattern?
> G.disp <- Gest(as(spcells, "ppp"))
> par(mfrow=c(1,2))
> plot(G.disp, cbind(rs,theo) ~ r,
      main="G-function, cells")
> plot(G.disp, cbind(rs,theo) ~ theo,
      main="G-function, cells")
> par(mfrow=c(1,1))
```



Q10 : *Describe and interpret the differences between this G function and that for the Japanese pines.* *Jump to A10* •

So far we have formed subjective opinions about which patterns are consistent with CSR. There is no formal test; the way this is assessed is to form an

envelope of how the empirical G function would look, under the null hypothesis of CSR and with the observed homogeneous intensity. This envelope is computed by a large number of **simulations** of the Poisson point process that gives rise to CSR. Each realization will be a bit different, because it's a discrete simulation, not the theoretical curve. Then the envelope is plotted along with the actual empirical G function, and we can see if it is contained inside, and if not, at what points it diverges.

Q11 : *Would simulations be more or less variable as the intensity of the process increases?* *Jump to A11* •

Task 7 : Compute an envelope for the G function of the Japanese pines, and plot it along with the observed empirical G function. •

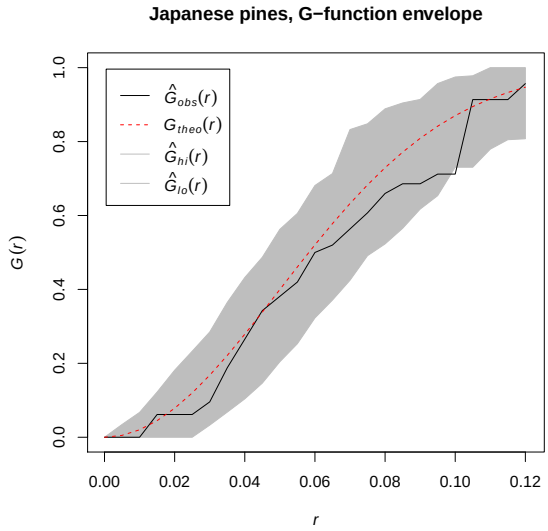
The `envelope` function of the `spatstat` package computes this for an object of class `ppp`. Arguments include the object, the function name (here, `Gest`), the radii at which to compute point-wise envelopes, the rank of the envelope value among the simulated values, and the number of simulations.

> **Note:** The `nrank` argument controls the width of the envelope. A low value, such as the default `nrank=1`, uses the extreme values (minimum and maximum) of the simulated distributions as the envelope; this is the widest and most conservative with respect to rejecting the null hypothesis of CSR. Using a higher rank corresponds roughly to setting the one-sided α for a t-test. In this case we have 99 simulations; so using `nrank=2` is excluding the single maximum and minimum at each point, thereby narrowing the envelope. This is roughly equivalent to one-sided $\alpha = (1-(2/99)) = 0.9798$, i.e., two-sided $\alpha \approx 96\%$ confidence level considering both minimum and maximum.

We try to simulate out to the radius where the empirical $G(r) = 1$. Also, we use the `set.seed` function to initialize the random number generator, so your results match ours. If we did not specify a large enough radius in the previous step, the G function may not reach 1, so we specify a somewhat smaller ending radius.

> **Note:** The argument to `set.seed` is arbitrary, it has no meaning.

```
> set.seed(30)
> rmax.jap <- G$r[min(which(G$rs > 0.98))]
> r <- seq(0, rmax.jap, by = 0.005)
> envjap <- envelope(as(spjpines1, "ppp"), fun=Gest,
                     r=r, nrank=2, nsim=99, verbose=F)
> plot(envjap, xlim=c(0, rmax.jap),
       main="Japanese pines, G-function envelope")
```

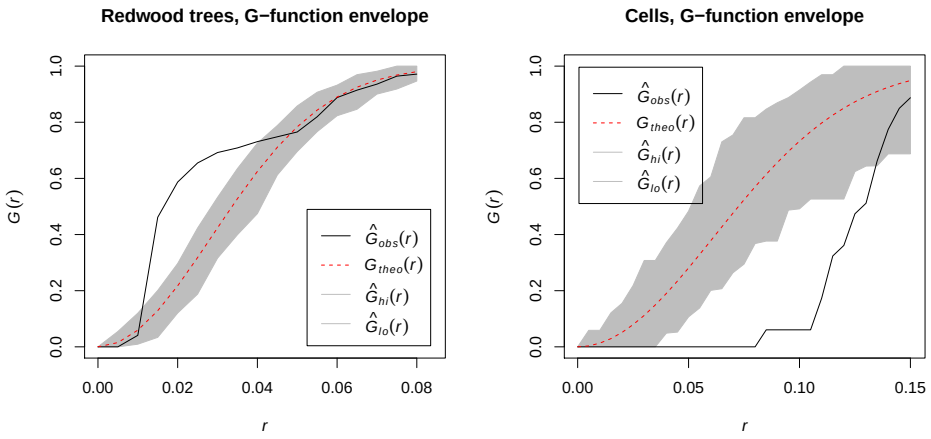


Q12 : *Is the empirical G function within the envelope throughout? Can we reject the null hypothesis of CSR?* *Jump to A12* •

Task 8 : Compute and plot the G function envelopes for the other two datasets. •

The maximum radius at which a point is encountered varies considerably; here we show each one with its own maximum:

```
> rmax.red <- G.clust$r[min(which(G.clust$rs > 0.98))]
> r <- seq(0, rmax.red, by = 0.005)
> envred <- envelope(as(spred, "ppp"), fun = Gest, r = r,
     nrank = 2, nsim = 99, verbose = F)
> rmax.cell <- G.disp$r[min(which(G.disp$rs > 0.98))]
> r <- seq(0, rmax.cell, by = 0.005)
> envcells <- envelope(as(spcells, "ppp"), fun = Gest,
     r = r, nrank = 2, nsim = 99, verbose = F)

> par(mfrow=c(1,2))
> plot(envred, xlim=c(0, rmax.red),
      main="Redwood trees, G-function envelope")
> plot(envcells, xlim=c(0, rmax.cell),
      main="Cells, G-function envelope")
> par(mfrow=c(1,1))
```



Q13 : *Describe and interpret the differences between the envelopes for these G functions and that for the Japanese pines.* *Jump to A13* •

3 Kernel density estimation

Supplementary reading:

- Bivand et al. [3, §7.4.1–3]: Statistical Analysis of Spatial Point Processes

An important question is whether the stochastic point-process is *homogeneous* (the same across the whole area) or *inhomogeneous*. Equivalently, is the *intensity* of the process the same everywhere? If not, we assume an *inhomogeneous Poisson process* (IPP), so that a single intensity λ describing the Poisson process is replaced by a spatial function $\lambda(\mathbf{x})$, where $\mathbf{x}$ is the spatial position. The question then arises: what is the spatial function of the density?

> **Note:** The assumption of homogeneity is often not realistic, from what we know about the process. For example, environmental factors favour some sub-areas over others for occurrence of a given tree species[3], so that the point-process by which a species is located can not be assumed to be homogeneous. However, the other assumption of the Poisson process still holds: given an intensity, events are independent and uniformly distributed.

Naturally, this depends on the scale at which we examine it. At broad scales, all points are taken together and the process is by definition homogeneous; at fine scales, very small neighbourhoods are considered and random fluctuations can lead to different intensity estimates. In other words, a small bandwidth will lead to a "spiky" map, whereas a large bandwidth will lead to a smooth map – which one best represents the (in)homogeneity of the point process? This is the *bandwidth* problem.

[3] E.g., the eastern hemlock (*Tsuga canadensis*) grows by preference in moist, shallow soils on hillsides, whereas beech (*Fagus* spp.) prefers well-drained hilltop positions

Among the proposals for selecting a bandwidth, a simple and effective one is from Berman and Diggle [2]. It is based on minimization of the mean square error (MSE) of the kernel smoothing estimator when compared to actual values, and has been implemented as function `mse2d` of the `splancs` "Spatial and Space-Time Point Pattern Analysis" package by Rowlingson and Diggle.

We apply this to the Redwood data, because we suspect from the previous section that these trees are clumped, so we should get a different process intensity over the area – some "holes" with low intensity, some "clumps" with high.

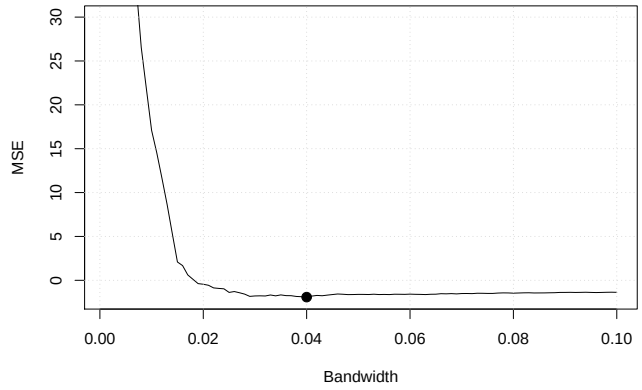
Task 9 : Compute the optimal bandwidth for the kernel density estimator for the Redwood dataset. •

The `mse2d` function expects a set of points, so we extract these from the `sp` object with the `coordinates` method; this returns a list of (x, y) points. This function takes as arguments (1) the points, (2) a bounding polygon, (3) the number of steps at which to compute the MSE, (4) the maximum bandwidth at which to compute the MSE. In this case we choose to compute from $[0 \ldots 0.1]$ (i.e., 1/10 of the bounding polygon in each dimension) in 100 steps.

```
> require(splancs)
> str(coordinates(spred))

 num [1:195, 1:2] 0.931 0.939 0.935 0.98 0.787 ...
 - attr(*, "dimnames")=List of 2
  ..$ : NULL
  ..$ : chr [1:2] "mx" "my"

> mse <- mse2d(pts=coordinates(spred),
              poly=as.points(list(x=c(0,1,1,0),
                  y=c(0,0,1,1))),
              nsmse=100,
              range=0.1)
> (bw <- mse$h[which.min(mse$mse)])

[1] 0.04
```

Task 10 : Graph the MSE vs. bandwidth. •

```
> plot(mse$h, mse$mse, xlab = "Bandwidth", ylab = "MSE",
     type = "l", ylim = c(-2, 30))
> grid()
> i <- which.min(mse$mse)
> points(mse$h[i], mse$mse[i], pch = 20, cex = 2)
```



Note that the minimum value of MSE is on a very flat part of the MSE vs. bandwidth curve, so the selected bandwidth is only barely "optimal".

The next question is the shape of the kernel. This is a function of the two coordinates, providing a smooth 2D surface with highest probability of finding a point at the centre and smoothly decreasing probability away from it. The default used by `spkernel2d` is the *quartic* kernel:

$$\kappa(u) = \begin{cases} \frac{3}{\pi}(1 - ||u||^2)^2 & \text{if } u \in (-1, 1) \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

where $||u||^2 = u_1^2 + u_2^2$, i.e., the squared norm, centred on the point to be estimated. Thus there is an inverse-square decrease in density outward from a point, to zero outside the unit circle; the "unit" is set by the bandwidth h, hence its importance:

$$u = ||x - x_i||/h \tag{5}$$

Although the kernel conceptually is spatially continuous, in practice it is computed at many points over a fine grid and displayed as a raster "image".

Task 11 : Plot the kernel density of the Redwoods dataset with the selected bandwidth. •

The kernel density, calculated by the `spkernel2d` function of the `splancs` package, takes the following arguments:

1. an `sp` object from which points can be extracted by the `coordinates` method;
2. one or more `splancs` polygons;
3. the bandwidth;
4. an object of class `GridTopology`; the kernel is evaluated at each grid intersection.

It returns returns a vector of kernel values in the order required by the `data` slot of a `SpatialGridDataFrame` object.

In the present case we make a single polygon, from the boundary as a list of coordinates $(0,0),(0,1),(1,1),(1,0)$:

```
> poly <- as.points(list(x = c(0, 0, 1, 1), y = c(0, 1,
    1, 0)))
```

Within this boundary we make a grid over which to apply the kernel, an object of class `GridTopology`, using the `GridTopology` method:

```
> grd <- GridTopology(cellcentre.offset = rep(0.005, 2),
    cellsize = rep(0.01, 2), cells.dim = rep(100, 2))
> str(grd)

Formal class 'GridTopology' [package "sp"] with 3 slots
  ..@ cellcentre.offset: num [1:2] 0.005 0.005
  ..@ cellsize         : num [1:2] 0.01 0.01
  ..@ cells.dim        : int [1:2] 100 100
```

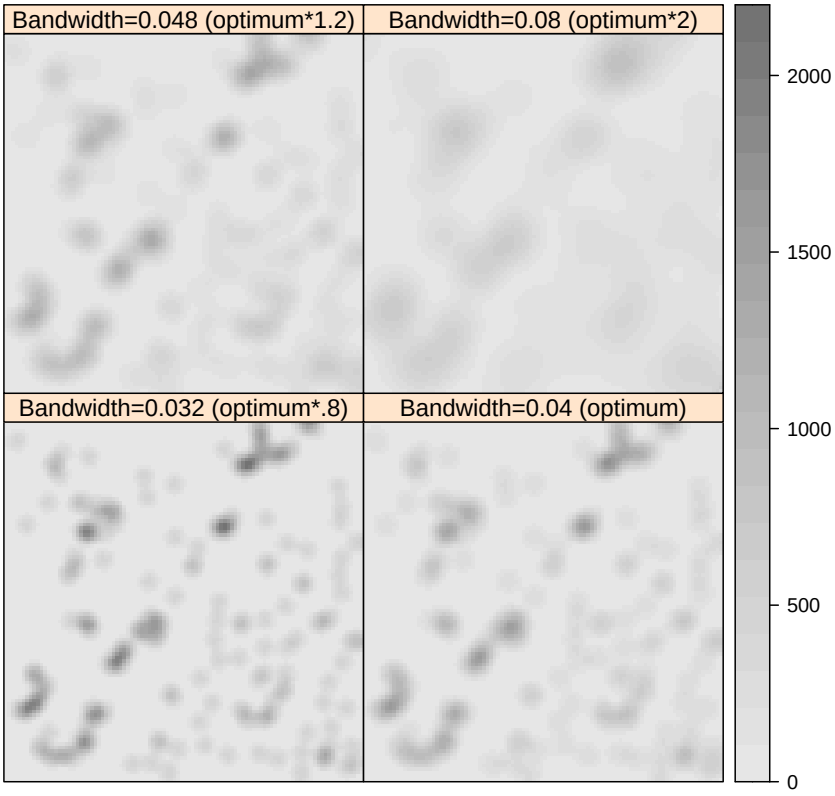
Now we can compute the kernel with various bandwidths: the "optimum" computed above and several multiples; we combine these as four fields of one `SpatialGridDataFrame`.

```
> k0 <- spkernel2d(spred, poly, h0 = bw * 0.8, grd)
> k1 <- spkernel2d(spred, poly, h0 = bw, grd)
> k2 <- spkernel2d(spred, poly, h0 = bw * 1.2, grd)
> k3 <- spkernel2d(spred, poly, h0 = bw * 2, grd)
> kernels <- SpatialGridDataFrame(grd, data = data.frame(k0 = k0,
    k1 = k1, k2 = k2, k3 = k3))
> summary(kernels)

Object of class SpatialGridDataFrame
Coordinates:
     min max
[1,]   0   1
[2,]   0   1
Is projected: NA
proj4string : [NA]
Grid attributes:
  cellcentre.offset cellsize cells.dim
1             0.005     0.01       100
2             0.005     0.01       100
Data attributes:
       k0                k1                k2                k3
 Min.   :   0.0   Min.   :   0.0   Min.   :   0.0   Min.   :  0.0
 1st Qu.:   0.0   1st Qu.:   0.0   1st Qu.:  11.6   1st Qu.: 88.1
 Median :  22.3   Median :  90.7   Median : 130.0   Median :165.1
 Mean   : 196.1   Mean   : 196.5   Mean   : 197.1   Mean   :200.0
 3rd Qu.: 284.0   3rd Qu.: 271.3   3rd Qu.: 263.2   3rd Qu.:256.2
 Max.   :2200.1   Max.   :1750.9   Max.   :1398.3   Max.   :937.7
```

Q14 : *What happens to the kernel density (intensity) statistics as the bandwidth increases?* *Jump to A14* •

Finally, these can all be graphed with the `spplot` method. We first visualize as continuous fields:

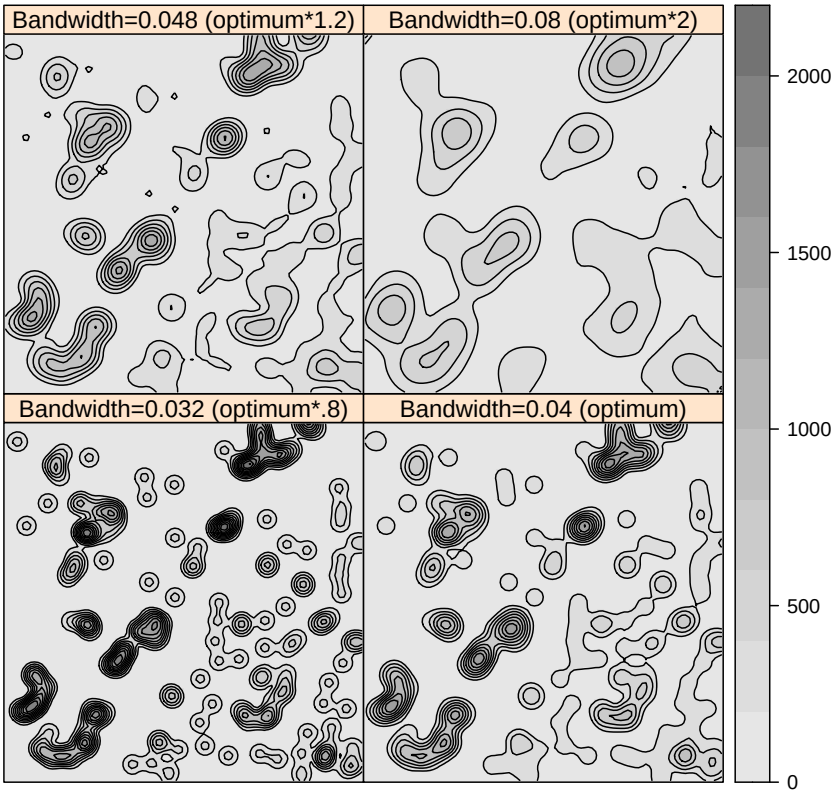
```
> gp <- grey.colors(5, 0.9, 0.45, 2.2)
> print(spplot(kernels,
              at=seq(0,max(k0,k1,k2,k3),length.out=22),
              col.regions=colorRampPalette(gp)(21),
 names.attr=c(
   paste("Bandwidth=",bw*.8, " (optimum*.8)", sep="", collapse=""),
   paste("Bandwidth=",bw, " (optimum)",sep="", collapse=""),
   paste("Bandwidth=",bw*1.2, " (optimum*1.2)",sep="", collapse=""),
   paste("Bandwidth=",bw*2, " (optimum*2)", sep="", collapse="")
   )))
```



We can emphasize the pattern by adding contours at the same intervals as the grey scale, using the optional `contour` argument to the `spplot` method. After some experimentation we reduce the number of intervals from 22 (above) to 12, to avoid too-dense contours:

```
> gp <- grey.colors(5, 0.9, 0.45, 2.2)
> print(spplot(kernels, at = seq(0, max(k0, k1, k2, k3),
     length.out = 12), col.regions = colorRampPalette(gp)(11),
     contour = T, names.attr = c(paste("Bandwidth=", bw *
         0.8, " (optimum*.8)", sep = "", collapse = ""),
         paste("Bandwidth=", bw, " (optimum)", sep = "",
             collapse = ""), paste("Bandwidth=", bw *
             1.2, " (optimum*1.2)", sep = "", collapse = ""),
```

```
    paste("Bandwidth=", bw * 2, " (optimum*2)", sep = "",
        collapse = ""))))
```



Q15 : *Describe the trend in kernel density as the bandwidth increases. Does the "optimum" found by minimizing the MSE seems to give the "best" view of the varying intensity of the point process that is assumed to be causing the Redwood point pattern ? (Hint: compare with the point pattern).* *Jump to A15* •

4 Second-order properties: the K function

Supplementary reading:

- Bivand et al. [3, §7.4.5]: Second-order properties

A *second-order* property of a point process refers to the *interactions* between points[4]. Examples are clustering (attraction) or competition (repulsion), with obvious ecological interest. Below (§6.2) we examine interactions between two types of points; here we consider one type of points, but at any distances. The F and G functions are "short-sighted", they only consider nearest neighbours to an arbitrary point or location, respectively. Here we consider any radius from an arbitrary point.

[4] Recall: the *first-order* property refers to properties at a single point, e.g., the intensity of the process

Ripley [12] proposed a K function for quantifying second-order properties for a HPP. It counts the number of points within a given distance of a point, and is defined as:

$$K(s) = \lambda^{-1} E[N_0(s)] \tag{6}$$

where $E[.]$ is the expectation and $N_0(s)$ is the number of points found within radius s of point x_0 (an arbitrary point of the point pattern). This expectation is computed as:

$$\hat{K}(s) = (n(n-1))^{-1}|A| \sum_{i=i}^{n} \sum_{j \neq i} w_{ij}^{-1} \cdot x_j \ : \ d(x_i, x_j) \leq s \tag{7}$$

where the *radius* $d(x_i, x_j)$ is the distance between two points, and the *weights* w_{ij} are the proportion of the area inside region A, of size $|A|$, of the circle centred on the target point x_i. The term $(n(n-1))^{-1}$ normalizes for the total number of point-pairs.

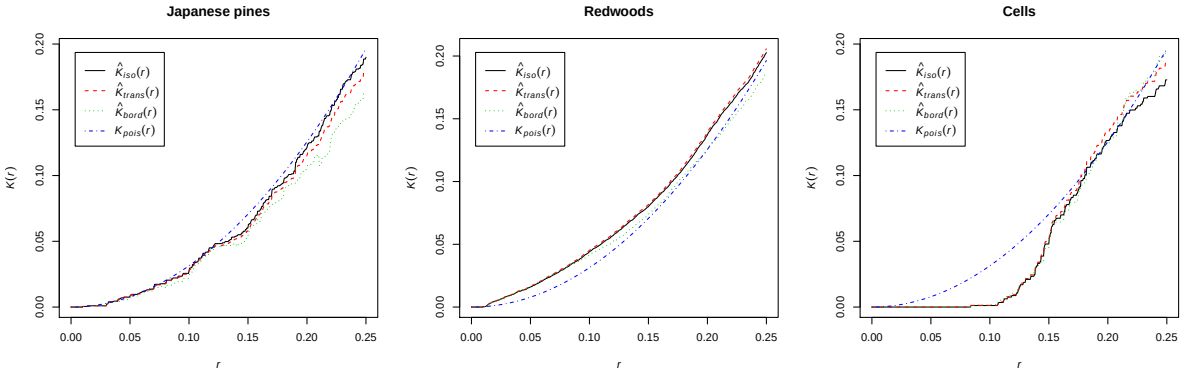
For the HPP, we have $K(s) = \pi s^2$, i.e., the area of a circle with radius s. If $K(s)$ is greater, this indicates clustering, i.e., more points than expected with the radius; the inverse indicates a regular (dispersed) process.

Task 12 : Compute and graph the K function for the three example datasets. •

We use the `Kest` function of the `spatstat` package:

```
> Kjap <- Kest(as(spjpines1, "ppp"))
> Kred <- Kest(as(spred, "ppp"))
> Kcells <- Kest(as(spcells, "ppp"))
```

```
> par(mfrow = c(1, 3))
> plot(Kjap, main = "Japanese pines")
> plot(Kred, main = "Redwoods")
> plot(Kcells, main = "Cells")
> par(mfrow = c(1, 1))
```



As with the G function, there are several border corrections; see `help(Kest)` for details.

Q16 : *Do these graphs provide evidence of second-order CSR, clustering, or dispersion?* *Jump to A16* •

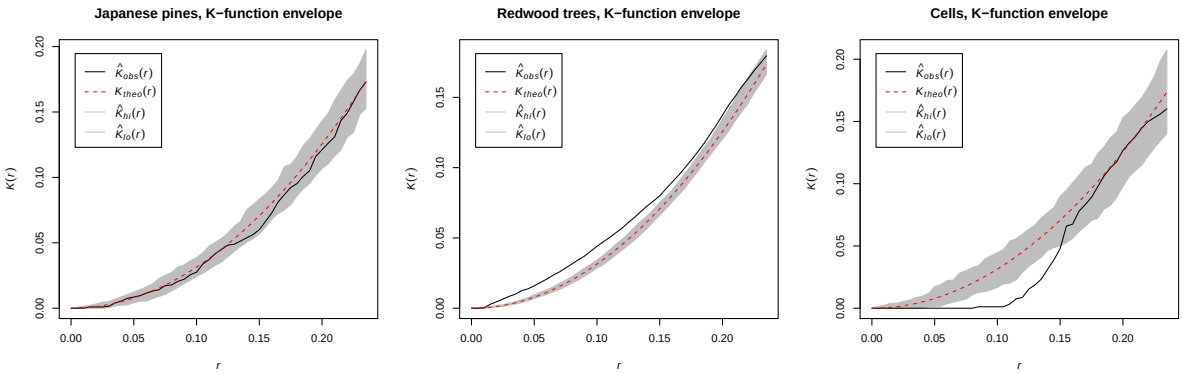
We can also compute a simulation envelope for the K function, in the same manner as for the G function.

Task 13 : Compute and graph simulation envelopes for the K function, for the three example datasets. •

We choose to display these to a radius that is 1/3 across the diagonal of the unit bounding box.

```
> r <- seq(0, sqrt(2)/6, by = 0.005)
> envjap <- envelope(as(spjpines1, "ppp"), fun = Kest,
      r = r, nrank = 2, nsim = 99, verbose = F)
> envred <- envelope(as(spred, "ppp"), fun = Kest, r = r,
      nrank = 2, nsim = 99, verbose = F)
> envcells <- envelope(as(spcells, "ppp"), fun = Kest,
      r = r, nrank = 2, nsim = 99, verbose = F)
```

```
> par(mfrow = c(1, 3))
> plot(envjap, main = "Japanese pines, K-function envelope")
> plot(envred, main = "Redwood trees, K-function envelope")
> plot(envcells, main = "Cells, K-function envelope")
> par(mfrow = c(1, 1))
```



These envelopes confirm the interpretations.

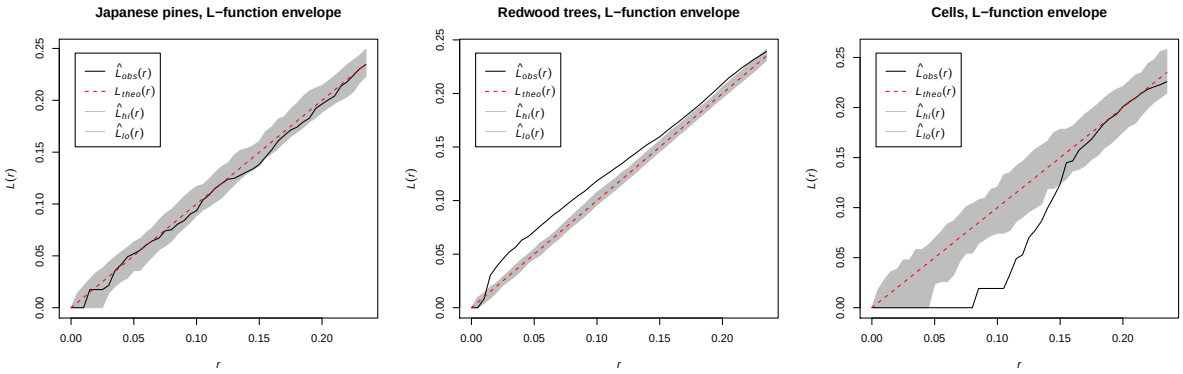
4.1 The L function: a linearized K function

A linear version of K may be easier to interpret; therefore Besag proposed a function $L(r) = \sqrt{K(r)/\pi}$. All this does is linearize the expected value, so it appears on the plot as a straight line,

Task 14 : Compute and plot the L function and their envelopes for the three patterns. •

```
> r <- seq(0, sqrt(2)/6, by = 0.005)
> envjap <- envelope(as(spjpines1, "ppp"), fun = Lest,
      r = r, nrank = 2, nsim = 99, verbose = F)
> envred <- envelope(as(spred, "ppp"), fun = Lest, r = r,
      nrank = 2, nsim = 99, verbose = F)
> envcells <- envelope(as(spcells, "ppp"), fun = Lest,
      r = r, nrank = 2, nsim = 99, verbose = F)
```

```
> par(mfrow = c(1, 3))
> plot(envjap, main = "Japanese pines, L-function envelope")
> plot(envred, main = "Redwood trees, L-function envelope")
> plot(envcells, main = "Cells, L-function envelope")
> par(mfrow = c(1, 1))
```



4.2 * Modifying the window

Recall that an object of class `ppp` includes a `window` of class `owin`, that defines the window in which the point-pattern is evaluated. Suppose we want to only evaluate the point-pattern in some smaller area. To do this, we can create a new window with the `owin` function, and then use it to extract just those points that fall in the window.

Task 15 : Create a window covering the upper left-hand (NW) quadrant of the Japanese pines point-pattern. Extract just the Japanese pines points in this window and plot them. •

We use the `owin` function to specify the new window, and then the `inside.owin` logical function to determine which points are in the new window. We then use the logical vector to select the points in the new window, and save these as a new point pattern.

```
> str(japanesepines, max.level = 1)
```

```
List of 5
 $ window    :List of 4
  ..- attr(*, "class")= chr "owin"
 $ n         : int 65
 $ x         : num [1:65] 0.09 0.29 0.38 0.39 0.48 0.59 0.65 0.67 0.73 0.79 ...
 $ y         : num [1:65] 0.09 0.02 0.03 0.18 0.03 0.02 0.16 0.13 0.13 0.03 ...
 $ markformat: chr "none"
 - attr(*, "class")= chr "ppp"
```

```
> print(japanesepines$window)
```

```
window: rectangle = [0, 1] x [0, 1] units (one unit = 5.7 metres)
```

```
> (window.nw <- owin(xrange = c(0, 0.5), yrange = c(0.5,
     1)))
```

```
window: rectangle = [0, 0.5] x [0.5, 1] units
```

```
> table(is.in <- inside.owin(japanesepines, w = window.nw))
```

```
FALSE  TRUE
   43    22
```

```
> japanesepines.nw <- japanesepines[is.in]
```

We see only 22 of the 65 Japanese pines are in this window.

However, this selection does not change the window size. We can see this with the `Windowunction` of the `spatstat` package:

```
> Window(japanesepines.nw)
```

```
window: rectangle = [0, 1] x [0, 1] units (one unit = 5.7 metres)
```

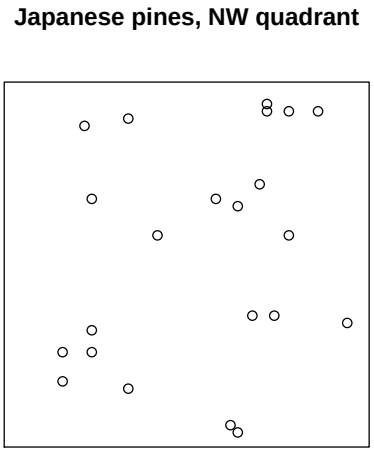
If we want to reduce the window size of the new point pattern, we again use the `Window` function to set the window size:

```
> Window(japanesepines.nw) <- window.nw
> Window(japanesepines.nw)
```

```
window: rectangle = [0, 0.5] x [0.5, 1] units (one unit = 5.7 metres)
```

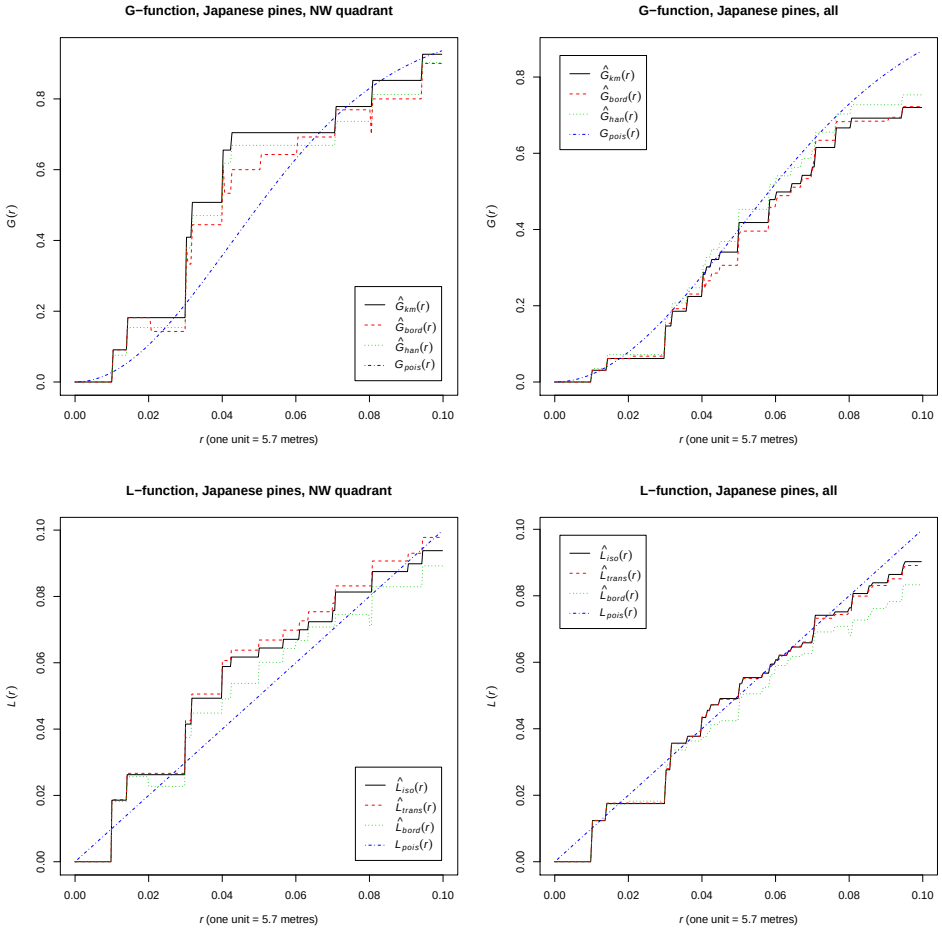
Now we can plot the reduced window and its points:

```
> plot(japanesepines.nw, main = "Japanese pines, NW quadrant")
```



Task 16 : Compute and plot the G and L functions for this subset; compare with these functions for the full set. Limit the plot radius to 0.10, i.e., the close-range part of the function •

```
> par(mfrow = c(2, 2))
> G <- Gest(japanesepines.nw)
> plot(G, main = "G-function, Japanese pines, NW quadrant",
      xlim = c(0, 0.1))
> G <- Gest(japanesepines)
> plot(G, main = "G-function, Japanese pines, all", xlim = c(0,
      0.1))
> L <- Lest(japanesepines.nw)
> plot(L, main = "L-function, Japanese pines, NW quadrant",
      xlim = c(0, 0.1))
> L <- Lest(japanesepines)
> plot(L, main = "L-function, Japanese pines, all", xlim = c(0,
      0.1))
> par(mfrow = c(1, 1))
```



It's clear that the smaller window, with fewer points, results in a more irregular function. The G-function is considerably different: for the quadrant we see some dispersal around $r = 0.05$; this is not seen in the full point-pattern.

5 The F function; non-rectangular windows

The G function explored in §2 is a **point-related** function, i.e., computed from each point in the pattern. Another way to examine point distribution is with a **location-related** function, i.e., computed from any location, whether or not it is a point. This provides information about empty space. Such a function developed from the theory of Poisson processes is the F "empty space" function, which we examine in this section.

However, there is a complication. The above examples used rectangular windows "filled" with the point pattern. The implicit assumption (which we now make explicit) is that the data-generating process (i.e., process by which the points were placed) operates over the whole window, and in some border area outside the window. The process may not be homogeneous, as we saw in the kernel density estimation (§3), but it does "fill" the window – there is a probability that any location in the window could have a point.

However, if the rectangular window includes areas that were not observed or not part of the study, there will be "white space" which appears as part of the pattern, but is not. In that case some statistics will be misleading, in particular, the "empty space" F function, and any plots will include areas that are not interesting.

Another issue is that we may be given a point-pattern that is clearly only filling part of a map, and we want to extract that area. An example is the point-pattern of trees from which we want to derive the boundary of a forest. The `spatstat` package has several useful functions for that purpose.

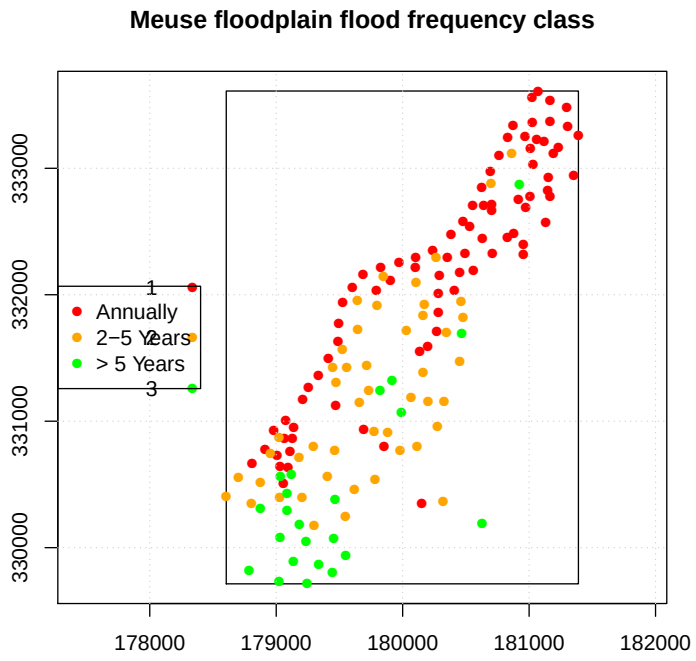
We illustrate the process of specifying a window with the `meuse` example dataset provided with the `sp` package. This is a set of observation points in the river Maas (Meuse) floodplain near the village of Stein, Limburg province, Netherlands.

Task 17 : Load the `meuse` example dataset, restrict it to just the Pb content and flooding frequency attributes, convert to a spatial object of class `SpatialPointsDataFrame`, make an equivalent point-pattern object of class `ppp`, and plot as a marked point-pattern, marked by the flooding frequency (field `ffreq`). •

Here we use the generic `as` method, specialized in the `maptools` package to convert from `sp` to `ppp`; the `meuse` data set is first converted to a `SpatialPointsDataFrame` with the `coordinates` method.

```
> require(sp)
> data(meuse)
> meuse <- meuse[, c("x", "y", "lead", "ffreq")]
> coordinates(meuse) <- ~x + y
> meuse.ppp <- as(meuse, "ppp")
> str(meuse.ppp)

List of 6
 $ window    :List of 4
  ..$ type  : chr "rectangle"
  ..$ xrange: num [1:2] 178605 181390
  ..$ yrange: num [1:2] 329714 333611
  ..$ units :List of 3
  .. ..$ singular  : chr "unit"
  .. ..$ plural    : chr "units"
  .. ..$ multiplier: num 1
  .. ..- attr(*, "class")= chr "unitname"
  ..- attr(*, "class")= chr "owin"
 $ n         : int 155
 $ x         : num [1:155] 181072 181025 181165 181298 181307 ...
 $ y         : num [1:155] 333611 333558 333537 333484 333330 ...
 $ markformat: chr "dataframe"
 $ marks     :'data.frame':        155 obs. of  2 variables:
  ..$ lead : num [1:155] 299 277 199 116 117 137 132 150 133 80 ...
  ..$ ffreq: Factor w/ 3 levels "1","2","3": 1 1 1 1 1 1 1 1 1 1 ...
 - attr(*, "class")= chr "ppp"
```

```
> tmp <- plot(meuse.ppp, use.marks = TRUE, cols = c("red",
      "orange", "green"), chars = 16, which.marks = "ffreq",
      maxsize = 100, main = "Meuse floodplain flood frequency class",
      axes = T)
> grid()
> legend("left", pch = 16, col = c("red", "orange", "green"),
      legend = c("Annually", "2-5 Years", "> 5 Years"))
```



We can see that the point-pattern only partially fills the bounding rectangle. By default, the type conversion to class `ppp` defines the window as the rectangular bounding box of the point-pattern; we can see this as the `window` field of the `ppp` object:

```
> meuse.ppp$window

window: rectangle = [178605, 181390] x [329714, 333611] units
```

Task 18 : Compute a bounding window and replace the rectangular boundary with it. •

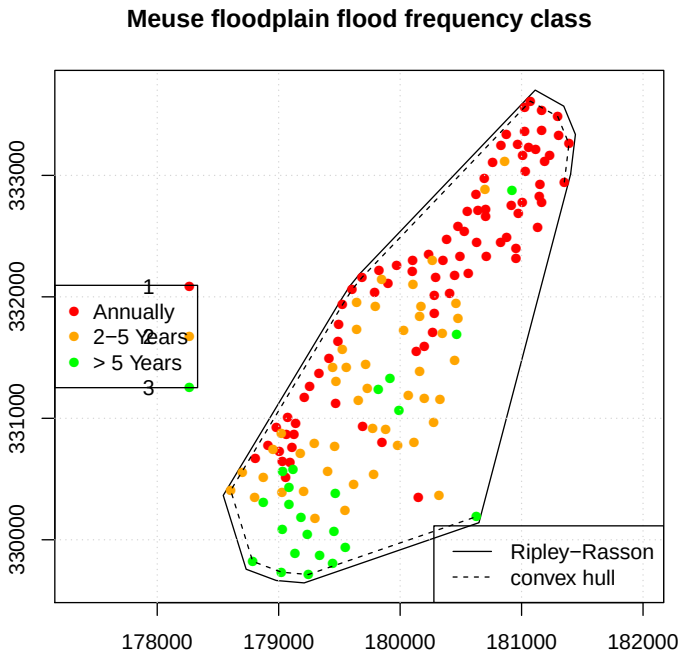
We compute the window as the Ripley and Rasson [13] estimate of the spatial domain. This is a clever way of expanding the convex hull (which contains the outermost points) consistent with the intensity of the pattern.

We use the `ripras` "Ripley–Rasson forest edge" function to compute the window; we plot this along with the convex hull computed by the `convexhull` function.

Note: Both `ripras` and `convexhull` return an object of class `owin`; this includes the boundary in field `bdry` as a list of coördinate vectors.

To plot a point pattern we use the `plot.ppp` function, which is automatically called by the generic `plot` method for an object of class `ppp`.

```
> meuse.ppp.r <- meuse.ppp
> (meuse.ppp.r$window <- ripras(meuse.ppp))

window: polygonal boundary
enclosing rectangle: [178544, 181443] x [329645, 333702] units

> tmp <- plot(meuse.ppp.r, use.marks = TRUE, cols = c("red",
     "orange", "green"), chars = 16, which.marks = "ffreq",
     maxsize = 100, main = "Meuse floodplain flood frequency class",
     boundary = 2, axes = T)
> grid()
> legend("left", pch = 16, col = c("red", "orange", "green"),
     legend = c("Annually", "2-5 Years", "> 5 Years"))
> ch <- convexhull(meuse.ppp)
> lines(ch$bdry[[1]]$x, ch$bdry[[1]]$y, lty = 2)
> legend("bottomright", lty = 1:2, legend = c("Ripley-Rasson",
     "convex hull"))
```



Q17 : *Describe the polygonal window.* *Jump to A17* •

Q18 : *How did changing the boundary affect the mean intensity of the point process?* *Jump to A18* •

The `intensity` function of the `spatstat` package computes the intensity from the number of points and the window area:

```
> str(meuse.ppp)

List of 6
 $ window    :List of 4
  ..$ type  : chr "rectangle"
  ..$ xrange: num [1:2] 178605 181390
  ..$ yrange: num [1:2] 329714 333611
  ..$ units :List of 3
  .. ..$ singular  : chr "unit"
  .. ..$ plural    : chr "units"
  .. ..$ multiplier: num 1
  .. ..- attr(*, "class")= chr "unitname"
  ..- attr(*, "class")= chr "owin"
 $ n         : int 155
 $ x         : num [1:155] 181072 181025 181165 181298 181307 ...
 $ y         : num [1:155] 333611 333558 333537 333484 333330 ...
 $ markformat: chr "dataframe"
 $ marks     :'data.frame':        155 obs. of  2 variables:
  ..$ lead : num [1:155] 299 277 199 116 117 137 132 150 133 80 ...
  ..$ ffreq: Factor w/ 3 levels "1","2","3": 1 1 1 1 1 1 1 1 1 1 ...
 - attr(*, "class")= chr "ppp"

> intensity(meuse.ppp)

[1] 1.4282e-05

> intensity(meuse.ppp.r)

[1] 2.6367e-05

> round(100 * intensity(meuse.ppp.r)/intensity(meuse.ppp))

[1] 185
```

Clearly, this is not a perfect boundary of the area from which the points were taken; we know from the documentation that it is bounded by a large meander of the river Maas (Meuse), and is limited on the east side by a canal (the Julianakanaal) and steep cliff, so ideally we'd have a bounding polygon of the actual study area. Absent this, the Ripley-Rasson method at least restricts the area.

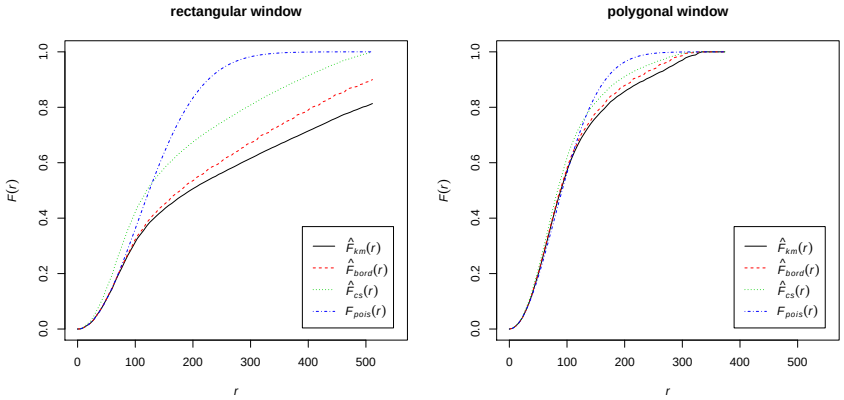
> **Note:** See §A.2 for how to import a polygonal boundary in ESRI shapefile format, and use it for the window.

A major effect of reducing the window to the actual area sampled is to properly estimate the "empty space" function, i.e., average distance from an arbitrary location in the window to the nearest point (event).

Task 19 : Compute and plot the "empty space" function F for the Meuse point-pattern in the rectangular and polygonal windows. •

The `GFest` function of the `spatstat` package computes this function on an object of class `ppp`. We specify the same x-axes to compare the functions side-by-side:

```
> par(mfrow = c(1, 2))
> plot(Fest(meuse.ppp), main = "rectangular window", xlim = c(0,
    550))
> plot(Fest(meuse.ppp.r), main = "polygonal window", xlim = c(0,
    550))
> par(mfrow = c(1, 2))
```



As with the G function, there are several cumulative functions; $F_{\text{pois}}(r)$ is the theoretical distribution in the case of CSR; $F_{\text{bord}}(r)$ is the same corrected for border effects; the estimates are marked $\hat{F}_{\text{method}}(r)$ for Kaplan-Meier ("km"), Chiu-Stoyan ("cs"). Note the different x-axis scales of the two plots.

Q19 : *What proportion of space is expected to have at least one point within 100 m for the rectangular and polygonal windows?* *Jump to A19* •

Q20 : *Describe the agreement (or lack thereof) of the observed $\hat{F}_{\text{km}}(r)$ with the theoretical for CSR $F_{\text{pois}}(r)$.* *Jump to A20* •

6 Marked point patterns

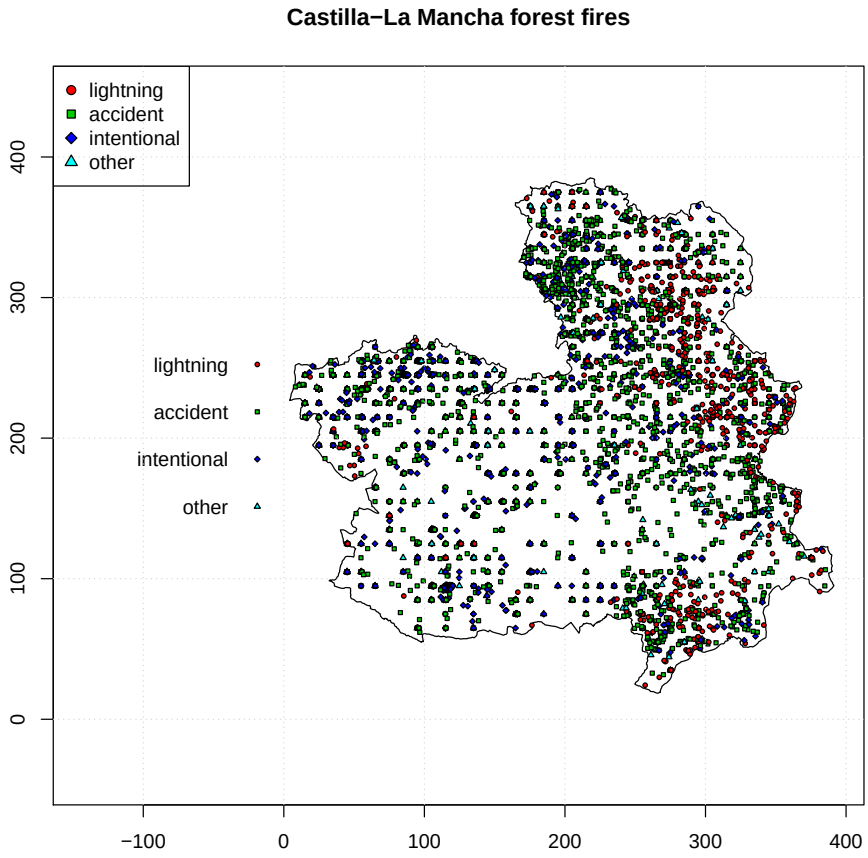
A **marked** point pattern is one where each point has some attribute; this can be a continuous value (e.g., tree size) (§6.4) or a categorical attribute (e.g., tree species, tree size class) (§6.1).

6.1 Categorical marks

We first examine a point-pattern marked with a **categorical** attribute: the "forest fires" dataset `clmfires`, supplied as an example in the `spatstat` package. This is a record of forest fires (1998-2007) in the Castilla-La Mancha region (E). For each fire there are four types of marks, i.e., attributes: cause, date, day of year, and size.

Task 20 : Load the `clmfires` dataset and display the locations of the fires, along with their cause. •

```
> data(clmfires)
> plot(clmfires, which.marks = "cause", bg = 2:5, chars = 21:24,
      cex = 0.5, axes = T, main = "Castilla-La Mancha forest fires")
> grid()
> legend("topleft", pch = 21:24, legend = levels(clmfires$marks$cause),
      pt.bg = 2:5)
```



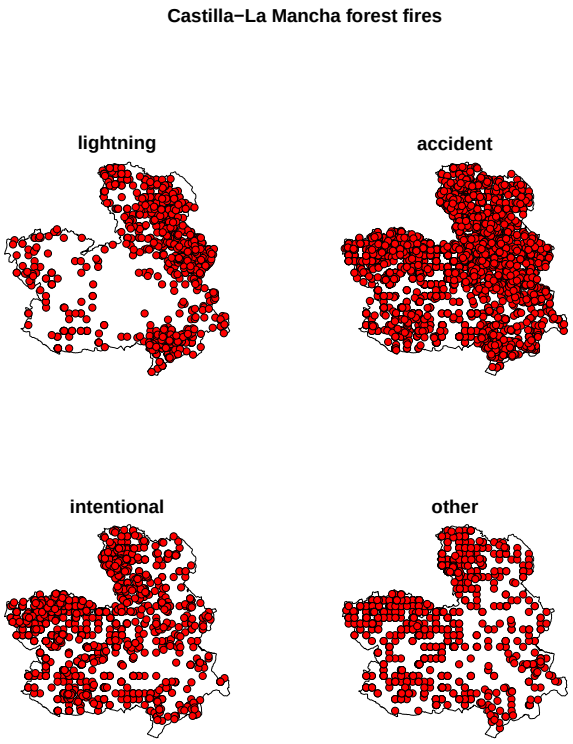
Q21 : *What do the marks represent? How many classes are there?* *Jump to A21* •

Task 21 : Split the marked point-pattern and show the pattern for each mark separately. •

The `split.ppp` method of the `spatstat` package specializes the generic `split` method. Similarly, the `plot.splitppp` method of the `spatstat` package specializes the generic `plot` method

Here we don't need to plot any mark types, since we've already split on the cause. So the `use.marks` argument to `plot.splitppp` is set to `FALSE`.

```
> clmfires.split <- split(clmfires)
> str(clmfires.split, max.level=1)

List of 4
 $ lightning  :List of 6
  ..- attr(*, "class")= chr "ppp"
 $ accident   :List of 6
  ..- attr(*, "class")= chr "ppp"
 $ intentional:List of 6
  ..- attr(*, "class")= chr "ppp"
 $ other      :List of 6
  ..- attr(*, "class")= chr "ppp"
 - attr(*, "class")= chr [1:4] "splitppp" "ppplist" "solist" "list"
 - attr(*, "fsplit")= Factor w/ 4 levels "lightning","accident",..: 3 1 1 1 4 4 2
 - attr(*, "fgroup")= Factor w/ 4 levels "lightning","accident",..: 3 1 1 1 4 4 2

> plot(clmfires.split, use.marks=FALSE,
       main="Castilla-La Mancha forest fires",
       pch=21, bg=2)
```



Q22 : *Do the fires with different causes appear to have different point patterns?* *Jump to A22* •

We could compare the separate patterns with the usual G, F, J, K or L functions and compare the function plots visually.

6.2 Interaction between point patterns: the cross-K function

Another question can be raised when there are several patterns covering the same area: what is the **interaction** between them? That is, do occurrences of one mark "attract" or "repel" those of other marks?

> **Note:** The quotes for "attract" and "repel" remind us that we need meta-statistical information to propose the causes of observed interactions.

We call such a process a **multi-type** process, that is, we assume that there may be some interaction between the types. In the current example, we may expect that an area burned with one kind of fire would not be susceptible to another kind of fire, because the necessary fuel would have been removed by the first fire. We assume that the multi-type process is stationary across

the area.

The appropriate statistic to investigate this is a so-called "cross" point-pattern function, from mark type i to mark type j or vice-versa. For example, a crossed K function of a stationary multi-type point process with intensity λ_j of point type j is defined so that $\lambda_j K_{ij}(r)$ is the expected number of *additional* random points of type j within a distance r of a typical point of type i.

The $K_{ij}(r)$ function plotted over a range of distances is used to form hypotheses about the multi-type point pattern. If the two point-processes are independent, the expected value $K_{ij}(r) = \pi r^2$, that is, the number of additional points just depends on the area of the circle centred on a source point. If the empirical K_{ij} function is above the theoretical function πr^2 (i.e., a parabola), there are more points of type j near to the source points of type i than expected; this suggest dependence between the processes. If the empirical function is below the theoretical, this suggests repulsion or avoidance.

Task 22 : Compute and plot the cross-K function for the relation between intentional and lightning-induced fires. •

The `Kcross` function computes Ripley's K, as for the univariate case (function `Kest`), but the measure is the number of neighbours *of another pattern* within a radius of a given point[5]. However, this function works on an "multi-type point-pattern" object, which is a point-pattern with a single mark. The `clmfires` object has four kinds of marks:

```
> str(clmfires$marks)

'data.frame':        8488 obs. of  4 variables:
 $ cause      : Factor w/ 4 levels "lightning","accident",..: 3 1 1 1 4 4 2 4 2 2
 $ burnt.area : num  0.4 0 0.4 0 1.05 3 0.1 0.02 0.4 2.85 ...
 $ date       : Date, format: "1998-01-07" ...
 $ julian.date: num  6 6 6 6 6 7 7 7 8 8 ...
```

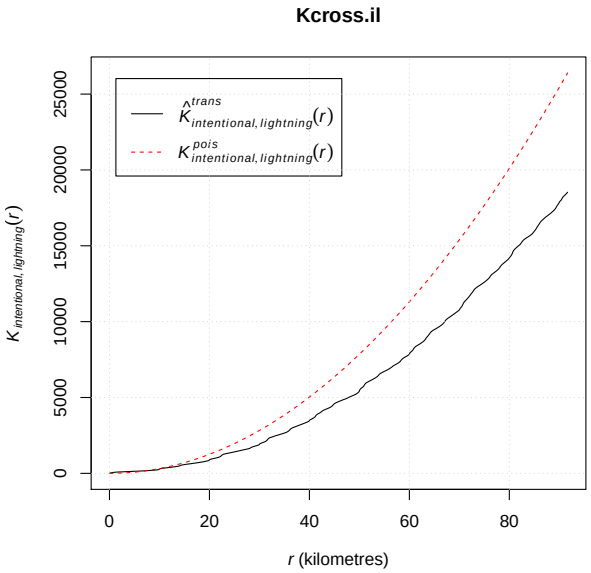
So, we make an object with just a single mark, i.e., the causes:

```
> clmfires.cause <- clmfires
> is.multitype(clmfires.cause)

[1] FALSE

> clmfires.cause$marks <- clmfires$marks$cause
> is.multitype(clmfires.cause)

[1] TRUE
```

Now we can compute the cross-K function, using `Kcross`. We specify the 'translation' edge correction, suitable for complex geometries such as the province boundaries:

[5] There are similar analogues of the L, G, and J functions, but not the F "empty space" function.

```
> Kcross.il <- Kcross(clmfires.cause, "intentional", "lightning",
      correction = "translate")
> plot(Kcross.il)
> grid()
```



Q23 : *Is there evidence for interaction between the processes that produce the intentional and accidental fires?* *Jump to A23* •

6.3 Combining point patterns

We may have two or more unmarked point patterns which represent different types of points, which we want to combine into a marked point pattern. For example, the `redwoodfull` and `japanesepines` point patterns represent two kinds of trees as unmarked point patterns. If we suppose these two patterns are from the same area[6] we may ask what is the relation between them. We can discover this with the cross-K function, if we can combine them into a single multi-type marked point pattern.

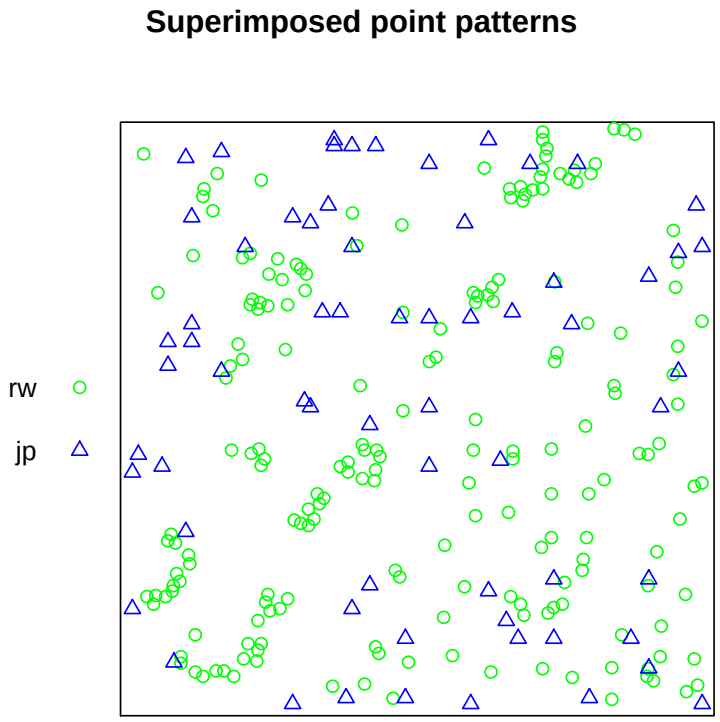
Task 23 : Combine the `redwoodfull` and `japanesepines` point patterns into a single multi-type marked point pattern. •

The `superimpose` function of the `spatstat` package superimpose several point patterns. These can optionally be supplied with marks applied to all points in each pattern.

```
> two.trees <- superimpose(rw = redwoodfull, jp = japanesepines)
> str(two.trees)
```

[6] which is not true, but allows us to illustrate the techniques

```
List of 6
 $ window    :List of 4
  ..$ type  : chr "rectangle"
  ..$ xrange: num [1:2] 1e-09 1e+00
  ..$ yrange: num [1:2] 1e-09 1e+00
  ..$ units :List of 3
  .. ..$ singular  : chr "metre"
  .. ..$ plural    : chr "metres"
  .. ..$ multiplier: num 5.7
  .. ..- attr(*, "class")= chr "unitname"
  ..- attr(*, "class")= chr "owin"
 $ n         : int 260
 $ x         : num [1:260] 0.931 0.939 0.935 0.98 0.787 ...
 $ y         : num [1:260] 0.818 0.764 0.722 0.665 0.661 ...
 $ markformat: chr "vector"
 $ marks     : Factor w/ 2 levels "rw","jp": 1 1 1 1 1 1 1 1 1 1 ...
 - attr(*, "class")= chr "ppp"

> plot(two.trees, main = "Superimposed point patterns",
     cols = c("green", "blue"))
```

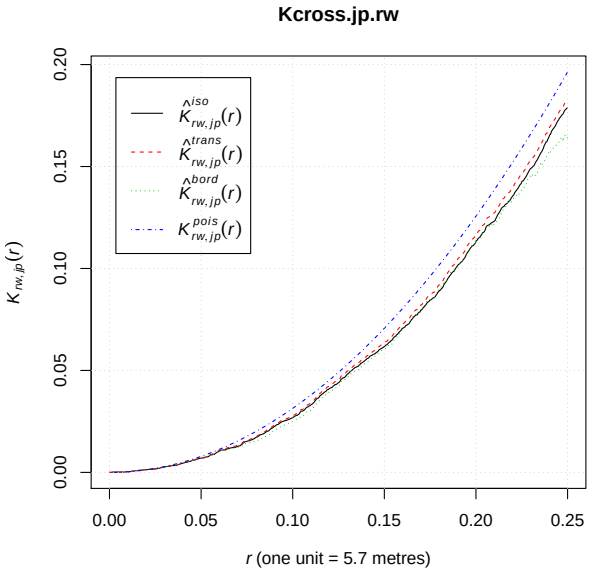


In this case the windows had the same extent; by default a union of the windows of the superimposed point patterns is used. The optional `W` "Window"

argument provides several additional ways to specify a window.

Task 24 : Compute the crossed K-function for these two tree species. •

```
> Kcross.jp.rw <- Kcross(two.trees)
> plot(Kcross.jp.rw)
> grid()
```



Q24 : *Does there appear to be any interaction between the point processes that produced these two tree species? Is this surprising?* *Jump to A24* •

6.4 Continuous marks

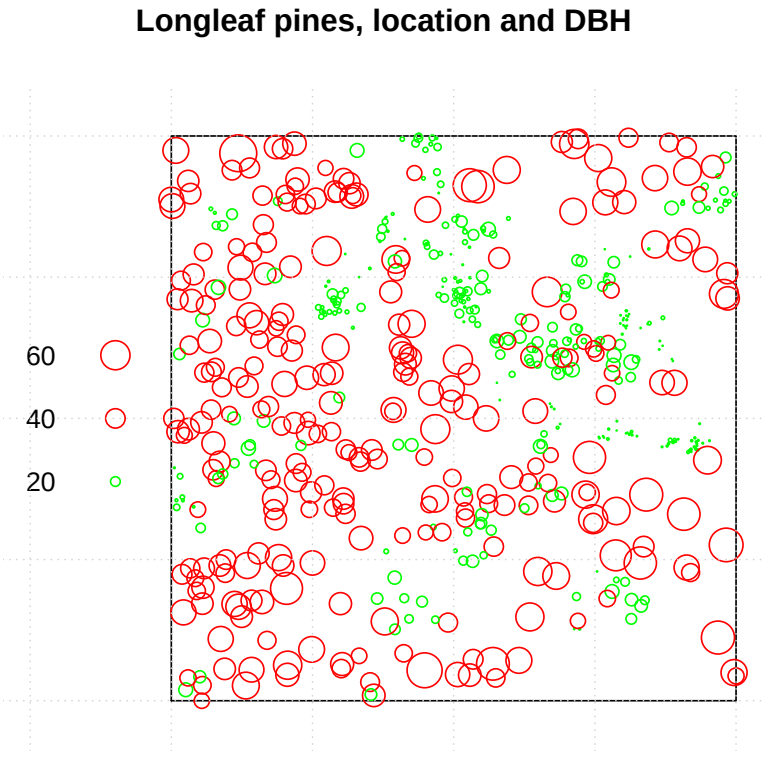
The marks on a point-pattern may be continuous variables rather than categories. An example is the `longleaf` dataset, which shows the locations and diameters at breast height (DBH) of 584 longleaf pines (*Pinus palustris*) in a 200 x 200 metre region in southern Georgia (USA)

Task 25 : Load and display this point pattern. Show the mature trees with a red symbol, and saplings with a green symbol. •

The `summary` function summarizes the dataset. To just see the window size, use the `Window` function (note the capital "W").

```
> data(longleaf)
> summary(longleaf)

Marked planar point pattern:  584 points
Average intensity 0.0146 points per square metre
```

```
Coordinates are given to 1 decimal place
i.e. rounded to the nearest multiple of 0.1 metres

marks are numeric, of type ‘Ă‘Ÿdouble‘Ă‘Ź
Summary:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    2.0     9.1    26.1    26.8    42.1    75.9

Window: rectangle = [0, 200] x [0, 200] metres
Window area = 40000 square metres
Unit of length: 1 metre

> Window(longleaf)

window: rectangle = [0, 200] x [0, 200] metres

> plot(longleaf, main = "Longleaf pines, location and DBH",
     cols = function(x) ifelse(x < 30, "green", "red"))
> grid()
```



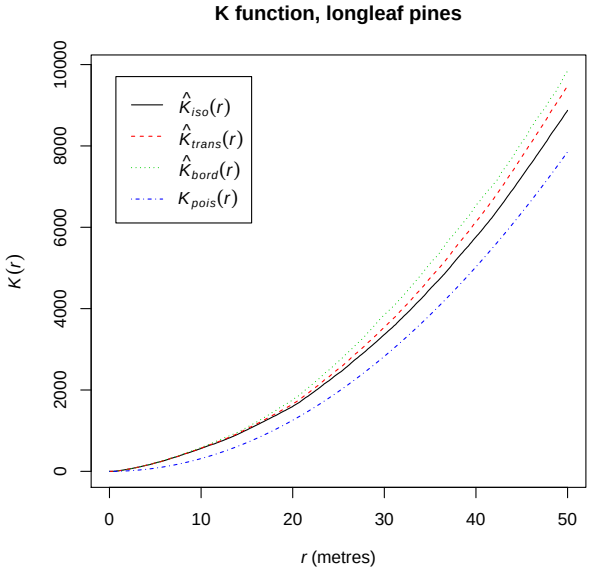
Q25 : *Do the trees appear to be clustered, regularly-spaced, or randomly placed? Do trees of similar size appear to be clustered? What appears to*

be the relation between mature trees and saplings? *Jump to A25* •

Task 26 : Compute and plot the K function for the longleaf pines. •

Again the estimated K function is computed with `Kest`.

```
> K.long <- Kest(longleaf)
> plot(K.long, main = "K function, longleaf pines")
```



The trees clearly show some clustering at all distances to 50 m.

7 Models of spatial processes

Supplementary reading:

- Bivand et al. [3, §7.4.4]: Likelihood of an inhomogeneous Poisson process

The observed point pattern is presumably the **realization** of some **point process**, i.e., a spatial data generating process (sDGP) by which points, also called "events", are placed on the landscape. These could be completely random with some intensity (a **homogeneous** Poisson process), random but with varying intensity across the region (an **inhomogeneous** Poisson process), a process depending on inter-point interactions, depending on a regional trend, depending on environmental covariables, or any combination. If we can fit a model to the observed process we can (1) infer the sDGP which generated it; (2) map the results of the process.

The result of such models is a **conditional intensity** $\lambda(u, \mathbf{x})$, a function of the location u and the observed point pattern $\mathbf{x}$. The units are the number of points per unit area. In practice we compute this over some "small" cell.

Baddeley and Turner [1] explain how to fit stochastic models to observed point patterns with the versatile `ppm` function of the `spatstat` package. The issue of modelling is quite deep and you are encouraged to read this paper before building your own models; here we only show some possibilities.

The general formula for models that can be fit with `ppm` is [1, Eqn. (4)]:

$$\lambda(u, \mathbf{x}) = \exp\left(\psi^T B(u) + \phi^T C(u, \mathbf{x})\right) \tag{8}$$

where the two components are:

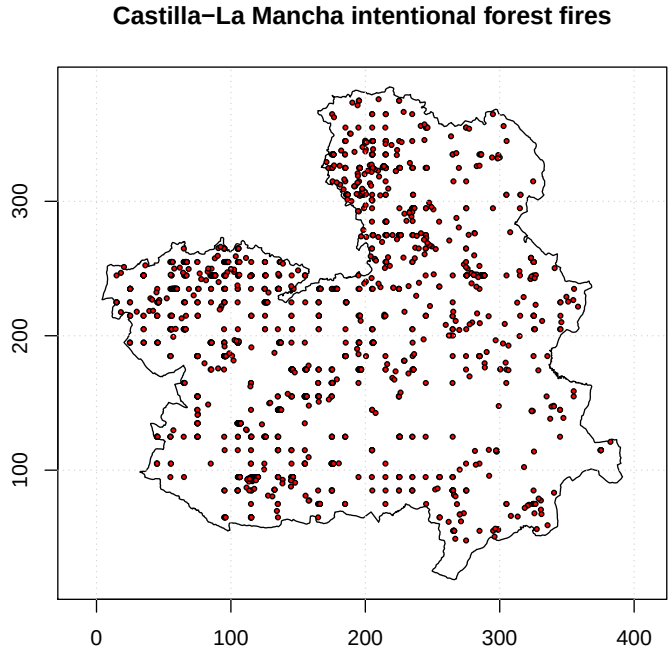
1. the **spatial trend** $B(u)$ which depends only on location; this could also include covariates at these locations;
2. the **stochastic interactions** $C(u, \mathbf{x})$, i.e., the dependence between the points of the point process.

The analyst specifies the forms of B and C and `ppm` estimates the coefficients (ψ, ϕ).

We continue with the Castilla-La Mancha forest fires example of §6. In this modelling exercise we subset the whole dataset to just one kind of fire; it should be easier to interpret the model results.

Task 27 : Restrict the dataset to intentional fires. •

```
> clmfires.i <- split(clmfires, "cause")$intentional
> plot(clmfires.i, chars = 21, cex = 0.5, bg = 2, axes = T,
     main = "Castilla-La Mancha intentional forest fires",
     use.marks = FALSE)
> grid()
```



Task 28 : Remove the marks from the point pattern. •

This is necessary because `ppm` is not yet implemented for marked patterns. Here we remove the marks for the entire pattern; it would also be possible to split the pattern according to a mark using the `split.ppp` function, remove the marks from each of the sub-patterns and analyze each one.

```
> marks(clmfires.i) <- NULL
```

7.1 Null model

Task 29 : Model the forest fire incidence as a Poisson process, i.e., complete spatial randomness (CSR). This is a "null" model because it is the simplest hypothesis about how points are placed. •

This is the simplest conditional intensity: $\lambda(u, \mathbf{x}) = \beta$, where β is a single intensity of a (presumed) homogeneous Poisson process. In the terminology of Eqn. (8), both B (trend) and C (interaction) are absent.

We specify this to `ppm` by setting the `trend` argument to `~1`, i.e., the process only has a mean intensity. We also specify the type of interaction between points by setting the `interaction` argument to `NULL`[7].

```
> print(m.pois <- ppm(clmfires.i, trend = ~1, interaction = NULL))
```

[7] These are both defaults and so don't have to be explicitly specified.

```
Stationary Poisson process
Intensity: 0.022507
            Estimate     S.E. CI95.lo CI95.hi Ztest    Zval
log(lambda)  -3.7939 0.023662 -3.8403 -3.7476   *** -160.34

> class(m.pois)

[1] "ppm"

> exp(coef(m.pois))

log(lambda)
   0.022507

> intensity(clmfires.i)

[1] 0.022507

> (clmfires.i$n/summary(clmfires.i)$window$areas)

[1] 0.022507
```

A model fitted by `ppm` is of class `ppm`.

This is not a very interesting model, since we could get the same result simply from the average intensity, using the `intensity` function or even direct computation from the number of points and the window area. The only complication is that `ppm` works with the logarithm of the parameters, in this case just the Poisson intensity β. Notice however the standard error and confidence intervals that are provided with the model summary.

7.2 Trend surface

A more complex model is $\lambda(u, \mathbf{x}) = \beta(u)$, where $\beta(u)$ is a variable intensity, dependent on the location u, of a (presumed) inhomogeneous Poisson process. This is termed a *trend*, which may be a function of the coördinates or of covariables. In the terminology of Eqn. (8), B (trend) is defined but C (interaction) are absent. The form of B is a trend surface, i.e., a polynomial function of the coördinates.

Q26 : *Does there seem to be a regional trend in intentional forest fire intensity?* *Jump to A26* •

Task 30 : Model the intentional forest fire incidence as first- and second-order regional trend plus a Poisson process, i.e., complete spatial randomness (CSR) after accounting for a trend. •

We specify the trend to `ppm` by setting the `trend` argument to a formula; for example `~x+y` for a first-order trend: the intensity changes linearly along some plane to be computed. Here we use the `polynom` function to specify both first- and second-order trend surface:

```
> (m.ts1 <- ppm(clmfires.i, trend = ~polynom(x, y, 1),
     interaction = NULL))

Nonstationary Poisson process

Log intensity:  ~x + y

Fitted trend coefficients:
(Intercept)           x           y
 -3.2878809  -0.0057327   0.0028847

              Estimate       S.E.    CI95.lo    CI95.hi Ztest    Zval
(Intercept) -3.2878809 0.07317041 -3.4312923 -3.1444695   *** -44.935
x           -0.0057327 0.00027073 -0.0062633 -0.0052021   *** -21.175
y            0.0028847 0.00030260  0.0022916  0.0034778   ***   9.533

> (m.ts2 <- ppm(clmfires.i, trend = ~polynom(x, y, 2),
     interaction = NULL))

Nonstationary Poisson process

Log intensity:  ~x + y + I(x^2) + I(x * y) + I(y^2)

Fitted trend coefficients:
(Intercept)           x           y      I(x^2)    I(x * y)
-5.2689e+00  1.2021e-02  9.5359e-03 -3.4094e-05 -2.9700e-05
     I(y^2)
-2.2298e-06

               Estimate       S.E.     CI95.lo     CI95.hi Ztest
(Intercept) -5.2689e+00 2.5420e-01 -5.7671e+00 -4.7707e+00   ***
x            1.2021e-02 1.6868e-03  8.7148e-03  1.5327e-02   ***
y            9.5359e-03 1.6157e-03  6.3692e-03  1.2703e-02   ***
I(x^2)      -3.4094e-05 3.5830e-06 -4.1116e-05 -2.7071e-05   ***
I(x * y)    -2.9700e-05 4.6701e-06 -3.8853e-05 -2.0547e-05   ***
I(y^2)      -2.2298e-06 3.5262e-06 -9.1410e-06  4.6813e-06
                 Zval
(Intercept) -20.72765
x             7.12658
y             5.90203
I(x^2)       -9.51551
I(x * y)     -6.35954
I(y^2)       -0.63237
```

Now we see the fitted coefficients β; the intercept is the overall log-intensity at $(0, 0)$ (the lower-left corner of the pattern) and the coefficients show the change intensity in the x and y directions, along with their standard errors and confidence intervals; recall these are logarithms, so we convert to original units to interpret them. Here we see an increase in intensity towards the WNW, almost equal in both axes. This accords with our visual estimate.

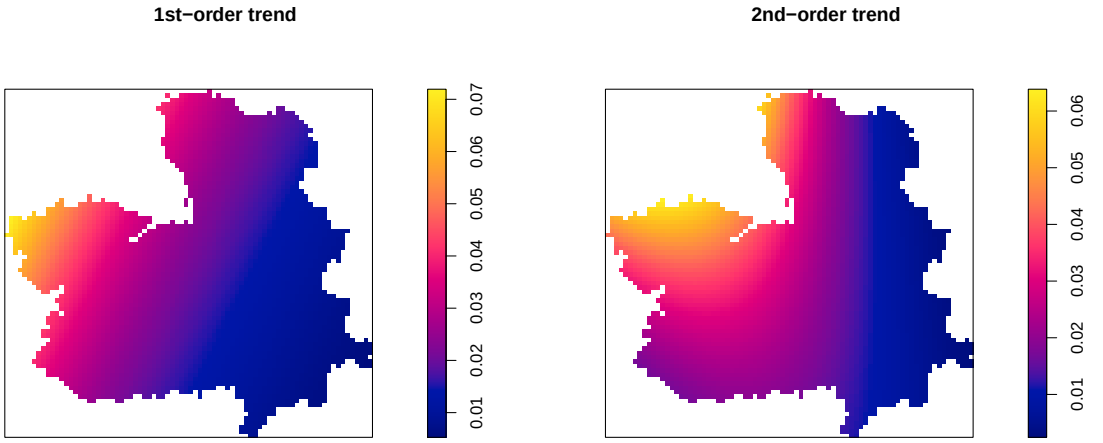
Task 31 : Plot the trend surfaces. •

The `plot.ppm` function calls `predict.ppm` (see below, §8) to compute the

spatial trend and conditional intensity of the fitted point process model on a grid, and then displays the result; by default the grid is 40 by 40 pixels filling the bounding box.

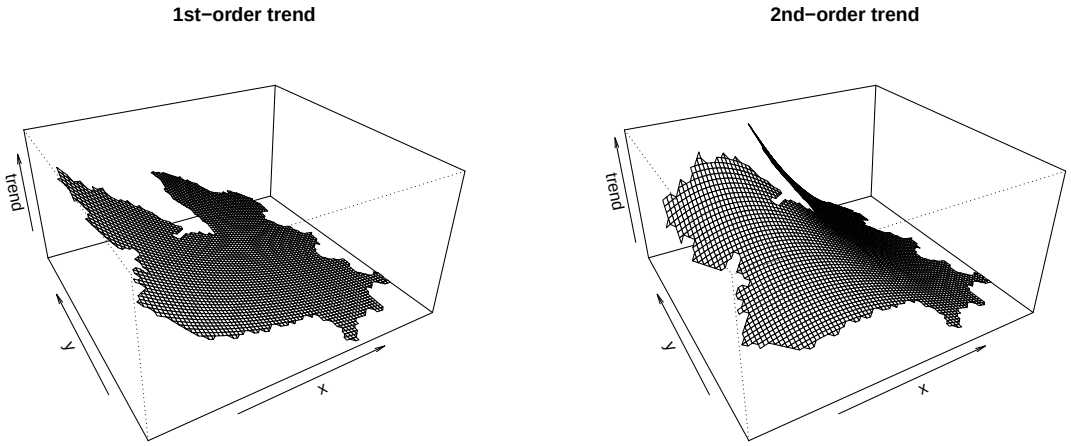
We first visualize these by a colour ramp 2.5D plot; the `how` argument specifies the type of plot:

```
> par(mfrow = c(1, 2))
> plot.ppm(m.ts1, ngrid = c(80, 80), how = "image", superimpose = F,
     trend = T, se = F, pause = F, main = "1st-order trend")
> plot.ppm(m.ts2, ngrid = c(80, 80), how = "image", superimpose = F,
     trend = T, se = F, pause = F, main = "2nd-order trend")
> par(mfrow = c(1, 1))
```



We can also see the trends as perspective plots:

```
> par(mfrow = c(1, 2))
> plot.ppm(m.ts1, ngrid = c(80, 80), how = "persp", theta = -30,
    phi = 30, trend = T, se = F, pause = F, main = "1st-order trend")
> plot.ppm(m.ts2, ngrid = c(80, 80), how = "persp", theta = -30,
    phi = 30, trend = T, se = F, pause = F, main = "2nd-order trend")
> par(mfrow = c(1, 1))
```



Task 32 : Compare the goodness-of-fit of the Poisson process and the trend models. •

The `anova.ppm` function performs analysis of deviance for two or more fitted models with Poisson interaction terms, i.e., independence:

```
> anova(m.ts2, m.ts1, m.pois)

Analysis of Deviance Table

Model 1: ~x + y + I(x^2) + I(x * y) + I(y^2)          Poisson
Model 2: ~x + y          Poisson
Model 3: ~1          Poisson
  Npar Df Deviance
1    6
2    3 -3     -126
3    1 -2     -505
```

Here we see that the trend surfaces uses more degrees of freedom but both reduce the residual deviance slightly, the second-order more than the first.

7.3 Strauss process

So far we've treated the observations as independent (a Poisson process), possibly influenced by a regional trend in overall intensity. Another possibility is that fires are not independent. In the terminology of Eqn. (8), B (trend)

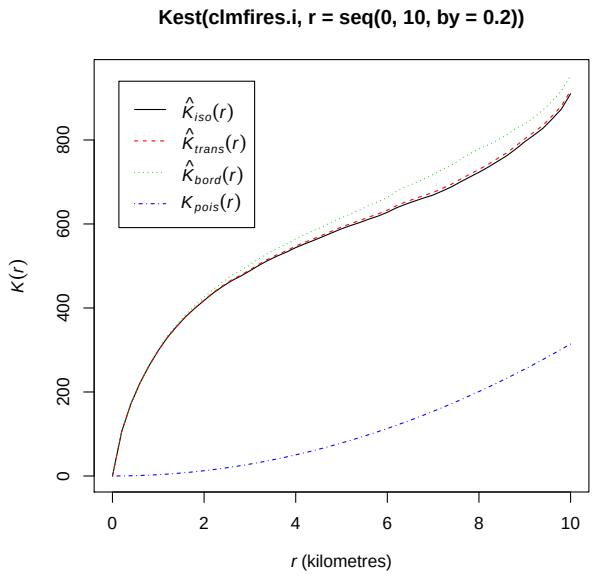
is absent but C (interaction) is present; the analyst must define the form of C. One way to model that is as a **Strauss process**: $\lambda(u, \mathbf{x}) = \beta\gamma^{t(u,\mathbf{x})}$, where β is the overall homogeneous intensity and γ is an **interaction** parameter $0 \leq \gamma \leq 1$ and $t(u, \mathbf{x})$ is the number of points of the pattern $\mathbf{x}$ closer than the **interaction radius** r of the location u. This has an interesting interpretation: $\gamma = 0 \Longrightarrow \lambda = 0$, that is, within the radius there is no chance of finding another point, perhaps because of the intrinsic size of a "point". With $\gamma < 1$ the chance of a second point is reduced, at $\gamma = 1$ this is equivalent to a Poisson process (no effect one way or the other).

> **Note:** A Strauss process is an example of a so-called **Gibbs process**, derived from physics to model repulsion; they include an intensity and an interaction function.

Task 33 : Fit a homogeneous Strauss process model to the forest fire incidence and plot the resulting surface. •

We must choose an interaction radius; this can be based on our hypothesis of how fires interact. We investigate this with the K function. Above (§4) we saw that this measures the number of points within a given radius of a given point, as a function of radius.

```
> plot(Kest(clmfires.i, r = seq(0, 10, by = 0.2)))
```



There seems to be an inflection point around 4 km, so we pick this as an interaction radius.

```
> (m.strauss.4 <- ppm(clmfires.i, trend=~1,
                      interaction=Strauss(4)))

Stationary Strauss process

First order term:  beta = 0.017687
```

```
Interaction distance:        4
Fitted interaction parameter gamma:         1.07813

Relevant coefficients:
Interaction
   0.075224

For standard errors, type coef(summary(x))

*** Model is not valid ***
*** Interaction parameters are outside valid range ***

> exp(coef(m.strauss.4))

(Intercept) Interaction
   0.017687    1.078125
```

There are two fitted parameters: the overall log intensity β and the strength-of-interaction parameter γ. This latter was fit as 1.0781, i.e., $\gamma > 1$, so there is on average clustering.

Task 34 : Compare the likelihood of the several fitted models. •

The `logLik` function shows the log-likelihood of the fitted parameters:

```
> data.frame(
     model=c("Poisson","1st order trend", "2nd order trend", "Strauss"),
     likelihood=c(logLik(m.pois, warn=F),
         logLik(m.ts1, warn=F),
         logLik(m.ts2, warn=F),
         logLik(m.strauss.4, warn=F)))

            model likelihood
1         Poisson    -8562.0
2 1st order trend    -8306.8
3 2nd order trend    -8243.5
4         Strauss    -6776.1
```

Clearly the Strauss model is superior: there is local clustering, not CSR, and this is a better fit to the observations than either trend surface.

7.4 Covariates

The observed point pattern may well depend on environmental factors. For example, density of trees in a forest may depend on soil type, elevation, temperature or rainfall. Baddeley and Turner [1, §7] explain how how to include covariates, such as environmental factors, in the model. The `clmfires` dataset is accompanied by a raster dataset `clmfires.extra`, a list of two objects of class `im`, also defined by `spatstat`, which is a matrix of images (i.e., a layer stack); one of the objects in the list is `clmcov200`, a 200 x 200 pixels grid in the same coördinate system as `clmfires`, showing four possible covariates that might affect fire incidence: elevation, orientation (aspect),

slope and landuse. The `anova.ppm` function uses this image to extract the value of the covariable(s) at the locations of observed events (points).

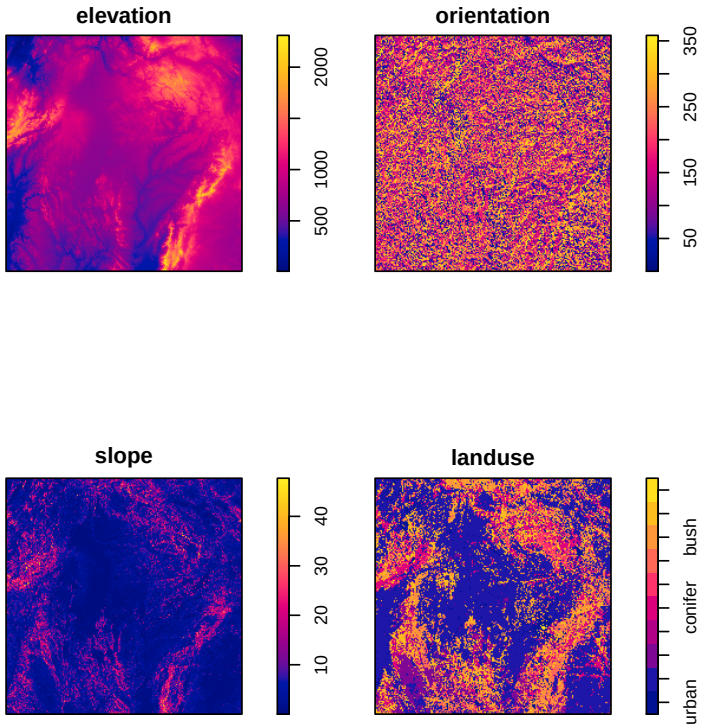
Loading `clmfires` also loaded the covariate images as object `clmfires.extra`:

```
> names(clmfires.extra)

[1] "clmcov100" "clmcov200"

> names(clmfires.extra$clmcov200)

[1] "elevation"   "orientation" "slope"       "landuse"

> names(clmfires.extra$clmcov200$landuse)

 [1] "v"      "dim"    "xrange" "yrange" "xstep"  "ystep"  "xcol"
 [8] "yrow"   "type"   "units"

> names(clmfires.extra$clmcov200$landuse)

 [1] "v"      "dim"    "xrange" "yrange" "xstep"  "ystep"  "xcol"
 [8] "yrow"   "type"   "units"

> levels(clmfires.extra$clmcov200$landuse$v)

 [1] "urban"       "farm"        "meadow"      "denseforest"
 [5] "conifer"     "mixedforest" "grassland"   "bush"
 [9] "scrub"       "artifgreen"
```

Task 35 : Display the covariate images. •

The generic `plot` method specializes to `plot.im` for objects of class `im`.

```
> plot(clmfires.extra$clmcov200, main = "200 m grid covariates")
```

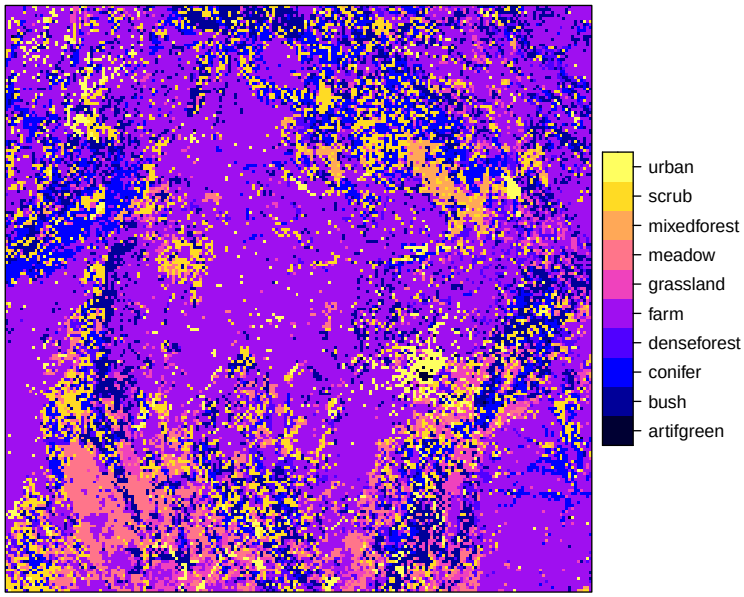


We can get a better view of the landuse classes by considering them as a `SpatialGridDataFrame` and displaying the classes with `spplot`:

```
> clmfires.lu.grid <- as(clmfires.extra$clmcov200$landuse,
     "SpatialGridDataFrame")
> summary(clmfires.lu.grid)

Object of class SpatialGridDataFrame
Coordinates:
        min    max
[1,] -1.125 398.88
[2,] -1.125 398.88
Is projected: NA
proj4string : [NA]
Grid attributes:
  cellcentre.offset cellsize cells.dim
1            -0.125        2       200
2            -0.125        2       200
Data attributes:
         v
 farm     :19457
 bush     : 4941
 scrub    : 4122
```

```
 conifer   : 3472
 meadow    : 2302
 grassland: 2039
 (Other)   : 3667
```

```
> spplot(clmfires.lu.grid)
```



An interesting question is whether different land uses have different forest fire incidences.

Task 36 : Model the incidence of intentional forest fire as a function of the "landuse" covariate, both by itself and also taking into account the presumed Strauss interaction process. Compare the model fit with the null model. •

In the terminology of Eqn. (8), B (trend) depends on the covariates (not the coördinates as in the trend surface), and C (interaction) is absent. The analyst must define the form of B, here, a linear model of the covariate.

```
> levels(clmfires.extra$clmcov200$landuse)
```

```
 [1] "urban"       "farm"        "meadow"      "denseforest"
 [5] "conifer"     "mixedforest" "grassland"   "bush"
 [9] "scrub"       "artifgreen"
```

```
> (m.lu <- ppm(clmfires.i, ~ landuse - 1,
              covariates=clmfires.extra$clmcov200,
              interaction=NULL))
```

```
Nonstationary Poisson process
```

```
Log intensity:  ~landuse - 1

Fitted trend coefficients:
      landuseurban        landusefarm      landusemeadow
           -4.2664            -3.7120            -4.1721
landusedenseforest     landuseconifer landusemixedforest
           -4.0006            -3.7163            -4.2117
  landusegrassland        landusebush       landusescrub
           -4.0030            -3.9672            -3.6268
 landuseartifgreen
          -15.3026

                   Estimate       S.E.   CI95.lo  CI95.hi Ztest
landuseurban        -4.2664   0.164399   -4.5886  -3.9442   ***
landusefarm         -3.7120   0.032513   -3.7757  -3.6482   ***
landusemeadow       -4.1721   0.137361   -4.4413  -3.9029   ***
landusedenseforest  -4.0006   0.127000   -4.2496  -3.7517   ***
landuseconifer      -3.7163   0.076696   -3.8667  -3.5660   ***
landusemixedforest  -4.2117   0.196116   -4.5961  -3.8273   ***
landusegrassland    -4.0030   0.117041   -4.2324  -3.7736   ***
landusebush         -3.9672   0.078326   -4.1207  -3.8137   ***
landusescrub        -3.6268   0.062500   -3.7493  -3.5043   ***
landuseartifgreen  -15.3026 251.201020 -507.6475 477.0424
                         Zval
landuseurban        -25.951557
landusefarm        -114.168873
landusemeadow       -30.373239
landusedenseforest  -31.501106
landuseconifer      -48.455254
landusemixedforest  -21.475642
landusegrassland    -34.202021
landusebush         -50.649459
landusescrub        -58.028362
landuseartifgreen    -0.060918

> data.frame(model=c("Poisson", "Landuse"),
             likelihood=c(logLik(m.pois),
                 logLik(m.lu, warn=F)))

    model likelihood
1 Poisson    -8562.0
2 Landuse    -8533.1
```

Q27 : *Does the landuse explain some of the intentional fire pattern? Which land use classes are more prone to fire?* *Jump to A27* •

Task 37 : Model the incidence of intentional forest fire as a function of the "landuse" covariate, also taking into account the presumed Strauss interaction process. Compare the model fit with the null, landuse-only and interaction-only models. •

In the terminology of Eqn. (8), B (trend) depends on the covariates (not

the coördinates as in the trend surface), and C (interaction) is present. The analyst must define the forms of both B (here, a linear model of the covariate) and C (here the Strauss process).

```
> (m.lu.strauss.4 <- ppm(clmfires.i, ~ landuse,
                         covariates=clmfires.extra$clmcov200,
                         interaction=Strauss(4)))

Nonstationary Strauss process

Log trend:  ~landuse

Fitted trend coefficients:
       (Intercept)        landusefarm      landusemeadow
          -4.42807            0.33941            0.29214
landusedenseforest     landuseconifer landusemixedforest
           0.46879            0.30725            0.21889
  landusegrassland        landusebush       landusescrub
           0.34572            0.42709            0.71876
 landuseartifgreen
         -10.87451

Interaction distance:        4
Fitted interaction parameter gamma:         1.0789

Relevant coefficients:
Interaction
   0.075944

For standard errors, type coef(summary(x))

*** Model is not valid ***
*** Interaction parameters are outside valid range ***

> data.frame(
     model=c("Poisson", "Landuse", "Strauss", "Landuse + Strauss"),
     likelihood=c(logLik(m.pois),
         logLik(m.lu, warn=F),
         logLik(m.strauss.4, warn=F),
         logLik(m.lu.strauss.4, warn=F)))

              model likelihood
1           Poisson    -8562.0
2           Landuse    -8533.1
3           Strauss    -6776.1
4 Landuse + Strauss    -6757.8
```

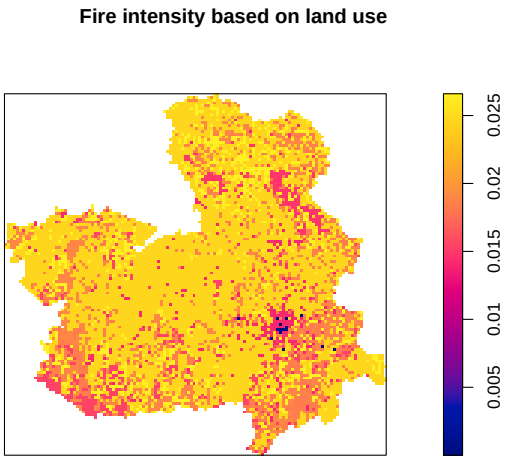
Q28 : *Which model is most likely, given the observations? What do you conclude about the origin of intentionally-set fires?* *Jump to A28* •

8 Prediction

The models fit with `ppm` can be used to predict the point-pattern intensity over a study area. Obviously, they can not predict individual events (e.g., new forest firest) but they can predict the conditional intensity $\lambda(u, \mathbf{x})$ of an occurrence at each location over a grid. The `predict.ppm` function (called just as `predict` on an object of class `ppm`, i.e., a "point pattern model") evaluates the intensity at each grid location.

Task 38 : Predict the conditional intensity of intentional fires using the landuse as predictor and plot it. •

```
> pred.lu <- predict(m.lu, covariates = clmfires.extra$clmcov200)
> summary(pred.lu)

real-valued pixel image
128 x 128 pixel array (ny, nx)
enclosing rectangle: [4.1311, 391.38] x [18.565, 385.19] kilometres
dimensions of each pixel: 3.03 x 2.8642 kilometres
Image is defined on a subset of the rectangular grid
Subset area = 79462.0730449286 square kilometres
Subset area fraction = 0.56
Pixel values (inside window):
        range = [2.2603e-07, 0.026602]
        integral = 1787
        mean = 0.022489

> image(pred.lu, main = "Fire intensity based on land use")
```



Notice the default grid 128 x 128 pixels, and the automatic calculation of the size of each grid cell. This can be changed with the optional `ngrid` argument. Intensity at any set of locations can be requested with the optional

`locations` argument. See `?predict.ppm` for details.

Q29 : *What is the pattern of conditional intensities? How is this derived from the model?* *Jump to A29* •

Another model was the trend surface; we already saw that prediction in the previous section.

The best model of the previous section was landuse + Strauss (interaction process).

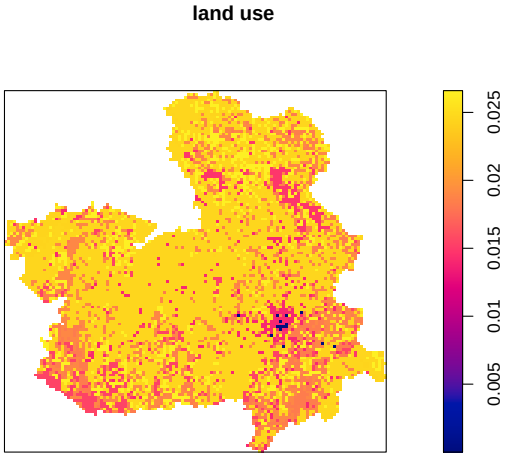
Task 39 : Predict the conditional intensity of intentional fires using the landuse + Strauss process as predictors. Compare the range and mean with the land use-only model. •

```
> pred.lu.strauss <- predict(m.lu.strauss.4,
                             covariates=clmfires.extra$clmcov200)
> summary(pred.lu.strauss)

real-valued pixel image
128 x 128 pixel array (ny, nx)
enclosing rectangle: [4.1311, 391.38] x [18.565, 385.19] kilometres
dimensions of each pixel: 3.03 x 2.8642 kilometres
Image is defined on a subset of the rectangular grid
Subset area = 79462.0730449286 square kilometres
Subset area fraction = 0.56
Pixel values (inside window):
        range = [2.2603e-07, 0.024494]
        integral = 1402.8
        mean = 0.017653

> range(pred.lu.strauss)

[1] 2.2603e-07 2.4494e-02

> range(pred.lu)

[1] 2.2603e-07 2.6602e-02

> mean(pred.lu.strauss)

[1] 0.017653

> mean(pred.lu)

[1] 0.022489
```

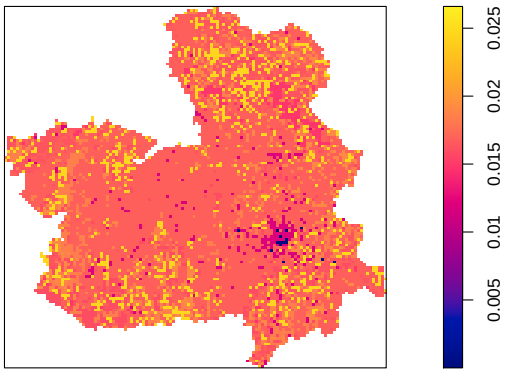
Q30 : *Which model has the higher predicted mean intensity and wider range?* *Jump to A30* •

Task 40 : Plot the two predicted maps with the same stretch. •

```
> zlim=c(min(pred.lu, pred.lu.strauss),
        max(pred.lu, pred.lu.strauss))
> image(pred.lu, zlim=zlim,
       main="land use")
```




```
> image(pred.lu.strauss, zlim=zlim,
       main="land use + Strauss process")
> par(mfrow=c(1,2))
```



9 * Spatio-temporal analysis

Point patterns can evolve over time. In this short section we introduce one way to analyze these: by comparing **time slices** of a point pattern where each point is associated with a **time stamp**, i.e., time of observation.

The objective is to analyze point-patterns which may change over time, for example:

- locations of live trees in a forest plot (some die, some new ones grow);
- locations of crime or disease incidences; these occur at known times.

There is a rich literature on spatio-temporal point process models, see Diggle [6] and Taylor et al. [14]. Here we only show some visualizations and simple analysis, without any attempt to build models.

We ask several questions about the point pattern:

1. Does the structure of the point-pattern change over time?
   - Evaluate with intensity, kernel density, G, F, K, L functions.
2. Does the point-pattern at one time affect the pattern at a later time?
   - Evaluate with the crossed K function.

And of course the aim is to interpret the answers in terms of the **process** that produced the spatio-temporal point pattern.

We use an example of occurrences of foot-and-mouth disease of cattle from North Cumbria (England), `fmd`, in the `stpp` "Spatio-temporal Point Patterns" package.

Task 41 : Load the foot-and-mouth disease temporal point-pattern dataset, and the study area boundary `northcumbria`. Summarize the dataset. •

```
> library("stpp")
> data("fmd")
> data("northcumbria")
> summary(fmd)
```

```
       X                Y           ReportedDay
 Min.   :295580   Min.   :494470   Min.   : 28.0
 1st Qu.:327742   1st Qu.:534362   1st Qu.: 51.0
 Median :340625   Median :544235   Median : 60.5
 Mean   :340190   Mean   :542980   Mean   : 71.8
 3rd Qu.:352670   3rd Qu.:553052   3rd Qu.: 76.0
 Max.   :384530   Max.   :575320   Max.   :198.0
```

```
> dim(fmd)
```

```
[1] 648   3
```

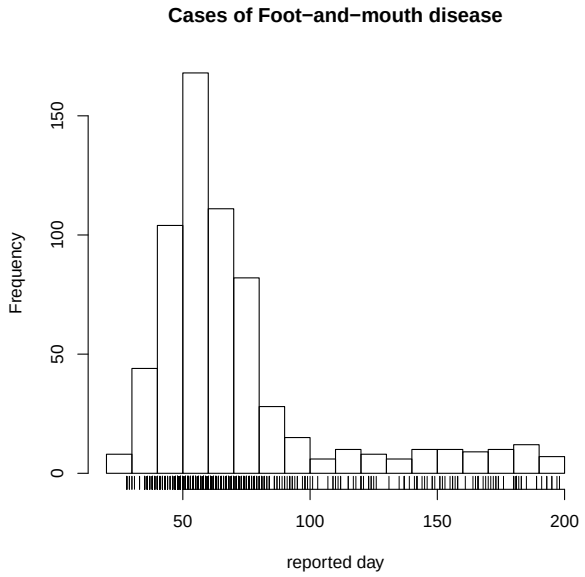
Task 42 : Examine the dataset description. •

```
> help(fmd)
```

Q31 : *What are the three fields? How many cases of foot-and-mouth disease were reported?* *Jump to A31* •

Task 43 : Display a histogram of the occurrences over time. •

```
> hist(fmd[,"ReportedDay"], xlab="reported day",
      main="Cases of Foot-and-mouth disease", breaks=16)
> rug(fmd[,"ReportedDay"])
```



Q32 : *Describe the temporal pattern of the epidemic.* *Jump to A32* •

Task 44 : Convert the point-pattern to an object of R class `stpp` "spatio-temporal point pattern". •

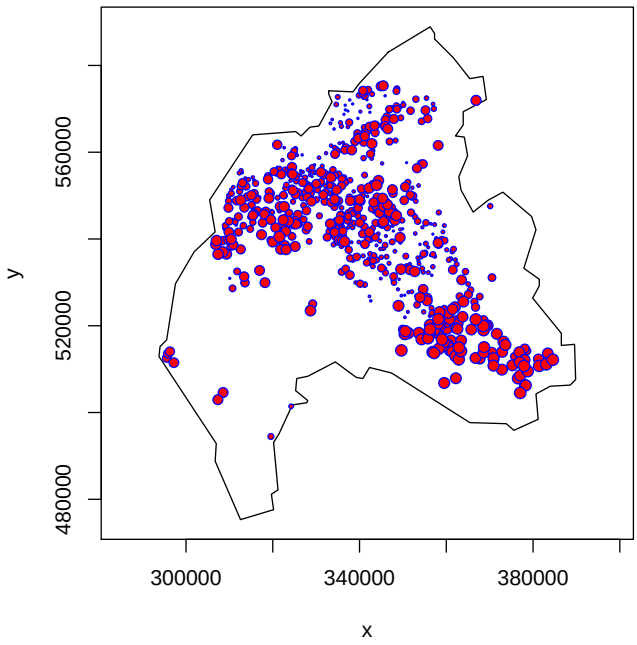
The function `as.3dpoints` performs this conversion:

```
> class(fmd)

[1] "matrix"

> fmd <- as.3dpoints(fmd)
> class(fmd)

[1] "stpp"
```

Task 45 : Plot the occurrence locations, with an indication of the data of occurrence. •

We show the occurrence by the size of the symbol, stretched from `mark.cexmin` to `mark.cexmin`:

```
> plot(fmd, s.region = northcumbria, pch = 21, mark = TRUE,
     mark.col = 0, mark.cexmin = 0.2, mark.cexmax = 1.2,
     col = "blue", bg = "red")
```



For reference, here is the study area from Google Maps. Northern Cumbria county does not include Windermere and further south.

Q33 : *Describe the overall spatial point pattern, not considering time of occurrence.* *Jump to A33* •

Q34 : *Describe the evolution of the spatial point pattern over time.* *Jump to A34* •

Now for some analysis. We will compare the G and F functions for **time-slices** of the point-pattern, and examine their interaction with the crossed K function. The time slices discretize the continuous evolution of the epidemic. In practice these would be set by the epidemiologist according to the presumed process; for convenience we chose 50-day time slices.

Note: You can experiment with different time slices.

Task 46 : Slice the data set into 50-day intervals; report the number of cases in each slice. •

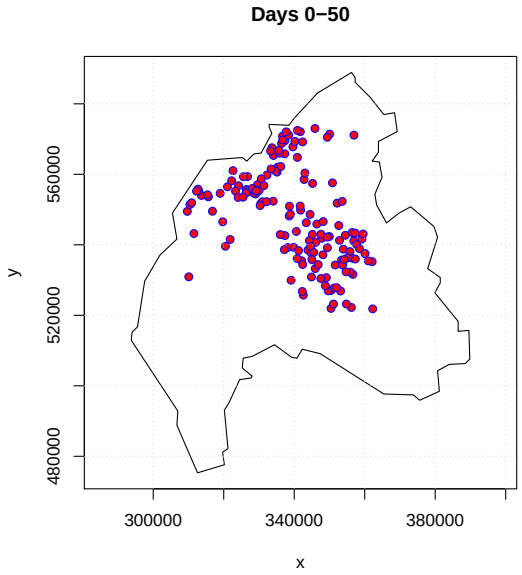
Slicing is with the `[]` selection operator and various logical operators, including `<` and `<=`, to form logical conditions.

```
> dim(fmd)[1]

[1] 648

> fmd.1 <- as.3dpoints(fmd[fmd[, 3] <= 50, ])
> fmd.2 <- as.3dpoints(fmd[(fmd[, 3] > 50) & (fmd[, 3] <=
      100), ])
> fmd.3 <- as.3dpoints(fmd[(fmd[, 3] > 100) & (fmd[, 3] <=
      150), ])
> fmd.4 <- as.3dpoints(fmd[fmd[, 3] > 150, ])
> dim(fmd.1)[1]

[1] 156

> dim(fmd.2)[1]

[1] 404

> dim(fmd.3)[1]

[1] 40

> dim(fmd.4)[1]

[1] 48
```

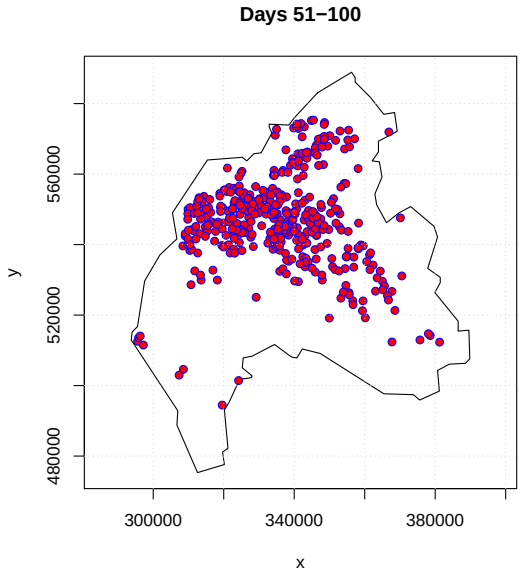
Q35 : *How many cases are in each slice?* *Jump to A35* •

Task 47 : Plot each slice's point pattern. •

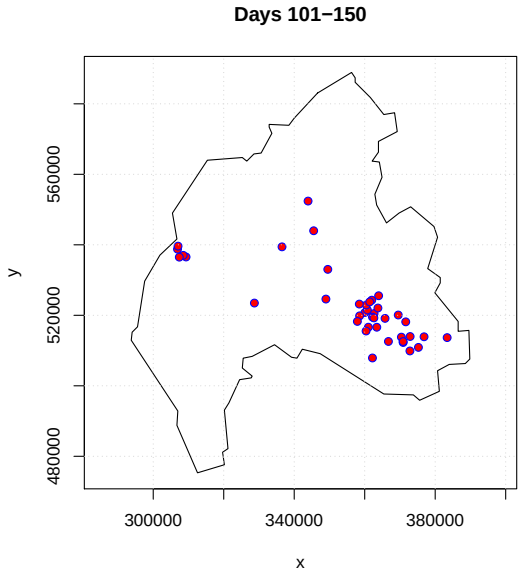
```
> plot(fmd.1, s.region = northcumbria, pch = 21, col = "blue",
      bg = "red", mark = T, mark.col = 0, mark.cexmin = 1,
      mark.cexmax = 1)
> title("Days 0-50")
> grid()
```



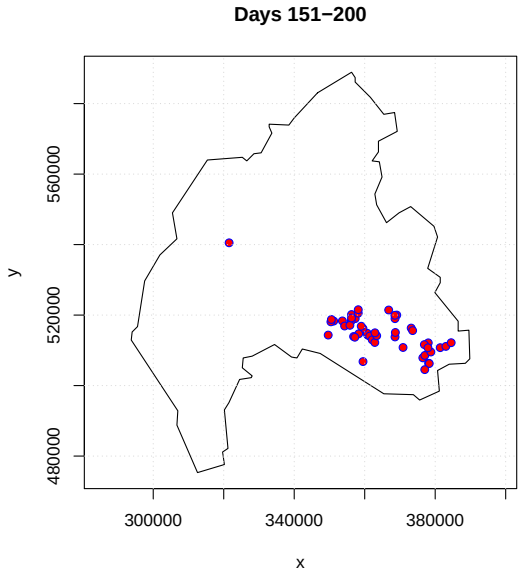

```
> plot(fmd.2, s.region = northcumbria, pch = 21, col = "blue",
     bg = "red", mark = T, mark.col = 0, mark.cexmin = 1,
     mark.cexmax = 1)
> title("Days 51-100")
> grid()
```




```
> plot(fmd.3, s.region = northcumbria, pch = 21, col = "blue",
     bg = "red", mark = T, mark.col = 0, mark.cexmin = 1,
     mark.cexmax = 1)
> title("Days 101-150")
> grid()
```




```
> plot(fmd.4, s.region = northcumbria, pch = 21, col = "blue",
     bg = "red", mark = T, mark.col = 0, mark.cexmin = 1,
     mark.cexmax = 1)
> title("Days 151-200")
> grid()
```



We see an obvious difference in location and clustering.

Task 48 : Compute an `owin` "point-pattern window" object, in order to compute intensity, G, F and K functions. •

```
> w <- owin(poly = list(x = northcumbria[, 1], y = northcumbria[,
     2]))
```

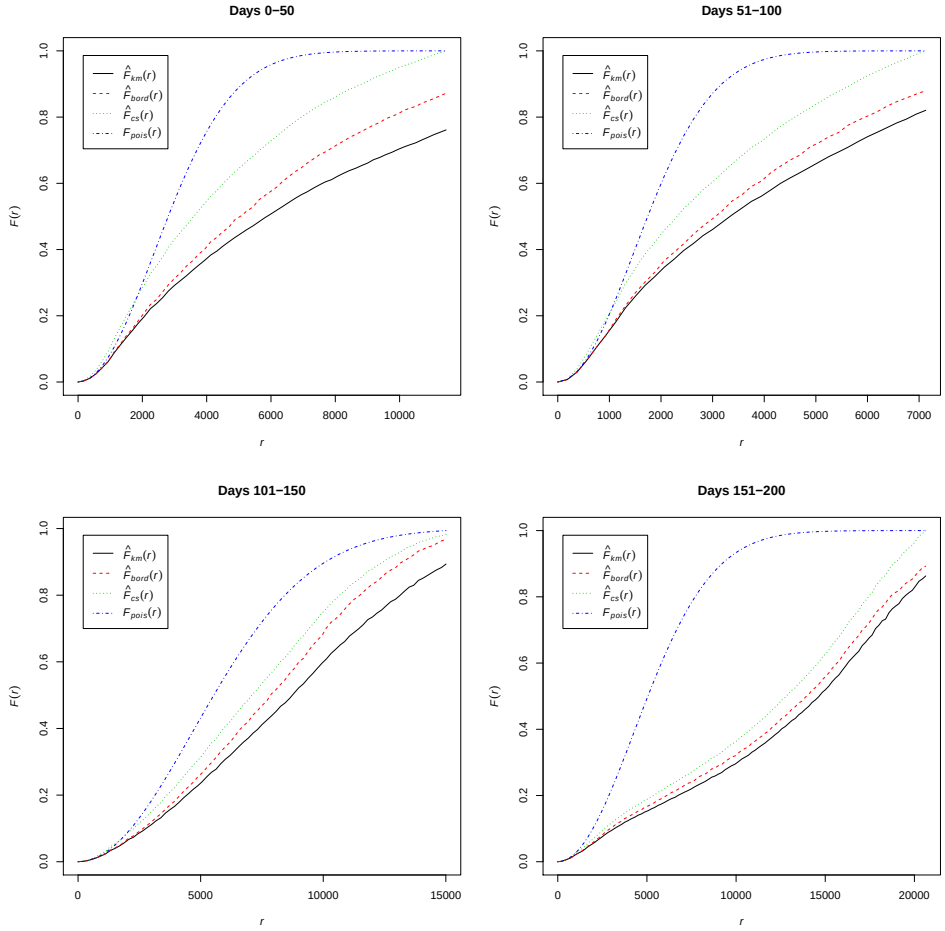
Task 49 : Compute the point-pattern intensity within the window, expressed as cases per km^2. At the same time, make a class `ppp` object from the point-pattern. •

```
> 1/(intensity(fmd.1.ppp <- ppp(fmd.1[, 1], fmd.1[, 2],
     window = w)) * 10^6)

[1] 35.617

> 1/(intensity(fmd.2.ppp <- ppp(fmd.2[, 1], fmd.2[, 2],
     window = w)) * 10^6)

[1] 13.753

> 1/(intensity(fmd.3.ppp <- ppp(fmd.3[, 1], fmd.3[, 2],
     window = w)) * 10^6)

[1] 138.91

> 1/(intensity(fmd.4.ppp <- ppp(fmd.4[, 1], fmd.4[, 2],
     window = w)) * 10^6)

[1] 115.76
```

As shown by the histogram, there is a big difference in intensity between the time slices.

Task 50 : Compare the F "empty space" functions for the four time slices. •

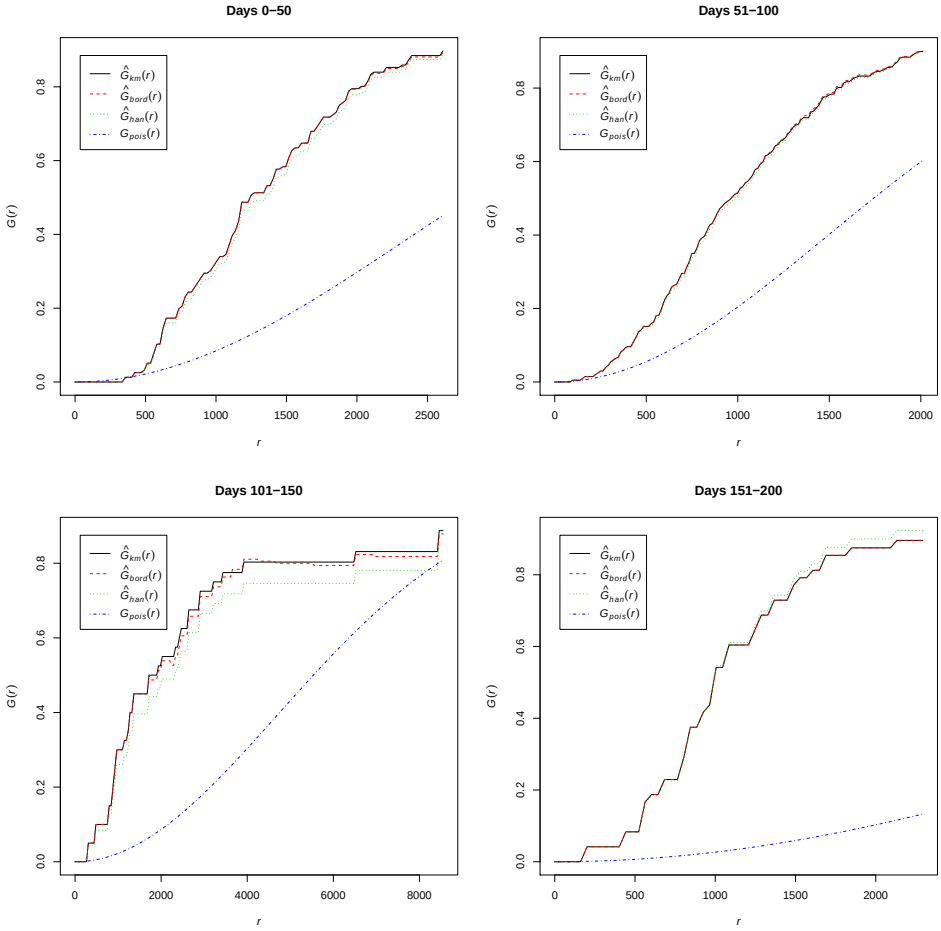
```
> par(mfrow = c(2, 2))
> plot(Fest(fmd.1.ppp), main = "Days 0-50")
> plot(Fest(fmd.2.ppp), main = "Days 51-100")
> plot(Fest(fmd.3.ppp), main = "Days 101-150")
> plot(Fest(fmd.4.ppp), main = "Days 151-200")
> par(mfrow = c(1, 1))
```



Q36 : *Describe the evolution of the F function over time.* *Jump to A36* •

Task 51 : Compare the G "closest point" functions for the four time slices. •

```
> par(mfrow = c(2, 2))
> plot(Gest(fmd.1.ppp), main = "Days 0-50")
> plot(Gest(fmd.2.ppp), main = "Days 51-100")
> plot(Gest(fmd.3.ppp), main = "Days 101-150")
> plot(Gest(fmd.4.ppp), main = "Days 151-200")
> par(mfrow = c(1, 1))
```



Q37 : *Describe the evolution of the G function over time.* *Jump to A37* •

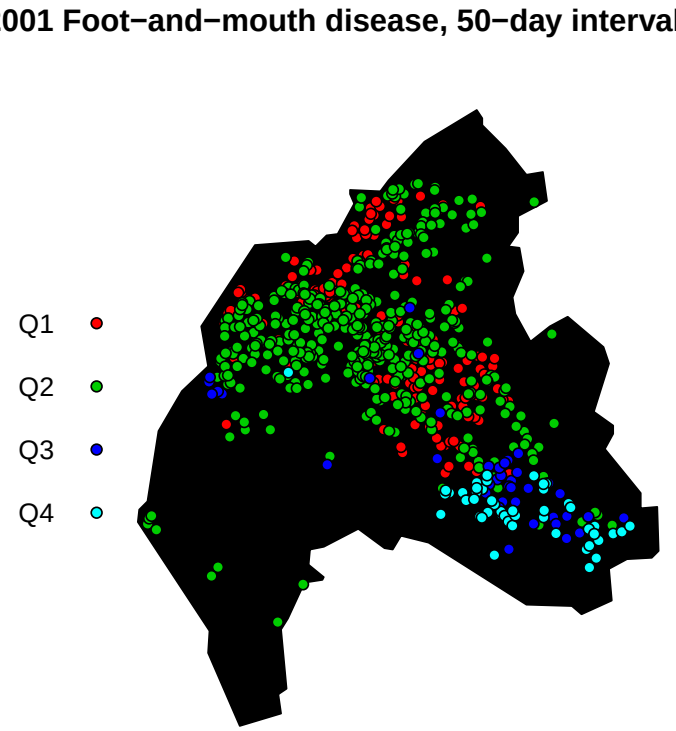
Now we want to evaluate the relation between time slices with the crossed K function. This will reveal if there is any interaction (attraction, dispersion, independence) between patterns. This can then be interpreted by the epidemiologist.

To compute the crossed K function the pattern must be marked.

Task 52 : Combine the four time slices into one marked point pattern. Plot the marked point pattern. •

We do this with the `superimpose` function, and name the four slices.

```
> fmd.all.ppp <- superimpose(Q1 = fmd.1.ppp, Q2 = fmd.2.ppp,
      Q3 = fmd.3.ppp, Q4 = fmd.4.ppp)
> plot(fmd.all.ppp, main = "2001 Foot-and-mouth disease, 50-day intervals",
      cex = 0.9, pch = 21, col = 1, bg = 2:5)
```

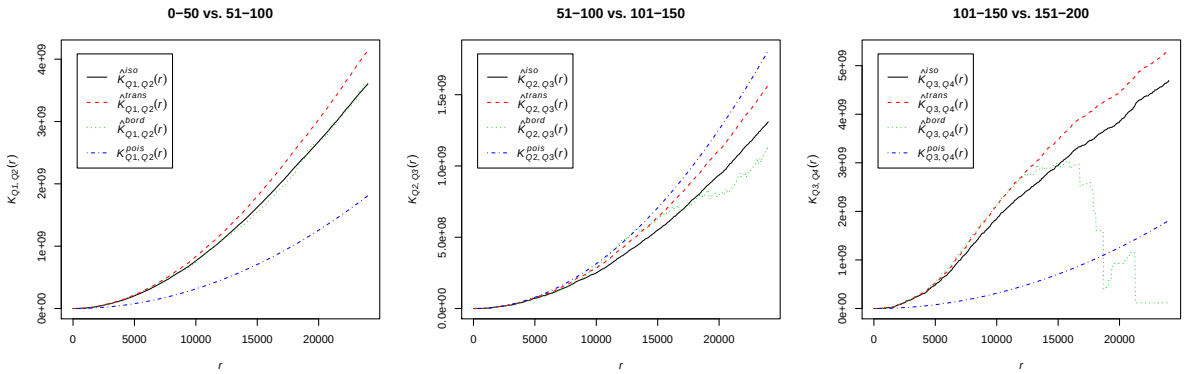


Task 53 : Compute and display the crossed K function for each time step. •

There are three of these.

```
> Kcross.1.2 <- Kcross(fmd.all.ppp, "Q1", "Q2")
> Kcross.2.3 <- Kcross(fmd.all.ppp, "Q2", "Q3")
> Kcross.3.4 <- Kcross(fmd.all.ppp, "Q3", "Q4")
```

```
> par(mfrow = c(1, 3))
> plot(Kcross.1.2, main = "0-50 vs. 51-100")
> plot(Kcross.2.3, main = "51-100 vs. 101-150")
> plot(Kcross.3.4, main = "101-150 vs. 151-200")
> par(mfrow = c(1, 1))
```



Q38 : *What is the relation between the point-patterns in successive time slices?* *Jump to A38* •

10 Further reading

Point-pattern analysis is based on theories of point processes. A modern review article is by Møller and Waagepetersen [11]. The text of Diggle [5] presents a detailed explanation and many worked examples of the concepts presented here, and many more. Bivand et al. [3, Ch. 7] presents worked examples of some of these questions; in particular §7.5 presents some applications in spatial epidemiology. Illian et al. [8] present a computational framework for fitting complex spatial point process models using a recently-developed methodology known as INLA. Spatio-temporal point pattern modelling is covered by Diggle [6] and Taylor et al. [14].

11 Answers

A1 : *65 trees.* *Return to Q1* •

A2 : *A square of 5.7 m x 5.7 m; the units have been normalized to* $[0 \ldots 1]$. *Return to Q2* •

A3 : *The pattern looks completely random; we will investigate this statistically later.* *Return to Q3* •

A4 : *They have been re-scaled from* $[0 \ldots 1]$ *to the original coordinates, here* $[0 \ldots 5.7]$ *m. Note that although the* `ppp` *object stated that the units are metres, that information has not been carried through to the* `sp` *object: it has an undefined projection and hence units.* *Return to Q4* •

A5 : *Yes, they are quite different. The Japanese pines appear to be completely randomly distributed; the Redwoods clustered, the cells more or less regular (anyway, dispersed).* *Return to Q5* •

A6 :

(a) The distance at which at least 95% of the points have a neighbour is 0.117 units.

(b) the proportion of points with a neighbour within 0.05 units is 0.388. *Return to Q6* •

A7 : *They all match well, with very little difference between them. Only at the furthest distance is the empirical function somewhat lower (fewer neighbours than expected) than expected by CSR.* *Return to Q7* •

A8 : *They match fairly well, again the discrepancy around* $G(r) = 0.8$ *where there are fewer points with first nearest neighbour in that range than expected.* *Return to Q8* •

A9 : *For the Redwoods dataset the empirical* G *function is well above (greater than) the theoretical after a radius of about 0.01 to about 0.05. This indicates strong clustering: nearest neighbours are found at closer distances than expected by CSR. An interesting feature is that there are no neighbours until 0.01 – a very short-range repulsion probably due to the size of an individual tree, making it impossible for two trees to be closer than the size of one tree.* *Return to Q9* •

A10 : *For the Cells dataset the empirical* G *function is well below (less than) the theoretical throughout the range. This indicates strong dispersion: nearest neighbours are found at further distances than expected by CSR. There are no neighbours until about 0.08, after which the function rises very steeply. This is approaching a pure regular grid, in which the empirical* G *function is a backwards "L" shape.* *Return to Q10* •

A11 : *Low intensities lead to more sampling error, so high intensities would have narrower envelopes.* *Return to Q11* •

A12 : *With this simulation the empirical* G *function is almost completely within the envelope throughout, so we can not reject the null hypothesis of CSR. The anomaly near* $r = 0.1$ *is clear, indeed at one point of this simulation the empirical*

value is below the lower limit of the 96% confidence envelope. *Return to Q12* •

A13 : *The empirical G function for both are well outside the simulation envelope for much of the radius range. The Redwood trees are almost surely clustered and the cells almost surely dispersed.* *Return to Q13* •

A14 : *As the bandwidth increases, the maximum intensity decreases: the "hot spots" are not as "hot" (dense). The minimum intensity in all cases is zero, meaning there are some regions with effectively no probability of a point occurrence. The mean intensity is almost the same; theoretically it should be the same but there are variable edge effects depending on bandwidth. The quartiles show a clear trend: first quartile and median increasing with bandwidth, third quartile decreasing. The intensities are concentrated below the mean at wider bandwidths. This is especially clear with the maximum intensity, which decreases as the counts are averaged across increasingly larger areas.* *Return to Q14* •

A15 : *As the bandwidth increases, the density becomes more uniform . By twice the optimum the density is almost homogeneous, at 80% of the optimum the density there are spurious patches. The optimum seems to give a good representation.* *Return to Q15* •

A16 : *The K-function for the Japanese pines is quite close to the theoretical for CSR, although slightly below (dispersed) for radiuses around* 0.15. *The K function for the Redwood trees is consistently above (clustered) at all radii but especially near zero-separation, except for the very close range. The K function for the cells is well below the theoretical for separations to about* 0.15*; after this is conforms to CSR, meaning that after the initial dispersion to that radius, the number of points within the radius is as expected by CSR. This shows that the dispersion is not on a regular grid.* *Return to Q16* •

A17 : *The convex hull has expanded slightly outward, consistent with average inter-point spacing. The point at the extreme SE controls the SSE and E boundaries, adding a large amount of unsampled area to the polygon.* *Return to Q17* •

A18 : *The removal of extraneous "white space" increases the intensity by 185 %.* *Return to Q18* •

A19 : *For the rectangular window about* 0.4*; for the polygonal* $\approx$ 0.55. *This is because of the smaller area of the rectangle with the same number of points, i.e., higher average intensity.* *Return to Q19* •

A20 : *In the rectangular window the observed proportion matches the theoretical under CSR up to about 80 m, after which the observed is much lower than the theoretical, i.e., much of the area is further from a point than expected under CSR. This is because of the large areas without any points in the rectangle. In the*

polygonal window the theoretical (under CSR) and actual match well till about 120 m, after which the observed is slightly lower than the theoretical, by about 0.1. Thus there is less area far from the nearest point; this indicates some larger areas of empty space compared to CSR; in this case these are the areas in the SSE and E controlled by the south-easternmost point and not part of the study area. *Return to Q20* •

A21 : *The marks are the cause of each forest fire; there are four classes (causes): lightning, accidental, intentional, and other.* *Return to Q21* •

A22 : *Yes; for example, the fires caused by lightning are heavily clustered in the east, with a large space in the centre with almost no fires; by contrast, the intentional fires are more prevalent in the NW.* *Return to Q22* •

A23 : *Up to about 15 km distances the expected and observed numbers of additional fires from the second process are almost the same; however, beyond that, they are consistently fewer than would be expected by chance, indicating dispersion of one process "caused by" the other. In this case the apparent dispersion may be an artefact of non-stationary intensities of both processes.* *Return to Q23* •

A24 : *The actual cross-K function is very close to the theoretical cross-K function from two unrelated processes. This makes sense because these are two independent patterns that we superimposed just to show that operation.* *Return to Q24* •

A25 : *The trees appear to be clustered; there are some areas with no trees. In some sections trees of similar size cluster together but there are also very small trees near very large (see centre E).* *Return to Q25* •

A26 : *Yes, it appears that there is trend from higher intensity in the NNW to lower in the SSE.* *Return to Q26* •

A27 : *The landuse is a somewhat more likely explanation for the observed pattern of fires than the null model. Scrubland, coniferous forests, and farmland have higher intensities of the Poisson process. Urban land has less.* *Return to Q27* •

A28 : *The model with both land use and Strauss process (interaction) is the most likely; the model with just land use is quite poor. The combined model is a bit better than the interaction-only model. There is definitely interaction between points, i.e., clustering within the 4 km radius. Fires are more likely on scrubland, conifer forest, and dense forest,* *Return to Q28* •

A29 : *The intensities follow the land use classes (compare with the figure of §7.4). The lowest intensities are in the "urban" and "artificial green" areas (the "cold spot"*

in the lower-right), the highest in mixed forests. *Return to Q29* •

A30 : *The landuse-only model has both a higher mean and wider range. It is not adjusted to account for inter-point interaction, which reduces the intensity. The Strauss interaction coefficient was* $0.76 > 0$*, indicating local clustering, accounting for some of the intensity within the most susceptible land uses.* *Return to Q30* •

A31 : *The three fields are x, y, and x. These are the east and north coördinates in an unspecified CRS, and the reported time of occurrence of a case of foot-and-mouth disease, in days from an unspecified 0 (maybe day of year 2001).* *Return to Q31* •

A32 : *There was a gradual start to the epidemic, then a very strong peak, and a long tail with a few additional cases.* *Return to Q32* •

A33 : *The cases form clear clusters, with a few scattered cases outside of these (e.g., in the SW and NE). The pattern within the clusters seems random. Note that the blank areas with no cases are probably because no cattle is raised there – the large central area is the Lake District national park, and the eastern edge are the North Pennine mountains.* *Return to Q33* •

A34 : *The earliest cases (smallest symbols) are in the centre and NW, then there are cases more towards the NW and W, and finally the most recent cases (largest symbols) are concentrated in the SE.* *Return to Q34* •

A35 : *0-50 days: 156 cases; 51-100 days: 404 cases; 101-150 days: 40 cases; 151-200 days: 48 cases.* *Return to Q35* •

A36 :

These all show a longer distance from an arbitrary location in the study area to the nearest case than would be expected by chance. The pattern changes: increasingly strong in the last time-slice, since most of the cases are found only in the SE of the study area. *Return to Q36* •

A37 : *These all show strong clustering: the observed distance to nearest neighbour is well above the theoretical line. Notice the different distance* (r) *scales. The clustering is strongest for the 151-200 day slice: almost all points have a neighbour within 2.2 km, whereas for the 101-150 time slice this is not reached until about 8 km.* *Return to Q37* •

A38 : *There is a clear repulsion influence of the first slice (0-50 days) on the second (51-100), and the third (101-150) on the fourth (151-200). That is, the nearest point in one pattern is further than expected by chance from the point in*

the other pattern. The second and third time-slice patterns are almost independent.
Return to Q38 •

12 Assignment

This is a small test of how well you mastered this exercise. You should be able to complete the tasks and answer the questions with the knowledge you have gained from the exercise.

We will use the "Lansing woods" dataset provided as an example with the `spatstat` package.

Task 1 : Load the example dataset `lansing` which is provided with the `spatstat` package and view its description (via the help system). •

Q1 : *Where was the data collected? How large was the study area? What is recorded, besides the location? What is its R class?* •

Task 2 : Summarize the dataset. •

Q2 : *How many black oaks were observed in the study area? What is the intensity of the point-process for black oaks, in trees per unit area? Explain why these are the same number.* •

Task 3 : Display this **marked** point pattern (i.e., the occurrences are marked by the species). •

> **Note:** You might try improving the plot with some combination of `cex` (zoom factor for plotting symbols), `chars` (plotting character), `cols` (plotting colours).

It is very difficult to pick out the patterns for different species in this combined graph. We can use the `split.ppp` function to divide the pattern by the marks.

Task 4 : Split the Lansing woods data into an object with separate point-pattern for each species. Display them in one graph with the default `plot` method. •

Q3 : *Which of the species appears to be most clustered? most dispersed? most random?* •

Task 5 : Compute and graph the G-function for the hickory trees. •

> **Note:** You can access one of the unmarked point-patterns in the split object with the `$` syntax, e.g., if the split object is named `lansing.split`, access the hickory point-pattern with `lansing.split$hickory`.

Q4 : *What is the interpretation of this G-function, with respect to the null hypothesis of complete spatial randomess?* •

Task 6 : Compute and graph a 99-simulation, 96% envelope for the hickory, to a radius indicated by the G-function graph of the previous task. •

Q5 : *Is the empirical G-function within the envelope throughout? Can we reject the null hypothesis of CSR?* •

Now we investigate if the process is homogeneous or not.

Task 7 : Compute the optimal bandwidth, and plot the kernel density of the hickory trees with the selected bandwidth. •

> **Note:** Recall that the `mse2d` function expects a set of points, so you will have to (1) convert from `ppp` to `sp`; (2) rescale to the unit square with the `elide` method of the `maptools` package, (3) extract the `sp` object's coordinates with the `coordinates` method.

Q6 : *Does this plot provide evidence for an inhomogeneous point process? Why or why not?* •

A Preparing data for point pattern analysis

As shown in §5, objects of class `ppp` can be converted from objects of class `SpatialPoints` (point locations only) or `SpatialPointsDataFrame` (point locations with attributes which can be used as marks) with the generic conversion method `as`, specialized to convert from `ppp` to `sp` (and vice-versa) by methods loaded with the `spatstats` package. The conversion to `ppp` is done with the `as.ppp` function.

So, all that is required is to import point data (possibly with attributes) into `SpatialPoints` or `SpatialPointsDataFrame` objects. There are two common methods:

1. Directly from shapefiles of points, using the `readOGR` function of the `rgdal` package (§A.1);
2. From data frames imported with the `read.table` function or its variants such as `read.csv` for comma-separated values (CSV) files (§A.2).

For records stored in Excel spreadsheets, see the R data import/export FAQ[8]. The easiest way to import Excel spreadsheets is to first export the sheet from Excel as a CSV file, and follow option (2) below (§A.2).

A.1 Shapefiles

A shapefile is a specification for geospatial data interchange among ESRI and other information systems, and is one of the native formats used by ArcGIS. It consists of three files with the same name and different file extensions: (1) `shp` for the geometry; (2) `shx` for the spatial index; (3) `dbf` for the attribute table. We illustrate how to read a shapefile for of the sample datasets provided with the `maptools` package, using `readOGR` function to read it.

```
> library(rgdal)
> library(maptools)
> tmp <- readOGR(system.file("shapes/baltim.shp", package = "maptools"))

OGR data source with driver: ESRI Shapefile
Source: "/Library/Frameworks/R.framework/Versions/3.6/Resources/library/maptools/
with 211 features
It has 17 fields

> class(tmp)

[1] "SpatialPointsDataFrame"
attr(,"package")
[1] "sp"

> proj4string(tmp)

[1] NA
```

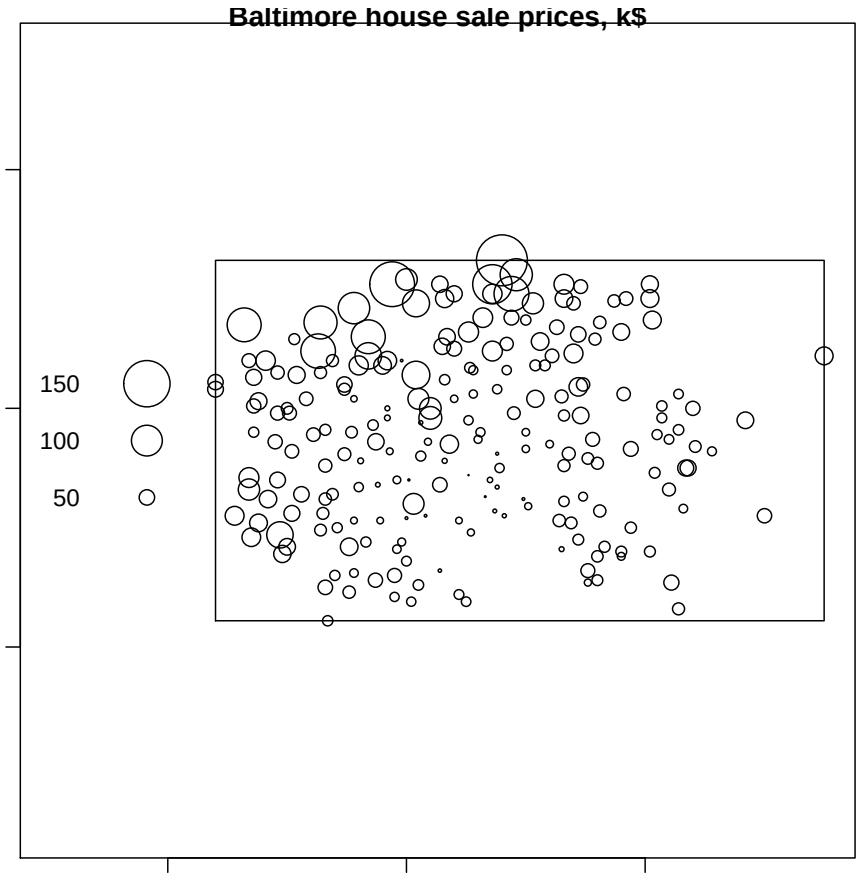
[8] http://cran.r-project.org/doc/manuals/R-data.html#Reading-Excel-spreadsheets

Note: Packages are stored in the directory found with the `.libPaths` function. The `system.file` function expands its argument with this, to give a full path and file name.

This is a `SpatialPointsDataFrame` since it has attributes; these appear to be information on house sales in Baltimore (USA) It does not have a defined coördinate reference system; if the source shapefile has one, it is imported.

This is then easily converted to a `ppp` object with `as`:

```
> require(spatstat)
> tmp <- as(tmp, "ppp")
> op <- par(no.readonly = TRUE)
> par(mar=rep(0.5, 4))
> plot(tmp, which.marks="PRICE", axes=T,
      main="Baltimore house sale prices, k$")
> par(op)
```



The bounding box is set to the exact limits of the point data set.

Note: The `par` function retrieves and sets base graphics parameters. Here we reduce the default margins, which are specified with the `mar` argument to `par`.

A.2 Text files

A text file to be converted to a point pattern typically has one header line giving the variable names, usually with the two coördinates as the first two

columns. The following lines are one record per point, with a number of fields. For a point pattern, at least two fields are needed, i.e., the coördinates. Others (the attributes) are optional. In a CSV file, fields within each record are separated by commas, and text is quoted. But this is only one possible format; the many arguments to `read.table` (see its help) allow almost any text file format to be read into a data frame. We use as an example the Meuse dataset `meuse` of the `sp` package. We write it to a text file using `write.csv`, examine its structure, and show how to import it using `read.csv`.

First the export:

```
> require(sp)
> data(meuse)
> class(meuse)

[1] "data.frame"

> write.csv(meuse, file = "tmp.csv")
```

We examine its structure with `file.show`, but you can also view in any plain-text editor.

```
> file.show("tmp.csv")
```

```
"","x","y","cadmium","copper","lead","zinc","elev","dist","om","ffreq","soil","lime","landuse","dis
"1",181072,333611,11.7,85,299,1022,7.909,0.00135803,13.6,"1","1","1","Ah",50
"2",181025,333558,8.6,81,277,1141,6.983,0.0122243,14,"1","1","1","Ah",30
"3",181165,333537,6.5,68,199,640,7.8,0.103029,13,"1","1","1","Ah",150
"4",181298,333484,2.6,81,116,257,7.655,0.190094,8,"1","2","0","Ga",270
...
"164",180627,330190,2.7,27,124,375,8.261,0.0122243,5.5,"3","3","0","W",40
```

Then the import; although here we just duplicate what we already had in the example data frame, for your own data this would be the point at which you bring your data into R.

Notice that the CSV file has no information on data types; `read.table` guesses but is not always right. Here it can not determine that `ffreq`, `soil` and `lime` are classes, because they are coded as integer labels. So they must be converted explicitly with `as.factor`.

```
> pp <- read.csv(file = "tmp.csv")
> class(pp)

[1] "data.frame"

> pp$ffreq <- as.factor(pp$ffreq)
> pp$soil <- as.factor(pp$soil)
> pp$lime <- as.factor(pp$lime)
> str(pp)

'data.frame':        155 obs. of  15 variables:
 $ X      : int  1 2 3 4 5 6 7 8 9 10 ...
```

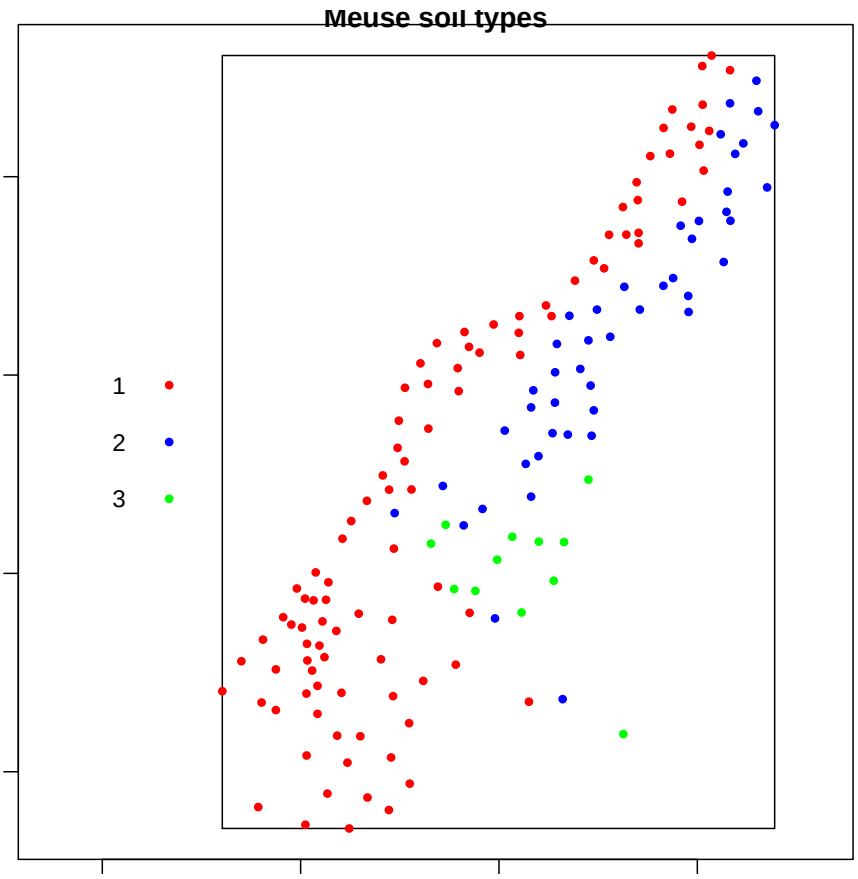
```
 $ x      : int  181072 181025 181165 181298 181307 181390 181165 181027 181060 1
 $ y      : int  333611 333558 333537 333484 333330 333260 333370 333363 333231 3
 $ cadmium: num  11.7 8.6 6.5 2.6 2.8 3 3.2 2.8 2.4 1.6 ...
 $ copper : int  85 81 68 81 48 61 31 29 37 24 ...
 $ lead   : int  299 277 199 116 117 137 132 150 133 80 ...
 $ zinc   : int  1022 1141 640 257 269 281 346 406 347 183 ...
 $ elev   : num  7.91 6.98 7.8 7.66 7.48 ...
 $ dist   : num  0.00136 0.01222 0.10303 0.19009 0.27709 ...
 $ om     : num  13.6 14 13 8 8.7 7.8 9.2 9.5 10.6 6.3 ...
 $ ffreq  : Factor w/ 3 levels "1","2","3": 1 1 1 1 1 1 1 1 1 1 ...
 $ soil   : Factor w/ 3 levels "1","2","3": 1 1 1 2 2 2 2 1 1 2 ...
 $ lime   : Factor w/ 2 levels "0","1": 2 2 2 1 1 1 1 1 1 1 ...
 $ landuse: Factor w/ 15 levels "Aa","Ab","Ag",..: 4 4 4 11 4 11 4 2 2 15 ...
 $ dist.m : int  50 30 150 270 380 470 240 120 240 420 ...
```

Convert to a `SpatialPointsDataFrame` using the `coordinates` method:

```
> coordinates(pp) <- ~x + y
> class(pp)

[1] "SpatialPointsDataFrame"
attr(,"package")
[1] "sp"
```

Finally, this can be converted to a point pattern:

```
> pp <- as(pp, "ppp")
> class(pp)

[1] "ppp"

> window(pp)

Marked planar point pattern: 6 points
Mark variables:
   X cadmium copper lead zinc elev dist om ffreq soil lime landuse
dist.m
window: rectangle = [178605, 181390] x [329714, 333611] units

> op <- par(no.readonly = TRUE)
> par(mar=rep(0.5, 4))
> plot(pp, which.marks="soil", cols=c("red","blue","green"),
      pch=20, main="Meuse soil types", axes=T)
> par(op)
```



The window boundary can be limited as explained in §5. If a bounding polygon is available, it can be imported and used as the window. I have prepared a crude boundary for this area, by polygonizing the interpolation grid `meuse.grid` supplied in the `sp` package, taking the union of the grid cell polygons, and writing the result as a shapefile `meuseBoundary`. This should have been supplied with this exercise, as a compressed folder with the four files which together make up the ESRI shapefile format.

The `readOGR` function of the `rgdal` package can read ESRI shapefiles in to an `sp` object. The `dsn` "data source name" argument to `readOGR` for a shapefile is the folder name in which the shapefile is located; in the code below it is given as `"."`, i.e., the current working directory[9]; you can change this as you wish. The `layer` "layer name" argument is the name of the shapefile, without extension.

```
> library(rgdal)
> meuseBoundary <- readOGR(dsn = ".", layer = "meuseBoundary")
> class(meuseBoundary)

OGR data source with driver: ESRI Shapefile
Source: "/Users/rossiter/data/edu/dgeostats/ex/ds/meuse", layer: "meuseBoundary"
with 1 features
It has 1 fields

[1] "SpatialPolygonsDataFrame"
```

[9] You can see what this is with the `getwd` function.

```
attr(,"package")
[1] "sp"
```

The polygon shapefile import creates a a `SpatialPolygonsDataFrame` object, which can be converted to an `owin` "observation window" object with the `as.owin` function of the `spatstat` package:

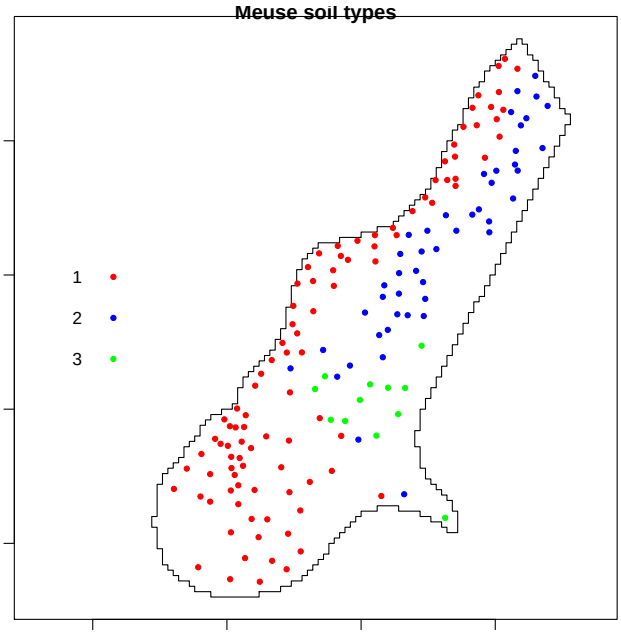
```
> (meuseBoundary.win <- as.owin(meuseBoundary))

window: polygonal boundary
enclosing rectangle: [178440, 181560] x [329600, 333760] units
```

Finally, this window can be substituted for the original rectangular window by direct assignment to the `window` field of the `ppp` object:

```
> pp$window <- meuseBoundary.win
```

When this is plotted we see the study area window:

```
> op <- par(no.readonly = TRUE)
> par(mar=rep(0.5, 4))
> plot(pp, which.marks="soil", cols=c("red","blue","green"),
      pch=20, main="Meuse soil types", axes=T)
> par(op)
```



Challenge: Repeat the analysis of §5 (the F function) with this polygonal window, and compare the results with those from the rectangular bounding box and the window found by the Ripley-Rasson method.

Clean up from this section; this includes removing the temporary file with the `unlink` function[10]:

```
> unlink("tmp.csv")
> rm(meuse, pp, meuseBoundary, meuseBoundary.win, op)
```

[10] This name for file deletion is inherited from the Unix operating system.

References

[1] A. Baddeley and R. Turner. Modelling spatial point patterns in R. In A. Baddeley, P. Gregori, J. Mateu, R. Stoica, D. Stoyan, J. Berger, S. Fienberg, J. Gani, K. Krickeberg, I. Olkin, and B. Singer, editors, *Case Studies in Spatial Point Process Modeling*, volume 185 of *Lecture Notes in Statistics*, pages 23–74. Springer New York, 2006. ISBN 978-0-387-31144-9. 41, 48

[2] M. Berman and P. Diggle. Estimating weighted integrals of the second-order intensity of a spatial point process. *Journal of the Royal Statistical Society. Series B (Methodological)*, 51(1):81–92, 1989. 16

[3] R. S. Bivand, E. J. Pebesma, and V. Gómez-Rubio. *Applied Spatial Data Analysis with R*. UseR! Springer, 2008. http://www.asdar-book.org/. 1, 2, 7, 15, 20, 40, 67

[4] B. N. Boots and A. Getis. *Point pattern analysis*. Number v. 8 in Scientific geography series. Sage Publications, Newbury Park, Calif, 1988. ISBN 0803922450. 1

[5] P. Diggle. *Statistical analysis of spatial point patterns*. Arnold, London, 2nd edition, 2003. ISBN 0122158504. 67

[6] P. J. Diggle. *Statistical analysis of spatial and spatio-temporal point patterns*. CRC Press, Boca Raton, 3rd edition, 2013. ISBN 978-1-4665-6023-9. 57, 67

[7] Peter Diggle. *Statistical Analysis of Spatial and Spatio-Temporal Point Patterns*. CRC Press, Boca Raton, third edition. edition, 2014. ISBN 978-1-4665-6024-6. doi: 10.1201/b15326. 1

[8] Janine B. Illian, Sigrunn H. Sorbye, and Havard Rue. A toolbox for fitting complex spatial point process models using integrated nested laplace approximation (inla). *Annals of Applied Statistics*, 6(4):1499–1530, Dec 2012. doi: 10.1214/11-AOAS530. 1, 67

[9] F Leisch. Sweave, part I: Mixing R and LaTeX. *R News*, 2(3):28–31, December 2002. URL http://CRAN.R-project.org/doc/Rnews/. 2

[10] F Leisch. *Sweave User's Manual*. TU Wein, Vienna (A), 2.7.1 edition, 2006. URL http://www.stat.uni-muenchen.de/~leisch/Sweave/. 2

[11] Jesper Møller and Rasmus P. Waagepetersen. Modern statistics for spatial point processes. *Scandinavian Journal of Statistics*, 34(4):643–684, Dec 2007. doi: 10.1111/j.1467-9469.2007.00569.x. 1, 67

[12] B. D. Ripley. Modelling spatial patterns. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(2):172–212, January 1977. 21

[13] B. D. Ripley and J. P. Rasson. Finding the edge of a Poisson forest. *Journal of Applied Probability*, 14(3):483–491, September 1977. doi: 10.2307/3213451. 28

[14] Benjamin M. Taylor, Tilman M. Davies, Barry S. Rowlingson, and Peter J. Diggle. lgcp: an R package for inference with spatial and

spatio-temporal log-Gaussian Cox processes. *Journal of Statistical Software*, 52(4), 2013. ISSN 1548-7660. doi: 10.18637/jss.v052.i04. URL http://www.jstatsoft.org/v52/i04/. 57, 67

Index of R Concepts