Analysing spatial point patterns in R

Adrian Baddeley
Adrian.Baddeley@csiro.au
adrian@maths.uwa.edu.au

Workshop Notes
February 2008

Abstract

This is a detailed set of notes for a workshop on *Analysing spatial point patterns* that has been held several times in Australia and New Zealand in 2006–2008.

It covers statistical methods that are currently feasible in practice and available in public domain software. Some of these techniques are well established in the applications literature, while some are very recent developments.

The workshop uses the statistical package R and is based on `spatstat`, an add-on library for R for the analysis of spatial data.

Topics covered include: statistical formulation and methodological issues; data input and handling; R concepts such as classes and methods; nonparametric intensity estimates; goodness-of-fit testing for Complete Spatial Randomness; maximum likelihood inference for Poisson processes; model validation for Poisson processes; distance methods and summary functions such as Ripley's K function; non-Poisson point process models; simulation techniques; fitting models using summary statistics; Gibbs point process models; fitting Gibbs models; simulating Gibbs models; validating Gibbs models; multitype and marked point patterns; exploratory analysis of marked point patterns; multitype Poisson process models and maximum likelihood inference; multitype Gibbs process models and maximum pseudolikelihood; and line segment data.

This workshop requires R **version 2.6.0** or later, and `spatstat` **version 1.12-6** or later.

Acknowledgements

The author gratefully acknowledges countless comments and suggestions from workshop participants, and the support of CSIRO Mathematical and Information Sciences, The New Zealand Statistical Association, The University of Waikato, The Statistical Society of Australia and The University of Western Australia.

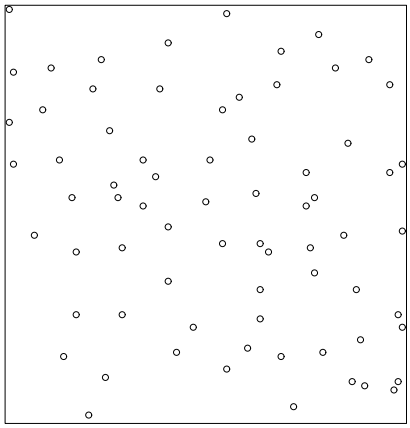
Contents

1 Introduction

1.1 Types of data

1.1.1 Points

A **point pattern** dataset gives the locations of objects/events occurring in a study region.

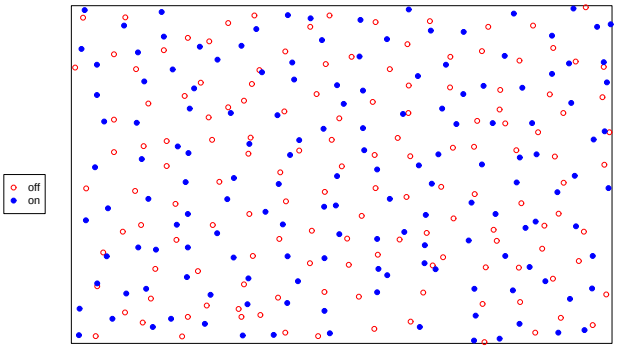


The points could represent trees, animal nests, earthquake epicentres, petty crimes, domiciles of new cases of influenza, galaxies, etc.

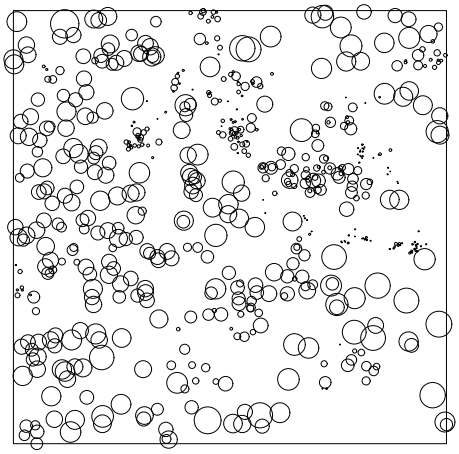
The points might be situated in a region of the two-dimensional (2D) plane, or on the Earth's surface, or a 3D volume, etc. They could be points in space-time (e.g. earthquake epicentre location and time). The software presented here is only applicable to 2D point patterns (but we're working on it).

1.1.2 Marks

The points may have extra information called **marks** attached to them. The mark represents an "attribute" of the point. The mark variable could be *categorical*, e.g. species or disease status:



The mark variable could be *continuous*, e.g. tree diameter:

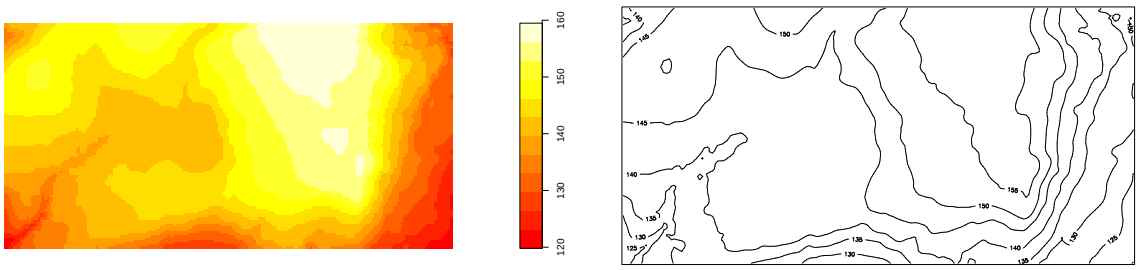


The mark could be multivariate, or even more complicated.

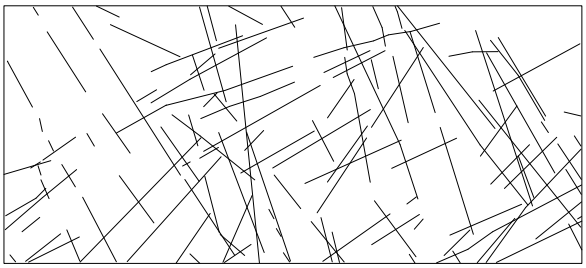
1.1.3 Covariates

Our dataset may also include **covariates** — any data that we treat as explanatory, rather than as part of the 'response'.

Covariate data may be a *spatial function* $Z(u)$ defined at all spatial locations u, e.g. altitude, soil pH, displayed as a pixel image or a contour plot:



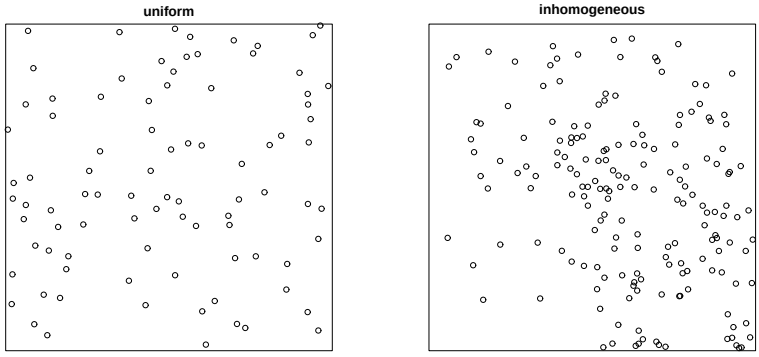
Covariate data may be another *spatial pattern* such as another point pattern, or a line segment pattern, e.g. a map of geological faults:



1.2 Typical scientific questions

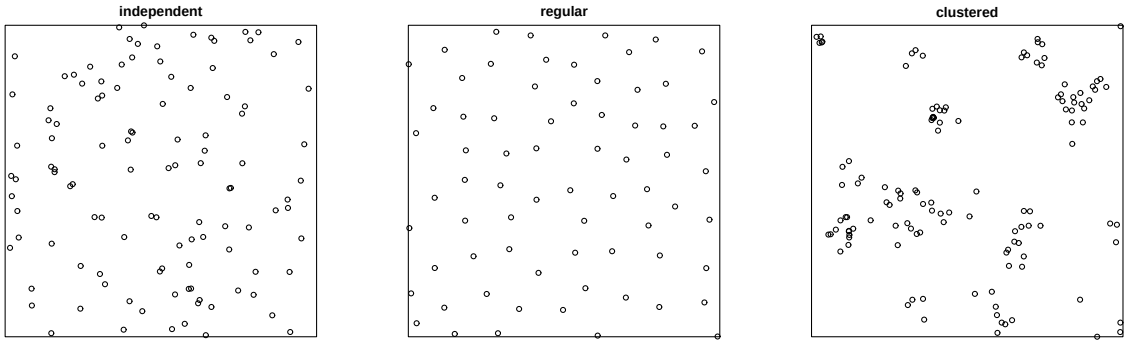
1.2.1 Intensity

'Intensity' is the average density of points (expected number of points per unit area). Intensity may be constant ('uniform') or may vary from location to location ('non-uniform' or 'inhomogeneous').



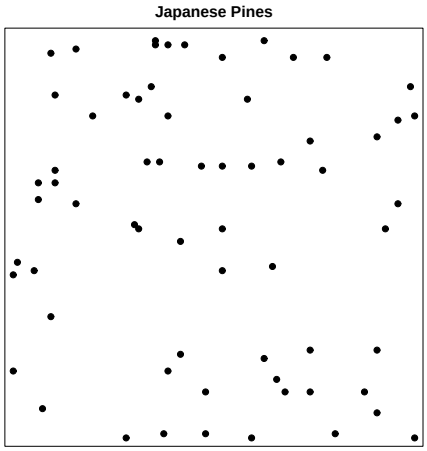
1.2.2 Interaction

'Interpoint interaction' is stochastic dependence between the points in a point pattern. Usually we expect dependence to be strongest between points that are close to one another.



Example 1 (Japanese pines) *Locations of 65 saplings of Japanese pine in a* 5.7×5.7 *metre square sampling region in a natural stand.*

Main question: is the spacing between saplings greater than would be expected for a random pattern? (reflecting competition for resources)



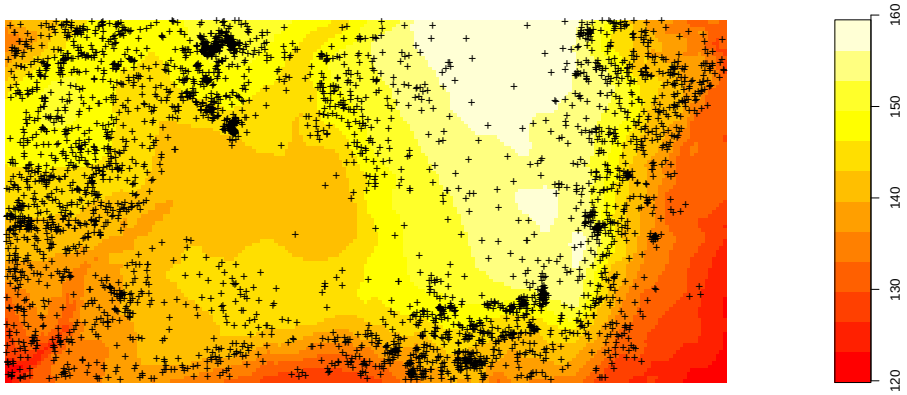
1.2.3 Covariate effects

For a point pattern dataset with covariate data, we typically

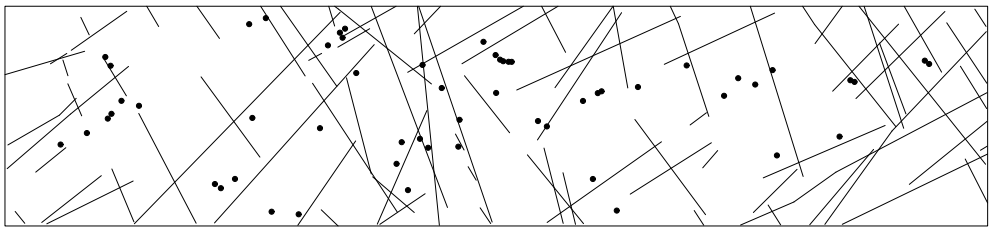
- investigate whether the intensity depends on the covariates
- allow for covariate effects on intensity before studying interaction between points

Example 2 (Tropical rainforest data) *Locations of 3605 trees in a tropical rainforest, with supplementary grid map of elevation (altitude).*

Main questions: (1) does tree density depend on slope? (2) after accounting for variation in tree density due to slope, is there evidence of clustering of trees?

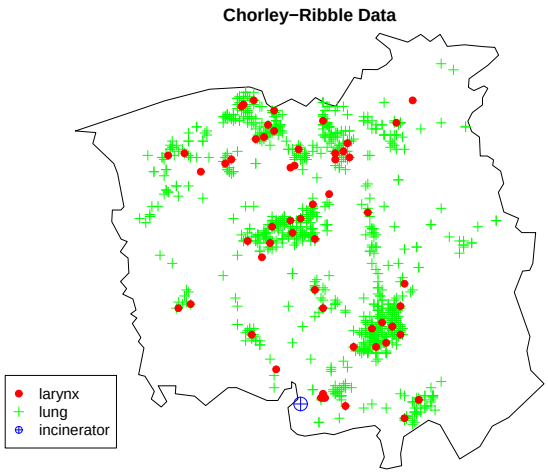


Example 3 (Queensland copper data) *A intensive mineralogical survey yields a map of copper deposits (essentially pointlike at this scale) and geological faults (straight lines). The faults can easily be observed from satellites, but the copper deposits are hard to find. The main question is whether the faults are 'predictive' for copper deposits (e.g. copper less/more likely to be found near faults).*



Example 4 (Chorley-Ribble data) *An apparent cluster of cases of cancer of the larynx occurred near a disused industrial incinerator. The area health authority mapped the domicile locations of all cases (58) of cancer of the larynx and, for control purposes, a random sample of cases (978) of lung cancer.*

Main question: after allowing for spatial variation in density of the susceptible population (for which the lung cancer cases are a surrogate), is there evidence of raised incidence of laryngeal cancer near the incinerator?

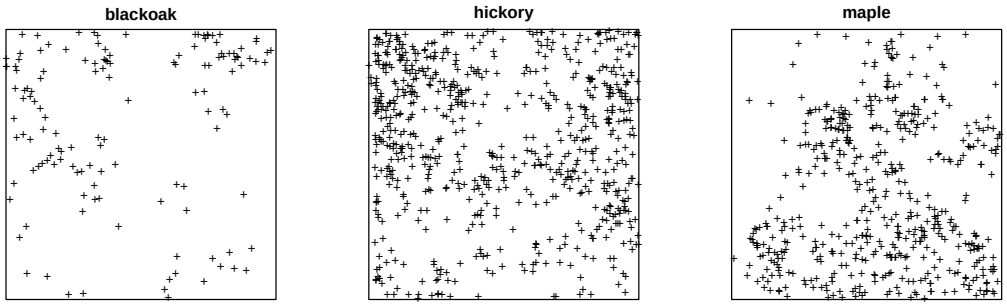


1.2.4 Segregation of points with different marks

In a marked point pattern, we need to investigate whether points with different mark values are 'segregated' (found in different parts of the study region).

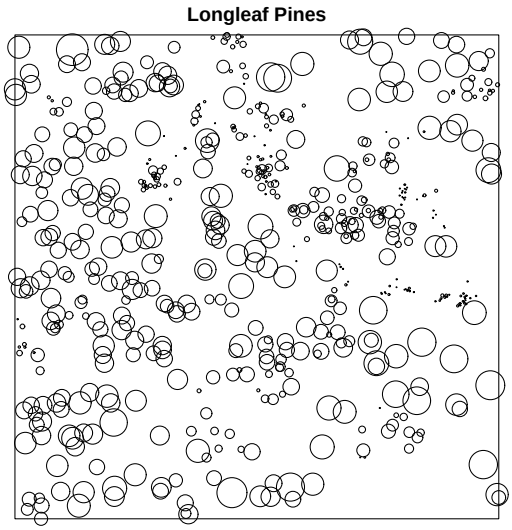
Example 5 (Lansing Woods) *In a 20-acre study region in Lansing Woods, Michigan, the locations of 2251 trees and the botanical classification of each tree were recorded.*

Main question: is the study region divided into domains where a single tree species dominates, or are the different species randomly interspersed?



Example 6 (Longleaf Pines) *In a forest of Longleaf Pine trees in Georgia, USA, the locations of 584 trees were recorded along with their diameter at breast height (dbh), a convenient surrogate measure of size and age.*

Main question: explain any spatial variation in the density and age of trees.



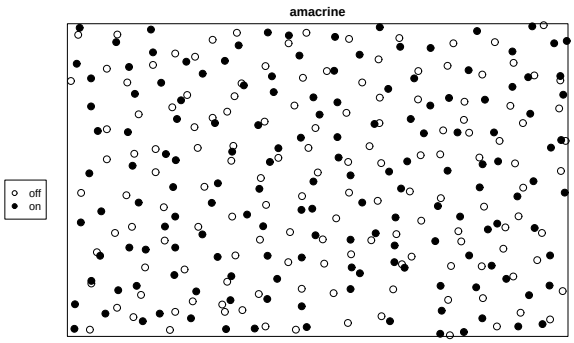
1.2.5 Dependence between points of different types

In a point pattern dataset with **categorical** marks, (aka *multitype point pattern*), dependence between the different types may be formulated either as

- interaction between the sub-pattern of points of type i and the sub-pattern of points of type j; or
- dependence between the mark values of points at two specified locations.

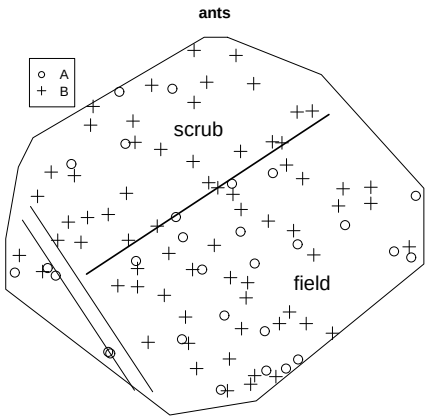
Example 7 (Amacrine cells) *The retina is a flat sheet containing several layers of cells. Amacrine cells occupy two adjacent layers, the 'on' and 'off' layers. In a microscope field of view, the locations of all amacrine cells were mapped, and classified into 'on' and 'off'.*

Main question: is there evidence that the 'on' and 'off' layers grew independently of one another?



Example 8 (Ants' nests) *The nests of two species of ants in a plot in Greece were mapped. Auxiliary information records a field/scrub boundary, and the position of a walking track.*

Main question: does species A intentionally place its nests close to species B?



1.3 Overview of statistical methods

Statistical methods for spatial point patterns have a quirky history, and have not yet coalesced into a mature statistical methodology. They include

- **summary statistics:** the applied literature is dominated by *ad hoc* methods based on evaluating a summary statistic (e.g. average distance from a point to its nearest neighbour) with very little statistical theory to support them.
- **comparison to Poisson process:** in the applied literature, hypothesis tests are invoked chiefly to decide whether the point pattern is 'completely random' (a uniform Poisson point process) whether or not this is scientifically relevant. Lots of misunderstandings prevail.
- **modelling:** only in the last decade has it finally become possible to formulate and fit realistic models to point pattern data. There's still a lot of work to be done e.g. in algorithms, model choice, goodness-of-fit.

We'll cover both classical and modern methods. Useful textbooks include [17, 21, 42, 33]. An important recent survey is [34].

2 Statistical formulation

2.1 Point processes

In this workshop, the observed point pattern **x** will be treated as a realisation of a random **point process X** in two-dimensional space. A point process is simply a random set of points; the *number* of points is random, as well as the locations of the points. Our goal is usually to estimate parameters of the distribution of **X**.

2.2 Should I treat the data as a point process?

Treating the point pattern as a point process effectively assumes that the pattern is *random* (the locations of the points, and the number of points, are random) and that the pattern is the *observation* or *'response'* of interest. A realisation of a point process is an unordered set of points, so the points do not have a serial order (unless there are marks attached).

Example 9 *A silicon wafer is inspected for defects in the crystal surface, and the locations of all defects are recorded.*

This can be analysed as a point process in two dimensions, assuming the defects are point-like. We're interested in the intensity of defects, spacing between defects, etc.

Example 10 *Earthquake aftershocks in Japan are detected and their latitude, longitude and time of occurrence are recorded.*

This can be analysed as a point process in space-time (where space is the two-dimensional plane or the Earth's surface). If the occurrence times are ignored, it becomes a spatial point process.

Example 11 *The locations of petty crimes that occurred in the past week are plotted on a street map of Chicago.*

This can be analysed as a point process. We're interested in the intensity (propensity for crimes to occur), any spatial variation in intensity, clusters of crimes, etc. One issue here is whether the recorded crime locations can be anywhere in two dimensional space, or whether they are actually restricted to locations on the streets (making them a point process on a 1-dimensional network).

Example 12 *A tiger shark is captured, tagged with a satellite transmitter, and released. Over the next month its location is reported daily. These points are plotted on a map.*

It is probably *not* appropriate to analyse these data as a spatial point process. At the very least, the time of each observation should be included. They could be treated as a space-time point process, except that it's a strange process, as it consists of exactly one point at each instant of time. These data should really be treated as a sparse sample of a continuous trajectory, and analysed using other methods [which, alas, are fairly underdeveloped.] See the R package `trip`.

Example 13 *A herd of deer is photographed from the air at noon each day for 10 days. Each photograph is processed to produce a point pattern of individual deer locations on a map.*

Each day produces a point pattern that could be analysed as a realisation of a point process. However, the observations on successive days are dependent (e.g. constant herd size, systematic foraging behaviour). Assuming individual deer cannot be identified from day to day, this is effectively a 'repeated measures' dataset where each response is a point pattern. Methods for this problem are in their infancy.

Example 14 *In a designed controlled experiment, silicon wafers are produced under various conditions. Each wafer is inspected for defects in the crystal surface, and the locations of all defects are recorded as a point pattern.*

This is a designed experiment in which the response is a point pattern. Methods for this problem are in their infancy. There are some methods for *replicated* spatial point patterns [9, 12, 22, 23, 26] that apply when each experimental group contains several point patterns.

Example 15 *The points are not the original data, but were obtained after processing the data. For example,*

- *the original dataset is a pattern of small blobs, and the points are the blob centres;*
- *the original dataset is a collection of line segments, and the points are the endpoints, crossing points, midpoints etc;*
- *the original dataset is a space-filling tessellation of biological cells, and the points are the centres of the cells.*

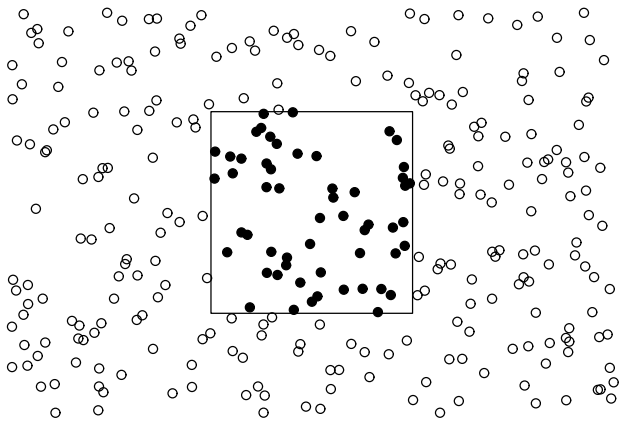
This is a grey area. Point process methodology can be applied, and may be more powerful or more flexible than existing methodology for the unprocessed data. However the origin of the point pattern may lead to artefacts (for example the centres of biological cells never lie very close together, because cells have nonzero size) which must be taken into account in the analysis.

2.3 Assumptions about the data

The "**standard model**" assumes that the point process $\mathbf{X}$ extends throughout 2-D space, but is observed only inside a region W, the *"sampling window"*. Our data consist of an unordered set

$$\mathbf{x} = \{x_1, \ldots, x_n\}, \quad x_i \in W, \quad n \geq 0$$

of points x_i in W. The window W is fixed and known. Usually our goal is inference about parameters of $\mathbf{X}$.



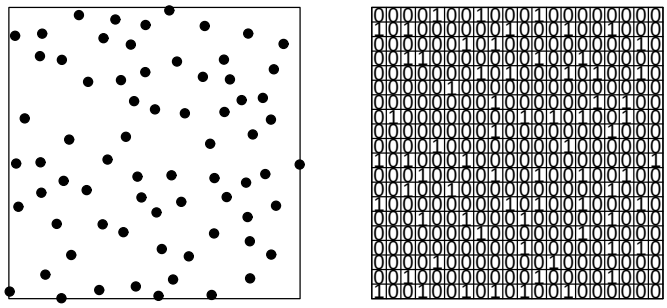
Data are often supplied without information about the sampling window W. **It is important to know the window** W, since we need to know where points were *not* observed. Even something as simple as estimating the density of points depends on the window. It would be wrong, or at least different, to analyze a point pattern dataset by "guessing" the appropriate window. An analogy may be drawn with the difference between sequential experiments and experiments in which the sample size is fixed *a priori*.

For the same reason, it is not sufficient to observe the values of covariates at the data points only. In order to investigate the dependence of the point process on the covariate, we need to have at least some observations of the covariate at other ("non-data") locations.

It's implicitly assumed that all points of $\mathbf{X}$ within W have been mapped without omission.

Most models we use will assume that random points *could* have been observed at any location in the window W, without further constraint. (Examples where this does not apply: GPS locations of cars will usually lie along roads; certain cells lie only inside certain tissues).

When thinking about methodological issues it's often useful to think about the discretised version of a point process. Suppose the window W is chopped into infinitely many 'pixels'. Each pixel is assigned the value $I = 1$ if it contains a point of $\mathbf{X}$, and $I = 0$ otherwise. This array of 0's and 1's constitutes the data that must be modelled. [e.g. obviously we can't model the dependence of these indicators on a covariate if we only observe the covariate value at the locations where $I = 1$.]



2.4 Marks and covariates

The main differences between marks and covariates are that

- marks are associated with data points;
- marks are part of the 'response' (the point pattern) while covariates are 'explanatory'.

2.4.1 Marks

A mark variable may be interpreted as an additional coordinate for the point: for example a point process of earthquake epicentre locations (longitude, latitude), with marks giving the occurrence time of each earthquake, can alternatively be viewed as a point process in space-time with coordinates (longitude, latitude, time).

A marked point process of points in space S with marks belonging to a set M is mathematically defined as a point process in the cartesian product $S \times M$. The space M of possible marks may be 'anything'. In current applications, typically the mark is either a categorical variable (so that the points are grouped into 'types') or a real number. Multivariate marks consisting of several such variables are also common.

A marked point pattern is an unordered set

$$\mathbf{y} = \{(x_1, m_1), \ldots, (x_n, m_n)\}, \quad x_i \in W, \quad m_i \in M$$

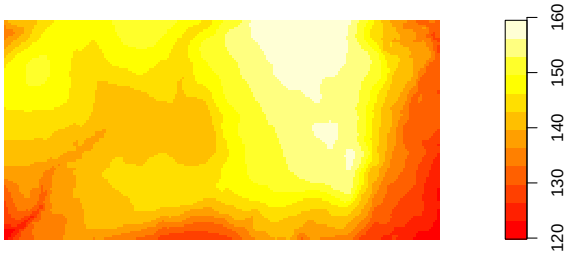
where x_i are the locations and m_i are the corresponding marks.

Marked point patterns are discussed in detail in section 21.

2.4.2 Covariates

Any kind of data may be recruited as an explanatory variable (covariate).

A 'spatial function', 'spatial covariate' or 'geostatistical covariate' is a function $Z(u)$ observable (potentially) at every spatial location $u \in W$. Values of $Z(u)$ may be available for a fine grid of locations u:



The values of a spatial function $Z(u)$ may only be observable at some scattered sampling locations u. An example is the measurement of soil pH at a few sampling locations. In this case, the value of the covariate Z must be observed for all points x_i of the point pattern $\mathbf{x}$, and must also be observed at some other 'non-data' or 'background' locations $u \in W$ with $u \notin \mathbf{x}$.

Alternatively, the covariate information may consist of another spatial pattern, such as a point pattern or a line segment pattern. The way in which this covariate information enters the analysis or statistical model depends very much on the context and the choice of model. Typically the covariate pattern would be used to define a surrogate spatial function Z, for example, $Z(u)$ may be the distance from u to the nearest line segment.

3 The R system

We will be using the statistical package R.

3.1 How to obtain R

R is free software with an open-source licence. You can download it from `r-project.org` and it should be easy to install on any computer (see the instructions at the website).

Books and online tutorials are available to help you learn to use R.

3.2 How commands are printed in the notes

You can run an R session using either a point-and-click interface or a line-by-line command interpreter. In these notes, R commands are printed as they would appear when typed at the command line. So a typical series of R commands looks like this:

```
> pi/2
> sin(pi/2)
> x <- sqrt(2)
> x
```

Note that you are not meant to type the > symbol; this is just the prompt for command input in R. To type the first command, just type `pi/2`.

In these notes we will sometimes also print the response that R gives to a set of commands. In the example above, it would look like this:

```
> pi/2

[1] 1.570796

> sin(pi/2)

[1] 1

> x <- sqrt(2)
> x

[1] 1.414214
```

If the input is too long, R will break it into several lines, and print the character + to indicate that the input continues from the previous line. (You don't type the +). Also if you type an expression involving brackets and hit Return before all the open brackets have been closed, then R will print a + indicating that it expects you to finish the expression.

```
> folderol <- 1.2
> sin(folderol * folderol * folderol * folderol * folderol * folderol *
+     folderol * folderol * folderol * folderol)

[1] -0.09132148
```

3.3 Contributed libraries for R

In addition to the basic R system, the R website also offers many add-on modules ('libraries' or 'packages') contributed by users. These can be downloaded from `cran.r-project.org` (under 'Contributed Packages').

Packages that may be useful for analysing spatial data include:

`ads`	spatial point pattern analysis
`DCluster`	detecting clusters in spatial count data
`fields`	curve and function fitting
`geoR`	model-based geostatistical methods
`geoRglm`	model-based geostatistical methods
`GeoXB`	interactive spatial exploratory data analysis
`grasp`	spatial prediction
`maptools`	geographical information systems
`rgdal`	interface to GDAL geographical data analysis
`sp`	base library for some spatial data analysis packages
`spatclus`	detecting clusters in spatial point pattern data
`spatialCovariance`	spatial covariance for data on grids
`spatialkernel`	interpolation and segregation of point patterns
`spatstat`	**Spatial point pattern analysis and modelling**
`spBayes`	Gaussian spatial process MCMC (grid data)
`spdep`	spatial statistics for variables observed at fixed sites
`spgwr`	geographically weighted regression
`splancs`	spatial and space-time point pattern analysis
`spsurvey`	spatial survey methods
`trip`	analysis of spatial trip data

To make use of a package, you need to:

1. download the package code (once only) *without unpacking*;
2. 'install' the package code on your system (once only);
3. 'load' the package into your current R session using the command `library` (each time you start a new R session).

The installation step is performed automatically using R, not by manually unpacking the code. Installation is usually a very easy process.

Instructions on how to install a package are given at `cran.r-project.org`. If you are running Windows, first start an R session. Then try the pull-down menu item `Packages — Install packages`. If this menu item is available, then you will be able to download and install any desired packages by simply selecting the package name from the pulldown list. If this menu item is not available (for internet security reasons), you can manually download packages by going to the CRAN website under `Contributed packages -- Windows binaries` and downloading the desired zip files of Windows binary files. To perform step 2, start an R session and use the menu item `Packages — Install from local zip files` to install.

If you are running Linux, step 1 is performed manually by going to the CRAN website under `Contributed Packages` and downloading the tar file `packagename.tar.gz`. Step 2 is performed by issuing the command `R CMD INSTALL packagename.tar.gz`.

4 Introduction to spatstat

4.1 The spatstat package

Spatstat is a contributed R package for analysing spatial data, written by Adrian Baddeley and Rolf Turner. Current versions of spatstat deal mainly with **spatial point patterns** in two dimensions. The package supports

- creation, manipulation and plotting of point patterns
- exploratory data analysis
- simulation of point process models
- parametric model-fitting
- hypothesis tests, residual plots, diagnostics

Spatstat is one of the largest contributed packages available for R, with over 300 user-level functions and a 500-page manual. It has its own web domain, www.spatstat.org, offering information about the package.

Spatstat can be downloaded from cran.r-project.org (under 'Contributed packages — spatstat'). To install spatstat you will also need to download the packages mgcv and sm.

4.2 Please acknowledge spatstat

If you use spatstat for research that leads to publications, it would be much appreciated if you could acknowledge spatstat in your publications, preferably citing [4]. Citations help us to justify the expenditure of time and effort on maintaining and developing the package.

4.3 Getting started

Here is a quick demonstration of spatstat in action. You can follow the demonstration by typing the commands into R.

To begin any analysis using spatstat, first start the R system, and type

```
> library(spatstat)
```

The response will be something like this:

```
> library(spatstat)

This is mgcv 1.3-20


spatstat 1.12-7
Type 'help(spatstat)' for information
```

The printout shows that, before loading spatstat, the system has loaded the package mgcv that is required by spatstat. Then it loads spatstat, showing the version number of the package.

For a list of the commands available in spatstat, type

```
> help(spatstat)
```

To get information on a particular *command*, type help(*command*).

To gain an impression of what is available in spatstat, you can run the package demonstration by typing demo(spatstat).

4.4 Inspecting data

For our first demonstration, we'll use one of the standard point pattern datasets that is installed with the package. The 'Swedish Pines' dataset represent the positions of 71 trees in a forest plot 9.6 by 10.0 metres.

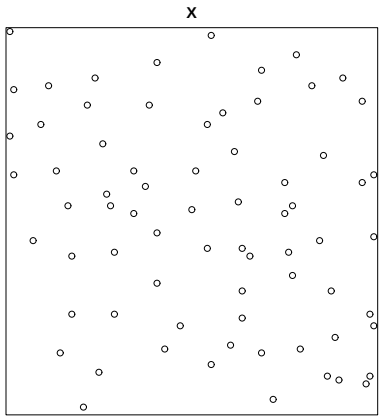
```
> data(swedishpines)
```

To avoid typing 'swedishpines' all the time, let us copy the data to another dataset with a shorter name:

```
> X <- swedishpines
```

You can immediately plot the point pattern by typing

```
> plot(X)
```



Simply typing the name of the dataset gives you some basic information:

```
> X
```

```
 planar point pattern: 71 points
window: rectangle = [0, 96] x [0, 100] units (one unit = 0.1 metres)
```

Let's study the intensity (density of points) in this point pattern. For a few basic summary statistics, type

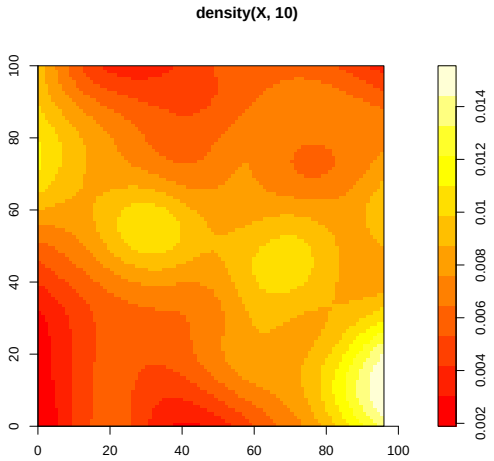
```
> summary(X)
```

```
Planar point pattern: 71 points
Average intensity 0.0074 points per square unit (one unit = 0.1 metres)

Window: rectangle = [0, 96] x [0, 100] units
Window area =  9600 square units
Unit of length: 0.1 metres
```

The coordinates are in decimetres (0.1 metre), so the average intensity is 0.0074 trees per square decimetre or 0.74 trees per square metre.

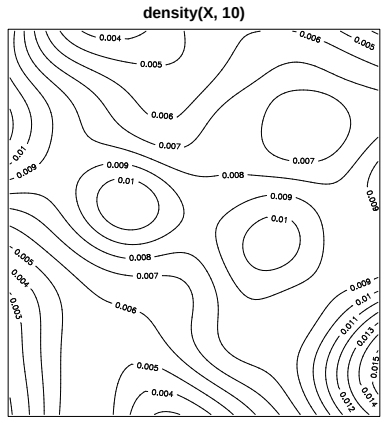
To get an impression of local spatial variations in intensity, we can plot a kernel estimate of intensity:

```
> plot(density(X, 10))
```



where `10` is my chosen value for the standard deviation of the Gaussian smoothing kernel. If you prefer a contour plot,

```
> contour(density(X, 10), axes = FALSE)
```



The contours are labelled in density units of "trees per square decimetre".

4.5 Exploratory data analysis

`Spatstat` is designed to support all the standard types of exploratory data analysis for point patterns.

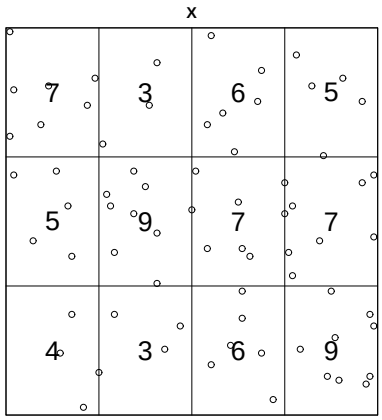
One example is *quadrat counting*. The study region is divided into rectangles ('quadrats') of equal size, and the number of points in each rectangle is counted.

```
> Q <- quadratcount(X, nx = 4, ny = 3)
> Q
```

```
          y
x          [0,33.3] (33.3,66.7] (66.7,100]
  [0,24]          4           5          7
  (24,48]         3           9          3
```

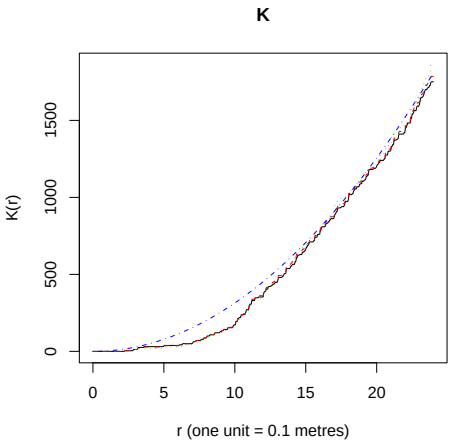
```
  (48,72]        6           7           6
  (72,96]        9           7           5
```

```
> plot(X)
> plot(Q, add = TRUE, cex = 2)
```



Another example is *Ripley's K function.* I'll explain more about the K function later. For now, we'll just demonstrate how easy it is to compute and plot it. To compute the K function for a point pattern `X`, type `Kest(X)`. This returns an object which can be plotted.

```
> K <- Kest(X)
> plot(K)
```



4.6 Multitype point patterns

A marked point pattern in which the marks are a categorical variable is usually called a *multitype* point pattern. The 'types' are the different values or levels of the mark variable.

Here is the famous Lansing Woods dataset recording the positions of 2251 trees of 6 different species (hickories, maples, red oaks, white oaks, black oaks and miscellaneous trees).

```
> data(lansing)
> lansing
```

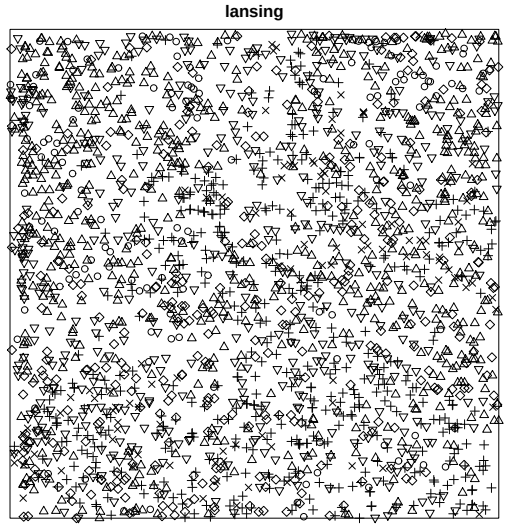
```
marked planar point pattern: 2251 points
multitype, with levels = blackoak        hickory        maple        misc        redoak
window: rectangle = [0, 1] x [0, 1] units (one unit = 924 feet)
```

```
> summary(lansing)
```

```
Marked planar point pattern: 2251 points
Average intensity 2250 points per square unit (one unit = 924 feet)

*Pattern contains duplicated points*
Multitype:
         frequency proportion intensity
blackoak       135     0.0600       135
hickory        703     0.3120       703
maple          514     0.2280       514
misc           105     0.0466       105
redoak         346     0.1540       346
whiteoak       448     0.1990       448

Window: rectangle = [0, 1] x [0, 1] units
Window area =  1 square unit
Unit of length: 924 feet
```

```
> plot(lansing)
```

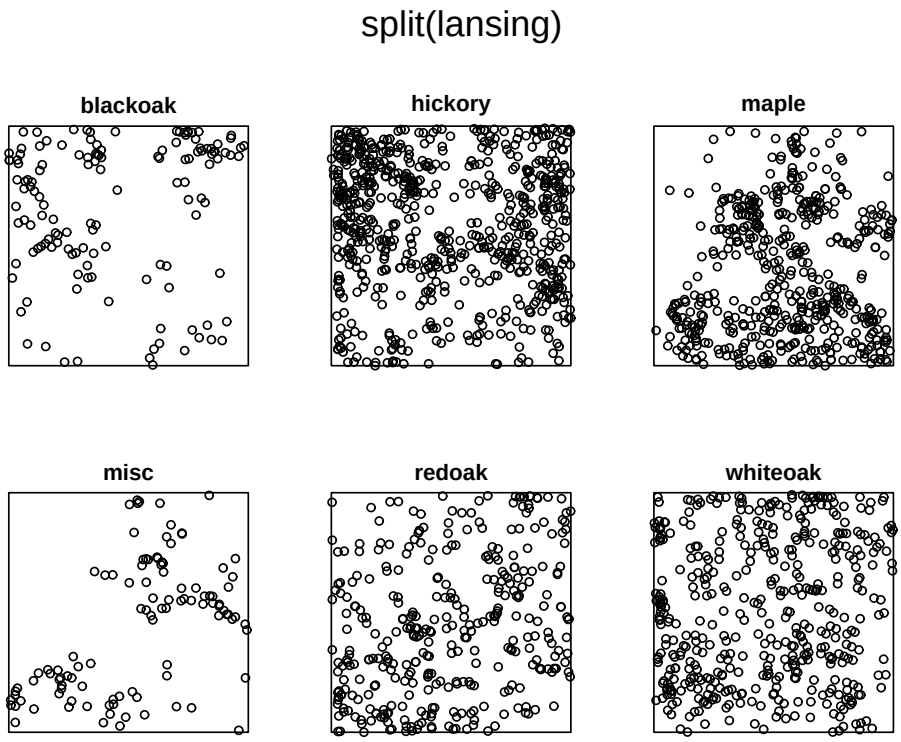
```
blackoak  hickory    maple     misc   redoak whiteoak
       1        2        3        4        5        6
```



In this plot, each type of point (i.e. each species of tree) is represented by a different plot symbol. The last line of output above explains the encoding: black oak is coded as symbol 1 (open circle) and so on.

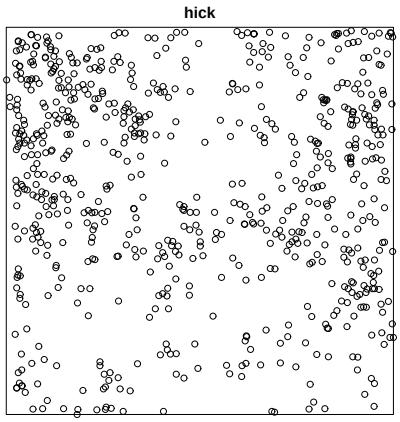
An alternative way to plot these data is to split them into 6 point patterns, each pattern containing the trees of one species. This is done using `split`:

```
> plot(split(lansing))
```



The result of `split(lansing)` is a list of point patterns. The names of the list entries are the names of the types (in this case `"blackoak"`,`"hickory"`, etc). To extract one of these patterns, e.g. the hickories,

```
> hick <- split(lansing)$hickory
> plot(hick)
```



4.7 Installed datasets

For reference, here is a list of the standard point pattern datasets that are supplied with the installation of `spatstat`:

name	description	marks	covariates	window
`amacrine`	Hughes' rabbit amacrine cells	2 types	·	☐
`anemones`	Upton-Fingleton sea anemones	diameter	·	☐
`ants`	Harkness-Isham ant nests	2 species	2 zones	convex poly
`bei`	Tropical rainforest trees	·	topography	☐
`betacells`	Wässle et al. cat retinal ganglia	2 types	·	☐
`bramblecanes`	Bramble Canes	3 ages	·	☐
`cells`	Crick-Ripley biological cells	·	·	☐
`chorley`	Chorley-South Ribble cancers	case/control	·	irregular
`copper`	Queensland copper deposits	·	fault lines	☐
`demopat`	artificial data	2 types	·	irregular
`finpines`	Finnish Pines	diameter	·	☐
`hamster`	Aherne's hamster tumour data	2 types	·	☐
`humberside`	Humberside child leukaemia	case/control	·	irregular
`japanesepines`	Japanese Pines	·	·	☐
`lansing`	Lansing Woods	6 species	·	☐
`longleaf`	Longleaf Pine trees	diameter	·	☐
`nbfires`	New Brunswick fires	several	·	irregular
`nztrees`	Mark-Esler-Ripley NZ trees	·	·	☐
`ponderosa`	Getis-Franklin Ponderosa pines	·	·	☐
`redwood`	Strauss-Ripley redwood saplings	·	·	☐
`redwoodfull`	Strauss redwood map (full set)	·	2 zones	☐
`simdat`	Simulated point pattern	·	·	☐
`spruces`	Spruce trees in Saxony	diameter	·	☐
`swedishpines`	Strand-Ripley Swedish pines	·	·	☐

The symbol ☐ indicates that the window for the pattern is a rectangle.

To flick through a nice display of all these datasets, type `demo(data)`.

To access one of these datasets, type `data(`*`name`*`)` where *`name`* is the name listed above. To see information about the dataset, type `help(`*`name`*`)`. To plot the dataset, type `plot(`*`name`*`)`.

4.8 Point-and-click on the screen

There is a graphical interface which allows you to draw a point pattern on the screen. Type

```
> X <- clickppp(10)
```

This opens a graphics window and invites you to point and click 10 times in the window. The result is a point pattern, consisting of 10 points, stored in the object named `X`. To plot it, type

```
> plot(X)
```

5 Objects, classes and methods

The tutorial examples above have used some of the 'object-oriented' features of R. It is very useful to know a little about how these work.

5.1 Classes in R

R is an 'object-oriented' language. A dataset with some kind of structure on it (e.g. a contingency table, a time series, a point pattern) is treated as a single 'object'.

For example, R includes a dataset `sunspots` which is a time series containing monthly sunspot counts from 1749 to 1983. This dataset can be manipulated as if it were a single object:

```
> plot(sunspots)
> summary(sunspots)
> X <- sunspots
```

Each object in R is identified as belonging to a particular type or **class** depending on its structure. For example, the `sunspots` dataset is a time series:

```
> class(sunspots)

[1] "ts"
```

Standard operations, such as printing, plotting, or calculating the sample mean, are defined separately for each class of object.

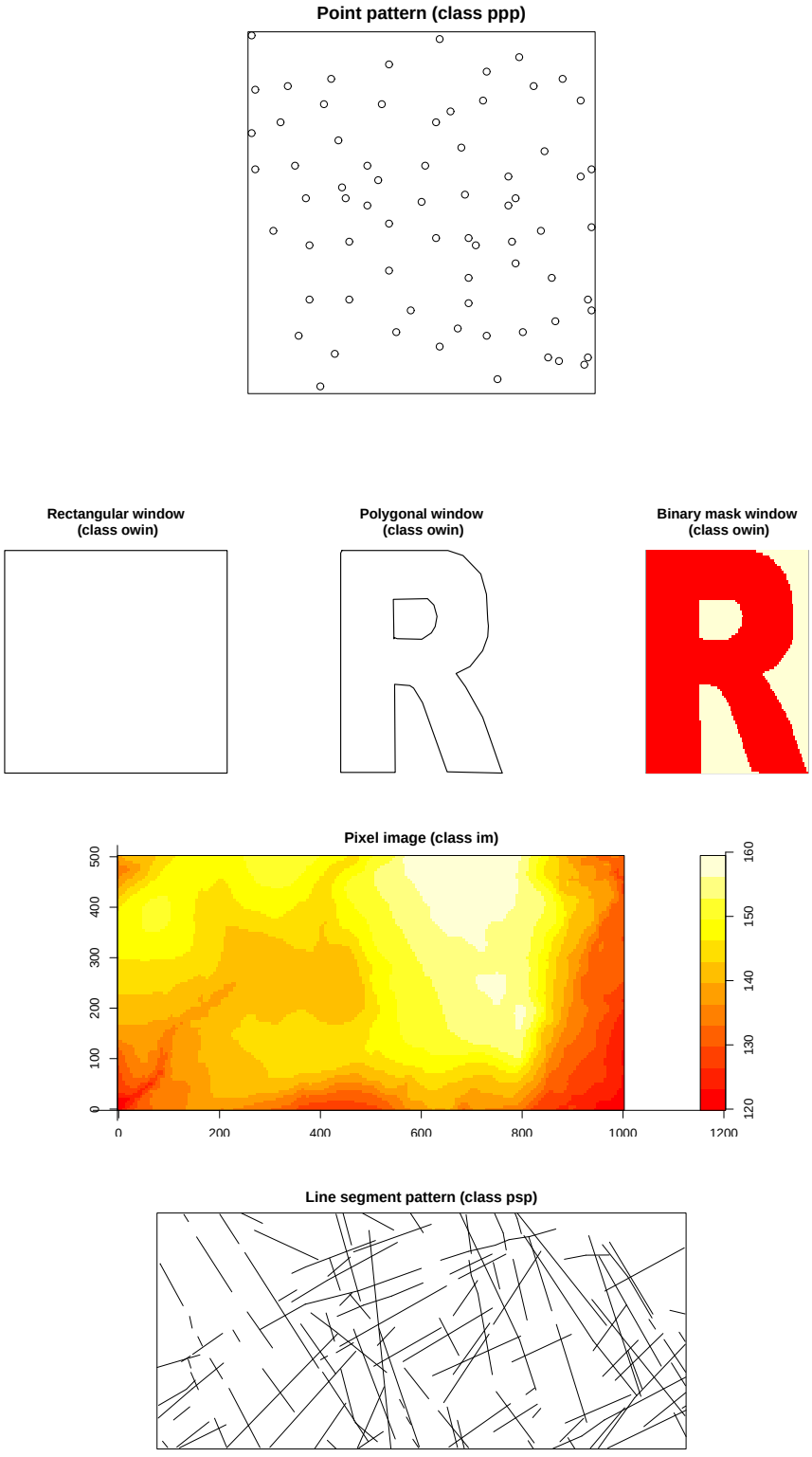
For example, typing `plot(sunspots)` invokes the generic command `plot`. Now `sunspots` is an object of class `"ts"` representing a time series, and there is a special **"method"** for plotting time series, called `plot.ts`. So the system executes `plot.ts(sunspots)`. It is said that the plot command is "dispatched" to the method `plot.ts`. The plot method for time series produces a display that is sensible for time series, with axes properly annotated.

> Tip: to find out how to modify the plot for an object of class `"foo"`, consult `help(plot.foo)` rather than `help(plot)`.

5.2 Classes in `spatstat`

To handle point pattern datasets and related data, the `spatstat` package defines the following classes of objects:

- `ppp`: planar point pattern
- `owin`: spatial region ('observation window')
- `im`: pixel image
- `psp`: pattern of line segments

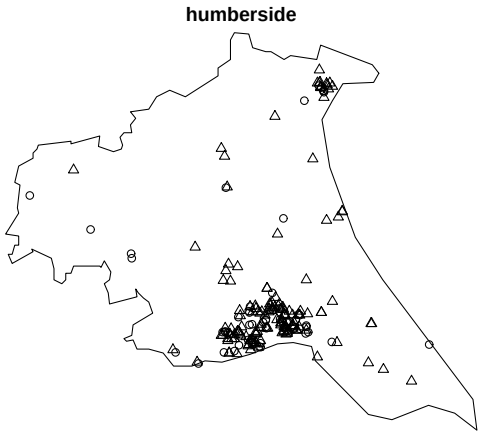


Most of the functionality in `spatstat` works on such objects. To use this functionality, you'll need to read your raw data into R and then convert it into an object of the appropriate format.

In particular `spatstat` has methods for `plot`, `print` and `summary` for each of these classes. For example, the `plot` method for point patterns, `plot.ppp`, ensures that the x and y scales

are equal, and does various other things that are sensible when plotting a spatial point pattern rather than just a list of (x, y) pairs.

```
> data(humberside)
> plot(humberside)
```



Exercise 1 *Find out how to modify the command* `plot(swedishpines)` *so that the title reads "Swedish Pines data" and the points are represented by plus-signs instead of circles.*

When you type `print(swedishpines)` or just `swedishpines`, this invokes the generic command `print`, which dispatches to the method `print.ppp`, which prints some sensible information about the point pattern `swedishpines` at the terminal.

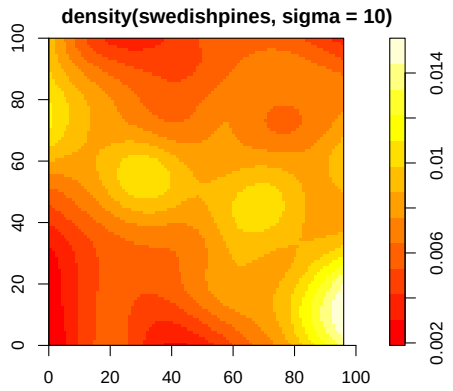
```
> swedishpines

 planar point pattern: 71 points
window: rectangle = [0, 96] x [0, 100] units (one unit = 0.1 metres)
```

The generic command `summary` is meant to provide basic summary statistics for a dataset. When you type `summary(swedishpines)` this is dispatched to the method `summary.ppp`, which computes a sensible set of summary statistics for a point pattern, and prints them at the terminal.

```
> summary(swedishpines)

Planar point pattern: 71 points
Average intensity 0.0074 points per square unit (one unit = 0.1 metres)

Window: rectangle = [0, 96] x [0, 100] units
Window area =  9600 square units
Unit of length: 0.1 metres
```

The command `density` is also generic. It is normally used to compute a kernel density estimate of a probability distribution from a vector of numbers. (This "default method" is called `density.default`.) But there is also a method for point patterns, so that when you type `density(swedishpines)`, this is dispatched to `density.ppp` which computes a two-dimensional kernel estimate of the intensity function.

```
> plot(density(swedishpines, sigma = 10))
```



To see a list of all methods available in R for a particular generic function such as `plot`:

```
> methods(plot)
```

To see a list of all methods that are available for a particular class such as `ppp`:

```
> methods(class = "ppp")
```

```
 [1] [.ppp              [<-.ppp            affine.ppp         as.data.frame.ppp
 [5] as.owin.ppp        as.ppp.ppp         crossdist.ppp      cut.ppp
 [9] density.ppp        distmap.ppp        duplicated.ppp     identify.ppp
[13] is.marked.ppp      is.multitype.ppp   kstest.ppp         markformat.ppp
[17] marks.ppp          marks<-.ppp        nndist.ppp         nnwhich.ppp
[21] pairdist.ppp       pcf.ppp            plot.ppp           print.ppp
[25] quadrat.test.ppp   rescale.ppp        rotate.ppp         rshift.ppp
[29] shift.ppp          split.ppp          split<-.ppp        subset.ppp
[33] summary.ppp        unique.ppp         unitname.ppp       unitname<-.ppp
```

5.3 Return values

5.3.1 The return value of a function

Every function in R returns a value. The return value may be 'null', or a single number, a list, or any kind of object. When you type an R expression on the command line, the result of evaluating the expression is printed.

```
> 1 + 1

[1] 2

> sin(pi/3)

[1] 0.8660254
```

Just to confuse matters, the result of a function may be tagged as *'invisible'* so that it is not printed.

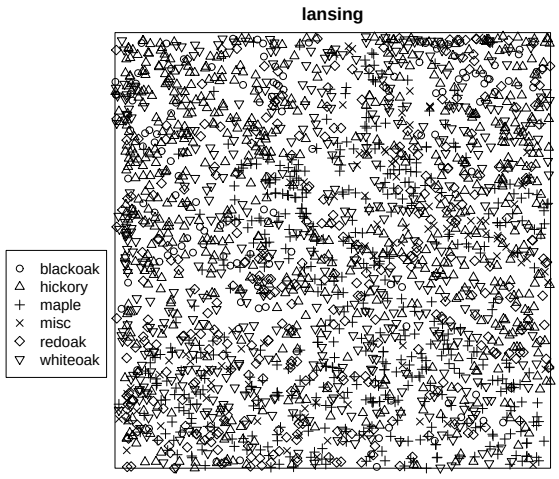
```
> data(cells)
> plot(cells)
```

There's still a return value from the function, which can be captured by assigning the result to a variable:

```
> a <- plot(cells)
> a

NULL
```

Tip: Many plotting commands return a value which is useful if you want to annotate the plot. In `spatstat` the function `plot.ppp` plots a point pattern and returns information about the encoding of the marks. After plotting a multitype pattern, to make a nice legend for the plot, save the result of the `plot` call and pass it to the `legend` command:

```
> a <- plot(lansing)
> legend(-0.25, 0.5, names(a), pch = a)
```



Tip: To find out the format of the output returned by a particular function `fun`, type `help(fun)` and read the section headed *'Value'*.

5.3.2 Returning an object

A function which performs a complicated analysis of your data will typically return an object belonging to a special class. This is a convenient way to handle calculations that yield large or complicated output. It enables you to store the result for later use, and provides methods for handling the result.

Many of the functions in `spatstat` return an object of a special class. For example, the value returned by `density.ppp` is a pixel image (an object of class `"im"`). This is effectively a large matrix, giving the values of the kernel estimate of intensity at each point in a fine regular grid of locations.

```
> Z <- density(swedishpines, sigma = 10)
> Z
```

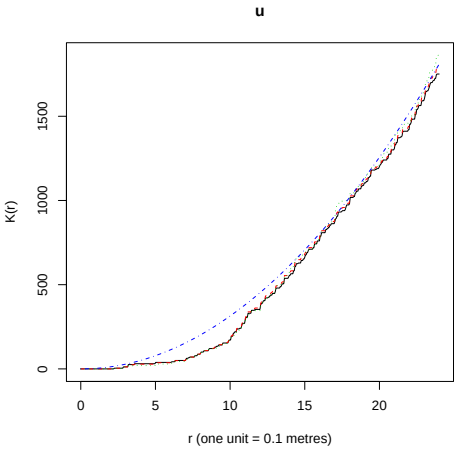
```
real-valued pixel image
100 x 100 pixel array (ny, nx)
enclosing rectangle: [0, 96] x [0, 100] units (one unit = 0.1 metres)
```

The class of pixel images in `spatstat` has methods for `print`, `summary`, `plot` and so on.

```
> summary(Z)

real-valued pixel image
100 x 100 pixel array (ny, nx)
enclosing rectangle: [0, 96] x [0, 100] units
dimensions of each pixel: 0.96 x 1 units
(one unit = 0.1 metres)
Image is defined on the full rectangular grid
Frame area =  9600 square units
Pixel values :
        range = [0.00188947243195950,0.0155470858797917]
        integral = 71.3036909843861
        mean = 0.00742746781087355
```

Another example is the command `Kest` which estimates Ripley's K-function. The value returned by `Kest` is an object of class `"fv"` ('function value table') containing the estimated values of $K(r)$, obtained using several different estimators, for a range of r values. This class has methods for `print`, `plot` and so on.

```
> u <- Kest(X)
> u

Function value object (class 'fv')
for the function r -> K(r)
Entries:
id       label       description
--       -----       -----------
r        r           distance argument r
theo     Kpois(r)    theoretical Poisson K(r)
border   Kbord(r)    border-corrected estimate of K(r)
trans    Ktrans(r)   translation-corrected estimate of K(r)
iso      Kiso(r)     Ripley isotropic correction estimate of K(r)
--------------------------------------

Default plot formula:
        . ~ r

Recommended range of argument r: [0, 24]
Unit of length: 0.1 metres

> plot(u)
```



6 Data input

To analyse your *own* point pattern data in `spatstat`, you'll need to read the raw data into R and convert them into an object of class `"ppp"`. This tutorial gives one basic recipe.

6.1 Basic recipe

In most cases, the observation window is a rectangle. The following steps will then be sufficient.

1. store the x and y coordinates for the points in two vectors `x` and `y`.
2. if there are marks attached to the points, store the corresponding marks in a vector `m`. (*Note:* only a single mark value per point is allowed; multivariate marks are not supported. But we're working on it.)
3. create the point pattern object by

   ```
   > ppp(x, y, xrange, yrange)
   ```

   or, if there are marks,

   ```
   > ppp(x, y, xrange, yrange, marks = m)
   ```

   where `xrange`, `yrange` are vectors of length 2 giving the x and y dimensions of the rectangular window.

The value returned by the function `ppp` is an object of class `"ppp"` representing a point pattern.

Entering coordinate data

Suppose we have recorded the x, y coordinates of 25 points that lie in a rectangle $[0, 2] \times [0, 1]$. They can be entered into R in various ways, for example by typing them directly:

```
> x <- scan()
```

```
1: 1.94  0.32 1.74 0.64 0.12 1.44 0.29 0.74
9: 0.32 1.35 1.23 0.53 0.98 0.96 0.91 1.28
17: 1.24 0.14 1.75 0.24 0.45 0.94 1.22 1.60 0.62
26:
Read 25 items
```

```
> y <- scan()
```

```
1: 0.40 0.70 0.91 0.92 0.13 0.92 0.72 0.15
9: 0.78 0.59 0.02 0.70 0.75 0.33 0.52 0.75
17: 0.19 0.32 0.87 0.13 0.63 0.08 0.72 0.67 0.96
26:
Read 25 items
```

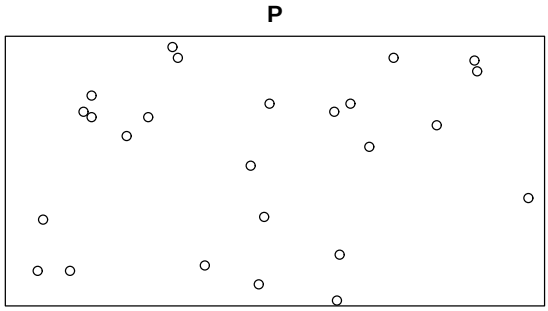
You can also use `scan(file="filename")` to read a stream of numbers from a file. Alternatively, if the file is nicely formatted as a table with a separate line for each data point, use `read.table`.

Unmarked point pattern

In the example above, the x coordinates are in the range $[0, 2]$ and the y coordinates in $[0, 1]$. To create the point pattern object we simply type

```
> P <- ppp(x, y, c(0, 2), c(0, 1))
> plot(P)
> P
```

```
 planar point pattern: 25 points
window: rectangle = [0, 2] x [0, 1] units
```



Marked point pattern

Mark values may have any atomic type: numeric, integer, character, logical, or complex. For example, let's take a vector of real numbers:

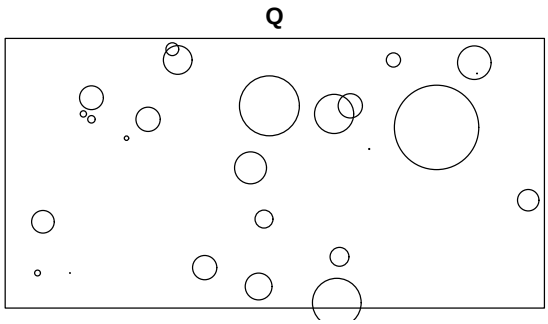
```
> m <- scan()
```

```
1: 9.2 3.2 14.4 12.3 2.5 6.1  2.7 10.4
9: 10.2  0.4 20.9 10.4 25.7  7.7 13.7
16: 10.4  8.1  9.7  0.3  0.2  1.9 11.5
```

```
23: 16.8 36.2  5.5
26:
Read 25 items
```

and include this as the marks vector for the point pattern:

```
> Q <- ppp(x, y, c(0, 2), c(0, 1), marks = m)
> Q

marked planar point pattern: 25 points
marks are numeric, of type 'double'
window: rectangle = [0, 2] x [0, 1] units

> plot(Q)

         0         10         20         30         40
0.00000000 0.04323888 0.08647777 0.12971665 0.17295553
```



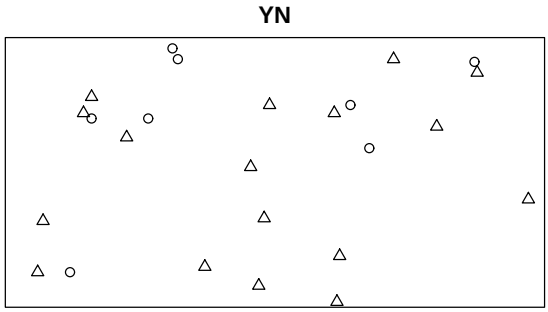
The last line of output is the return value from `plot(Q)`, which indicates the scale used to plot the marks. The mark value 10 was plotted as a circle of radius 0.0432.

Categorical marks

When the mark is a categorical variable, we have a *multitype point pattern*. The 'types' are the different levels of the mark variable. **The mark values should be stored as a 'factor' in R.**

For example, let's attach random marks to the pattern, taking two possible values `Yes` and `No` with equal probability.

```
> m <- sample(c("Yes", "No"), 25, replace = TRUE)
> m <- factor(m)
> YN <- ppp(x, y, c(0, 2), c(0, 1), marks = m)
> YN

marked planar point pattern: 25 points
multitype, with levels = No        Yes
window: rectangle = [0, 2] x [0, 1] units

> plot(YN)

 No Yes
  1   2
```



If the marks are intended to be a categorical variable, ensure that `m` is stored as a 'factor'.

The last line of output indicates how the marks were plotted: the mark `No` was plotted as symbol 1 (circle) and mark `Yes` was plotted as symbol 2 (triangle).

Notice that the factor levels have been re-sorted alphabetically (by default). This is one of the common slip-ups with factors in R. To stipulate a different ordering of the levels,

```
> m <- factor(m, levels = c("Yes", "No"))
> YN <- ppp(x, y, c(0, 2), c(0, 1), marks = m)
> YN

marked planar point pattern: 25 points
multitype, with levels = Yes        No
window: rectangle = [0, 2] x [0, 1] units
```

> Tip: whenever you create a factor, check that the factor levels are as you intended, using `levels(x)`.

Other ways of adding marks to a point pattern will be described in Section 23.

6.2 Checking data

It is prudent to check for quirks in the data.

- Print out the coordinate values and marks to check for errors in data entry, and to determine whether the coordinates have been rounded.
- *Duplicated points* are surprisingly common in data files (i.e. where two records in the file refer to the same (x, y) location). Once you have entered the coordinates into R as a two-column matrix or a data frame `D` say, you can check for duplication using the command `any(duplicated(D))`. If your data are already in the form of a point pattern `X`, you can also type `any(duplicated(X))` to detect duplication. To remove duplicated points, type `Y <- unique(X)`.
- Plotting the point pattern is always wise. Look for unexpected patterns, and points that lie outside the window.
- On a plot of a point pattern `X`, you can identify an individual point by typing `plot(X); identify(X)` then clicking on the point.

The function `ppp` automatically checks for duplicated points, and for points that lie outside the specified window.

6.3 Units

A point pattern `X` may include information about the units of length in which the x and y coordinates are recorded. This information is optional; it merely enables the package to print better reports and to annotate the axes in plots.

If the x and y coordinates in the point pattern `P` were recorded in metres, type

```
> unitname(P) <- c("metre", "metres")
```

at least in Australia or New Zealand. The two strings are the singular and plural forms of the unit. In Scandinavia and Germany you would type

```
> unitname(P) <- "meter"
```

The measurement unit can also be given as some multiple of a standard unit. If, for example, one unit for the x and y coordinates equals 42 centimetres, type

```
> unitname(P) <- list("cm", "cm", 42)
```

Beware that the `unitname` applies only to the coordinates, and not to the marks, of a point pattern.

Altering the `unitname` in an existing dataset is usually not sensible; it simply alters the name of the unit, without changing the entries in the x and y vectors. If you want to convert to different units (e.g. from metres to kilometres or from imperial to metric units), use the command `rescale` as described in Section 9.2.5. If you want to actually change the coordinates by a linear transformation, producing a dataset that is not equivalent to the original one, use `affine`.

6.4 Other ways to make point patterns

To create a point pattern object we can either

- create one from raw data using the function `ppp`
- convert data from other formats (including other packages) using `as.ppp`
- point-and-click on a graphics device using `clickppp`
- read data from a file using `scanpp`
- transform an existing point pattern using a variety of tools
- generate a random pattern using one of the simulation routines
- use one of the standard point pattern datasets supplied with the package.

The package help file `help(spatstat)` lists all the available options.

Note that it is a standard naming convention in R that, for a class `"foo"`, there should be a 'creator' function `foo` that creates objects of this class from raw numerical data, and a 'converter' function `as.foo` that converts data from other formats into objects of class `"foo"`. We adhere to this convention in `spatstat`:

Class	Creator	Converter
`"ppp"`	`ppp`	`as.ppp`
`"owin"`	`owin`	`as.owin`
`"im"`	`im`	`as.im`

More alternatives for using `ppp` will be covered in Section 8.4.

7 Methods 1: Investigating intensity

Finally we can start working on statistical methods for analysing point pattern data.

When we analyse numerical data, we often begin by taking the sample mean. The analogue of the mean or expected value of a random variable is the *intensity* of a point process.

'Intensity' is the average density of points (expected number of points per unit area). Intensity may be constant ('uniform' or 'homogeneous') or may vary from location to location ('inhomogeneous'). Investigation of the intensity should be one of the first steps in analysing a point pattern.

7.1 Uniform intensity

7.1.1 Theory

If the point process $\mathbf{X}$ is homogeneous, then for any sub-region B of two-dimensional space, the expected number of points in B is proportional to the area of B:

$$\mathbb{E}[N(\mathbf{X} \cap B)] = \lambda \, \mathrm{area}(B)$$

and the constant of proportionality λ is the intensity. Intensity units are numbers per unit area (length^{-2}). If we know that a point process is homogeneous, then the empirical density of points,

$$\overline{\lambda} = \frac{n(\mathbf{x})}{\mathrm{area}(W)}$$

is an unbiased estimator of the true intensity λ.

7.1.2 Implementation in `spatstat`

To compute the estimator $\overline{\lambda}$ in `spatstat`, use `summary.ppp`:

```
> data(bei)
> summary(bei)

Planar point pattern: 3604 points
Average intensity 0.00721 points per square metre

Window: rectangle = [0, 1000] x [0, 500] metres
Window area =  5e+05 square metres
Unit of length: 1 metre
```

The estimated intensity is $\overline{\lambda} = 0.00721$ points per square metre. To extract this intensity value, type

```
> lamb <- summary(bei)$intensity
> lamb

[1] 0.007208
```

7.2 Inhomogeneous intensity

7.2.1 Theory

In general the intensity of a point process will vary from place to place. Assume that the expected number of points falling in a small region of area $\mathrm{d}u$ around a location u is equal to $\lambda(u)\,\mathrm{d}u$. Then $\lambda(u)$ is the *"intensity function"* of the process, satisfying

$$\mathbb{E}[N(\mathbf{X} \cap B)] = \int_B \lambda(u)\,\mathrm{d}u$$

for all regions B.

More generally there could be singular concentrations of intensity (e.g. earthquake epicentres may be concentrated along a fault line) so that an intensity function does not exist. Then we speak of the *"intensity measure"* Λ defined by

$$\Lambda(B) = \mathbb{E}[N(\mathbf{X} \cap B)]$$

for each $B \subset \mathbb{R}^2$, assuming the expectation is finite.

If it is suspected that the intensity may be inhomogeneous, the intensity function or intensity measure can be estimated nonparametrically by techniques such as quadrat counting and kernel smoothing.

In quadrat counting, the window W is divided into subregions ('quadrats') $B_1, \ldots, B_m$ of equal area. We count the numbers of points falling in each quadrat, $n_j = n(\mathbf{x} \cap B_j)$ for $j = 1, \ldots, m$. These are unbiased estimators of the corresponding intensity measure values $\Lambda(B_j)$.

The usual *kernel estimator* of the intensity function is

$$\widetilde{\lambda}(u) = e(u) \sum_{i=1}^{n} \kappa(u - x_i), \tag{1}$$

where $\kappa(u)$ is the kernel (an arbitrary probability density) and

$$e(u)^{-1} = \int_W \kappa(u - v)\,\mathrm{d}v \tag{2}$$

is an edge effect bias correction. Clearly $\widetilde{\lambda}(u)$ is an unbiased estimator of

$$\lambda^*(u) = e(u) \int_W \kappa(u - v)\lambda(v)\,\mathrm{d}v,$$

a smoothed version of the true intensity function $\lambda(u)$. The choice of smoothing kernel κ involves a tradeoff between bias and variance.

Intensity can also be estimated using parametric methods, as we explain in Section 11.

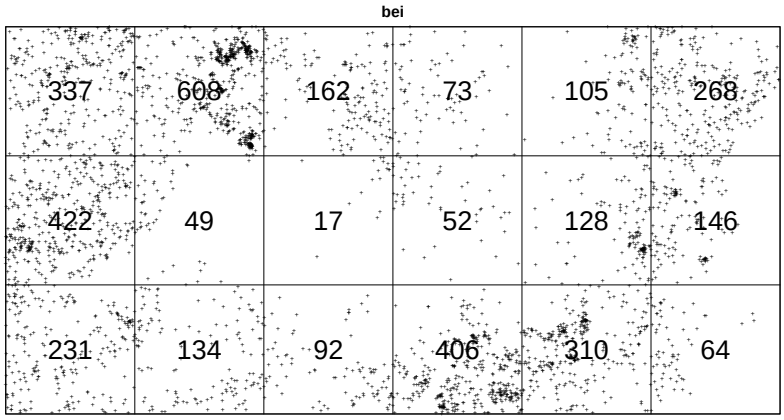
7.2.2 Implementation in `spatstat`

Quadrat counting is performed in `spatstat` by the function `quadratcount`.

```
> quadratcount(bei, nx = 4, ny = 2)
```

```
             y
x             [0,250] (250,500]
  [0,250]         544       666
  (250,500]       165       677
  (500,750]       643       130
  (750,1e+03]     298       481
```

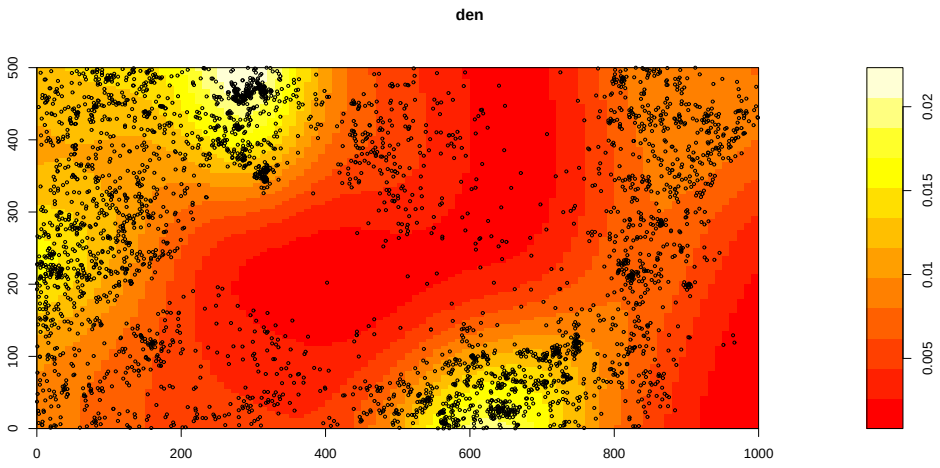
```
> Q <- quadratcount(bei, nx = 6, ny = 3)
> plot(bei, cex = 0.5, pch = "+")
> plot(Q, add = TRUE, cex = 2)
```



The value returned by `quadratcount` is an object belonging to the special class `"quadratcount"`. We have used the plot method for this class to get the display above.

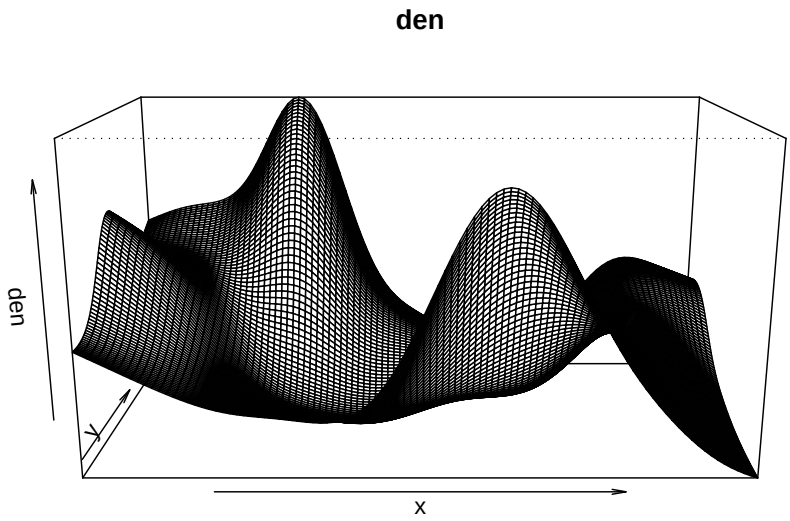
Kernel density (or *intensity*) estimation using an isotropic Gaussian kernel is implemented in `spatstat` by the function `density.ppp`, a method for the generic command `density`.

```
> den <- density(bei, sigma = 70)
> plot(den)
> plot(bei, add = TRUE, cex = 0.5)
```

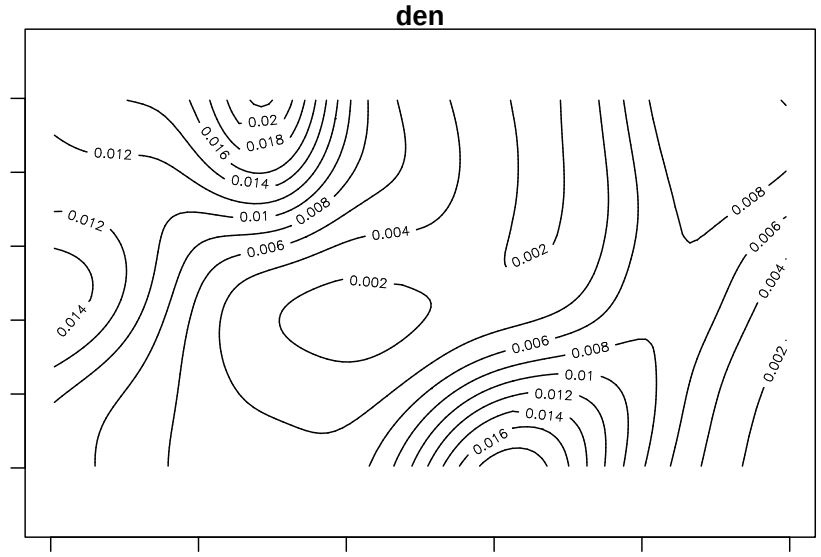


The value returned by `density.ppp` is a pixel image (object of class `"im"`). This class has methods for `print`, `summary`, `plot`, `contour` (contour plots), `persp` (perspective plots) and so on.

```
> persp(den)
```




```
> contour(den)
```

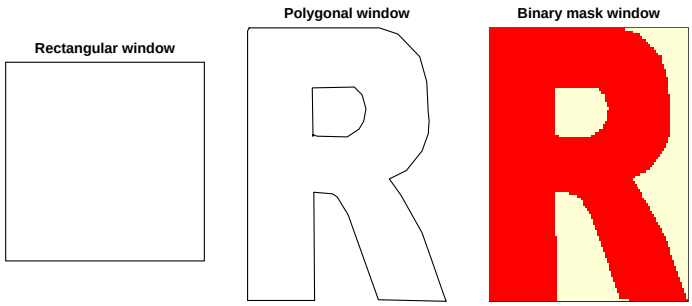


8 Defining the window

Many commands in `spatstat` require us to specify a window, study region or domain. It will be handy to know more about windows in `spatstat`.

An object of class `"owin"` ("observation window") represents a region or window in two-dimensional space. The window may be

- a rectangle;
- a polygon or polygons, with polygonal holes; or
- an irregular shape represented by a binary pixel image mask.



Objects of this class are created by the function `owin`. There are methods for printing and plotting windows, and numerous geometrical operations.

8.1 Making windows

8.1.1 Rectangular window

To create a rectangular window, type

```
> owin(xrange, yrange)
```

where `xrange, yrange` are vectors of length 2 giving the x and y dimensions, respectively, of the rectangle.

```
> owin(c(0, 3), c(1, 2))
```

```
window: rectangle = [0, 3] x [1, 2] units
```

For a square window you can also use `square`:

```
> square(5)
```

```
window: rectangle = [0, 5] x [0, 5] units
```

8.1.2 Circular window

For a circular window use `disc`:

```
> W <- disc(radius = 3, centre = c(0, 0))
```

Currently a circular window is represented as a polygon with a large number of edges.

8.1.3 Polygonal window

`Spatstat` supports polygonal windows of arbitrary shape and topology. That is, the boundary of the window may consist of one or more closed polygonal curves, which do not intersect themselves or each other. The window may have 'holes'. Type

```
> owin(poly = p)
```

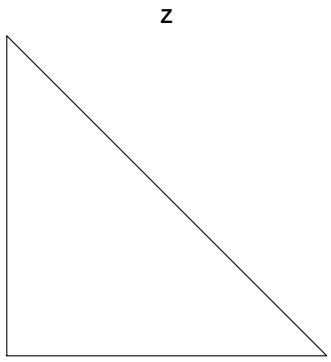
or

```
> owin(poly = p, xrange, yrange)
```

to create a polygonal window. The argument `poly=p` indicates that the window is polygonal and its boundary is given by the dataset `p`. Note we must use the "name=value" syntax to give the argument `poly`. The arguments `xrange` and `yrange` are optional here; if they are absent, the x and y dimensions of the bounding rectangle will be computed from the polygon.

If the window boundary is a single polygon, then `p` should be a list with components `x` and `y` giving the coordinates of the vertices of the window boundary, **traversed anticlockwise**. For example, the triangle with corners $(0, 0)$, $(1, 0)$ and $(0, 1)$ is created by

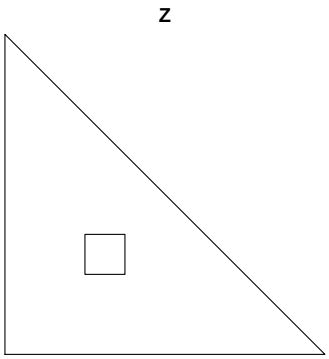
```
> Z <- owin(poly = list(x = c(0, 1, 0), y = c(0, 0, 1)))
> plot(Z)
```



Note that polygons should **not** be closed, i.e. the last vertex should **not** equal the first vertex. The same convention is used in the standard plotting function `polygon()`.

If the window boundary consists of several separate polygons, then `p` should be a list, each of whose components `p[[i]]` is a list with components `x` and `y` describing one of the polygons. The vertices of each polygon should be traversed **anticlockwise for external boundaries** and **clockwise for internal boundaries (holes)**. For example, the following creates a triangle with a square hole.

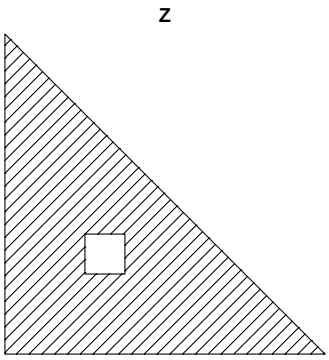
```
> Z <- owin(poly = list(list(x = c(0, 8, 0), y = c(0, 0, 8)), list(x = c(2,
+     2, 3, 3), y = c(2, 3, 3, 2))))
> plot(Z)
```



Notice that the first boundary polygon is traversed anticlockwise and the second clockwise, because it is a hole.

It is often useful to plot a polygonal window with line shading:

```
> plot(Z, hatch = TRUE)
```



8.1.4 Binary mask

A window may be defined by a discrete pixel approximation. Type

```
owin(mask=m, xrange, yrange)
```

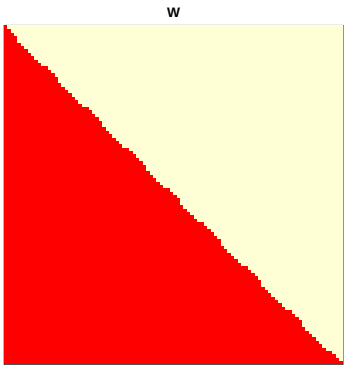
to create the window object. Here `m` should be a matrix with logical entries; it will be interpreted as a binary pixel image whose entries are `TRUE` where the corresponding pixel belongs to the window.

The rectangle with dimensions `xrange, yrange` is divided into equal rectangular pixels. The correspondence between matrix indices `m[i,j]` and cartesian coordinates is slightly idiosyncratic: the **rows** of `m` correspond to the y coordinate, and the columns to the x coordinate. The entry `m[i,j]` is `TRUE` if the point `(xx[j],yy[i])` (sic) belongs to the window, where `xx, yy` are vectors of pixel coordinates equally spaced over `xrange` and `yrange` respectively. The length of `xx` is `ncol(m)` while the length of `yy` is `nrow(m)`.

In some GIS applications the study region will be given as a binary pixel image. A safe strategy is to dump the data from the GIS system to a text file, and read the text file into R using `scan`. Then reformat it as a matrix, and use `owin` to create the window object.

To convert a rectangle or polygonal window to a binary mask, use `as.mask`.

```
> Z <- owin(poly = list(x = c(0, 1, 0), y = c(0, 0, 1)))
> W <- as.mask(Z)
> plot(W)
```



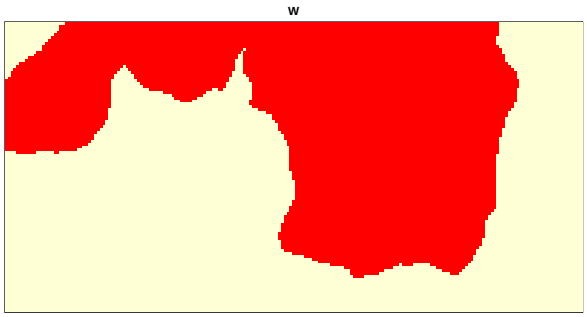
8.2 Functions that return a window

Some functions return a window object. They include

`as.owin`	Convert other data to a window object
`disc`	Create a circular window
`clickpoly`	The user draws a polygon on the screen
`bounding.box`	Bounding box of a window
`bounding.box.xy`	Bounding box of a point pattern
`convexhull.xy`	Convex hull of a point pattern
`ripras`	Ripley-Rasson estimator of window, given only the points
`trim.rectangle`	Cut off side(s) of a rectangle
`levelset`	Level set of a pixel image
`solutionset`	Solution of an equation involving pixel image(s)

For example, the dataset `bei.extra$elev` is a pixel image containing altitude (elevation) values for a study region. To find the subset where altitude exceeds 145,

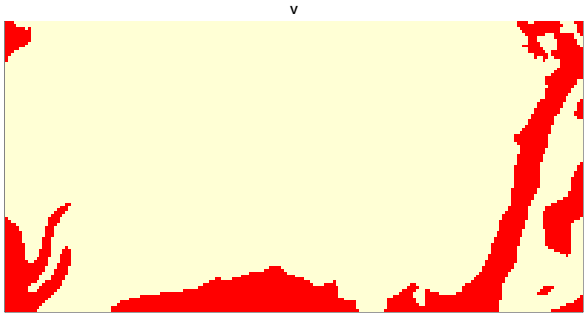
```
> elev <- bei.extra$elev
> W <- levelset(elev, 145, ">")
> plot(W)
```



The result `W` is a window.

The accompanying dataset `bei.extra$grad` is a pixel image of the slope (gradient) of the terrain. To find the subset where altitude is below 140 and slope exceeds 0.1,

```
> grad <- bei.extra$grad
> V <- solutionset(elev <= 140 & grad > 0.1)
> plot(V)
```



8.3 Operations on windows

Basic methods for the class `"owin"` include

`print.owin`	print short description of a window
`summary.owin`	print detailed summary of a window
`plot.owin`	plot a window

Numerous geometrical operations are implemented for window objects. They include:

`area.owin`	compute window's area
`diameter`	compute window's diameter
`intersect.owin`	intersection of two windows
`union.owin`	union of two windows
`bounding.box`	Find a tight bounding box for the window
`complement.owin`	swap inside and outside
`rotate`	rotate window
`shift`	translate window
`affine`	apply affine transformation
`rescale`	change scale and adjust units
`as.mask`	convert to binary image mask
`dilate.owin`	morphological dilation
`erode.owin`	morphological erosion
`eroded.areas`	compute areas of eroded windows
`inside.owin`	determine whether a point is inside a window
`distmap.owin`	distance transform image
`centroid.owin`	compute centroid (centre of mass) of window
`is.subset.owin`	determine whether one window contains another

8.4 Creating a point pattern in any window

As we saw in Section 6.1, the function `ppp()` will create a point pattern (an object of class `"ppp"`) from raw numerical data in R.

Suppose the x, y coordinates of the points of the pattern are contained in vectors `x` and `y` of equal length. Then

```
ppp(x, y, other.arguments)
```

will create the point pattern. The 'other arguments' must determine a window for the pattern, in one of two ways:

- the other arguments can be passed to `owin` to determine a window:

`ppp(x, y, xrange, yrange)`	point pattern in rectangle
`ppp(x, y, poly=p)`	point pattern in polygonal window
`ppp(x, y, poly=p, xrange, yrange)`	point pattern in polygonal window
`ppp(x, y, mask=m, xrange, yrange)`	point pattern in binary mask window

- if `W` is a window object (class `"owin"`) then

  ```
  > ppp(x, y, window = W)
  ```

  will create the point pattern.

You may already have a window `W` (an object of class `"owin"`) ready to hand, and now want to create a pattern of points in this window. For example you may want to put a new point pattern inside the window of an existing point pattern `X`; the window is accessed as `X$window`, so type

```
ppp(x, y, window=X$window)
```

9 Manipulating point patterns

Before proceeding, we need to know more about how to manipulate and interrogate point pattern data.

9.1 Format of `ppp` objects

A point pattern is represented in `spatstat` by an object of the class `"ppp"`. This contains the coordinates of the points, optional 'mark' values attached to the points, and a description of the study region or spatial 'window'.

9.1.1 Format

A point pattern object `P` has the following components:

- `P$n` is the number of points (which may be zero).
- `P$x` is a numeric vector containing the x coordinates of the points. Its length equals `P$n` (and may be zero).
- `P$y` is a numeric vector containing the y coordinates of the points. Its length also equals `P$n`.
- `P$marks` contains the marks. It is either `NULL`, or a vector of length `P$n` containing the mark values. The entries of `P$marks` may be of any atomic type (character, numeric, logical, complex).
- `P$window` is an object of class `"owin"` ("observation window") determining the study region or spatial 'window'.

You can extract these components individually; for example, to make a histogram of the x coordinates just type `hist(P$x)`. However, **do not assign values to these components directly**, or you may create inconsistencies in the data which cause `spatstat` to crash. To manipulate point patterns, use the functions provided.

Although a point pattern should be treated as an unordered set, the coordinates are obviously stored in a particular order, and can be addressed using that order.

```
> data(longleaf)
> x <- longleaf$x
> y <- longleaf$y
> diameter <- longleaf$marks
> cbind(x, y, diameter)[1:5, ]

         x    y diameter
[1,] 200.0  8.8     32.9
[2,] 199.3 10.0     53.5
[3,] 193.6 22.4     68.0
[4,] 167.7 35.6     17.7
[5,] 183.9 45.4     36.9
```

If the marks are a categorical variable, then `P$marks` is a `factor`.

```
> data(chorley)
> x <- chorley$x
> y <- chorley$y
> type <- chorley$marks
> data.frame(x, y, type)[55:60, ]

       x     y   type
55 355.6 413.9 larynx
56 355.5 413.9 larynx
57 355.7 413.9 larynx
58 355.6 414.1 larynx
59 359.0 417.3   lung
60 353.1 426.9   lung

> is.factor(type)

[1] TRUE

> levels(type)

[1] "larynx" "lung"

> table(type)

type
larynx   lung
    58    978
```

9.1.2 A point pattern needs a window

Note especially that, when you create a new point pattern object, you need to specify the spatial region or window in which the pattern was observed. In `spatstat`, the observation window is an integral part of the point pattern. A point pattern dataset consists of knowledge about where points were *not* observed, as well as the locations where they *were* observed. Even something as simple as estimating the intensity of the pattern depends on the window of observation. It would be wrong, or at least different, to analyze a point pattern dataset by "guessing" the appropriate window (e.g. by computing the convex hull of the points). An analogy may be drawn with the difference between sequential experiments and experiments in which the sample size is fixed *a priori*.

Often, the window of observation is a rectangle, so this requirement just means that we have to specify the x and y dimensions of the rectangle when we create the point pattern. Windows with a more complicated shape can easily be represented in `spatstat`, as described below.

For situations where the window is really unknown, `spatstat` provides the function `ripras` to compute the Ripley-Rasson estimator of the window, given only the point locations.

9.2 Operations on ppp objects

Directly manipulating the entries inside an object is not safe. It is also unnecessary, because these manipulations can be performed using functions or operators.

For point patterns (objects of class `"ppp"`) there are the following operations.

9.2.1 Extracting subsets

Recall that in R the subset operator is `[ ]`. If `x` is a vector of numbers, then `x[s]` extracts an element or subset of `x`. The subset index `s` can be

- a positive integer: `x[3]` means the third element of `x`;
- a vector of positive integers indicating which elements to extract: `x[c(2,4,6)]` extracts the 2nd, 4th and 6th elements of `x`;
- a vector of negative integers indicating which elements *not* to extract: `x[-1]` means all elements of `x` except the first one;
- a vector of logical values, of the same length as `x`, with each `TRUE` entry of `s` indicating that the corresponding entry of `x` should be extracted, and `FALSE` indicating that it should not be extracted. For example `x[x > 3.1]` extracts those elements of `x` which are greater than 3.1.

To extract a subset of a point pattern in `spatstat`, we also use the subset operator `[ ]`. If `X` is a point pattern then `X[s]` is also a point pattern, consisting of those points of `X` selected by the subset index `s`, where `s` can be any of the three types listed above, (Recall that the points in a point pattern object are stored in a particular order; this is the order in which they are indexed by `s`.)

```
> data(bei)
> bei

 planar point pattern: 3604 points
window: rectangle = [0, 1000] x [0, 500] metres
```

```
> bei[1:10]

 planar point pattern: 10 points
window: rectangle = [0, 1000] x [0, 500] metres
```

It is also possible to extract the subset defined by a spatial region. If `X` is a point pattern and `W` is a spatial window (object of class `"owin"`) then `X[W]` is the point pattern consisting of all points of `X` that lie inside `W`.

```
> W <- owin(c(100, 800), c(100, 400))
> W

window: rectangle = [100, 800] x [100, 400] units

> bei[W]

 planar point pattern: 918 points
window: rectangle = [100, 800] x [100, 400] units
```

> Tip: You may need to put quotes around the subset operator in some contexts. The generic subset operator is `[` but the help file is summoned by typing `help("[")`. The subset method for point patterns is called `[.ppp` but the help file is summoned by typing `help("[.ppp")`.

9.2.2 Fiddling with marks

To extract the marks from a point pattern, use `marks`:

```
> m <- marks(X)
```

To add or change marks, use `marks<-`

```
> marks(X) <- whatever
```

To delete marks from a point pattern, assign the marks to `NULL`:

```
> marks(X) <- NULL
```

For convenience, you can also perform these operations inside an expression, using the function `unmark` to remove marks and the binary operator `%mark%` to add marks:

```
> data(redwood)
> radii <- rexp(redwood$n, rate = 10)
> X <- redwood %mark% radii
> X

marked planar point pattern: 62 points
marks are numeric, of type 'double'
window: rectangle = [0, 1] x [-1, 0] units

> unmark(X)
```

```
 planar point pattern: 62 points
window: rectangle = [0, 1] x [-1, 0] units
```

For a point pattern with real-valued marks, the method `cut.ppp` for the generic function `cut` will divide the range of mark values into several discrete bands, yielding a point pattern with categorical marks:

```
> Y <- cut(X, 3)
> Y <- cut(X, breaks = c(0, 1, 10, Inf))
> Y

marked planar point pattern: 62 points
multitype, with levels = (0,1]        (1,10]        (10,Inf]
window: rectangle = [0, 1] x [-1, 0] units
```

9.2.3 Combining point patterns

Any number of point patterns can be combined to make a single pattern, using `superimpose`.

```
> X <- runifpoint(20)
> Y <- runifpoint(10)
> superimpose(X, Y)

 planar point pattern: 30 points
window: rectangle = [0, 1] x [0, 1] units
```

The argument `W`, if given, specifies the window for the combined point pattern.

```
> superimpose(X, Y, W = square(2))

 planar point pattern: 30 points
window: rectangle = [0, 2] x [0, 2] units
```

To attach a separate mark to each component pattern, use argument names:

```
> superimpose(Hooray = X, Boo = Y)

marked planar point pattern: 30 points
multitype, with levels = Hooray        Boo
window: rectangle = [0, 1] x [0, 1] units
```

9.2.4 Geometrical transformations

The commands `rotate`, `shift` and `affine` apply two-dimensional rotation, vector shifts, and affine transformations, respectively.

9.2.5 Changing scales and units

A scalar dilation can be applied using `affine`. For example, the Swedish Pines data were recorded in decimetres. To convert the coordinates to metres, we could type

```
> data(swedishpines)
> X <- affine(swedishpines, mat = diag(c(1/10, 1/10)))
> unitname(X) <- c("metre", "metres")
> X

 planar point pattern: 71 points
window: rectangle = [0, 9.6] x [0, 10] metres
```

The command `rescale` performs the same function:

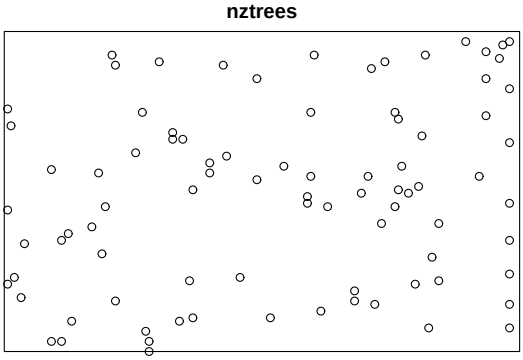
```
> data(swedishpines)
> X <- rescale(swedishpines, 10)
> X

 planar point pattern: 71 points
window: rectangle = [0, 9.6] x [0, 10] metres
```

Beware that this does not change the marks in the point pattern. If your marks represent tree diameter and you want to rescale them as well, this must be done by hand.

9.3 Example

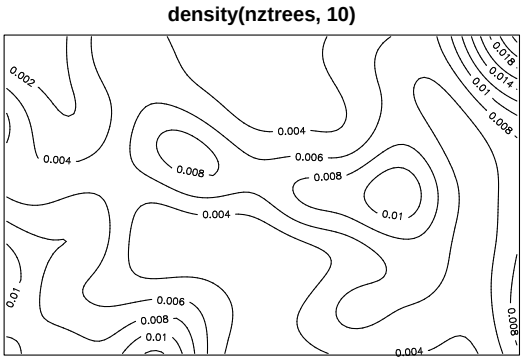
We will use one of the standard point pattern datasets that is installed with the package. The NZ trees dataset represent the positions of 86 trees in a forest plot 153 by 95 feet.

```
> data(nztrees)
> nztrees

 planar point pattern: 86 points
window: rectangle = [0, 153] x [0, 95] feet

> plot(nztrees)
```



To get an impression of local spatial variations in intensity, we plot a kernel density estimate of intensity.

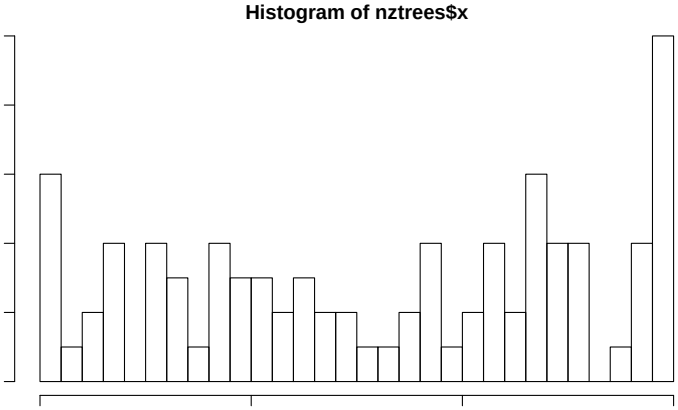
```
> contour(density(nztrees, 10), axes = FALSE)
```



The density surface has a steep slope at the top right-hand corner of the study region. Looking at the plot of the point pattern itself, we can see a cluster of trees at the top right.

You may also notice a line of trees at the right-hand edge of the study region. It looks as though the study region may have included some trees that were planted as a boundary or avenue. This sticks out like a sore thumb if we plot the x coordinates of the trees:

```
> hist(nztrees$x, nclass = 25)
```



We might want to exclude the right-hand boundary from the study region, to focus on the pattern of the remaining trees. Let's say we decide to trim a 5-foot margin off the right-hand side.

First we create the new, trimmed study region:

```
> chopped <- owin(c(0, 148), c(0, 95))
```

or more slickly,

```
> win <- nztrees$window
> chopped <- trim.rectangle(win, xmargin = c(0, 5), ymargin = 0)
> chopped

window: rectangle = [0, 148] x [0, 95] feet
```

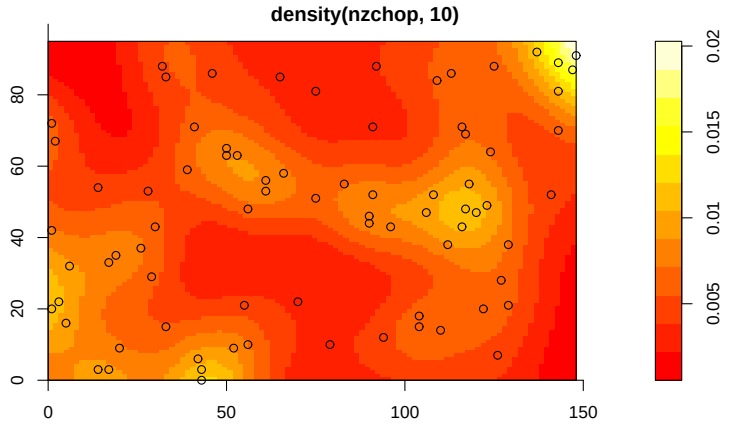
(Notice that `chopped` is not a point pattern, but simply a rectangle in the plane.)

Then, using the subset operator `[.ppp`, we simply extract the subset of the original point pattern that lies inside the new window:

```
> nzchop <- nztrees[chopped]
```

We can now study the 'chopped' point pattern:

```
> summary(nzchop)

Planar point pattern: 78 points
Average intensity 0.00555 points per square foot

Window: rectangle = [0, 148] x [0, 95] feet
Window area =  14060 square feet
Unit of length: 1 foot

> plot(density(nzchop, 10))
> plot(nzchop, add = TRUE)
```



Removing the right margin seems to have produced a much more uniform pattern.

10 Methods 2: Tests of Complete Spatial Randomness

The basic 'reference' or 'benchmark' model of a point process is the *uniform Poisson point process* in the plane with intensity λ, sometimes called *Complete Spatial Randomness (CSR)*. Its basic properties are

- the number of points falling in any region A has a Poisson distribution with mean $\lambda \text{area}(A)$
- given that there are n points inside region A, the locations of these points are i.i.d. and uniformly distributed inside A
- the contents of two disjoint regions A and B are independent.

The uniform Poisson process is often the 'null model' in an analysis. For historical reasons, many applied writers focus on establishing that their data do not conform to a uniform Poisson process.

10.1 Definition

The *homogeneous Poisson process* of intensity $\lambda > 0$ has the properties

(PP1): the number $N(\mathbf{X} \cap B)$ of points falling in any region B is a Poisson random variable;

(PP2): the expected number of points falling in B is $\mathbb{E}[N(\mathbf{X} \cap B)] = \lambda \cdot \mathsf{area}(B)$;

(PP3): if B_1, B_2 are disjoint sets then $N(\mathbf{X} \cap B_1)$ and $N(\mathbf{X} \cap B_2)$ are independent random variables;

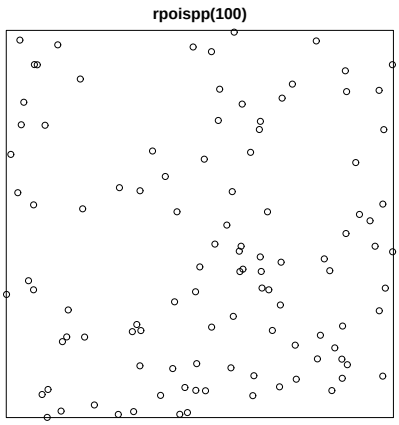
(PP4): given that $N(\mathbf{X} \cap B) = n$, the n points are independent and uniformly distributed in B.

The list is redundant; (PP2) and (PP3) are sufficient.

This process is often called *"Complete Spatial Randomness" (CSR)* especially in biological science. Under CSR, points are independent of each other and have the same propensity to be found at any location.

It is easy to simulate the Poisson process directly by following the properties (PP1)–(PP4). In `spatstat`, use the command `rpoispp` (by convention, random data generators have names beginning with `r`).

```
> plot(rpoispp(100))
```

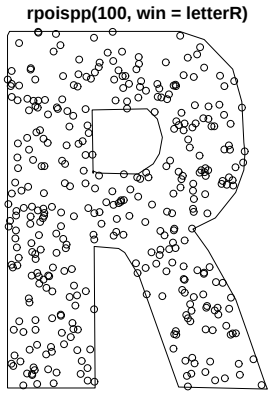


Conceptually, if we discretise a homogeneous Poisson process into infinitesimal pixels, the indicators I are independent and identically distributed, with success probability $\mathbb{P}\{I = 1\} = \lambda \, \mathrm{d}A$ where $\mathrm{d}A$ is the infinitesimal area of a pixel.

To develop some intuition about completely random patterns, it's useful to repeat the command `plot(rpoispp(100))` several times (use the up-arrow key to recall the previous command line) so that you see several replicates of the Poisson process. In particular you will notice that the points in a homogeneous Poisson process are not 'uniformly spread': there are empty gaps and clusters of points.

The command `rpoispp` has arguments `lambda` (the intensity) and `win` (the window in which to simulate). The default window is the unit square.

```
> data(letterR)
> plot(rpoispp(100, win = letterR))
```



If you want to simulate a Poisson process *conditionally* on a fixed number of points, use the command `runifpoint`.

```
> runifpoint(100)
```

```
 planar point pattern: 100 points
window: rectangle = [0, 1] x [0, 1] units
```

10.2 Quadrat counting tests for CSR

In classical literature, the homogeneous Poisson process (CSR) is usually taken as the appropriate 'null' model for a point pattern. Our basic task in analysing a point pattern is to find evidence against CSR.

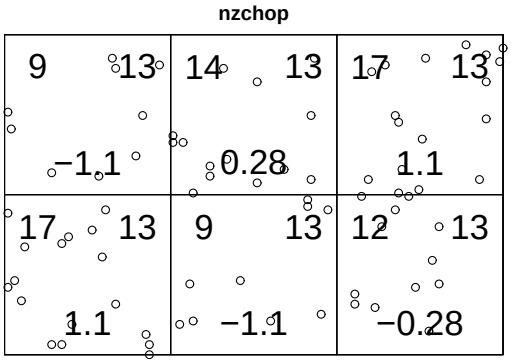
A classical test for the null hypothesis of CSR is the χ^2 test based on quadrat counts. As explained earlier, the window W is divided into subregions ('quadrats') $B_1, \ldots, B_m$ of equal area. We count the numbers of points falling in each quadrat, $n_j = n(\mathbf{x} \cap B_j)$ for $j = 1, \ldots, m$. Under the null hypothesis of CSR, the n_j are i.i.d. Poisson random variables with the same expected value. The Pearson χ^2 goodness-of-fit test can be used.

```
> quadrat.test(nzchop, nx = 3, ny = 2)

        Chi-squared test of CSR using quadrat counts

data:  nzchop
X-squared = 5.0769, df = 5, p-value = 0.4066
```

The value returned by `quadrat.test` is an object of class `"htest"` (the standard R class for hypothesis tests). Printing the object (as shown above) gives comprehensible output about the outcome of the test. Inspecting the p-value, we see that the test does not reject the null hypothesis of CSR for the (chopped) New Zealand trees data.

The return value `quadrat.test` also belongs to the special class `"quadrat.test"`. Plotting the object will display the quadrats, annotated by their observed and expected counts and the Pearson residuals (observed counts n_j at top left; expected count at top right; Pearson residuals at bottom).

```
> M <- quadrat.test(nzchop, nx = 3, ny = 2)
> M

        Chi-squared test of CSR using quadrat counts

data:  nzchop
X-squared = 5.0769, df = 5, p-value = 0.4066

> plot(nzchop)
> plot(M, add = TRUE, cex = 2)
```



The p-value can also be extracted by

```
> M$p.value

[1] 0.4065648
```

10.3 Critique

Since this kind of technique is often used in the applied literature, a few comments are appropriate.

The main critique of the quadrat test approach is the lack of information. This is a goodness-of-fit test in which the alternative hypothesis H_1 is simply the negation of H_0, that is, the alternative is that "the process is not a homogeneous Poisson process". A point process may fail to satisfy properties (PP1)–(PP4) either because it violates (PP2) by having non-uniform intensity, or because it violates (PP3)–(PP4) by exhibiting dependence between points. There are too many types of departure from H_0.

The usual justification for the classical χ^2 goodness-of-fit test is to assume that the counts are independent, and derive a test of the null hypothesis that all counts have the same expected value. Invoking it here is slightly naive, since the independence of counts is also open to question here.

Indeed we can also turn things around and view the χ^2 test as a test of the Poisson distributional properties (PP2)–(PP3) assuming that the intensity is uniform. The Pearson χ^2 test statistic

$$X^2 = \frac{\sum_j (n_j - n/m)^2}{n/m}$$

(where $n = \sum_j n_j$ is the total number of points) coincides, up to a constant factor, with the sample variance-to-mean ratio of the counts n_j, which is often interpreted as a measure of over/under-dispersion of the counts n_j assuming they have constant mean.

The power of the quadrat test depends on the size of quadrats, and falls to zero for quadrats which are either very large or very small. The power also depends on the alternative hypothesis, in particular on the 'spatial scale' of any departures from the assumptions of constant intensity and independence of points. The choice of quadrat size carries an implicit assumption about the spatial scale.

10.4 Kolmogorov-Smirnov test of CSR

Typically a more powerful test of CSR is the Kolmogorov-Smirnov test in which we compare the observed and expected distributions of the values of some function T.

We specify a real-valued function $T(x, y)$ defined at all locations (x, y) in the window. We evaluate this function at each of the data points. Then we compare this empirical distribution of values of T with the predicted distribution of values of T under CSR, using the classical Kolmogorov-Smirnov test.

In `spatstat` the spatial Kolmogorov-Smirnov test is performed by `kstest`. This function is generic. The method for point patterns, `kstest.ppp`, performs the Kolmogorov-Smirnov test for CSR.

If `X` is the data point pattern, then

```
> kstest(X, fun)
```

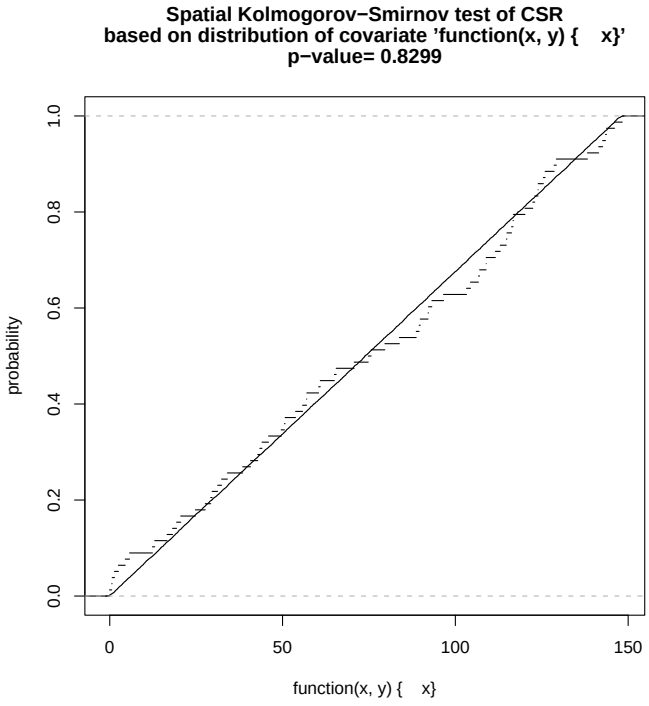
performs the test, where `fun` is a `function(x,y)` in the R language.

For example, let's consider the `nzchop` data and choose the function T to be the x coordinate, $T(x, y) = x$. This means we are simply comparing the observed and expected distributions of the x coordinate.

```
> kstest(nzchop, function(x, y) {
+     x
+ })

        Spatial Kolmogorov-Smirnov test of CSR

data:  covariate 'function(x, y) {    x}' evaluated at points of 'nzchop'
         and transformed to uniform distribution under CSR
D = 0.0741, p-value = 0.7566
alternative hypothesis: two-sided
```

The result of `kstest` is an object of class `"htest"` (the standard R class for hypothesis tests) and also of class `"kstest"` so that it can be printed and plotted. The print method (demonstrated above) reports information about the hypothesis test such as the p-value. The plot method displays the observed and expected distribution functions.

```
> KS <- kstest(nzchop, function(x, y) {
+     x
+ })
> plot(KS)
> pval <- KS$p.value
```



Sometimes this test generates a warning message about tied values. Typically this occurs because the coordinates in the dataset have been rounded to the nearest integer, so that there are tied observations.

11 Methods 3: Maximum likelihood for Poisson processes

If we are willing to assume (tentatively) that the points are independent, then we can apply some decent statistical methods to the investigation of the intensity.

11.1 Inhomogeneous Poisson process

The *inhomogeneous* Poisson process with intensity function $\lambda(u)$, $u \in \mathbb{R}^2$, is a modification of the homogeneous Poisson process, in which properties (PP2) and (PP4) above are replaced by

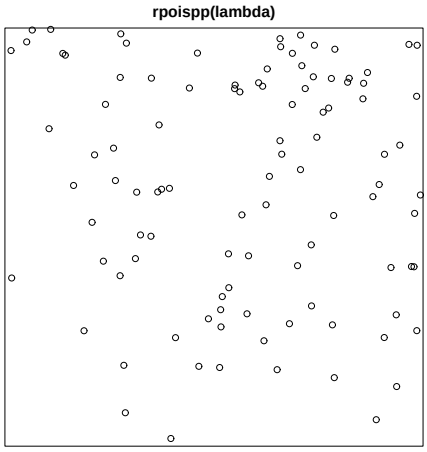
(PP2'): the number $N(\mathbf{X} \cap B)$ of points falling in a region B has expectation

$$\mathbb{E}[N(\mathbf{X} \cap B) = \int_B \lambda(u)\, \mathrm{d}u.$$

(PP4'): given that $N(\mathbf{X} \cap B) = n$, the n points are independent and identically distributed, with common probability density $f(u) = \lambda(u)/I$, where $I = \int_B \lambda(u)\, \mathrm{d}u$.

This process can also be simulated using `rpoispp` using the same properties. The intensity argument `lambda` can be a constant, a `function(x,y)` giving the values of the intensity function at coordinates x, y, or a pixel image containing the intensity values at a grid of locations.

```
> lambda <- function(x, y) {
+     100 * (x + y)
+ }
> plot(rpoispp(lambda))
```



If we discretise an inhomogeneous Poisson process, the indicators I are independent, but have unequal success probabilities, $\mathbb{P}\{I(u) = 1\} = \lambda(u)\, \mathrm{d}A$.

The inhomogeneous Poisson process is a plausible model for point patterns under several scenarios. One is **random thinning**: suppose that a homogeneous Poisson process of intensity β is generated, and that each point is either deleted or retained, independently of other points. Suppose the probability of retaining a point at the location u is $p(u)$. Then the resulting process of retained points is inhomogeneous Poisson, with intensity $\lambda(u) = \beta p(u)$.

Consider, for example, a model of plant propagation which assumes that seeds are randomly dispersed according to a Poisson process, and seeds randomly germinate or do not germinate, independently of each other, with a germination probability that depends on the local soil conditions. The resulting pattern of plants is an inhomogeneous Poisson process.

11.2 Likelihood methods

The log-likelihood for the homogeneous Poisson process with intensity λ is

$$\log L(\lambda; \mathbf{x}) = n(\mathbf{x}) \log \lambda - \lambda \operatorname{area}(W) \tag{3}$$

where $n(\mathbf{x})$ is the number of points in the dataset $\mathbf{x}$. The maximum likelihood estimator of λ is

$$\widehat{\lambda} = \frac{n(\mathbf{x})}{\operatorname{area}(W)}$$

which is also an unbiased estimator. The variance of $\widehat{\lambda}$ is $\mathsf{var}[\widehat{\lambda}] = \lambda/\operatorname{area}(W)$.

Consider an inhomogeneous Poisson process with intensity function $\lambda_\theta(u)$ depending on a parameter θ. The log-likelihood for θ is

$$\log L(\theta; \mathbf{x}) = \sum_{i=1}^{n} \log \lambda_\theta(x_i) - \int_W \lambda_\theta(u)\, \mathrm{d}u \tag{4}$$

This is a well-behaved likelihood; for example if $\log \lambda_\theta(u)$ is linear in θ, then the log-likelihood is concave, so there is a unique MLE. However, the MLE $\widehat{\theta}$ is not analytically tractable, so it must be computed using numerical algorithms such as Newton's method.

The usual asymptotic theory of maximum likelihood applies: under suitable large sample conditions, the MLE of θ is asymptotically normal. If we wish to test CSR, the likelihood ratio test statistic

$$R = 2 \log \frac{L(\widehat{\theta})}{L(\widehat{\lambda})}$$

is asymptotically χ^2 under CSR, and this gives an asymptotically optimal test of CSR against the alternative of an inhomogeneous Poisson process with intensity $\lambda_\theta(u)$.

11.3 Fitting Poisson processes in `spatstat`

Mark Berman and Rolf Turner [13] (see also [30, 16, 31]) developed a clever computational device for finding the MLE of θ by exploiting a formal similarity between the Poisson log-likelihood (4) and that of a loglinear Poisson regression.

The Berman-Turner algorithm is implemented in `spatstat`. The intensity function $\lambda_\theta(u)$ must be loglinear in the parameter θ:

$$\log \lambda_\theta(u) = \theta \cdot S(u) \tag{5}$$

where $S(u)$ is a real-valued or vector-valued function of location u. The form of S is arbitrary so this is not much of a restriction. In practice $S(u)$ could be a function of the spatial coordinates of u, or an observed covariate, or a mixture of both. Assuming (5), the log-likelihood (4) is a convex function of θ, so maximum likelihood is well-behaved.

11.3.1 Model-fitting function

The fitting function is called `ppm` ('point process model') and is very closely analogous to the model fitting functions in R such as `lm` and `glm`. The statistic $S(u)$ is specified by an R language formula, like the formulas used to specify the systematic relationship in a linear model or generalised linear model. The basic syntax is:

```
> ppm(X, ~trend)
```

where `X` is the point pattern dataset, and `~trend` is an R formula with no left-hand side. This should be viewed as a model with log link, so *the formula* `~trend` *specifies the form of the* **logarithm** *of the intensity function.*

To fit the homogeneous Poisson model:

```
> ppm(bei, ~1)

Stationary Poisson process

Uniform intensity:         0.007208
```

To fit an inhomogeneous Poisson model with an intensity that is log-linear in the cartesian coordinates, i.e. $\lambda_\theta((x,y)) = \exp(\theta_0 + \theta_1 x + \theta_2 y)$,

```
> ppm(bei, ~x + y)

Nonstationary Poisson process

Trend formula: ~x + y

Fitted coefficients for trend formula:
  (Intercept)             x             y
-4.7245290274 -0.0008031288  0.0006496090
```

Here `x` and `y` are reserved names that always refer to the cartesian coordinates. In the output, the 'fitted coefficients' are the maximum likelihood estimates of $\theta_0, \theta_1, \theta_2$, the coefficients of the 'linear predictor'. The fitted intensity function is

$$\lambda_\theta((x,y)) = \exp\left(-4.724529 + -0.000803\, x + 0.00065\, y\right).$$

To fit an inhomogeneous Poisson model with an intensity that is log-quadratic in the cartesian coordinates, i.e. such that $\log \lambda_\theta((x,y))$ is a quadratic in x and y:

```
> ppm(bei, ~polynom(x, y, 2))

Nonstationary Poisson process

Trend formula: ~polynom(x, y, 2)

Fitted coefficients for trend formula:
          (Intercept)   polynom(x, y, 2)[x]   polynom(x, y, 2)[y]
        -4.275762e+00         -1.609187e-03         -4.895166e-03
polynom(x, y, 2)[x^2] polynom(x, y, 2)[x.y] polynom(x, y, 2)[y^2]
         1.625968e-06         -2.836387e-06          1.331331e-05
```

Essentially any kind of model formula can be used, involving the reserved names `x` and `y` and any covariates (as we explain later).

To fit a model with constant but unequal intensities on each side of the vertical line $x = 500$, the explanatory variable $S(u)$ should be a factor with two levels, `Left` and `Right` say, taking the value `Left` when the location u is to the left of the line $x = 500$.

```
> side <- function(z) factor(ifelse(z < 500, "left", "right"))
> ppm(bei, ~side(x))
```

```
Nonstationary Poisson process

Trend formula: ~side(x)

Fitted coefficients for trend formula:
 (Intercept) side(x)right
  -4.8026460   -0.2792705
```

When factors are involved, the interpretation of the coefficients depends on which 'contrasts' are in force. By default the 'treatment contrasts' are assumed. This means that the treatment effect is taken to be zero for the first level of the factor, and the estimated treatment effects for other levels are effectively estimates of the difference from the first level. In this case `"left"` comes alphabetically before `"right"`, so by default, the first level is `"left"`. The fitted model is

$$\lambda_\theta((x,y)) = \begin{cases} \exp(-4.8026) & \text{if } x < 500 \\ \exp(-4.8026 + (-0.2793)) & \text{if } x \geq 500 \end{cases}$$

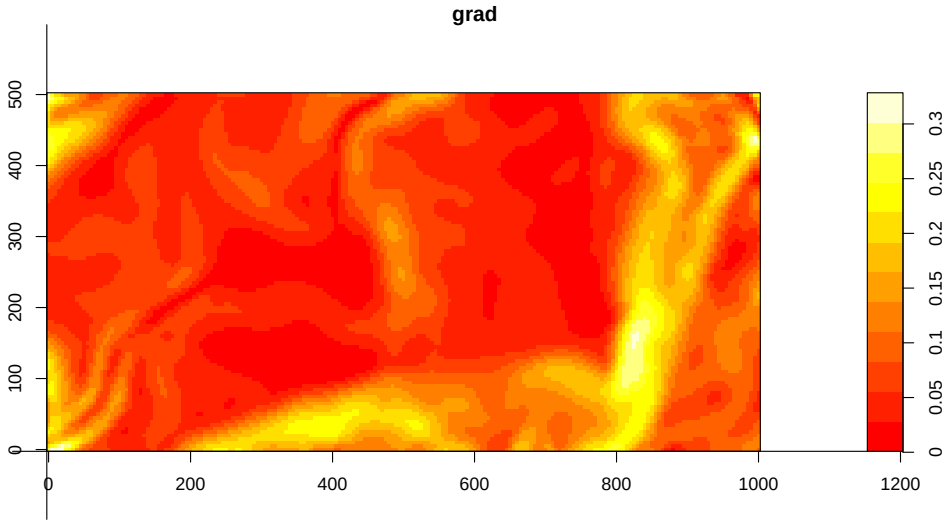
Rather than relying on such interpretations, it is prudent to use the command `predict` to compute predicted values of the model, as explained in Section 11.4 below.

11.3.2 Models involving spatial covariates

It is also possible to fit an inhomogeneous Poisson process model with an intensity function that depends on an observed covariate. Let $Z(u)$ be a covariate that has been measured at every location u in the study window. Then $Z(u)$, or any transformation of it, can serve as the statistic $S(u)$ in the parametric form (5) for the intensity function.

The point pattern dataset `bei` is supplied with accompanying covariate data `bei.extra`. The covariates are the elevation (altitude) and the slope of the terrain at each location in the window, given as two pixel images `bei.extra$elev` and `bei.extra$grad`.

```
> data(bei)
> grad <- bei.extra$grad
> plot(grad)
```



To fit the inhomogeneous Poisson model with intensity which is a loglinear function of slope, i.e.

$$\lambda(u) = \exp(\beta_0 + \beta_1 Z(u)) \tag{6}$$

where β_0, β_1 are parameters and $Z(u)$ is the slope at location u, we type

```
> ppm(bei, ~slope, covariates = list(slope = grad))

Nonstationary Poisson process

Trend formula: ~slope

Fitted coefficients for trend formula:
(Intercept)       slope
  -5.390553    5.022021
```

In the call to `ppm`, the argument `covariates` should be a list of name=value pairs. The names should match the variables appearing in the model formula. The values should be pixel images.

The printout includes the fitted coefficients β_0, β_1 so the fitted model is

$$\lambda(u) = \exp(-5.390553 + 5.022021\, Z(u)). \tag{7}$$

It might be more appropriate to fit the inhomogeneous Poisson model with intensity that is *proportional* to slope,

$$\lambda(u) = \beta Z(u) \tag{8}$$

where again $Z(u)$ is the slope at u. Equivalently

$$\log \lambda(u) = \log \beta + \log Z(u). \tag{9}$$

There is no coefficient in front of the term $\log Z(u)$ in (9), so this term is an 'offset'. To fit this model,

```
> ppm(bei, ~offset(log(slope)), covariates = list(slope = grad))
```

```
Nonstationary Poisson process

Trend formula: ~offset(log(slope))

Fitted coefficients for trend formula:
(Intercept)
  -2.427127
```

The fitted coefficient is the constant $\log \beta$ appearing in (9), so converting back to the form (8), the fitted model is

$$\lambda(u) = e^{-2.427127}\, Z(u) = 0.0883\, Z(u).$$

11.4 Fitted models

The value returned by the model-fitting function `ppm` is an object of class `"ppm"` that represents the fitted model. This is analogous to the fitting of linear models (`lm`), generalised linear models (`glm`) and so on.

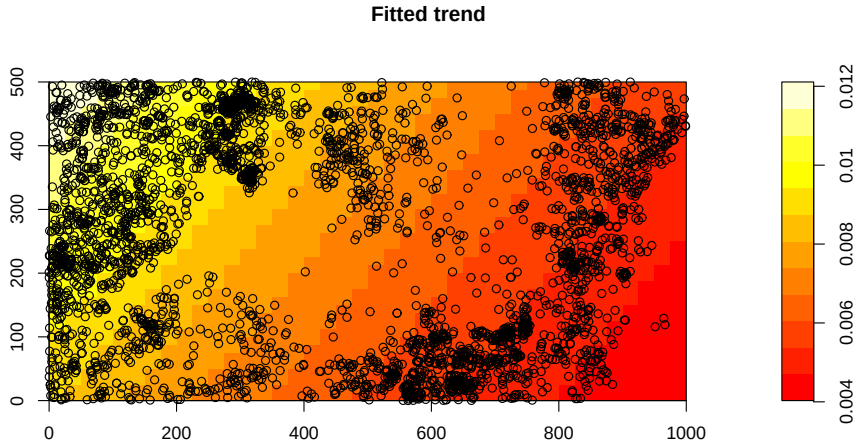
11.4.1 Standard operations

The following standard operations on fitted models in R can be applied to point process models (i.e. these operations have methods for the class `"ppm"`):

`print`	print basic information
`summary`	print detailed summary information
`plot`	plot the fitted intensity
`predict`	compute the fitted intensity
`fitted`	compute the fitted intensity at data points
`update`	re-fit the model
`coef`	extract the fitted coefficient vector $\widehat{\theta}$
`vcov`	variance-covariance matrix of $\widehat{\theta}$
`anova`	analysis of deviance
`logLik`	log-likelihood value

For information on these methods, see `print.ppm`, `summary.ppm`, `plot.ppm` etc.

```
> fit <- ppm(bei, ~x + y)
> fit

Nonstationary Poisson process

Trend formula: ~x + y

Fitted coefficients for trend formula:
  (Intercept)             x             y
-4.7245290274 -0.0008031288  0.0006496090

> plot(fit, how = "image")
```




```
> predict(fit, type = "trend")

real-valued pixel image
50 x 50 pixel array (ny, nx)
enclosing rectangle: [0, 1000] x [0, 500] metres

> predict(fit, type = "cif", ngrid = 256)

real-valued pixel image
256 x 256 pixel array (ny, nx)
enclosing rectangle: [0, 1000] x [0, 500] metres

> coef(fit)

  (Intercept)             x             y
-4.7245290274 -0.0008031288  0.0006496090

> vcov(fit)

              (Intercept)             x             y
(Intercept)  1.854091e-03 -1.491267e-06 -3.528289e-06
x           -1.491267e-06  3.437842e-09  1.208410e-14
y           -3.528289e-06  1.208410e-14  1.338955e-08

> sqrt(diag(vcov(fit)))

 (Intercept)            x            y
4.305915e-02 5.863311e-05 1.157132e-04

> round(vcov(fit, what = "corr"), 2)

            (Intercept)     x     y
(Intercept)        1.00 -0.59 -0.71
x                 -0.59  1.00  0.00
y                 -0.71  0.00  1.00
```

This is the fitted model with intensity function

$$\lambda_\theta((x,y)) = \exp(\theta_0 + \theta_1 x + \theta_2 y) \tag{10}$$

with the following estimates:

i	θ_i	$\mathsf{var}(\widehat{\theta_i})$	standard deviation
0	-4.724529	0.001854091	0.04305915
1	-0.0008031288	3.437842e-09	5.863311e-05
2	0.000649609	1.338955e-08	0.0001157132

11.4.2 Model selection

Analysis of deviance for nested Poisson point process models is implemented in `spatstat` as `anova.ppm`. The first model should be a sub-model of the second.

```
> fit <- ppm(bei, ~slope, covariates = list(slope = grad))
> fitnull <- update(fit, ~1)
> anova(fitnull, fit, test = "Chi")

Analysis of Deviance Table

Model 1: .mpl.Y ~ 1
Model 2: .mpl.Y ~ slope
  Resid. Df Resid. Dev    Df Deviance P(>|Chi|)
1     20507    18728.4
2     20506    18346.1     1    382.3 4.018e-85
```

This effectively performs the **likelihood ratio test** of the null hypothesis of a homogeneous Poisson process (CSR) against the alternative of an inhomogeneous Poisson process with intensity that is a loglinear function of the slope covariate (6). The p-value is extremely small, indicating rejection of CSR in favour of the alternative. (Please ignore the columns `Resid.Df` and `Resid.Dev` which are artefacts of the discretisation. Only the deviance difference and the difference in degrees of freedom are valid.)

At the time of writing, automatic model selection (using `step`) does not work for the class `"ppm"`.

Note that standard Analysis of Deviance requires the null hypothesis to be a sub-model of the alternative. Unfortunately the model (8), in which intensity is proportional to slope, does *not* include the homogeneous Poisson process as a special case, so we cannot use analysis of deviance to test the null hypothesis of homogeneous Poisson against the alternative of an inhomogeneous Poisson with intensity (8).

One possibility here is to use the Akaike Information Criterion **AIC** for model selection.

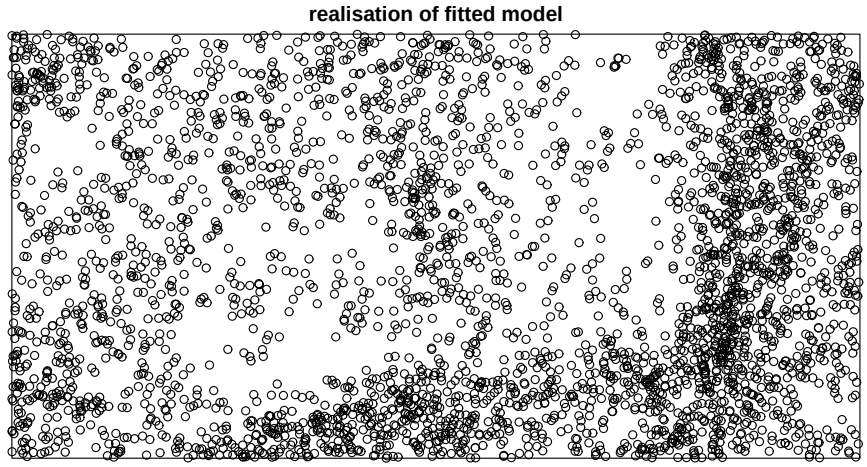
```
> fitprop <- ppm(bei, ~offset(log(slope)), covariates = list(slope = grad))
> fitnull <- ppm(bei, ~1)
> AIC(fitprop)

[1] 42496.65

> AIC(fitnull)

[1] 42763.92
```

The smaller AIC favours the model (8) with intensity is proportional to slope.

11.5 Simulating the fitted model

A fitted Poisson model can be simulated automatically using the function `rmh`.

```
> X <- rmh(fitprop)
> plot(X, main = "realisation of fitted model")
```



12 Methods 4: checking a fitted Poisson model

After fitting a point process model to a point pattern dataset, we should check that the model is a good fit ('goodness-of-fit'), and that each component assumption of the model was appropriate ('validation'). This section presents some techniques available for checking a fitted Poisson model.

Model checking can be either 'formal' or 'informal'. Formal techniques are based on detailed probabilistic assumptions about the data, and allow us to make probabilistic statements about the outcome. They include hypothesis tests, p-values, Bayesian model selection, χ^2 tests, goodness-of-fit tests and Monte Carlo tests. These have been presented in the previous sections.

In contrast, 'informal' tools do not impose assumptions on the data and their interpretation depends on human judgement. A typical example is the residual, defined for each observation by `(residual) = (observed) - (fitted)`. If the model is a good fit, then the residuals should be 'noise', centred around zero.

12.1 Goodness-of-fit

A goodness-of-fit test is a formal test of the null hypothesis that the model is true, against the very general alternative that the model is not true.

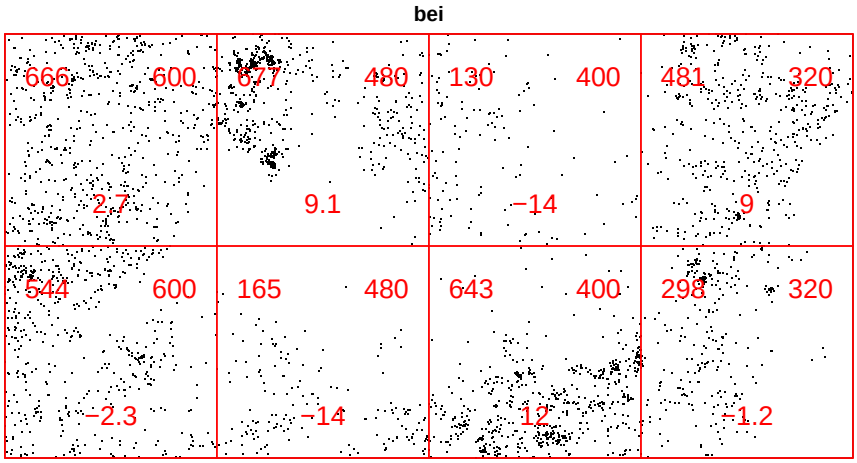
The χ^2 goodness-of-fit test based on quadrat counts can be applied to a fitted Poisson model, homogeneous or inhomogeneous. Under the null hypothesis, the quadrat counts are independent Poisson variables with different mean values, and the means are estimated by the fitted model.

```
> data(bei)
> fit <- ppm(bei, ~x)
> M <- quadrat.test(fit, nx = 4, ny = 2)
> M
```

```
        Chi-squared test of fitted model 'fit' using quadrat counts

data:  data from fit
X-squared = 711.5036, df = 6, p-value < 2.2e-16
```

If (as in this case) the formal goodness-of-fit test rejects the fitted model, we would then like to gain an informal impression of the type of departure from the model (i.e. in what way the data appear to depart from the predictions of the model) so that we may formulate a better model. To do this we can inspect the residual counts.

```
> plot(bei, pch = ".")
> plot(M, add = TRUE, cex = 1.5, col = "red")
```



The plot displays, for each quadrat, the observed number of points (top left), the predicted number of points according to the model (top right), and the Pearson residual (bottom) defined by

$$\text{Pearson residual} = \frac{(\text{observed}) - (\text{expected})}{\sqrt{\text{expected}}}$$

If the original data were Poisson, this transformation approximately standardises the residuals so that they have mean zero and variance 1 when the model is true. A Pearson residual of -14 is a gross departure from the fitted model.

The Kolmogorov-Smirnov test can also be applied to a fitted Poisson model, with homogeneous or inhomogeneous intensity.

```
> kstest(fit, function(x, y) {
+     y
+ })

        Spatial Kolmogorov-Smirnov test of inhomogeneous Poisson process

data:  covariate 'function(x, y) {    y}' evaluated at points of 'bei'
         and transformed to uniform distribution under 'fit'
D = 0.1026, p-value < 2.2e-16
alternative hypothesis: two-sided
```

This uses the method `kstest.ppm` for the generic function `kstest`.

12.2 Validation using residuals

12.2.1 Residuals

Residuals from the fitted model are an important diagnostic tool in other areas of applied statistics, but in spatial statistics they have only recently been developed ([35, 41], [40, pp. 49–50], [6]).

For a fitted Poisson process model, with fitted intensity $\widehat{\lambda}(u)$, the predicted number of points falling in any region B is $\int_B \widehat{\lambda}(u)\,\mathrm{d}u$. Hence the **residual** in each region $B \subset \mathbb{R}^2$ is defined [6] to be the *observed minus predicted* number of points falling in B: [6]

$$R(B) = n(\mathbf{x} \cap B) - \int_B \widehat{\lambda}(u)\,\mathrm{d}u \tag{11}$$

where $\mathbf{x}$ is the observed point pattern, $n(\mathbf{x} \cap B)$ the number of points of $\mathbf{x}$ in the region B, and $\widehat{\lambda}(u)$ is the intensity of the *fitted* model.

These residuals are closely related to the residuals for quadrat counts that were used above. Taking the set B to be one of our quadrats, the 'observed' quadrat count is $n(\mathbf{x} \cap B)$. The 'expected' quadrat count is $\widehat{\lambda}\,\text{area}(B)$ if the model is CSR, or more generally $\int_B \widehat{\lambda}(u)\,\mathrm{d}u$ if the model is an inhomogeneous Poisson process. Hence the 'raw residual' is `observed - expected` $= n(\mathbf{x} \cap B) - \int_B \widehat{\lambda}(u)\,\mathrm{d}u$.

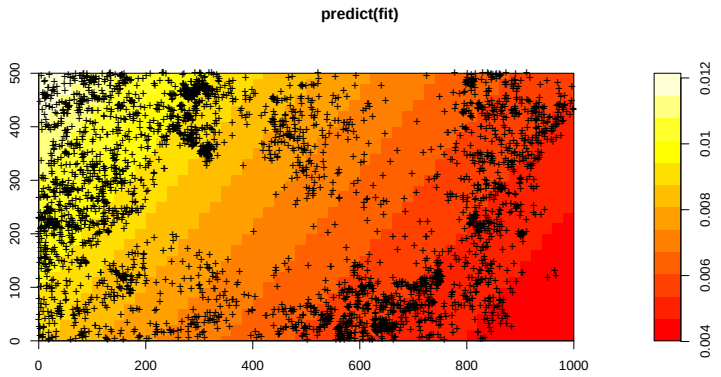
12.2.2 Residual measure

Equation (11) defines the total residual for any region B, large or small.

Intuitively the residuals can be visualised as an electric charge, with unit positive charge at each data point, and a diffuse negative charge at all other locations u, with density $\hat{\lambda}(u)$. If the model is true, then these charges should approximately cancel.

If we'd like to visualise this electric charge, one way is to plot the observed points and the fitted intensity function together:

```
> data(bei)
> fit <- ppm(bei, ~x + y)
> plot(predict(fit))
> plot(bei, add = TRUE, pch = "+")
```

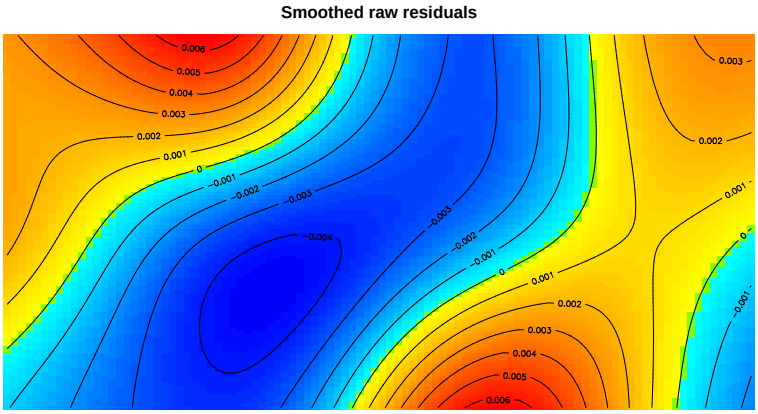


Each data point should be visualised as a charge of +1, while the colour image indicates a negative charge density.

12.2.3 Smoothed residuals

A more useful way to visualise the residuals is to smooth them.

```
> data(bei)
> fitx <- ppm(bei, ~x)
> diagnose.ppm(fitx, which = "smooth")
```



This is an image plot of the 'smoothed residual field'

$$s(u) = \widehat{\lambda}(u) - \lambda^{\dagger}(u) \tag{12}$$

where $\widehat{\lambda}(u)$ is the nonparametric, kernel estimate of the intensity,

$$\widehat{\lambda}(u) = e(u) \sum_{i=1}^{n(\mathbf{x})} \kappa(u - x_i)$$

while $\lambda^{\dagger}(u)$ is a correspondingly-smoothed version of the parametric estimate of the intensity according to the fitted model,

$$\lambda^{\dagger}(u) = e(u) \int_W \kappa(u - v) \lambda_{\hat{\theta}}(v) \, \mathrm{d}v.$$

Here κ is the smoothing kernel and $e(u)$ is the edge correction (2) on page 37. The difference (12) should be approximately zero if the model is true.

In this example the smoothed residual image contains a visible trend, suggesting that the model is inappropriate.

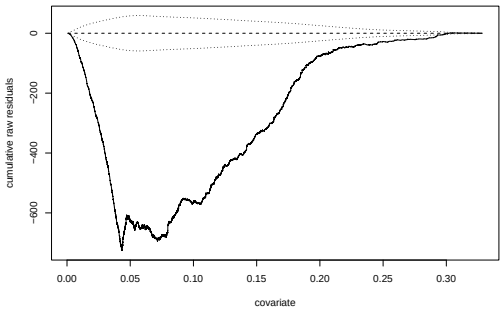
12.2.4 Lurking variable plot

If there is a spatial covariate $Z(u)$ that plays an important role in the analysis, it may be useful to display a *lurking variable plot* of the residuals against Z. This is a plot of $C(z) = R(B(z))$ against z, where

$$B(z) = \{u \in W : Z(u) \le z\}$$

is the region of space where the covariate value is less than or equal to z.

```
> grad <- bei.extra$grad
> lurking(fitx, grad, type = "raw")
```



Note that the lurking variable plot typically starts and ends at the horizontal axis, since (for any model with an intercept term) the total residual for the entire window W must equal zero. This is analogous to the fact that the residuals in linear regression sum to zero.

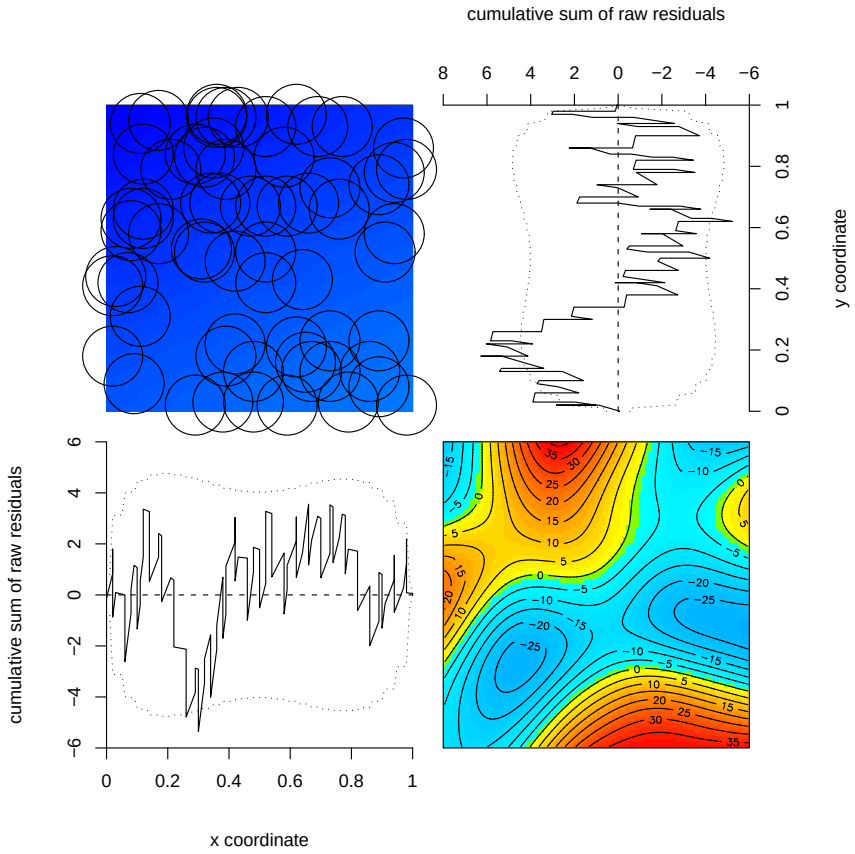
The plot also shows approximate 5% significance bands for the cumulative residual $C(x)$ or $C(y)$, obtained from the asymptotic variance under the model.

This plot indicates that the model is grossly inadequate; the fitted intensity function fails to capture the dependence of intensity on slope.

12.2.5 Four-panel plot

If there are no spatial covariates, use the command `diagnose.ppm` to plot the residuals:

```
> data(japanesepines)
> fit <- ppm(japanesepines, ~x + y)
> diagnose.ppm(fit)
```



This combination of four plots has proved to be a useful quick indication of departure from the trend in the model.

The bottom right panel is an image of the smoothed residual field.

The top left panel is a direct representation of the residual 'charge', with circles representing the data points (positive residuals) and a colour scheme representing the fitted intensity (negative residuals). However, it is often difficult to interpret.

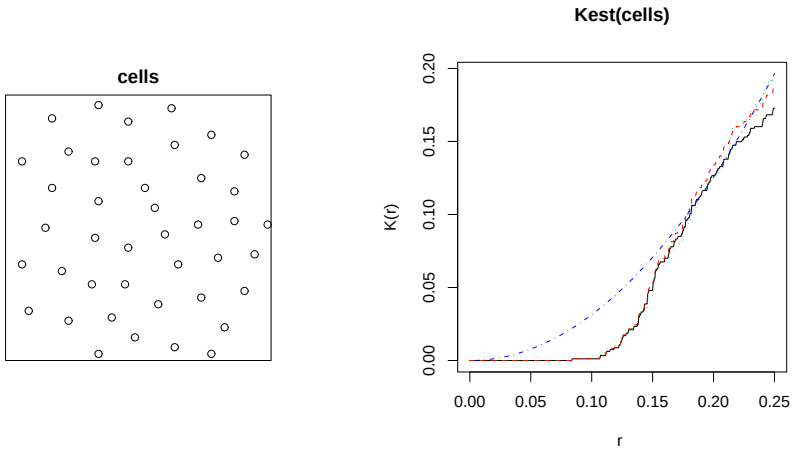
The two other panels are lurking variables against one of the cartesian coordinates. For example, the bottom left panel is a lurking variable plot for the x-coordinate. Imagine a vertical line which sweeps from left to right across the window. The progressive total residual to the left of the line is plotted against the position of the line.

In this example, the lurking variable plot for the y coordinate suggests a lack of fit at about $y = 0.15$, and the image of the smoothed residual field suggests an excess of positive residuals at about $x = 0.8, y = 0.15$, both indicating that the model *underestimates* the true intensity of points in this vicinity.

12.2.6 Q–Q plot

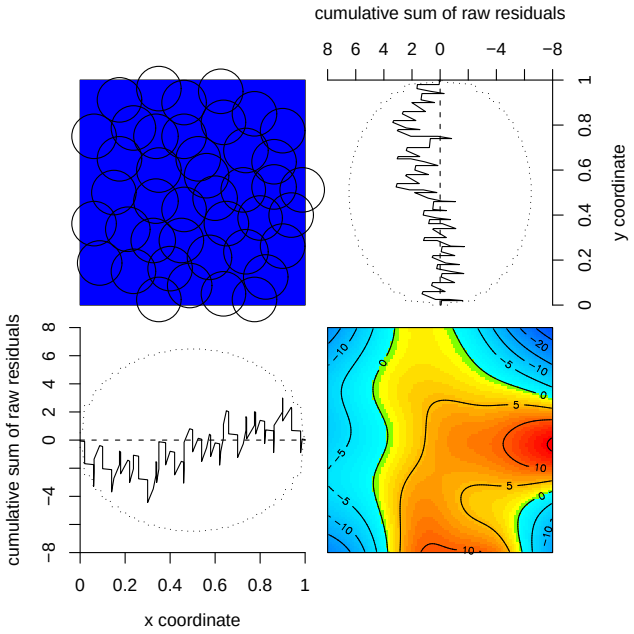
The residual plots described above are only useful for detecting misspecification of the *trend* in the fitted model. For example, the `cells` dataset has a uniform intensity but is clearly not a Poisson pattern:

```
> data(cells)
> par(mfrow = c(1, 2))
> plot(cells)
> plot(Kest(cells))
> par(mfrow = c(1, 1))
```



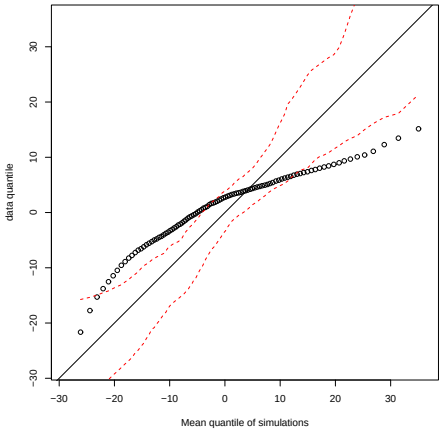
yet the residual plots appear to show nothing is wrong:

```
> fitPois <- ppm(cells, ~1)
> diagnose.ppm(fitPois)
```



Interaction between points in a point process corresponds roughly to the distribution of the responses in loglinear regression. To validate the interaction terms in a point process model, we should plot the distribution of the residuals. The appropriate tool is a *Q–Q plot*.

```
> qqplot.ppm(fitPois, nsim = 39)
```



This shows a Q–Q plot of the smoothed residuals, with pointwise 5% critical envelopes from simulations of the fitted model. This indicates that the uniform Poisson model is grossly inappropriate.

13 Images in spatstat

It's time to learn some more about pixel images in `spatstat`. They represent spatial functions $Z(u)$ in many different contexts.

An object of class `"im"` represents a pixel image. It specifies a rectangular grid of locations ("pixels") in two dimensional space, and a numerical value for each pixel. The pixel values can be real numbers, integers, complex numbers, single characters or strings, logical values or categorical values. A pixel's value can also be `NA`, meaning that it is not defined at that location.

13.1 Creating a pixel image

13.1.1 Creating an image from raw data

To create a pixel image from raw data, use `im`:

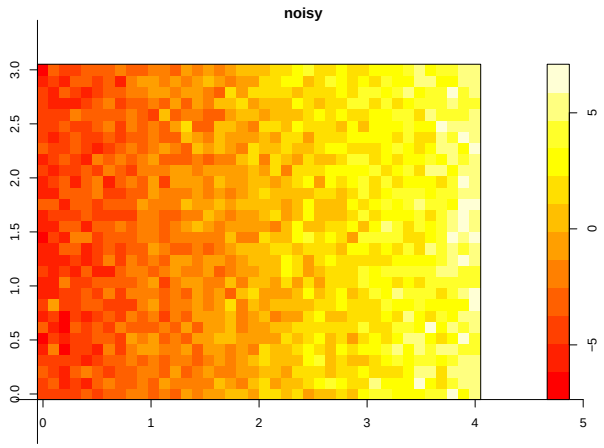
```
> im(mat, xcol, yrow)
```

where `mat` is a matrix containing the pixel values. The pixel values could have been generated by hand, or read from a file.

The correspondence between matrix indices `mat[i,j]` and cartesian coordinates is slightly idiosyncratic: the **rows** of `m` correspond to the y coordinate, and the columns to the x coordinate.

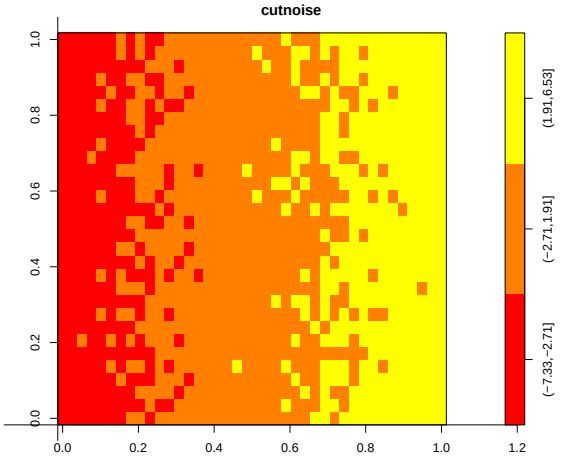
The argument `xcol` is a vector of equally-spaced x coordinate values corresponding to the **columns** of `mat`, and `yrow` is a vector of equally-spaced y coordinate values corresponding to the **rows** of `mat`. These vectors determine the spatial position of the pixel grid. The length of `xcol` is `ncol(mat)` while the length of `yrow` is `nrow(mat)`. If `mat` is not a matrix, it will be converted into a matrix with `nrow(mat) = length(yrow)` and `ncol(mat) = length(xcol)`.

```
> vec <- seq(-5, 5, length = 1200) + rnorm(1200)
> mat <- matrix(vec, nrow = 30, ncol = 40)
> noisy <- im(mat, xcol = seq(0, 4, length = 40), yrow = seq(0,
+     3, length = 30))
> plot(noisy)
```



For some strange reason, R does not allow matrices with categorical (factor) values. To create a pixel image with categorical values, leave the pixel values as a vector. The `im` command will reshape it:

```
> cutvec <- cut(mat, 3)
> cutnoise <- im(cutvec, xcol = seq(0, 1, length = 40), yrow = seq(0,
+     1, length = 30))
> plot(cutnoise)
```



Although `mat` was a matrix, `cutvec` is a vector, with factor values. Finally `cutnoise` is a factor-valued image.

13.1.2 Converting a function to an image

The command `as.im` will convert other types of data to a pixel image.

A function `f(x,y)` can be converted into a pixel image. This makes it easy to create a pixel image in which the pixel values are defined by an algebraic formula in the x and y coordinates.

```
> f <- function(x, y) {
+     x^2 + y^2
+ }
> w <- owin(c(-1, 1), c(-1, 1))
> Z <- as.im(f, w)
```

The second argument of `as.im` is a window object (class `"owin"`) specifying the domain of the image.

13.1.3 Functions that return a pixel image

Functions that return an object of class `"im"` include:

`as.im`	converts other data to a pixel image
`density.ppp`	kernel smoothing of point pattern
`density.psp`	kernel smoothing of line segment pattern
`distmap.owin`	distance function of window
`distmap.ppp`	distance function of point pattern
`distmap.psp`	distance function of line segment pattern
`setcov`	geometric covariance function of a window
`predict.ppm`	fitted intensity of a point process model
`[.im`	subset of an image (or look up pixel values)
`shift.im`	vector shift of an image
`eval.im`	evaluate any expression involving images
`cut.im`	convert numeric image to factor image
`interp.im`	spatial interpolation of image

13.2 Inspecting an image

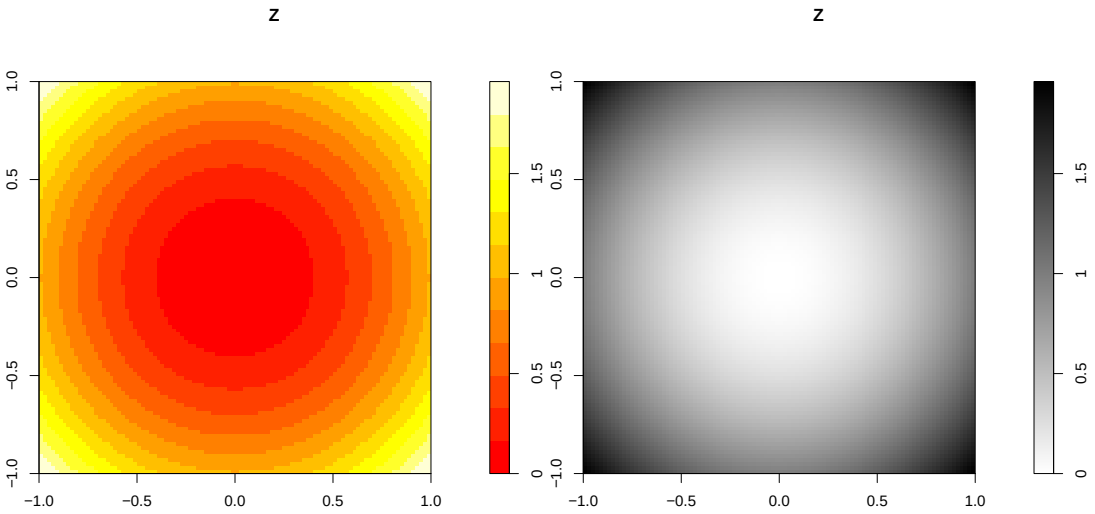
13.2.1 Plotting an image

Methods for plotting an image object include:

`plot.im`	display as colour image
`contour.im`	contour plot
`persp.im`	perspective plot of surface

Note that the default colour map for image plots in R has only 12 colours and can convey a misleading impression of the gradation of pixel values in the image. Use the argument `col` to control the colour map.

```
> opa <- par(mfrow = c(1, 2))
> plot(Z)
> plot(Z, col = grey(seq(1, 0, length = 512)))
> par(opa)
```



13.2.2 Exploratory analysis

To inspect an image, the following are useful.

`as.matrix`	extract matrix of pixel values from image
`cut.im`	convert numeric image to factor image
`hist.im`	histogram of pixel values

For an image `Z` with any type of values, `plot(cut(Z, 3))` will divide the pixel values into 3 bands, and display the image with the 3 bands rendered in 3 different colours.

To compute numerical summaries of pixel values, like the median or order statistics of the pixel values, extract the pixel values using `as.matrix(Z)` then apply the summary operation.

13.3 Manipulating images

13.3.1 Subsets of an image

The subset operator `[` has a method for pixel images, `[.im`:

```
> X[S]
> X[S, drop = TRUE]
```

The subset to be extracted is determined by the index argument `S`.

- If `S` is a point pattern, or a `list(x,y)`, then the values of the pixel image `X` at these points are extracted, and returned as a vector.
- If `S` is a window (an object of class `"owin"`), the values of the image inside this window are extracted. The result is a pixel image if possible, and a numeric vector otherwise (see `help("[.im")` for details).
- If `S` is a pixel image with logical values, it is interpreted as a window (with `TRUE` inside the window).

The logical argument `drop` determines whether pixel values that are undefined are omitted (`drop = TRUE`) or returned as the value `NA` (`drop=FALSE`).

See `help("[.im")` for full details.

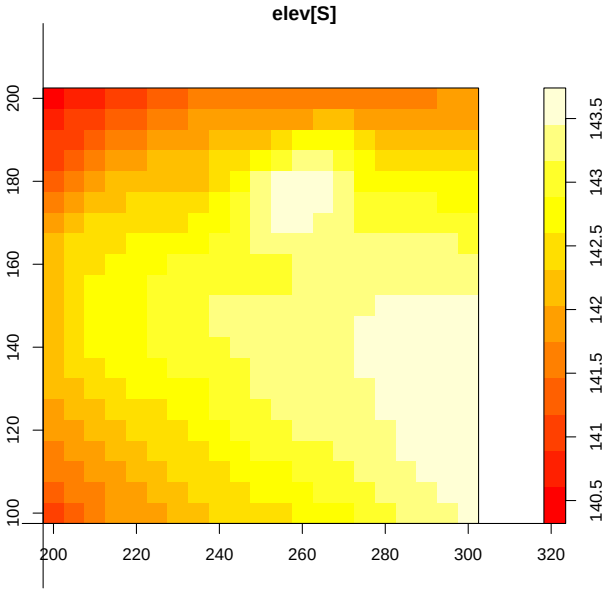
The subset operator can be used to look up the value of a pixel image at a single point:

```
> data(bei)
> elev <- bei.extra$elev
> elev[list(x = 142, y = 356)]
```

```
[1] 147.08
```

or to display a subregion:

```
> S <- owin(c(200, 300), c(100, 200))
> plot(elev[S])
```



This can even be performed interactively, using the R function `locator` to click on a point in the window:

```
> elev[locator(1)]
```

13.3.2 Computation with images

The handy function `eval.im` allows us to perform pixel-by-pixel calculations on an image or on several compatible images.

If `Z` is a pixel image, to take the logarithm of each pixel value,

```
> logZ <- eval.im(log(Z))
```

If `A` and `B` are two pixel images with compatible grids of pixels (i.e. having the same numbers of pixels and the same coordinate locations), then to find the sum of the corresponding pixel values,

```
> C <- eval.im(A + B)
```

The expressions may involve constants and functions as well, so long as the expression is 'parallelised'.

```
> W <- eval.im(sin(pi * Z))
> V <- eval.im(Z > 3)
> U <- eval.im(ifelse(Z > 3, 42, Z))
```

Other functions which manipulate images include the following:

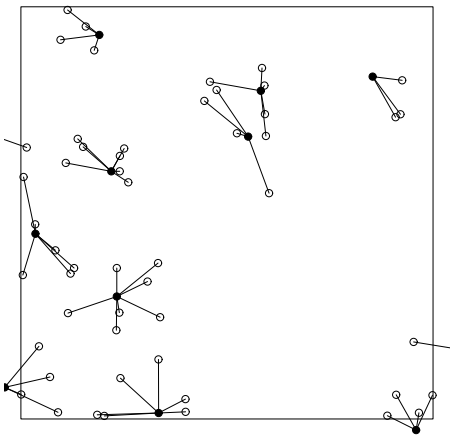
`shift.im`	vector shift of an image
`cut.im`	convert numeric image to factor image
`interp.im`	spatially interpolate an image
`levelset`	threshold an image (produces a window)
`solutionset`	find the region where a statement is true (produces a window)

14 Simple models of non-Poisson patterns

A point process that is not Poisson can be said to exhibit 'interaction' or dependence between the points. It's time to introduce some models for such processes. This section covers simple models that are derived from the Poisson process, and still retain some of the tractable features of the Poisson model.

14.1 Poisson cluster processes

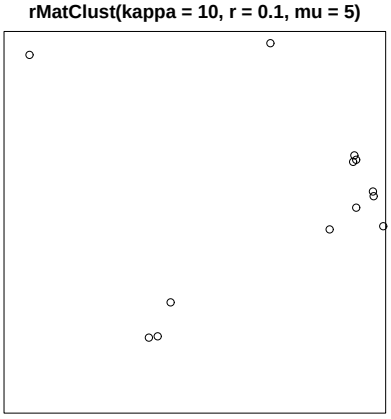
In a Poisson cluster process, we begin with a Poisson process $\mathbf{Y}$ of 'parent' points. Each 'parent' point $y_i \in \mathbf{Y}$ then gives rise to a finite set Z_i of 'offspring' points according to some stochastic mechanism. The set comprising all the offspring points forms a point process $\mathbf{X}$. Only $\mathbf{X}$ is observed.



An example is the *Matérn cluster process* in which the parent points come from a homogeneous Poisson process with intensity κ, and each parent has a Poisson (μ) number of offspring, independently and uniformly distributed in a disc of radius r centred around the parent.

The Matérn cluster process can be generated in `spatstat` using the command `rMatClust`. [By convention, random data generators in R always have names beginning with `r`.]

```
> plot(rMatClust(kappa = 10, r = 0.1, mu = 5))
```



Other Poisson cluster processes implemented in `spatstat` are

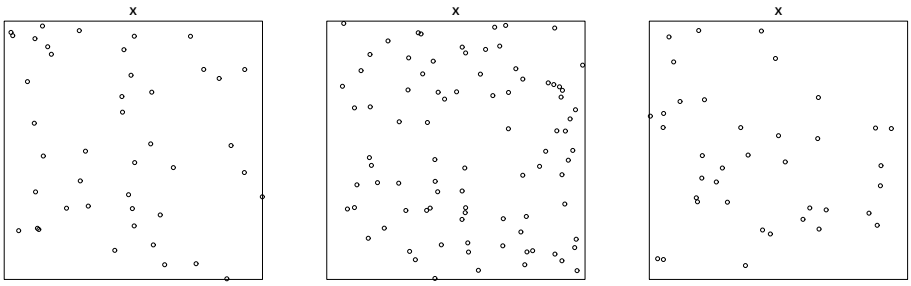
- `rThomas`: the *Thomas process,* in which each cluster consists of a Poisson(μ) number of random points, each having an isotropic Gaussian $N(0, \sigma^2 I)$ displacement from its parent.
- `rNeymanScott`: the general *Neyman-Scott* cluster process in which the cluster mechanism is arbitrary.

14.2 Cox processes

A Cox point process is effectively a Poisson process with a random intensity function. Let $\Lambda(u)$ be a random function with non-negative values, defined at all locations $u \in \mathbb{R}^2$. Conditional on Λ, let $\mathbf{X}$ be a Poisson process with intensity function Λ. Then $\mathbf{X}$ is a Cox process.

A trivial example is the "mixed Poisson" process in which we generate a random variable Λ and, conditional on Λ, generate a uniform Poisson process with intensity Λ. Following are three different realisations of this process:

```
> par(mfrow = c(1, 3))
> for (i in 1:3) {
+     lambda <- rexp(1, 1/100)
+     X <- rpoispp(lambda)
+     plot(X)
+ }
> par(mfrow = c(1, 1))
```



Moments of Cox processes are tractable (in terms of the moments of Λ). The intensity function of $\mathbf{X}$ is $\lambda(u) = \mathbb{E}[\Lambda(u)]$.

A Cox model is the analogue of a 'random effects' model. It is always overdispersed relative to a Poisson process (i.e. the variance of the number of points falling in a region, is greater than the mean). Cox processes are the most convenient models for clustered point patterns. A particularly interesting and useful class is that of *log-Gaussian Cox processes (LGCP)* in which $\log \Lambda(u)$ is a Gaussian random function [33, 34].

The Matérn Cluster process and the Thomas process are both Cox processes.

Currently there are no functions in `spatstat` for generating the general Cox process, but if you have a way of generating realisations of a random function Λ of interest, then you can use `rpoispp` to generate the Cox process. The intensity argument `lambda` of `rpoispp` can be a `function(x,y)` or a pixel image.

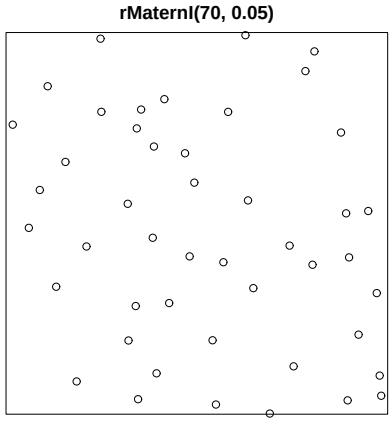
14.3 Thinned processes

'Thinning' means deleting some of the points from a point pattern. Under *'independent thinning'* the fate of each point is independent of other points. When independent thinning is applied to a

Poisson process, the resulting process of retained points is Poisson. To get a non-Poisson process we need some kind of *dependent thinning* mechanism.

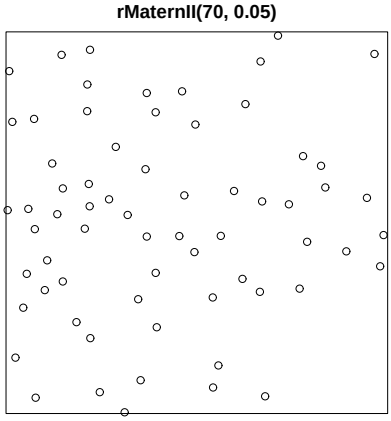
In *Matérn's Model I*, a homogeneous Poisson process $\mathbf{Y}$ is first generated. Any point in $\mathbf{Y}$ that lies closer than a distance r from the nearest other point of $\mathbf{Y}$, is deleted. Thus, pairs of close neighbours annihilate each other.

```
> plot(rMaternI(70, 0.05))
```



In *Matérn's Model II*, the points of the homogeneous Poisson process $\mathbf{Y}$ are marked by 'arrival times' t_i which are independent and uniformly distributed in $[0, 1]$. Any point in $\mathbf{Y}$ that lies closer than distance r from another point that has an earlier arrival time, is deleted.

```
> plot(rMaternII(70, 0.05))
```

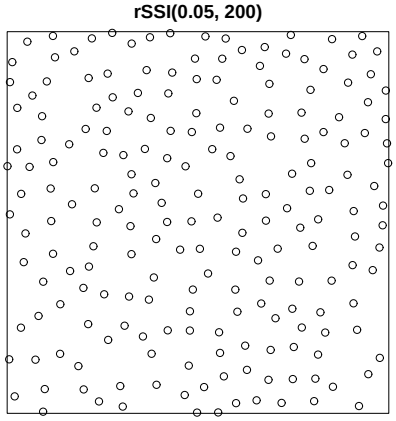


14.4 Sequential models

In a sequential model, we start with an empty window, and the points are placed into the window one-at-a-time, according to some criterion.

In Simple Sequential Inhibition, each new point is generated uniformly in the window and independently of preceding points. If the new point lies closer than r units from an existing point, then it is rejected and another random point is generated. The process terminates when no further points can be added.

```
> plot(rSSI(0.05, 200))
```

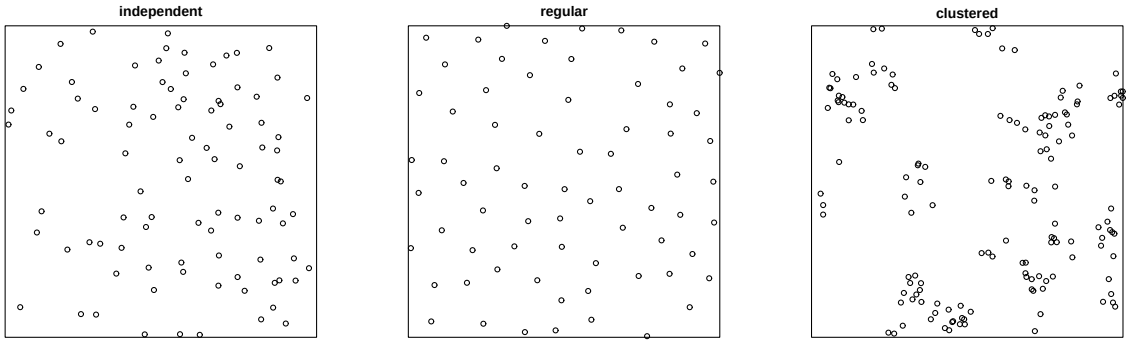


Sequential point processes are the easiest way to generate highly ordered patterns with high intensity.

15 Methods 5: Distance methods for point patterns

Suppose that a point pattern appears to have constant intensity, and we wish to assess whether the pattern is Poisson. The alternative is that the points are dependent (they exhibit 'interaction').

Classical writers suggested a simple trichotomy between 'independence' (the Poisson process), 'regularity' (where points tend to avoid each other), and 'clustering' (where points tend to be close together). [The concept of 'clustering' does not imply that the points are organised into identifiable 'clusters'; merely that they are closer together than expected for a Poisson process.]



15.1 Distances

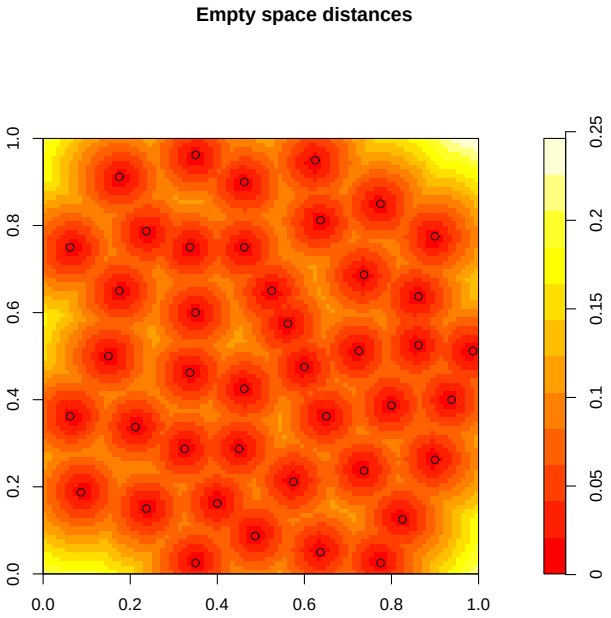
The classical techniques for investigating interpoint interaction are *distance methods,* based on measuring the distances between points. Specifically we may consider

- **pairwise distances** $s_{ij} = ||x_i - x_j||$ between all distinct pairs of points x_i and x_j $(i \neq j)$ in the pattern;
- **nearest neighbour distances** $t_i = \min_{j \neq i} s_{ij}$, the distance from each point x_i to its nearest neighbour;
- **empty space distances** $d(u) = \min_i ||u - x_i||$, the distance from a fixed reference location u in the window to the nearest data point.

If you need to compute these directly, they are available in `spatstat` using the functions `pairdist`, `nndist` and `distmap` respectively. If `X` is a point pattern object,

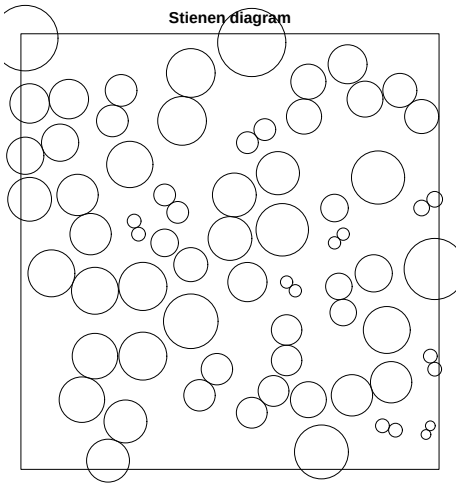
- `pairdist(X)` returns the matrix of pairwise distances.
- `nndist(X)` returns the vector of nearest neighbour distances.
- `distmap(X)` returns a pixel image whose pixel values are the empty space distances to the pattern `X` measured from every pixel.

```
> data(cells)
> emp <- distmap(cells)
> plot(emp, main = "Empty space distances")
> plot(cells, add = TRUE)
```



Tip: Quite a useful exploratory tool is the *Stienen diagram* obtained by drawing a circle around each data point of diameter equal to its nearest neighbour distance:

```
> plot(X %mark% (nndist(X)/2), markscale = 1, main = "Stienen diagram")
```



15.2 Empty space distances

It's easiest to start by explaining the analysis of the empty space distances

The distance

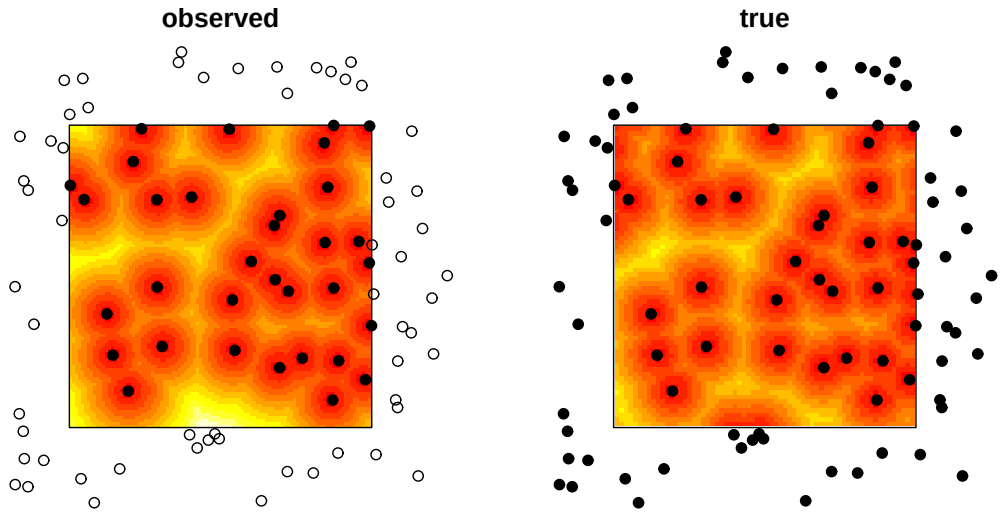
$$d(u, \mathbf{x}) = \min\{||u - x_i|| : x_i \in \mathbf{x}\}$$

from a fixed location $u \in \mathbb{R}^2$ to the nearest point in a point pattern $\mathbf{x}$, is called the 'empty space distance' or 'void distance'. It can be computed for all locations u on a fine grid, using the `spatstat` function `distmap` as we saw above.

15.2.1 Edge effects

It is not easy to interpret a histogram of the empty space distances. The empirical distribution of the empty space distances depends on the geometry of the window W as well as on characteristics of the point process $\mathbf{X}$.

Another viewpoint is that the window introduces a sampling bias. Recall that under the 'standard model' (Section 2.3) the point process $\mathbf{X}$ extends throughout 2-D space, but is observed only inside W. This leads to bias in the distance measurements. Confining observations to a window W implies that the observed distance $d(u, \mathbf{x}) = d(u, \mathbf{X} \cap W)$ to the nearest data point inside W, may be greater than the true distance $d(u, \mathbf{X})$ to the nearest point of the complete point process $\mathbf{X}$.



15.2.2 Empty space function F

Ignoring the edge problems for a moment, let us focus on the entire point process $\mathbf{X}$.

Assuming $\mathbf{X}$ is *stationary* (statistically invariant under translations), we can define the cumulative distribution function of the empty space distance

$$F(r) = \mathbb{P}\{d(u, \mathbf{X}) \le r\} \tag{13}$$

where u is an arbitrary reference location. If the process is stationary then this definition does not depend on u.

The empirical distribution function of the observed empty space distances on a grid of locations u_j, $j = 1, \ldots, m$,

$$F^*(r) = \frac{1}{m} \sum_j \mathbf{1}\{d(u_j, \mathbf{x}) \le r\} \tag{14}$$

is a negatively biased estimator of $F(r)$, for reasons explained above.

Corrections for this 'edge effect bias' are required. Many edge corrections are available. Typically they are weighted versions of the ecdf,

$$\widehat{F}(r) = \sum_j e(u_j, r) \mathbf{1}\{d(u_j, \mathbf{x}) \le r\} \tag{15}$$

where $e(u, r)$ is an edge correction weight designed so that $\widehat{F}(r)$ is unbiased. These corrections are effectively forms of the Horvitz-Thompson estimator of survey sampling fame.

The edge effect problem can also be regarded as a form of censoring (analogous to right-censoring in survival data), as first pointed out by CSIRO researcher Geoff Laslett [29]. A counterpart of the Kaplan-Meier estimator is available. For further information see [7].

Thus, *assuming that the point process is homogeneous*, we are able to compute an unbiased and reasonably accurate estimate of the empty space function F defined by (13).

To interpret this estimate, a useful benchmark is the Poisson process. Notice that $d(u, \mathbf{X}) > r$ if and only if there are no points of X in the disc $b(u, r)$ of radius r centred on u. For a homogeneous Poisson process of intensity λ, the number of points falling in $b(u, r)$ is Poisson with mean $\mu = \lambda \text{area}(b(u, r)) = \lambda\pi r^2$, so the probability that there are no points in this region is $\exp(-\mu) = \exp(-\lambda\pi r^2)$. Hence for a Poisson process

$$F_{\text{pois}}(r) = 1 - \exp(-\lambda\pi r^2). \tag{16}$$

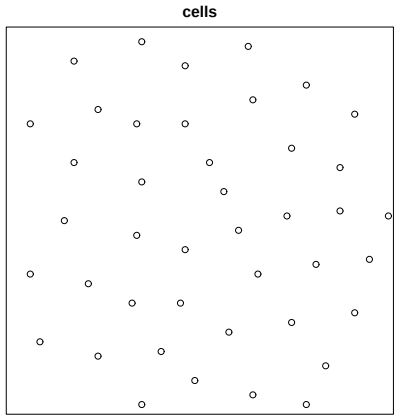
Typically we compare $\widehat{F}(r)$ with the value of $F_{\text{pois}}(r)$ obtained by plugging in the estimated intensity $\hat{\lambda} = n(\mathbf{x})/\text{area}(W)$. Values $\widehat{F}(r) > F_{\text{pois}}(r)$ suggest that empty space distances in the point pattern are shorter than for a Poisson process, suggesting a regularly space pattern; while values $\widehat{F}(r) < F_{\text{pois}}(r)$ suggest a clustered pattern.

15.2.3 Implementation in `spatstat`

The function `Fest` computes estimates of $F(r)$ using several edge corrections, and the benchmark value for the Poisson process.

```
> data(cells)
> plot(cells)
> Fc <- Fest(cells)
> Fc
```

```
Function value object (class 'fv')
for the function r -> F(r)
Entries:
id       label       description
--       -----       -----------
r        r           distance argument r
theo     Fpois(r)    theoretical Poisson F(r)
rs       Fbord(r)    border corrected estimate of F(r)
km       Fkm(r)      Kaplan-Meier estimate of F(r)
hazard   lambda(r)   Kaplan-Meier estimate of hazard function lambda(r)
raw      Fraw(r)     uncorrected estimate of F(r)
--------------------------------------

Default plot formula:
        . ~ r

Recommended range of argument r: [0, 0.085]
```

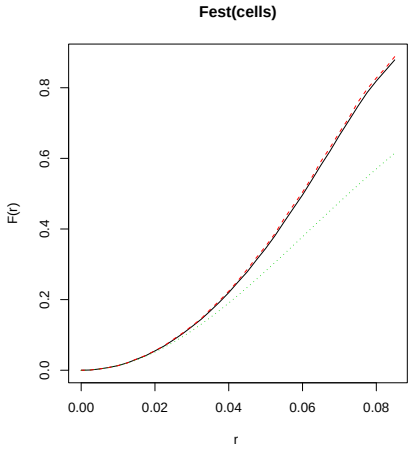


Tip: Don't use `F` as a variable name! It's a reserved word — an abbreviation for `FALSE`.

The value returned by `Fest` is an object of class `"fv"` ("function value table"). This is effectively a data frame with some extra information. The printout for `Fc` indicates that the columns in the data frame are named `r`, `theo`, `rs`, `km`, `hazard` and `raw`. The first column `r` contains a sequence of values of the function argument r. The next column `theo` contains the corresponding values of $F(r)$ for a homogeneous Poisson process. The columns `rs`, `km` and `raw` contain different estimates of the empty space function F, namely the 'reduced sample' estimator, the Kaplan-Meier estimator, and the uncorrected empirical distribution function, respectively. The column `hazard` contains an estimate of the hazard rate of F, i.e. $h(r) = (d/dr)\log(1-F(r))$, a by-product of the Kaplan-Meier estimate.

```
> par(pty = "s")
> plot(Fest(cells))
```

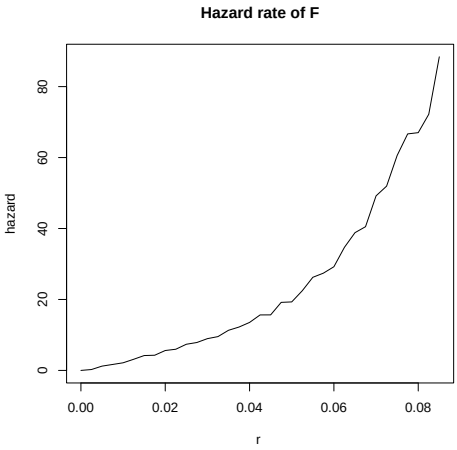
```
      lty col
km      1   1
rs      2   2
theo    3   3
```



This is a call to `plot.fv`. The printed output is the return value from `plot.fv`, which explains the encoding of the different function estimates using the R graphics parameters `lty` (line type) and `col` (line colour).

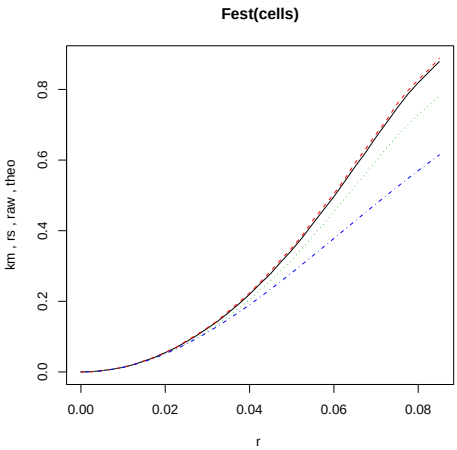
You'll notice that, by default, the uncorrected estimate `raw` and the hazard rate `hazard` were not plotted. The choice of estimates to be plotted, and the style in which they are plotted, are controlled by the second argument to `plot.fv`, which should be an R language formula involving the identifier names `r`, `theo`, `rs`, `km`, `hazard` and `raw`. To plot the hazard rate against r,

```
> plot(Fest(cells), hazard ~ r, main = "Hazard rate of F")
```



To plot all the estimates of $F(r)$, including the uncorrected estimate:

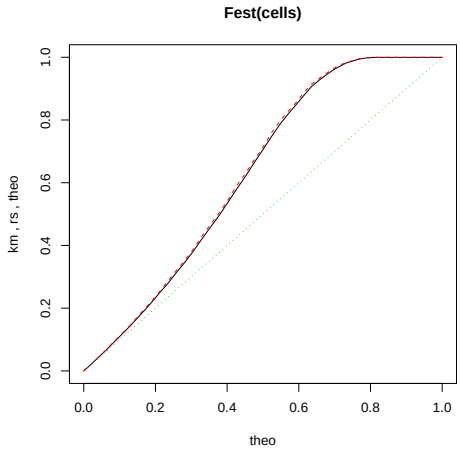
```
> plot(Fest(cells), cbind(km, rs, raw, theo) ~ r)
```



Notice the use of `cbind` to specify several different graphs on the same plot.

To plot the estimates of $F(r)$ against the Poisson value, in the style of a P–P plot:

```
> plot(Fest(cells), cbind(km, rs, theo) ~ theo)
```



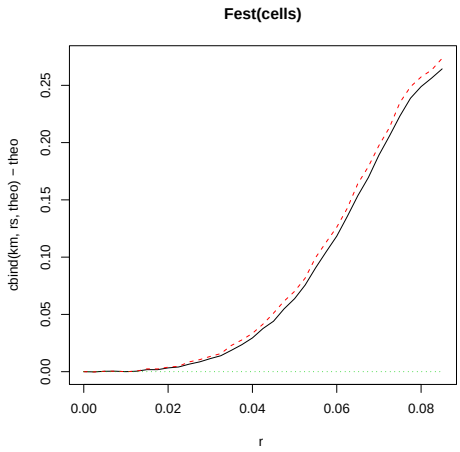
(including `theo` on the left side here gives us the diagonal line).

The symbol `.` stands for 'all recommended estimates of the function'. So an abbreviation for the last command is

```
> plot(Fest(cells), . ~ theo)
```

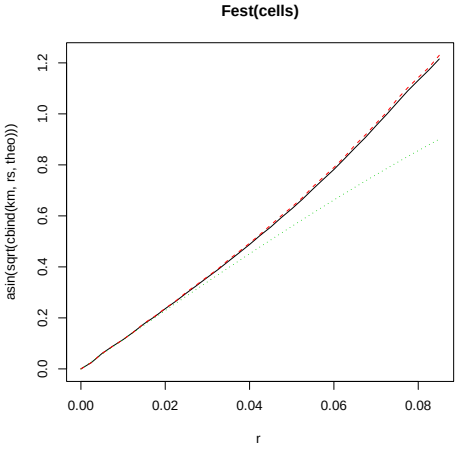
Transformations can be applied to these function values. For example, to subtract the theoretical Poisson value from the estimates,

```
> plot(Fest(cells), . - theo ~ r)
```



To apply Fisher's variance stabilising transformation $\phi(\hat{F}(t)) = \sin^{-1}(\sqrt{(}\hat{F}(t)))$,

```
> plot(Fest(cells), asin(sqrt(.)) ~ r)
```



15.3 Nearest neighbour distances

For other types of distances we encounter similar problems. For the nearest neighbour distances $t_i = \min_{j \neq i} ||x_i - x|j||$, again it is not easy to interpret a histogram of the observed distances. The empirical distribution of the nearest neighbour distances depends on the geometry of the window W as well as on characteristics of the point process $\mathbf{X}$. Confining observations to a window W implies that the observed nearest-neighbour distances are larger, in general, than the 'true' nearest neighbour distances of points in the entire point process $\mathbf{X}$. Corrections for this edge effect bias are required.

15.3.1 G function

Assuming the point process $\mathbf{X}$ is stationary, we can define the cumulative distribution function of the nearest-neighbour distance for a typical point in the pattern,

$$G(r) = \mathbb{P}\left\{d(u, \mathbf{X} \setminus \{u\}) \leq r \mid u \in \mathbf{X}\right\} \tag{17}$$

where u is an arbitrary location, and $d(u, \mathbf{X} \setminus \{u\})$ is the shortest distance from u to the point pattern $\mathbf{X}$ excluding u itself. If the process is stationary then this definition does not depend on u.

The empirical distribution function of the observed nearest-neighbour distances

$$G^*(r) = \frac{1}{n(\mathbf{x})} \sum_i \mathbf{1}\left\{t_i \leq r\right\} \tag{18}$$

is a negatively biased estimator of $G(r)$, for reasons we explained above. Many edge corrections are available. Typically they are weighted versions of the ecdf,

$$\widehat{G}(r) = \sum_i e(x_i, r)\mathbf{1}\left\{t_i \leq r\right\} \tag{19}$$

where $e(x_i, r)$ is an edge correction weight designed so that $\widehat{G}(r)$ is approximately unbiased. A counterpart of the Kaplan-Meier estimator is also available.

For a homogeneous Poisson point process of intensity λ, the nearest-neighbour distance distribution function is known to be

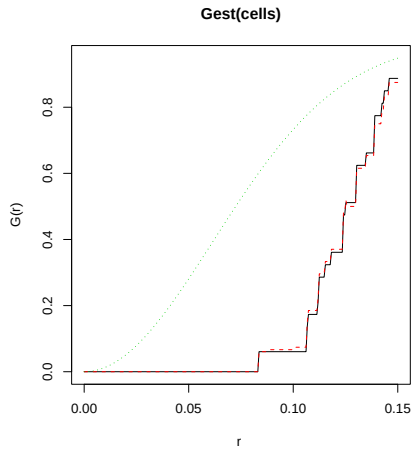
$$G_{\text{pois}}(r) = 1 - \exp(-\lambda \pi r^2). \tag{20}$$

This is identical to the empty space function for the Poisson process. Intuitively, because points of the Poisson process are independent of each other, the knowledge that u is a point of $\mathbf{X}$ does not affect any other points of the process, hence G is equivalent to F.

Interpretation of $\widehat{G}(r)$ is the reverse of $\widehat{F}(r)$. Values $\widehat{G}(r) > G_{\text{pois}}(r)$ suggest that nearest neighbour distances in the point pattern are shorter than for a Poisson process, suggesting a clustered pattern; while values $\widehat{G}(r) < G_{\text{pois}}(r)$ suggest a regular (inhibited) pattern.

The function `Gest` computes estimates of $G(r)$ using several edge corrections, and the benchmark value for the Poisson process.

```
> Gc <- Gest(cells)
> Gc

Function value object (class 'fv')
for the function r -> G(r)
Entries:
id       label       description
--       -----       -----------
r        r           distance argument r
theo     Gpois(r)    theoretical Poisson G(r)
rs       Gbord(r)    border corrected estimate of G(r)
km       Gkm(r)      Kaplan-Meier estimate of G(r)
hazard   lambda(r)   Kaplan-Meier estimate of hazard function lambda(r)
raw      Graw(r)     uncorrected estimate of G(r)
---------------------------------------

Default plot formula:
        . ~ r

Recommended range of argument r: [0, 0.15]

> par(pty = "s")
> plot(Gest(cells))
```



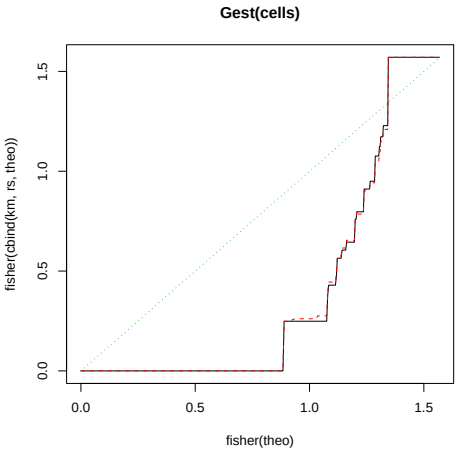
The estimate of $G(r)$ suggests strongly that the pattern is regular. Indeed, $\widehat{G}(r)$ is zero for $r \leq 0.07$ which indicates that there are no nearest-neighbour distances shorter than 0.07.

Common ways of plotting $\widehat{G}$ include:

$\widehat{G}(r)$ and $G_{\text{pois}}(r)$ plotted against r	`plot(Gest(X))`
$\widehat{G}(r) - G_{\text{pois}}(r)$ plotted against r	`plot(Gest(X), . - theo ~ r)`
$\widehat{G}(r)$ plotted against $G_{\text{pois}}(r)$ in P–P style	`plot(Gest(X), . ~ theo)`

and Fisher's variance-stabilising transformation $\phi(G(t)) = \sin^{-1}(\sqrt{G(t)})$ applied to the P–P plot:

```
> fisher <- function(x) {
+     asin(sqrt(x))
+ }
> plot(Gest(cells), fisher(.) ~ fisher(theo))
```



15.4 Pairwise distances and the K function

The observed pairwise distances $s_{ij} = ||x_i - x_j||$ in the data pattern $\mathbf{x}$ constitute a biased sample of pairwise distances in the point process, with a bias in favour of smaller distances. For example, we can never observe a pairwise distance greater than the diameter of the window.

Ripley [36] defined the K-function for a stationary point process so that $\lambda K(r)$ is the expected number of other points of the process within a distance r of a typical point of the process. Formally

$$K(r) = \frac{1}{\lambda}\mathbb{E}\left[n(\mathbf{X} \cap b(u,r) \setminus \{u\}) \mid u \in \mathbf{X}\right]. \tag{21}$$

For a homogeneous Poisson process, intuitively, the knowledge that u is a point of $\mathbf{X}$ does not affect the other points of the process, so that $\mathbf{X} \setminus \{u\}$ is conditionally a Poisson process. The expected number of points falling in $b(u,r)$ is $\lambda\pi r^2$. Thus for a homogeneous Poisson process

$$K_{\text{pois}}(r) = \pi r^2 \tag{22}$$

regardless of the intensity.

Numerous estimators of K have been proposed. Most of them are weighted and renormalised empirical distribution functions of the pairwise distances, of the general form

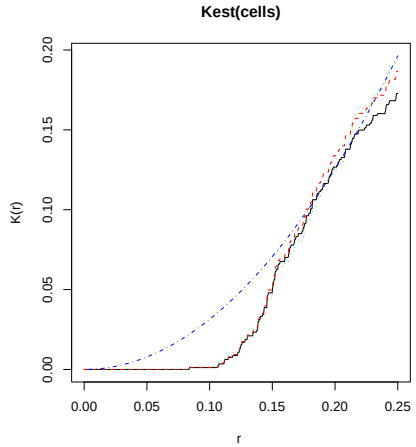
$$\widehat{K}(r) = \frac{1}{\widehat{\lambda}^2 \text{area}(W)} \sum_i \sum_{j \neq i} \mathbf{1}\left\{||x_i - x_j|| \leq r\right\} e(x_i, x_j; r) \tag{23}$$

where $e(u,v,r)$ is an edge correction weight. The choice of estimator does not seem to be very important, as long as *some* edge correction is applied.

Again we usually compare the estimate $\widehat{K}(r)$ with the Poisson K function. Values $\widehat{K}(r) > \pi r^2$ suggest clustering, while $\widehat{K}(r) < \pi r^2$ suggests a regular pattern.

In `spatstat` the function `Kest` computes several estimates of the K-function.

```
> Gc <- Kest(cells)
> Gc

Function value object (class 'fv')
for the function r -> K(r)
Entries:
id       label       description
--       -----       -----------
r        r           distance argument r
theo     Kpois(r)    theoretical Poisson K(r)
border   Kbord(r)    border-corrected estimate of K(r)
trans    Ktrans(r)   translation-corrected estimate of K(r)
iso      Kiso(r)     Ripley isotropic correction estimate of K(r)
--------------------------------------

Default plot formula:
        . ~ r

Recommended range of argument r: [0, 0.25]

> par(pty = "s")
> plot(Kest(cells))
```



In this case, the interpretation of all three summary statistics F, G and K is the same: emphatic evidence of a regular pattern. It is not always the case that these three summaries give equivalent messages.

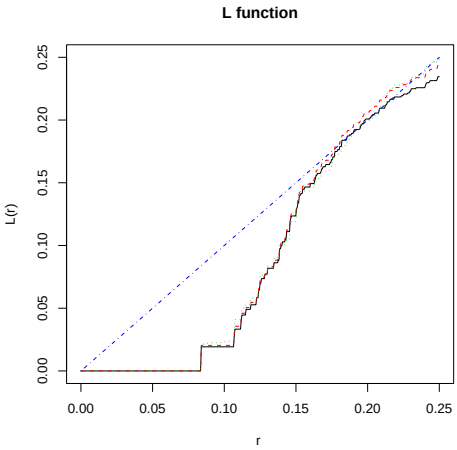
A commonly-used transformation of K is the L-function

$$L(r) = \sqrt{\frac{K(r)}{\pi}}$$

which transforms the Poisson K function to the straight line $L_{\text{pois}}(r) = r$, making visual assessment of the graph much easier. The square root transformation also approximately stabilises the variance of the estimator, making it easier to assess deviations.

To compute the estimated L function, use `Lest`.

```
> L <- Lest(cells)
> plot(L, main = "L function")
```



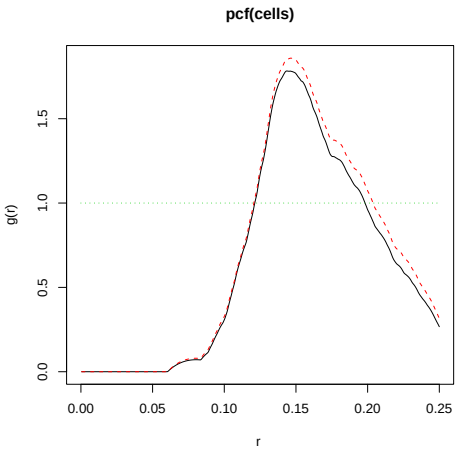
Another related summary function is the *pair correlation function*

$$g(r) = \frac{K'(r)}{2\pi r}$$

where $K'(r)$ is the derivative of K. The pair correlation is in some ways easier to interpret than either K or L, although it is more difficult to estimate. Roughly speaking, the pair correlation $g(r)$ is the probability of observing a pair of points separated by a distance r, divided by the corresponding probability for a Poisson process. This is a non-centred correlation which may take any nonnegative value. The value $g(r) = 1$ corresponds to complete randomness; for the Poisson process the pair correlation is $g_{\text{pois}}(r) \equiv 1$. For other processes, values $g(r) > 1$ suggest clustering or attraction at distance r, while values $g(r) < 1$ suggest inhibition or regularity.

To compute the estimated pair correlation function, use `pcf`.

```
> plot(pcf(cells))
```



Here we have used the method `pcf.ppp`. This computes a standard kernel estimate which performs well except at very small values of r. So it is prudent not to read too much into the behaviour of the pcf close to $r = 0$.

If you want to try another algebraic transformation of a summary function, the transformation can be computed using `eval.fv`. You can also plot algebraic transformations of a summary function using the 'plotting formula' argument to `plot.fv`. For example, if we have already computed the K function, we can plot the L function by

```
> K <- Kest(cells)
> plot(K, sqrt(./pi) ~ r)
```

and compute the L function using `eval.fv`:

```
> K <- Kest(cells)
> L <- eval.fv(sqrt(K/pi))
```

If you have already computed the K function and wish to derive the pair correlation, there is another algorithm `pcf.fv` that calculates $g(r) = K'(r)/(2\pi r)$ by numerical differentiation.

```
> K <- Kest(cells)
> g <- pcf(K)
```

15.5 J function

A useful combination of F and G is the J function [44]

$$J(r) = \frac{1 - G(r)}{1 - F(r)} \tag{24}$$

defined for all $r \geq 0$ such that $F(r) < 1$. For a homogeneous Poisson process, $F_{\text{pois}} = G_{\text{pois}}$, so that

$$J_{\text{pois}}(r) \equiv 1. \tag{25}$$

Values $J(r) > 1$ suggest regularity, and $J(r) < 1$ suggest clustering.

An appealing property of the J function is that the superposition $\mathbf{X}_\bullet = \mathbf{X}_1 \cup \mathbf{X}_2$ of two *independent* point processes $\mathbf{X}_1, \mathbf{X}_2$ has J-function

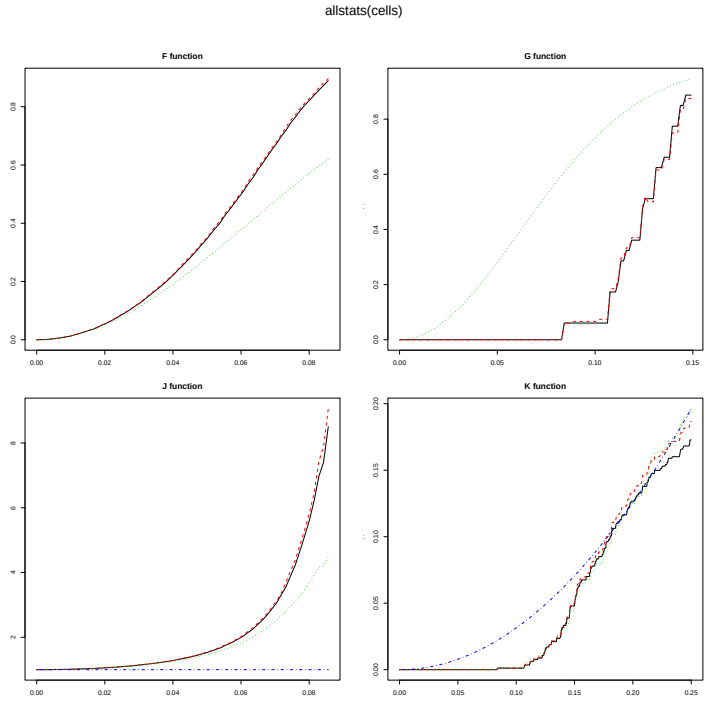
$$J(t) = \frac{\lambda_1}{\lambda_1 + \lambda_2} J_1(t) + \frac{\lambda_2}{\lambda_1 + \lambda_2} J_2(t)$$

where J_1, J_2 are the J-functions of $\mathbf{X}_1, \mathbf{X}_2$ respectively and λ_1, λ_2 are their intensities.

The J function is computed by `Jest`.

The convenient function `allstats` efficiently computes the F, G, J and K functions for a dataset. They can be plotted automatically.

```
> plot(allstats(cells))
```



15.6 Caveats

The use of summary functions for analysing point patterns has become established across wide areas of applied science, following Ripley's influential paper [36] and many subsequent textbooks [17, 19, 21, 43, 38, 39, 42] until quite recently.

There is a tendency to apply them uncritically and exclusively. It's important to remember that

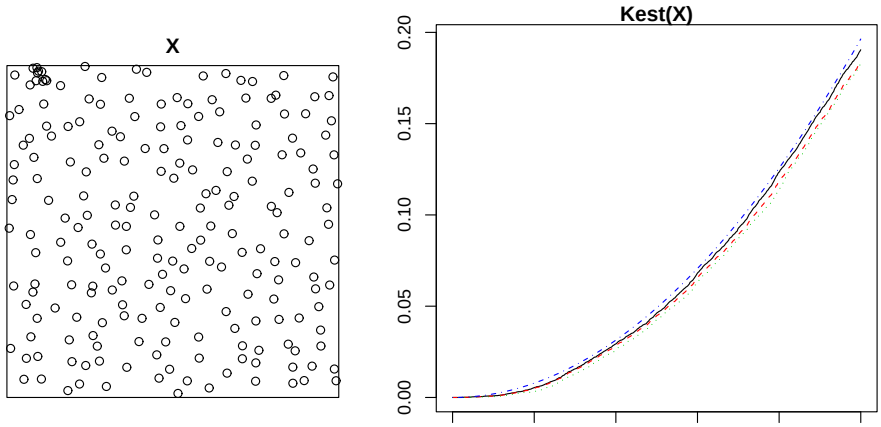
1. the functions F, G and K are defined and estimated under the *assumption that the point process is stationary (homogeneous).*

2. these summary functions *do not completely characterise the process.*

3. if the process is not stationary, deviations between the empirical and theoretical functions (e.g. $\widehat{K}$ and K_{pois}) are not necessarily evidence of interpoint interaction, since they may also be attributable to variations in intensity.

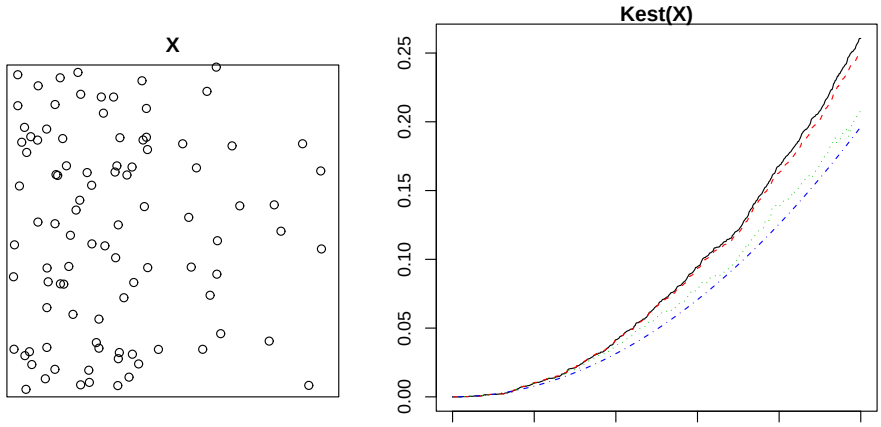
For an example of caveat 2, here is a point process constructed by Baddeley and Silverman [10] which has the same K function as the homogeneous Poisson process:

```
> par(mfrow = c(1, 2))
> X <- rcell(nx = 15)
> plot(X)
> plot(Kest(X))
```



For an example of caveat 3, we generate an inhomogeneous Poisson pattern and apply the ordinary K function estimator. The result appears to show clustering, but this is an artefact of the spatial inhomogeneity.

```
> par(mfrow = c(1, 2))
> X <- rpoispp(function(x, y) {
+     300 * exp(-3 * x)
+ })
> plot(X)
> plot(Kest(X))
```



16 Methods 6: inference using summary statistics

Although summary statistics such as the K-function are intended primarily for exploratory purposes, it is also possible to use them as a basis for parameter estimation and hypothesis testing.

16.1 Envelopes and Monte Carlo tests

The graphical comparison of $\widehat{K}$ with $K_{\rm pois}$, etc, can be formalised in terms of hypothesis testing. The null hypothesis is Complete Spatial Randomness (a homogeneous Poisson process) and the alternative comprises all other processes.

16.1.1 Pointwise Monte Carlo test

Following Besag [14] and Ripley [36, 38], formal hypothesis tests are conducted using the *Monte Carlo test* principle [25, 15] rather than the Neyman-Pearson lemma. Suppose the reference curve is the theoretical K function for a completely random (uniform Poisson) point process. Generate M independent simulations of this process inside the study region W. Compute the estimated K functions for each of these realisations, say $\widehat{K}^{(j)}(r)$ for $j = 1, \ldots, M$. Obtain the pointwise upper and lower envelopes of these simulated curves,

$$\begin{aligned} L(r) &= \min_j \widehat{K}^{(j)}(r) \\ U(r) &= \max_j \widehat{K}^{(j)}(r). \end{aligned}$$

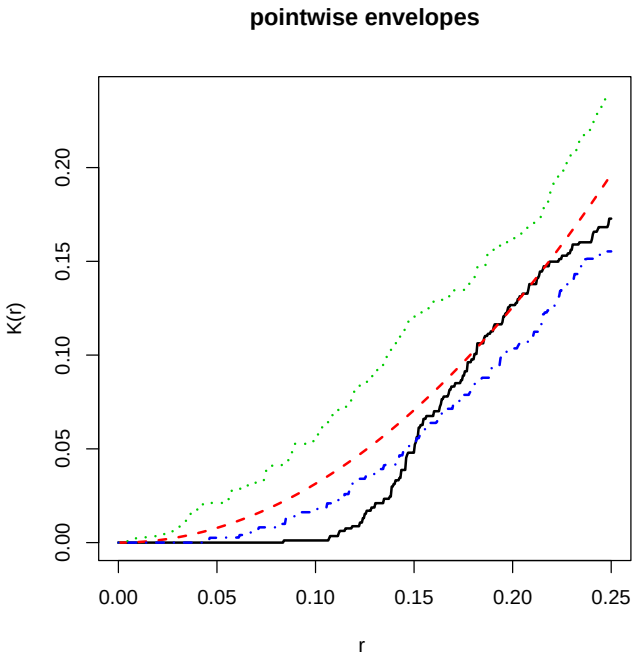
For any fixed value of r, consider the probability that $\widehat{K}(r)$ lies outside the envelope $[L(r), U(r)]$ for the simulated curves. If the data came from a uniform Poisson process, then $\widehat{K}(r)$ and $\widehat{K}^{(1)}(r), \ldots, \widehat{K}^{(M)}(r)$ are statistically equivalent and independent, so this probability is equal to $2/(M+1)$ by symmetry. That is, the test which rejects the null hypothesis of a uniform Poisson process when $\widehat{K}(r)$ lies outside $[L(r), U(r)]$, has exact significance level $\alpha = 2/(M+1)$. Instead of the pointwise maximum and minimum, one could use the pointwise order statistics (the pointwise kth largest and k smallest values) giving a test of exact size $\alpha = 2k/(M+1)$.

16.1.2 Envelopes in `spatstat`

In `spatstat` the function `envelope` computes the pointwise envelopes.

```
> data(cells)
> E <- envelope(cells, Kest, nsim = 39, rank = 1)

> E

Pointwise critical envelopes for K(r)
Obtained from 39 simulations of simulations of CSR
Significance level of pointwise Monte Carlo test: 2/40 = 0.05
Data: cells
Function value object (class 'fv')
for the function r -> K(r)
Entries:
id      label     description
--      -----     -----------
```

```
r       r         distance argument r
obs     obs(r)    function value for data pattern
theo    theo(r)   theoretical value for CSR
lo      lo(r)     lower pointwise envelope of simulations
hi      hi(r)     upper pointwise envelope of simulations
----------------------------------------

Default plot formula:
        . ~ r

Recommended range of argument r: [0, 0.25]

> plot(E, main = "pointwise envelopes")
```



For example if r had been fixed at $r = 0.10$ we would have rejected the null hypothesis of CSR at the 5% level. The value $M = 39$ is the smallest to yield a two-sided test with significance level 5%.

> Tip: A common and dangerous mistake is to misinterpret the simulation envelopes as "confidence intervals" around $\hat{K}$. They cannot be interpreted as a measure of accuracy of the estimated K function! They are the critical values for a test of the hypothesis that $K(r) = \pi r^2$.

16.1.3 Simultaneous Monte Carlo test

Note that the theory of the Monte Carlo test, as presented above, requires that r be fixed in advance. If we plot the envelope and check whether the empirical K function ever wanders outside the envelope, this is equivalent to choosing the value of r in a data-dependent way, and the true significance level is higher (less 'significant').

To avoid this problem we can construct *simultaneous critical bands* which have the property that, under H_0, the probability that $\widehat{K}$ ever wanders outside the critical bands is exactly 5%.

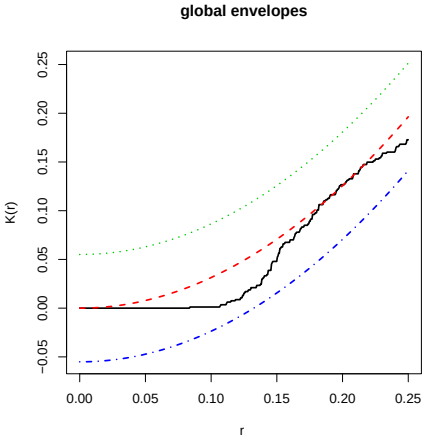
One simple way to achieve this is to compute, for each estimate $\widehat{K}(r)$, its maximum deviation from the Poisson K function, $D = \max_r |\widehat{K}(r) - K_{\rm pois}(r)|$. This is computed for each of the M simulated datasets, and the maximum value $D_{\rm max}$ obtained. Then the upper and lower limits are

$$\begin{aligned} L(r) &= \pi r^2 - D_{\rm max} \\ U(r) &= \pi r^2 + D_{\rm max}. \end{aligned}$$

The estimated K function for the data transgresses these limits if and only if the D-value for the data exceeds $D_{\rm max}$. Under H_0 this occurs with probability $1/(M+1)$. Thus, a test of size 5% is obtained by taking $M = 19$.

```
> E <- envelope(cells, Kest, nsim = 19, rank = 1, global = TRUE)
```

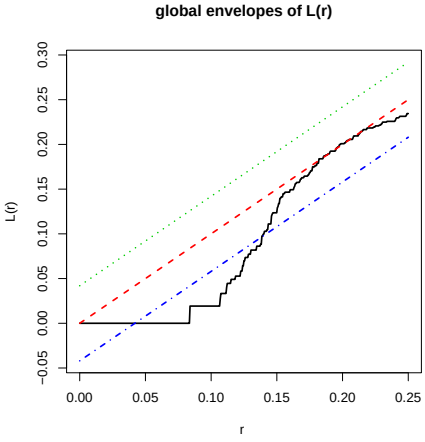
```
> plot(E, main = "global envelopes")
```



A more powerful test is obtained if we (approximately) stabilise the variance, by using the L function in place of K.

```
> E <- envelope(cells, Lest, nsim = 19, rank = 1, global = TRUE)
```

```
> plot(E, main = "global envelopes of L(r)")
```



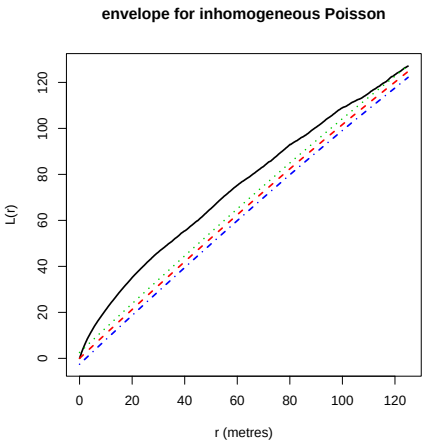
16.1.4 Envelopes for any fitted model

In the explanation above, we assumed that the null hypothesis was CSR (complete spatial randomness, a uniform Poisson process). In fact the Monte Carlo testing rationale can be applied to any point process model serving as a null hypothesis. We simply have to generate simulated realisations from the null hypothesis, and compute the summary function for each simulated realisation.

To simulate from a fitted point process model (object of class `"ppm"`), call the `envelope` function, giving the fitted model as the first argument of `envelope`. Then the simulated patterns will be generated according to this fitted model. The original data point pattern, to which the model was fitted, is stored in the fitted model object; the original data are extracted and the summary function for the data is also computed.

The following code fits an inhomogeneous Poisson process to the Beilschmiedia pattern, then generates simulation envelopes of the L function by simulating from the fitted inhomogeneous Poisson model.

```
> data(bei)
> fit <- ppm(bei, ~elev + grad, covariates = bei.extra)
> E <- envelope(fit, Lest, nsim = 19, global = TRUE, correction = "border")

> plot(E, main = "envelope for inhomogeneous Poisson")
```



16.1.5 Envelopes based on any simulation procedure

Envelopes can also be computed using any user-specified procedure to generate the simulated realisations. This allows us to perform randomisation tests, for example.

The simulation procedure should be encoded as an R expression, which will be evaluated each time a simulation is required. For example if we type

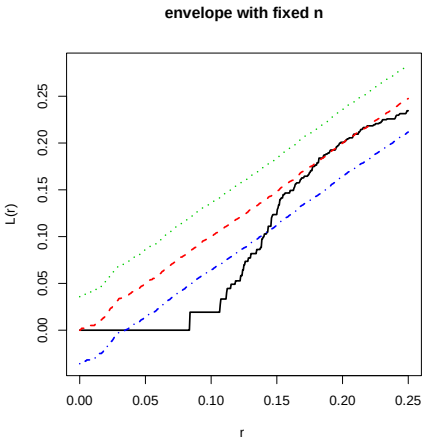
```
> sim <- expression(rpoispp(100))
```

then each time the expression `sim` is evaluated, it will yield a different random outcome of the Poisson process with intensity 100 in the unit square.

This expression should be passed to the `envelope` function as the argument `simulate`.

The following code generates simulation envelopes for the L function based on simulations of CSR which have the same number of points as the data pattern.

```
> data(cells)
> e <- expression(runifpoint(cells$n, cells$window))
> E <- envelope(cells, Lest, nsim = 19, global = TRUE, simulate = e)

> plot(E, main = "envelope with fixed n")
```



16.1.6 Envelopes based on a set of point patterns

Envelopes can also be computed from a user-supplied list of point patterns, instead of the simulated point patterns generated by a chosen simulation procedure.

This improves efficiency and consistency if, for example, we are going to calculate the envelopes of several different summary statistics.

```
> data(cells)
> SimPatList <- list()
> for (i in 1:1000) SimPatList[[i]] <- runifpoint(cells$n)
> EK <- envelope(cells, Kest, simulate = SimPatList, nsim = 1000)
> Ep <- envelope(cells, pcf, simulate = SimPatList, nsim = 1000)
```

16.2 Model-fitting using summary statistics

In the 'method of moments' we estimate a parameter θ by solving

$$\mathbb{E}_\theta[S(\mathbf{X})] = S(\mathbf{x})$$

where $S(\mathbf{x})$ is the observed value of a statistic S for our data $\mathbf{x}$, and the left side is the theoretical mean of S for the model governed by parameter θ.

The analogue for point process models is to fit the model by matching a summary statistic such as the K function to its theoretical value under the model.

16.2.1 Theoretical mean known analytically

In a precious few cases, the K function of a point process is known exactly as an analytic expression in terms of the model parameters. These include many Neyman-Scott processes. For example, the K-function of the Thomas process with parameters $\theta = (\kappa, \mu, \sigma)$ is

$$K_\theta(r) = \pi r^2 + \frac{1}{\kappa}(1 - \exp(-\frac{r^2}{4\sigma^2})). \tag{26}$$

We may thus fit a Thomas model by solving $K_\theta(r) = \widehat{K}(r)$ for some values of r. More efficiently we choose θ to minimise the discrepancy between the two functions over some range $[a, b]$:

$$D = \int_a^b \left| \widehat{K}(r)^q - K_\theta(r)^q \right|^p \, \mathrm{d}r \tag{27}$$

where $0 \le a < b$, and where $p, q > 0$ are indices. This method was originally advocated by Peter Diggle and collaborators, and is now known as the *method of minimum contrast*. See [21].

To fit the Thomas model by minimum contrast to the K function, use `thomas.estK`.

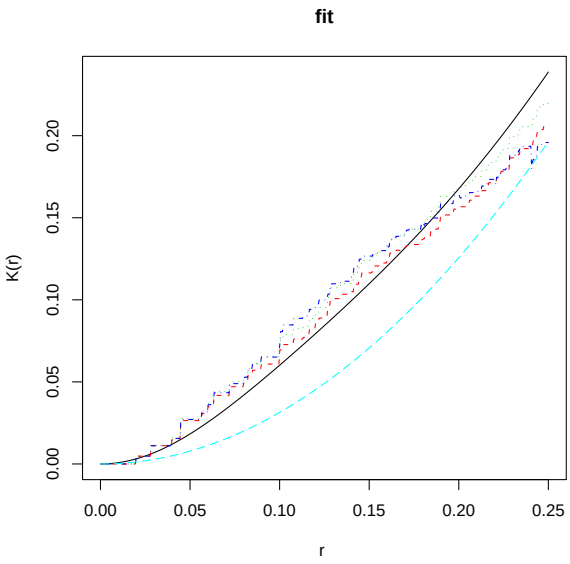
```
> data(redwood)
> fit <- thomas.estK(redwood, c(kappa = 10, sigma2 = 0.1))
```

The second argument to `thomas.estK` gives a set of starting values for the parameters, used in the minimisation search.

The fitted model, `fit`, is an object of class `minconfit`. There are methods for printing and plotting objects of this class.

```
> fit

Minimum contrast fit (object of class "minconfit")
Model: Thomas process
Fitted by matching theoretical K function to Kest(redwood)
Parameters fitted by minimum contrast ($par):
       kappa       sigma2
23.545183910  0.002214530
Derived parameters of Thomas process ($modelpar):
      kappa       sigma          mu
23.54518391  0.04705879  2.63323490
Converged successfully after 139 iterations.
Domain of integration: [ 0 , 0.25 ]
Exponents: p= 2, q= 0.25
```

```
> plot(fit)
```



The plot shows the theoretical K function of the fitted Thomas process (`fit`), three non-parametric estimates of the K function (`iso, trans, border`) and the Poisson K function (`theo`).

Other models can be fitted using `matclust.estK` (Matérn cluster process), `lgcp.estK` (log-Gaussian Cox process), or `mincontrast` (generic fitting algorithm for method of minimum contrast).

16.2.2 Monte Carlo

For the vast majority of point process models, the true K function $K_\theta(r)$ is not known analytically in terms of the parameter θ. In principle we could use Monte Carlo simulation to determine an approximation to $K_\theta(r)$, for any given θ, by generating a large number of simulated realisations of the process with parameter θ, computing the estimated K-function for each realisation, and taking the pointwise sample average. It's possible to do this in `spatstat` using the generic algorithm `mincontrast`. Details are not given here as it is rather fiddly at present, and will change soon.

17 Methods 7: adjusting for inhomogeneity

If a point pattern is known or suspected to be spatially inhomogeneous, then our statistical analysis of the pattern should take account of this inhomogeneity.

17.1 Inhomogeneous K function

There is a modification of the K function that applies to inhomogeneous processes [2]. If $\lambda(u)$ is the true intensity function of the point process $\mathbf{X}$, then the idea is that each point x_i will be weighted by $w_i = 1/\lambda(x_i)$.

The *inhomogeneous K-function* is defined as

$$K_{\mathrm{inhom}}(r) = \mathbb{E}\left[\frac{1}{\lambda(u)}\sum_{x_j \in \mathbf{X}} \frac{1}{\lambda(x_j)} \mathbf{1}\left\{0 < ||u - x_j|| \le r\right\} \;\middle|\; u \in \mathbf{X}\right] \tag{28}$$

assuming that this does not depend on location u. Thus, $\lambda(u)K(r)$ is the expected total 'weight' of all random points within a distance r of the point u, where the 'weight' of a point x_i is $1/\lambda(x_i)$.

If the process is actually homogeneous, then $\lambda(u)$ is constant and $K_{\mathrm{inhom}}(r)$ reduces to the usual K function (21).

It turns out that, for an inhomogeneous Poisson process with intensity function $\lambda(u)$, the inhomogeneous K function is

$$K_{\mathrm{inhom,\ pois}}(r) = \pi r^2 \tag{29}$$

exactly as for the homogeneous case.

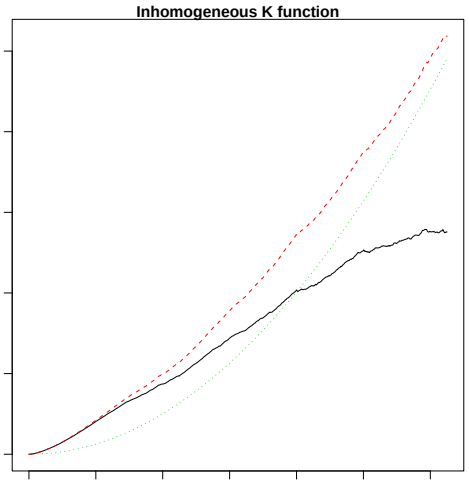
The standard estimators of K can be extended to the inhomogeneous K function:

$$\widehat{K}_{\mathrm{inhom}}(r) = \frac{1}{\mathrm{area}(W)} \sum_i \sum_{j \ne i} \frac{\mathbf{1}\left\{||x_i - x_j|| \le r\right\}}{\widehat{\lambda}(x_i)\widehat{\lambda}(x_j)} e(x_i, x_j; r) \tag{30}$$

where $e(u, v, r)$ is an edge correction weight as before, and $\widehat{\lambda}(u)$ is an estimate of the intensity function $\lambda(u)$.

There remains the question of how to estimate the intensity function $\lambda(u)$. It is usually advisable to obtain the intensity estimate $\widehat{\lambda}(u)$ by fitting a parametric model, to avoid overfitting. Here is an example for the tropical rainforest data, using the covariate data to suggest a model for the intensity.

```
> data(bei)
> fit <- ppm(bei, ~elev + grad, covariates = bei.extra)
> lam <- predict(fit, locations = bei)
> Ki <- Kinhom(bei, lam)
> plot(Ki, main = "Inhomogeneous K function")
```



The plot suggests that, even after accounting for dependence on altitude and slope, the trees still appear to be clustered.

The intensity function $\lambda(u)$ could also be estimated by kernel smoothing the point pattern data. However, notice that the estimator (30) of the inhomogeneous K function depends on the estimated intensity values at the *data points*, $\widehat{\lambda}(x_i)$. These are positively biased estimates of the true values $\lambda(x_i)$. In order to avoid bias, the value $\widehat{\lambda}(x_i)$ should be estimated by kernel smoothing of the point pattern with the point x_i *deleted*. This "leave-one-out" estimator is implemented in `Kinhom` and is invoked when the argument `lambda` is not given:

```
> Ki2 <- Kinhom(bei)
> plot(Ki2, main = "Kinhom using leave-one-out")
```

```
           lty col
bord.modif   1   1
border       2   2
theo         3   3
```

(the smoothing parameter σ can also be controlled.)

The inhomogeneous analogue of the L-function is defined by

$$\widehat{L}_{\mathrm{inhom}}(r) = \sqrt{\frac{\widehat{K}_{\mathrm{inhom}}(r)}{2\pi r}}.$$

This can be computed using `Linhom`. For an inhomogeneous Poisson process, $L_{\mathrm{inhom}}(r) \equiv r$.

The inhomogeneous analogue of the pair correlation function can be defined, similarly to the homogeneous case, as

$$g_{\mathrm{inhom}}(r) = \frac{K'_{\mathrm{inhom}}(r)}{2\pi r}.$$

It has the same interpretation, namely, that $g_{\mathrm{inhom}}(r)$ is the probability of observing a pair of points at certain locations separated by a distance r, divided by the corresponding probability for a Poisson process of the same (inhomogeneous) intensity.

The inhomogeneous pair correlation function is currently computed by calling `Kinhom` followed by `pcf.fv` (which does numerical differentiation):

```
> g <- pcf(Kinhom(bei))
```

17.2 Inhomogeneous cluster models

The inhomogeneous Poisson process was described in Section 11.1. We can also introduce spatial inhomogeneity into any of the non-Poisson models described in Section 14.

In the case of Poisson cluster processes (Section 14.1) we can introduce inhomogeneity in either the parent process or the offspring processes.

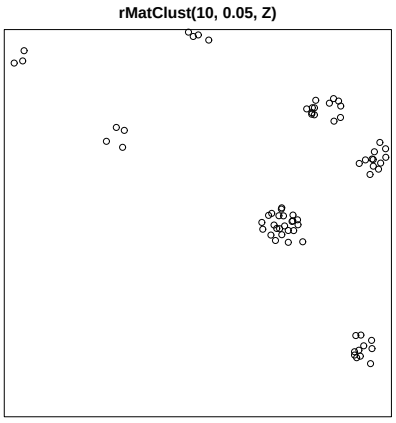
To make the *parents* inhomogeneous, we simply generate the parent points from an inhomogeneous Poisson process with some intensity function $\kappa(u)$.

To make the *clusters* inhomogeneous, we use a clever construction by Waagepetersen [45]. For a parent point at location (x_0, y_0), the offspring are generated from a Poisson process with intensity $\beta(x, y) = \mu(x, y) f(x - x_0, y - y_0)$, where $f(u, v)$ is either the Gaussian probability density (for the Thomas process) or the uniform probability density in a disc (for the Matérn cluster process), and $\mu(x, y)$ is the *reference* or *modulating* intensity. The number of offspring from a given parent (x_0, y_0) is a Poisson random variable with mean

$$B(x_0, y_0) = \int \beta(x, y)\, \mathrm{d}x\, \mathrm{d}y = \int f(x - x_0, y - y_0)\mu(x, y)\, \mathrm{d}x\, \mathrm{d}y.$$

The simulation algorithms `rMatClust` and `rThomas` allow these options. If the parent intensity parameter `kappa` is given as a `function(x,y)` or a pixel image, then the parents are Poisson with inhomogeneous intensity `kappa`. If the offspring mean parameter `mu` is given as a `function(x,y)` or a pixel image, then this determines an inhomogeneous reference density for the clusters.

```
> Z <- as.im(function(x, y) {
+     6 * exp(2 * x - 1)
+ }, owin())
> plot(rMatClust(10, 0.05, Z))
```



17.3 Fitting inhomogeneous models by minimum contrast

Minimum contrast methods can be applied to inhomogeneous point process models.

In principle we could fit any model (homogeneous or inhomogeneous) by the method of minimum contrast using any summary statistic. However, the method works best when we have an exact formula for the true value of the summary function for the model, expressed as a function of the parameters of the model.

Waagepetersen [45] pointed out that, if we take a Thomas process or Matérn cluster process (or in general a Neyman-Scott process) with **homogeneous** parent intensity κ and **inhomogeneous** cluster reference density $\mu(u)$, then the overall intensity of the process is

$$\lambda(u) = \kappa\,\mu(u)$$

and the *inhomogeneous* K-function is the same as it would be if μ were constant.

Thus, we can fit a Thomas process or Matérn cluster process with inhomogeneous clusters as follows:

1. estimate the inhomogeneous intensity $\lambda(u)$ of the process.
2. derive an estimate of the inhomogeneous K-function.
3. use the method of minimum contrast to estimate the parent intensity κ and the cluster scale parameter (Gaussian standard deviation or disc radius), exactly as we would in the homogeneous case.

Here is an application to the rainforest data.

```
> data(bei)
> fit <- ppm(bei, ~elev + grad, covariates = bei.extra)
> lam <- predict(fit, locations = bei)
> Ki <- Kinhom(bei, lam)
> thomas.estK(Ki, c(kappa = 4e-04, sigma2 = 1))

Minimum contrast fit (object of class "minconfit")
Model: Thomas process
Fitted by matching theoretical K function to Ki
Parameters fitted by minimum contrast ($par):
       kappa       sigma2
4.267423e-04 2.941906e+01
Derived parameters of Thomas process ($modelpar):
       kappa        sigma           mu
0.0004267423 5.4239342345           NA
Converged successfully after 113 iterations.
Domain of integration: [ 0 , 125 ]
Exponents: p= 2, q= 0.25
```

18 Gibbs models

One way to construct a statistical model (in any field of statistics) is to write down its probability density. Advantages of doing this are:

- the functional form of the density reflects its probabilistic properties.
- terms or factors in the density often have an interpretation as 'components' of the model.
- it is easy to introduce terms that represent the dependence of the model on covariates, etc.

This approach is useful provided the density *can* be written down, and provided the density is tractable.

Spatial point process models that are constructed by writing down their probability densities are called **'Gibbs processes'**. Good references on Gibbs point processes are [43, 18].

18.1 Probability densities

It is possible to define probability densities for spatial point processes that live inside a bounded window W.

The probability density will be a function $f(\mathbf{x})$ defined for each finite configuration $\mathbf{x} = \{x_1, \ldots, x_n\}$ of points $x_i \in W$ for any $n \geq 0$. Notice that the number of points n is not fixed, and may be zero. Apart from this peculiarity, probability densities for point processes behave much like probability densities in more familiar contexts.

That's all you need to know for applications. **If you're interested in the mathematical technicalities, read on; otherwise, skip to section 18.2.**

A point process $\mathbf{X}$ inside W is defined to have probability density f if and only if, for any nonnegative integrable function h,

$$\mathbb{E}[h(\mathbf{X})] = e^{-|W|} h(\emptyset) f(\emptyset) + e^{-|W|} \sum_{n=1}^{\infty} \frac{1}{n!} \int_W \cdots \int_W h(\{x_1, \ldots, x_n\}) f(\{x_1, \ldots, x_n\}) \, \mathrm{d}x_1 \cdots \mathrm{d}x_n \tag{31}$$

where $|W|$ denotes the area of W.

In particular, the probability that $\mathbf{X}$ contains exactly n points is

$$p_n = \mathbb{P}\{n(\mathbf{X}) = n\} = \frac{e^{-|W|}}{n!} \int_W \cdots \int_W f(\{x_1, \ldots, x_n\}) \, \mathrm{d}x_1 \cdots \mathrm{d}x_n$$

for $n \geq 1$ and $p_0 = \mathbb{P}\{n(\mathbf{X}) = 0\} = e^{-|W|} f(\emptyset)$. Given that there are exactly n points, the conditional joint density of the locations $x_1, \ldots, x_n$ is $f(\{x_1, \ldots, x_n\})/p_n$.

18.2 Poisson processes

The uniform Poisson process with intensity 1 has probability density $f(\mathbf{x}) \equiv 1$.

The uniform Poisson process in W with intensity λ has probability density

$$f(\mathbf{x}) = \alpha \, \lambda^{n(\mathbf{x})} \tag{32}$$

where $n(\mathbf{x})$ is the number of points in the configuration $\mathbf{x}$, and the constant α is

$$\alpha = e^{(1-\lambda)|W|}.$$

The inhomogeneous Poisson process in W with intensity function $\lambda(u)$ has probability density

$$f(\mathbf{x}) = \alpha \prod_{i=1}^{n} \lambda(x_i). \tag{33}$$

where the constant α is

$$\alpha = \exp\left[\int_W (1 - \lambda(u))\,\mathrm{d}u\right].$$

The densities (32) and (33) are products of terms associated with individual points x_i. This reflects the conditional independence property (PP4) of the Poisson process.

18.3 Pairwise interaction models

In order to construct spatial point processes which exhibit interpoint interaction (stochastic dependence between points), we need to introduce terms in the density that depend on more than one point. The simplest are *pairwise interaction models*, which have probability densities of the form

$$f(\mathbf{x}) = \alpha \left[\prod_{i=1}^{n(\mathbf{x})} b(x_i)\right] \left[\prod_{i<j} c(x_i, x_j)\right] \tag{34}$$

where α is a normalising constant, $b(u),\ u \in W$ is the 'first order' term, and $c(u, v),\ u, v \in W$ is the 'second order' or 'pairwise interaction' term. The pairwise interaction term introduces dependence between points. The interaction function must be symmetric, $c(u, v) = c(v, u)$. In principle we are free to choose any functions b and c, provided the resulting density is integrable (the right side of (31) should be finite when $h \equiv 1$).

18.3.1 Hard core process

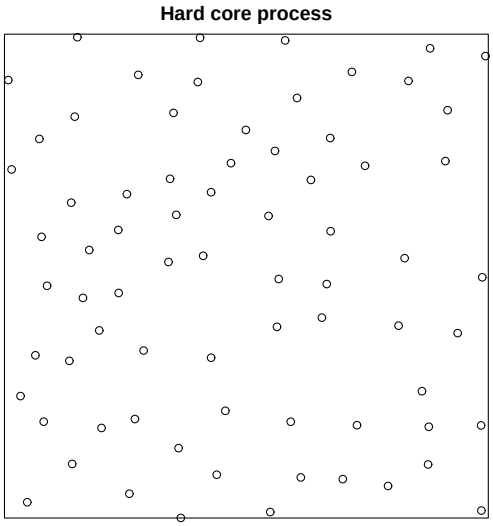
If we take $b(u) \equiv \beta$ and

$$c(u, v) = \begin{cases} 1 & \text{if } ||u - v|| > r \\ 0 & \text{if } ||u - v|| \le r \end{cases} \tag{35}$$

where $||u - v||$ denotes the distance between u and v, and $r > 0$ is a fixed distance, then the density becomes

$$f(\mathbf{x}) = \begin{cases} \alpha\beta^{n(\mathbf{x})} & \text{if } ||x_i - x_j|| > r \text{ for all } i \ne j \\ 0 & \text{otherwise} \end{cases}$$

This is the density of the Poisson process of intensity β in W conditioned on the event that no two points of the pattern lie closer than r units apart. It is known as the (classical) *hard core process*.



18.3.2 Strauss process

Generalising the hard core process, suppose we take $b(u) \equiv \beta$ and

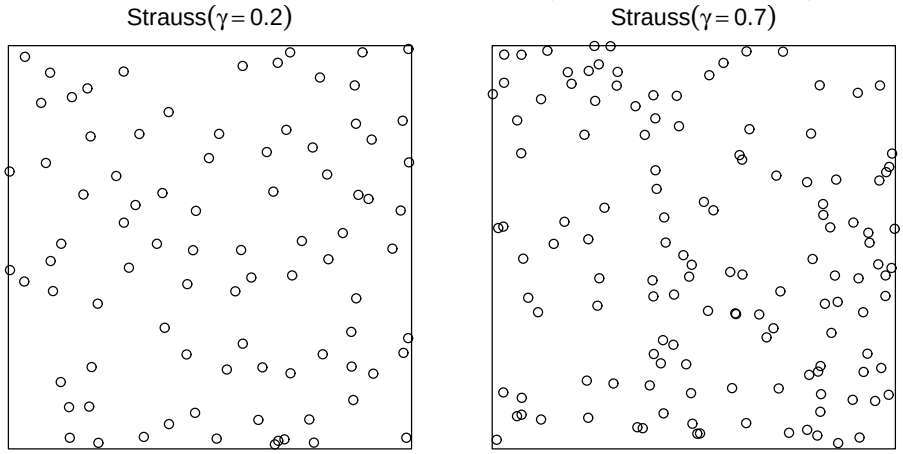
$$c(u,v) = \begin{cases} 1 & \text{if } ||u-v|| > r \\ \gamma & \text{if } ||u-v|| \leq r \end{cases} \tag{36}$$

where γ is a parameter. Then the density becomes

$$f(\mathbf{x}) = \alpha\beta^{n(\mathbf{x})}\gamma^{s(\mathbf{x})} \tag{37}$$

where $s(\mathbf{x})$ is the number of pairs of distinct points in $\mathbf{x}$ that lie closer than r units apart.

The parameter γ controls the 'strength' of interaction between points. If $\gamma = 1$ the model reduces to a Poisson process with intensity β. If $\gamma = 0$ the model is a hard core process. For values $0 < \gamma < 1$, the process exhibits inhibition (negative association) between points.



For $\gamma > 1$, the density (37) is not integrable. Hence the Strauss process is defined only for $0 \leq \gamma \leq 1$ and is a model for inhibition between points. This is typical of most Gibbs models.

18.3.3 Other pairwise interaction models

Other pairwise interactions that are considered in `spatstat` include the *Strauss-hard core* interaction (with hard core distance $h > 0$ and interaction distance $r > h$)

$$c(u,v) = \begin{cases} 0 & \text{if } ||u-v|| \le h \\ \gamma & \text{if } h < ||u-v|| \le r \\ 1 & \text{if } ||u-v|| > r \end{cases},$$

the *soft-core* interaction (with scale $\sigma > 0$ and index $0 < \kappa < 1$)

$$c(u,v) = \left(\frac{\sigma}{||u-v||}\right)^{2/\kappa},$$

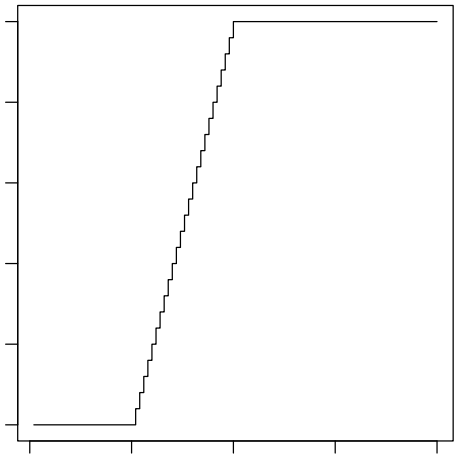
the *Diggle-Gates-Stibbard* interaction (with interaction range ρ)

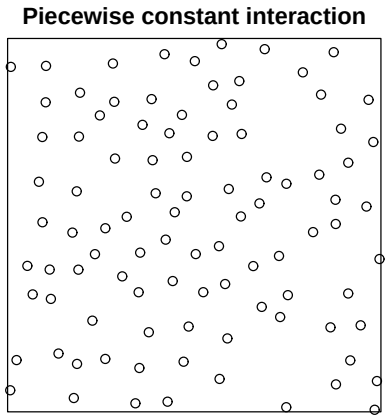
$$c(u,v) = \begin{cases} \sin\left(\frac{\pi||u-v||}{2\rho}\right)^2 & \text{if } ||u-v|| \le \rho \\ 1 & \text{if } ||u-v|| > \rho \end{cases},$$

the *Diggle-Gratton* interaction (with hard core distance δ, interaction distance ρ and index κ)

$$c(u,v) = \begin{cases} 0 & \text{if } ||u-v|| \le \delta \\ \left(\frac{||u-v||-\delta}{\rho-\delta}\right)^{\kappa} & \text{if } \delta < ||u-v|| \le \rho \\ 1 & \text{if } ||u-v|| > \rho \end{cases},$$

and the general *piecewise constant* interaction in which $c(||u-v||)$ is a step function of $||u-v||$.





18.4 Higher-order interactions

There are some useful Gibbs point process models which exhibit interactions of higher order, that is, in which the probability density has contributions from m-tuples of points, where $m > 2$.

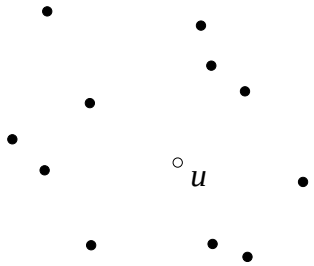
One example is the *area-interaction* or Widom-Rowlinson process [11] with probability density

$$f(\mathbf{x}) = \alpha\beta^{n(\mathbf{x})}\gamma^{-A(\mathbf{x})} \tag{38}$$

where α is the normalising constant, $\beta > 0$ is an intensity parameter, and $\gamma > 0$ is an interaction parameter. Here $A(\mathbf{x})$ denotes the area of the region obtained by drawing a disc of radius r centred at each point x_i, and taking the union of these discs. The value $\gamma = 1$ again corresponds to a Poisson process, while $\gamma < 1$ produces a regular process and $\gamma > 1$ a clustered process. This process has interactions of all orders. It can be used as a model for moderate regularity or clustering.

18.5 Conditional intensity

The main tool for analysing a Gibbs point process is its *conditional intensity* $\lambda(u, \mathbf{X})$. Intuitively this determines the conditional probability of finding a point of the process at the location u given complete information about the rest of the process. For formal definitions see [18]. Informally, the conditional probability of finding a point of the process inside an infinitesimal neighbourhood of the location u, given the complete point pattern at all other locations, is $\lambda(u, \mathbf{X})\,\mathrm{d}u$.



For point processes in a bounded window, the conditional intensity at a location u given the configuration $\mathbf{x}$ is related to the probability density f by

$$\lambda(u, \mathbf{x}) = \frac{f(\mathbf{x} \cup \{u\})}{f(\mathbf{x})} \tag{39}$$

(for $u \notin \mathbf{x}$), the ratio of the probability densities for the configuration $\mathbf{x}$ with and without the point u added.

The homogeneous Poisson process with intensity λ has conditional intensity

$$\lambda(u, \mathbf{x}) = \lambda$$

while the inhomogeneous Poisson process with intensity function $\lambda(u)$ has conditional intensity

$$\lambda(u, \mathbf{x}) = \lambda(u)$$

. The conditional intensity for a Poisson process does not depend on the configuration $\mathbf{x}$, because the points of a Poisson process are independent.

For the general pairwise interaction process (34) the conditional intensity is

$$\lambda(u, \mathbf{x}) = b(u) \prod_{i=1}^{n(\mathbf{x})} c(u, x_i). \tag{40}$$

For the hard core process,

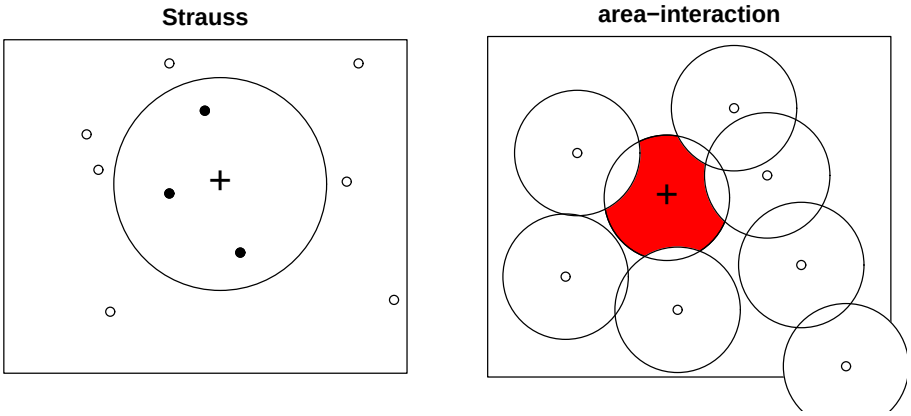
$$\lambda(u, \mathbf{x}) = \begin{cases} \beta & \text{if } ||u - x_i|| > r \text{ for all } i \\ 0 & \text{otherwise} \end{cases} \tag{41}$$

which has the nice interpretation that a point u is either 'permitted' or 'not permitted' depending on whether it satisfies the hard core requirement.

For the Strauss process

$$\lambda(u, \mathbf{x}) = \beta \gamma^{t(u, \mathbf{x})} \tag{42}$$

where $t(u, \mathbf{x}) = s(\mathbf{x} \cup \{u\}) - s(\mathbf{x})$ is the number of points of $\mathbf{x}$ that lie within a distance r of the location u. For $\gamma < 1$, this has the interpretation that a random point is less likely to occur at the location u if there are many points in the neighbourhood.



For the area-interaction process,

$$\lambda(u, \mathbf{x}) = \beta \gamma^{-B(u,\mathbf{x})} \tag{43}$$

where $B(u, \mathbf{x}) = A(\mathbf{x} \cup \{u\}) - A(\mathbf{x})$ is the area of that part of the disc of radius r centred on u that is not covered by discs of radius r centred at the other points $x_i \in \mathbf{x}$. If the points represent trees or plants, we may imagine that each tree takes nutrients and water from the soil inside a circle of radius r. Then we may interpret $B(u, \mathbf{x})$ as the area of the 'unclaimed zone' where a new plant at location u would be able to draw nutrients and water without competition from other plants. For $\gamma < 1$ we can interpret (43) as saying that a random point is less likely to occur when the unclaimed area is small.

The conditional intensity of a point process determines the probability density, through (39). Hence we can use the conditional intensity to define a point process. The conditional intensity is the preferred modelling tool for Gibbs processes: it has a direct interpretation, and it is easier to handle than the probability density.

18.6 Simulating Gibbs models

Gibbs models can be simulated by Markov chain Monte Carlo algorithms. Indeed, MCMC algorithms were invented to simulate Gibbs processes [32, 37].

In brief, these algorithms simulate a Markov chain whose states are point patterns. The chain is designed so that its equilibrium distribution is the distribution of the point process we want to simulate. If the chain were run for an infinite time, the state would converge in distribution to the desired point process. In practice the chain is run for a long finite time. Further details are beyond the scope of this workshop; consult [33, 34] for more information.

Currently `spatstat` offers the function `rmh` which simulates Gibbs processes using the Metropolis-Hastings algorithm.

```
> rmh(model, start, control)
```

- `model` determines the point process model to be simulated (see `help(rmhmodel)`).
- `start` determines the initial state of the Markov chain (see `help(rmhstart)`).
- `control` specifies control parameters for running the Markov chain, such as the number of iteration steps (see `help(rmhcontrol)`).

In the simplest uses of `rmh`, the three arguments are lists of parameter values. To generate a simulated realisation of the Strauss process with parameters $\beta = 2, \gamma = 0.7, r = 0.7$ in a square of side 10,

```
> mo <- list(cif = "strauss", par = c(beta = 2, gamma = 0.2, r = 0.7),
+     w = square(10))
> X <- rmh(model = mo, start = list(n.start = 42), control = list(nrep = 1e+06))
```

The other arguments specify a random initial state of 42 points, and that the algorithm shall be run for a million iterations.

19 Methods 8: fitting Gibbs models

19.1 Maximum pseudolikelihood

Maximum likelihood estimation is intractable for most point process models. At the very least it requires Monte Carlo simulation to evaluate the likelihood (or the score and the Fisher information).

A workable alternative, at least for investigative purposes, is to maximise the log *pseudolikelihood*

$$\log \mathrm{PL}\,(\theta; \mathbf{x}) = \sum_i \log \lambda(x_i; \mathbf{x}) - \int_W \lambda(u, \mathbf{x})\,\mathrm{d}u. \tag{44}$$

You may recognise this as being very similar to the likelihood (4) of the Poisson process. In general it is not a likelihood, but the analogue of the score equation

$$\frac{\partial}{\partial \theta} \log \mathrm{PL}\,(\theta) = 0$$

is an unbiased estimating equation. Thus the maximum pseudolikelihood estimator is asymptotically unbiased, consistent and asymptotically normal under appropriate conditions.

The main advantage of maximum pseudolikelihood is that, at least for popular Gibbs models, the conditional intensity $\lambda(u, \mathbf{x})$ is easily computable, so that the pseudolikelihood is easy to compute and to maximise. The main disadvantage is the bias and inefficiency of maximum pseudolikelihood in small samples.

More computationally-intensive estimation procedures typically use the maximum pseudolikelihood estimate as their initial guess. We are implementing such procedures in `spatstat` as well.

19.2 Fitting Gibbs models in `spatstat`

We have already met the function `ppm` for fitting Poisson point process models. In fact this function will fit a wide class of Gibbs models.

`ppm` contains an implementation of the algorithm of Baddeley and Turner [3] for maximum pseudolikelihood (which extends the Berman-Turner device for Poisson processes to a general Gibbs process). The conditional intensity of the model, $\lambda_\theta(u, \mathbf{x})$, must be loglinear in the parameters θ:

$$\log \lambda_\theta(u, \mathbf{x}) = \theta \cdot S(u, \mathbf{x}), \tag{45}$$

generalising (5), where $S(u, \mathbf{x})$ is a real-valued or vector-valued function of location u and configuration $\mathbf{x}$. Parameters θ appearing in the loglinear form (45) are called 'regular' parameters, and all other parameters are 'irregular' parameters. For example, the Strauss process conditional intensity (42) can be recast as

$$\log \lambda(u, \mathbf{x}) = \log \beta + (\log \gamma) t(u, \mathbf{x})$$

so that $\theta = (\log \beta, \log \gamma)$ are regular parameters, but the interaction distance r is an irregular parameter (technically called a 'bloody nuisance parameter').

In `spatstat` we split the conditional intensity into first-order and higher-order terms:

$$\log \lambda_\theta(u, \mathbf{x}) = \eta \cdot S(u) + \varphi \cdot V(u, \mathbf{x}). \tag{46}$$

The 'first order term' $S(u)$ describes spatial inhomogeneity and/or covariate effects. The 'higher order term' $V(u, \mathbf{x})$ describes interpoint interaction.

The model with conditional intensity (46) is fitted by calling `ppm` in the form

```
ppm(X, ~ terms, V)
```

The first argument `X` is the point pattern dataset. The second argument `~terms` is a model formula, specifying the first order term $S(u)$ in (46), in the manner described in Section 11. Thus the first order term $S(u)$ in (46) may take very general forms.

The third argument `V` is an object of the special class `"interact"` which describes the interpoint interaction term $V(u, \mathbf{x})$ in (46). It may be compared to the 'family' argument which determines the distribution of the responses in a linear model or generalised linear model. Only a limited number of canned interactions are available in `spatstat`, because they must be constructed carefully to ensure that the point process exists.

To fit the Strauss process to the cells data using `ppm`,

```
> data(cells)
> ppm(cells, ~1, Strauss(r = 0.1))

Stationary Strauss process

First order term:
    beta
294.2333

Interaction: Strauss process
interaction distance:        0.1
Fitted interaction parameter gamma:        0.0128

Relevant coefficients:
Interaction
  -4.359277
```

Here `Strauss` is a special function that creates an 'interaction' object (class `"interact"`) describing the interaction structure of the Strauss process. Notice that we had to specify the value of the irregular parameter r (more about that later).

To fit the inhomogeneous Strauss process with conditional intensity

$$\lambda(u, \mathbf{x}) = b(u)\gamma^{t(u,\mathbf{x})}$$

where, say, $b(u)$ is loglinear in the Cartesian coordinates,

$$\log b((x, y)) = \beta_0 + \beta_1 x + \beta_2 y$$

we simply type

```
> ppm(cells, ~x + y, Strauss(r = 0.1))

Nonstationary Strauss process

Trend formula: ~x + y

Fitted coefficients for trend formula:
(Intercept)           x           y
  5.7460724   0.1465176  -0.2724205
```

```
Interaction: Strauss process
interaction distance:        0.1
Fitted interaction parameter gamma:        0.0128

Relevant coefficients:
Interaction
  -4.357253
```

To fit an inhomogeneous Strauss process with log-quadratic first order term,

```
> ppm(cells, ~polynom(x, y, 2), Strauss(r = 0.1))

Nonstationary Strauss process

Trend formula: ~polynom(x, y, 2)

Fitted coefficients for trend formula:
          (Intercept)   polynom(x, y, 2)[x]   polynom(x, y, 2)[y]
             3.019133             11.064005              6.154949
polynom(x, y, 2)[x^2] polynom(x, y, 2)[x.y] polynom(x, y, 2)[y^2]
            -9.853849             -1.761367             -5.579568

Interaction: Strauss process
interaction distance:        0.1
Fitted interaction parameter gamma:        0.0071

Relevant coefficients:
Interaction
  -4.945833
```

19.3 Interpoint interactions

Instead of `Strauss` we may use any of the following functions to create an interaction:

`Poisson()`	the Poisson point process (the default)
`Strauss()`	the Strauss process
`StraussHard()`	the Strauss/hard core point process
`Softcore()`	pairwise interaction, soft core potential
`PairPiece()`	pairwise interaction, piecewise constant
`DiggleGratton()`	Diggle-Gratton potential
`LennardJones()`	Lennard-Jones potential
`Pairwise()`	pairwise interaction, user-supplied potential
`AreaInter()`	area-interaction process
`Geyer()`	Geyer's saturation process
`Saturated()`	Saturated pair model, user-supplied potential
`OrdThresh()`	Ord process, threshold potential
`Ord()`	Ord model, user-supplied potential

(There are two additional ones for multitype point processes, described in section 25.3.2.)

The area-interaction model and the Geyer saturation model are quite handy, as they can be used to model both clustering and regularity.

```
> data(redwood)
> ppm(redwood, ~1, Geyer(r = 0.07, sat = 2))


Stationary Geyer saturation process

First order term:
   beta
17.0143

Interaction: Geyer saturation process
interaction distance:        0.07
 saturation parameter:        2
Fitted interaction parameter gamma:        2.3509

Relevant coefficients:
Interaction
  0.8547814


> ppm(redwood, ~1, AreaInter(r = 0.03))


Stationary Area-interaction process

First order term:
    beta
571.5617

Interaction: Area-interaction process
disc radius:        0.03
Fitted interaction parameter eta:        19.11

Relevant coefficients:
Interaction
   2.950212
```

For more detailed explanation of modelling, see [5].

19.4 Fitted point process models

The result of the `ppm` call is an object of class `"ppm"` ('point process model'). This is very closely analogous to a fitted linear model (`lm`) or fitted generalised linear model (`glm`).

Standard R operations that are defined for fitted point process models (i.e. that have methods for the class `"ppm"`) include:

`print`	print basic information
`summary`	print detailed summary information
`plot`	plot the fitted (conditional) intensity
`predict`	fitted (conditional) intensity
`fitted`	fitted (conditional) intensity at data points
`update`	re-fit the model
`coef`	extract the fitted coefficient vector $\widehat{\theta}$
`vcov`	variance-covariance matrix of $\widehat{\theta}$
`anova`	analysis of deviance
`logLik`	evaluate log-pseudolikelihood

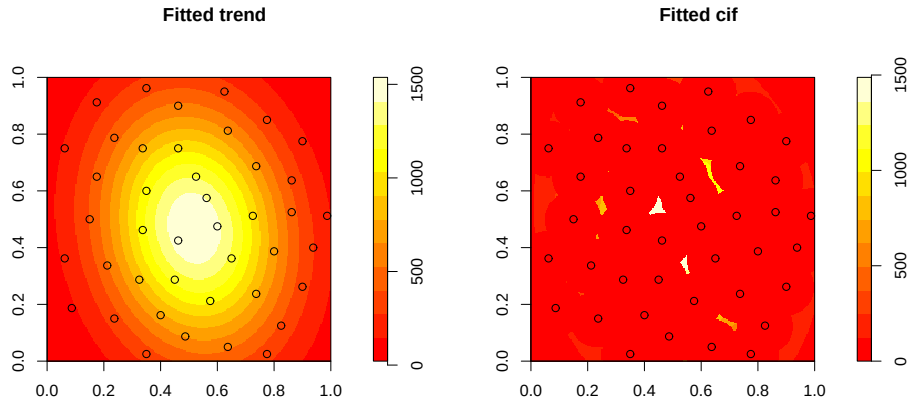
(the methods for `anova` and `vcov` are only available for Poisson models).

Plotting a fitted model generates a series of image and contour plots of

- the fitted first order term $\exp(\hat{\eta} \cdot S(u))$
- the fitted conditional intensity $\lambda_{\hat{\theta}}(u, \mathbf{x})$ evaluated for the data pattern $\mathbf{x}$

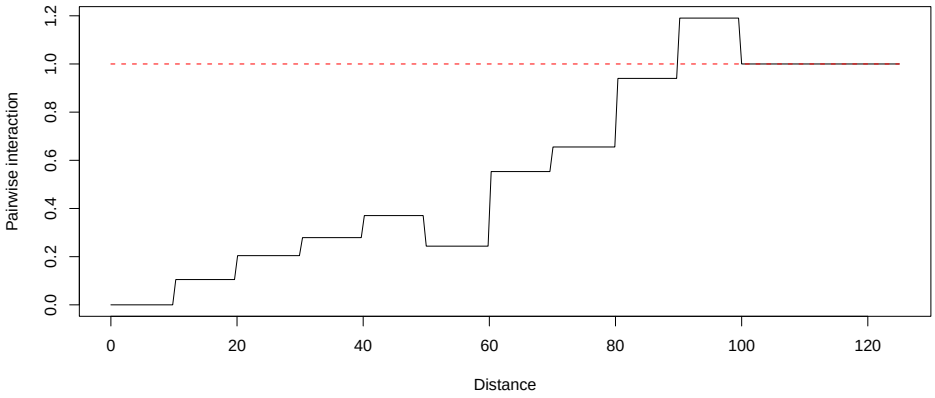
For Poisson models, the two plots are equivalent, and give the fitted intensity function.

```
> fit <- ppm(cells, ~polynom(x, y, 2), Strauss(r = 0.1))
> par(mfrow = c(1, 2))
> plot(fit, how = "image", ngrid = 256)
```



For non-Poisson models, it is also possible to extract and plot the interpoint interaction function, using `fitin`.

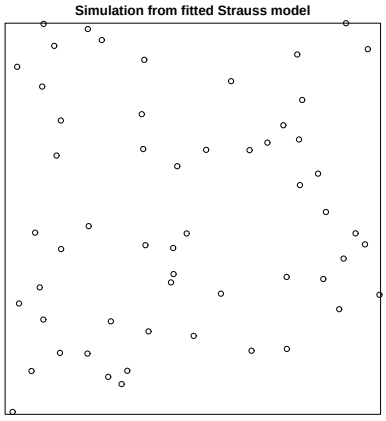
```
> model <- ppm(X, ~1, PairPiece(seq(10, 100, by = 10)))
> f <- fitin(model)
> plot(f)
```



19.5 Simulation from fitted models

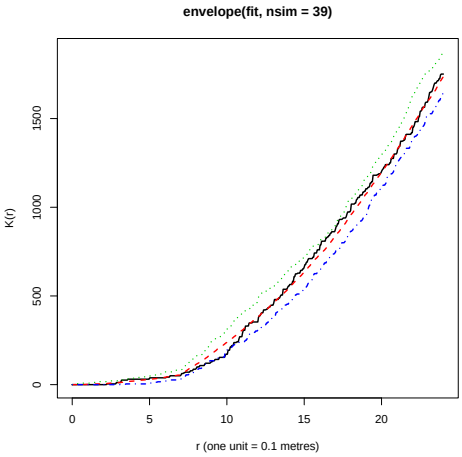
A fitted Gibbs model can also be simulated automatically using `rmh`.

```
> fit <- ppm(swedishpines, ~1, Strauss(r = 7))
> Xsim <- rmh(fit)
> plot(Xsim, main = "Simulation from fitted Strauss model")
```



The `envelope` command will also generate simulation envelopes for a fitted model.

```
> plot(envelope(fit, nsim = 39))
```



19.6 Dealing with nuisance parameters

Irregular parameters, such as the interaction radius r in the Strauss process, cannot be estimated directly using `ppm`. Indeed the statistical theory for estimating such parameters is unclear.

For some special cases, a maximum likelihood estimator of the nuisance parameter is available. For example, for the 'hard core process' (Strauss process with interaction parameter $\gamma = 0$) with interaction radius r, the maximum likelihood estimator is the minimum nearest-neighbour distance. Thus the following is a reasonable approach to the `cells` dataset:

```
> rhat <- min(nndist(cells))
> rhat <- rhat * 0.99999
> ppm(cells, ~1, Strauss(r = rhat))

Stationary Strauss process

First order term:
    beta
168.2692

Interaction: Strauss process
interaction distance:        0.0836293018068393
Fitted interaction parameter gamma:        0

Relevant coefficients:
Interaction
  -19.29955
```

The analogue of profile likelihood, *profile pseudolikelihood*, provides a general solution which may or may not perform well. If $\theta = (\phi, \eta)$ where ϕ denotes the nuisance parameters and η the regular parameters, define the profile log pseudolikelihood by

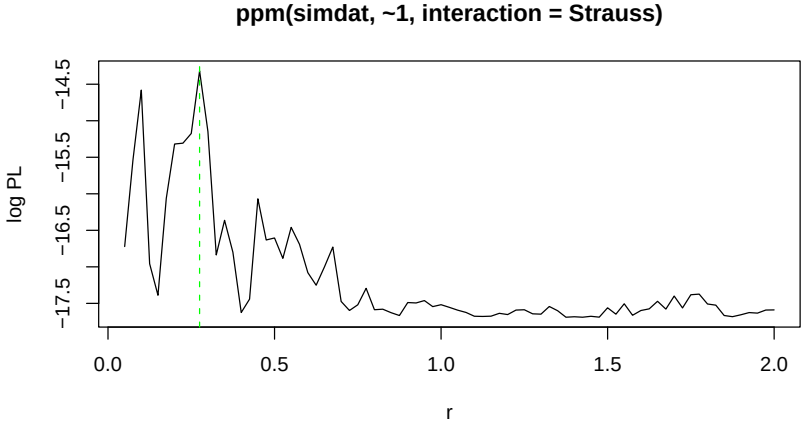
$$\mathrm{PLP}(\phi, \mathbf{x}) = \max_{\eta} \log \mathrm{PL}\left((\phi, \eta); \mathbf{x}\right).$$

The right hand side can be computed, for each fixed value of ϕ, by the algorithm `ppm`. Then we just have to maximise $\mathrm{PLP}(\phi)$ over ϕ. This is done by the command `profilepl`:

```
> data(simdat)
> df <- data.frame(r = seq(0.05, 2, by = 0.025))
> pfit <- profilepl(df, Strauss, simdat, ~1)

> pfit

Profile log pseudolikelihood values
for model:        ppm(simdat, ~1, interaction = Strauss)
fitted with rbord= 2
Interaction: Strauss
 with irregular parameter 'r' in [0.05, 2]
Optimum value of irregular parameter: r = 0.275
```

The result is an object of class `profilepl` containing the profile log pseudolikelihood function, the optimised value of the irregular parameter r, and the final fitted model. To plot the profile log pseudolikelihood,

```
> plot(pfit)
```



To extract the final fitted model,

```
> pfit$fit

Stationary Strauss process

First order term:
    beta
2.583110

Interaction: Strauss process
interaction distance:        0.275
Fitted interaction parameter gamma:        0.5631

Relevant coefficients:
Interaction
 -0.5743608
```

There is a `summary` method for these objects as well.

19.7 Improvements over maximum pseudolikelihood

Maximum pseudolikelihood is quick and dirty. There are statistically more efficient alternatives, but they are computationally intensive.

Currently we have implemented the easiest of these alternatives, the Huang-Ogata [27] one-step approximation to maximum likelihood. Starting from the maximum pseudolikelihood estimate $\hat{\theta}_{PL}$, we simulate M independent realisations of the model with parameters $\hat{\theta}_{PL}$, evaluate the canonical sufficient statistics, and use them to form estimates of the score and Fisher information at $\theta = \hat{\theta}_{PL}$. Then we take one Newton-Raphson step, updating the value of θ. The rationale is that the log-likelihood is approximately quadratic in a neighbourhood of the maximum pseudolikelihood estimator, so that one Newton-Raphson step is almost enough.

To use the Huang-Ogata method instead of maximum pseudolikelihood, add the argument `method="ho"`.

```
> fit <- ppm(simdat, ~1, Strauss(r = 0.275), method = "ho")

> fit

Stationary Strauss process

First order term:
    beta
2.500546

Interaction: Strauss process
interaction distance:        0.275
Fitted interaction parameter gamma:        0.6951

Relevant coefficients:
Interaction
 -0.3637451

> vcov(fit)

             [,1]        [,2]
[1,]  0.01070257 -0.01264063
[2,] -0.01264063  0.03635432
```

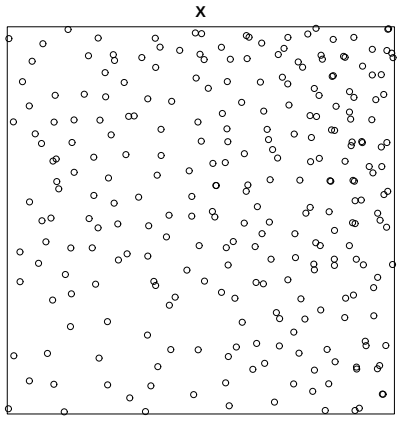
For models fitted by Huang-Ogata, the variance-covariance matrix returned by `vcov` is computed from the simulations.

20 Methods 9: validation of fitted Gibbs models

Goodness-of-fit testing and model validation for Poisson models were described in Section 12. Checking a fitted Gibbs point process model is more difficult. There is little theory available to support goodness-of-fit tests and the like.

As an example, consider the following data:

```
> data(residualspaper)
> X <- residualspaper$Fig4b
> plot(X)
```



We fit a Strauss process model with a log-quadratic intensity term:

```
> fit <- ppm(X, ~polynom(x, y, 2), Strauss(0.05), correction = "isotropic")
```

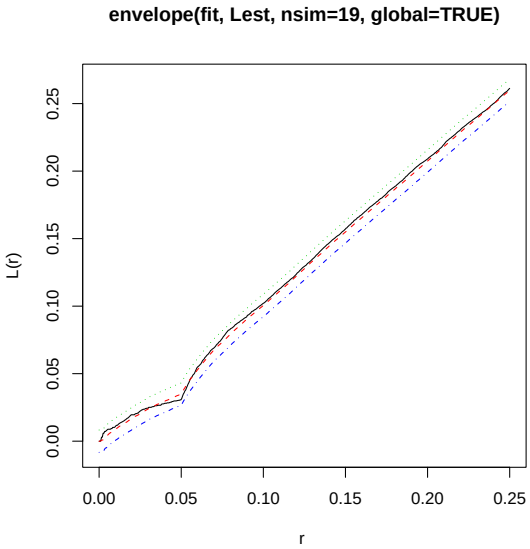
The question is how to confirm or validate this model.

20.1 Goodness-of-fit testing for Gibbs processes

For a fitted Gibbs process, no theory is available to support the χ^2 goodness-of-fit test or the Kolmogorov-Smirnov test. The predicted mean number of points in a given region is not known in closed form for a Gibbs process. Thus, the appropriate test statistic for a χ^2 test is not even available in closed form, let alone the null distribution of this statistic.

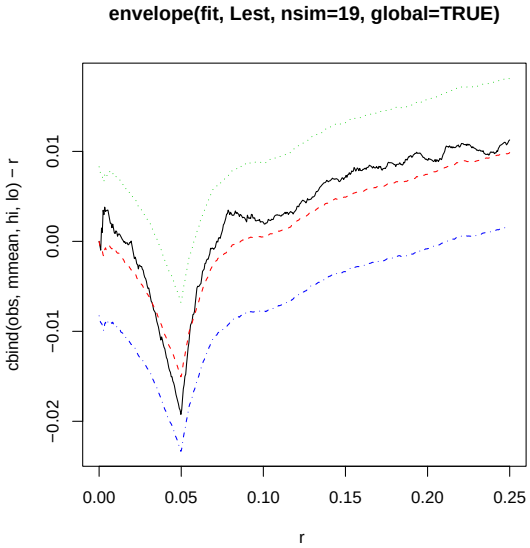
Instead, goodness-of-fit for fitted Gibbs models often relies on the summary functions K and G. The command `envelope` will accept as its first argument a fitted Gibbs model, and will simulate from this model to determine the critical envelope.

```
> plot(envelope(fit, Lest, nsim = 19, global = TRUE))
```



Let's subtract the theoretical Poisson value $L(r) = r$ to get a more readable plot:

```
> plot(envelope(fit, Lest, nsim = 19, global = TRUE), . - r ~ r)
```



This is fairly consistent with a Strauss process.

20.2 Residuals for Gibbs processes

Residuals for a general Gibbs model were defined only recently [6, 1]. The total residual in a region $B \subset \mathbb{R}^2$ is defined as

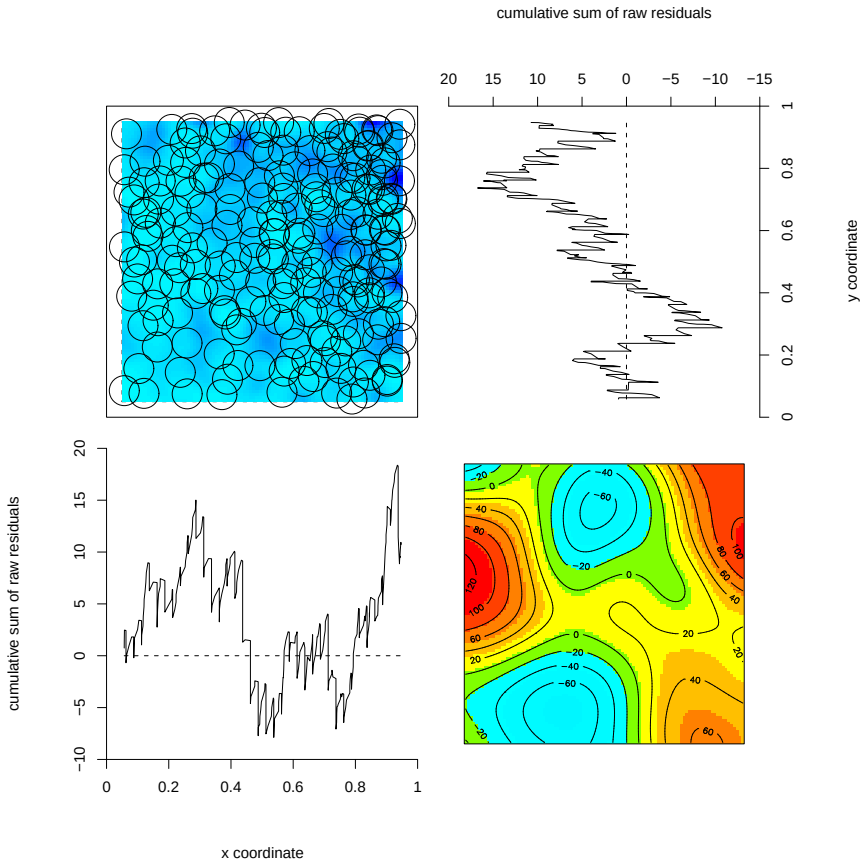
$$R(B) = n(\mathbf{x} \cap B) - \int_B \widehat{\lambda}(u, \mathbf{x})\, \mathrm{d}u \tag{47}$$

where again $n(\mathbf{x} \cap B)$ is the observed number of points in the region B, and $\widehat{\lambda}(u, \mathbf{x})$ is the **conditional** intensity of the fitted model, *evaluated for the data point pattern* $\mathbf{x}$. If the fitted model is correct, the residuals have mean zero.

This definition is similar to the definition of residuals for Poisson processes (Section 12.2) except that the intensity $\widehat{\lambda}(u)$ of the fitted Poisson process has been replaced by the *conditional* intensity $\widehat{\lambda}(u, \mathbf{x})$ of the fitted Gibbs process evaluated for the data point pattern $\mathbf{x}$.

Residuals for Gibbs processes can be plotted as explained in Section 12.2.

```
> diagnose.ppm(fit)
```

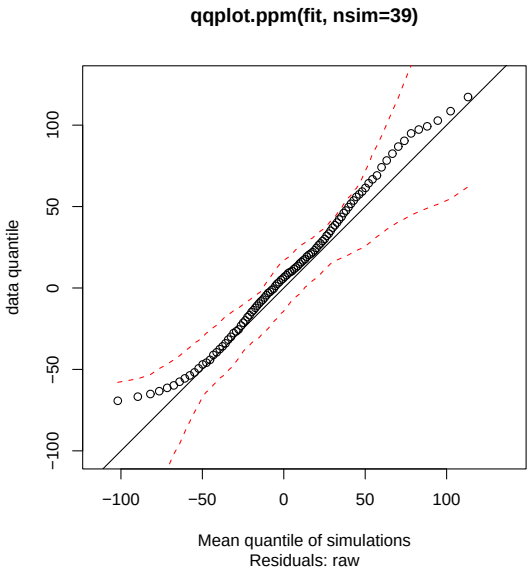


At the time of writing, `spatstat` does not yet display 2σ significance bands for the lurking variable plots when the fitted model is not Poisson. The interpretation of the lurking variable

plots is a little more difficult without the significance bands. One tends to place a little more emphasis on the smoothed residual field.

Interaction between points in a point process corresponds roughly to the distribution of the responses in loglinear regression. To validate the interaction terms in a point process model, we should plot the distribution of the residuals.

```
> qqplot.ppm(fit, nsim = 39)
```



This shows a Q–Q plot of the smoothed residuals, with pointwise 5% critical envelopes from simulations of the fitted model. This suggests that the Strauss model is reasonable.

These validation techniques generalise and unify many existing exploratory methods. For particular models of interpoint interaction, the Q–Q plot is closely related to the summary functions F, G and K. See [6].

21 Marked point patterns

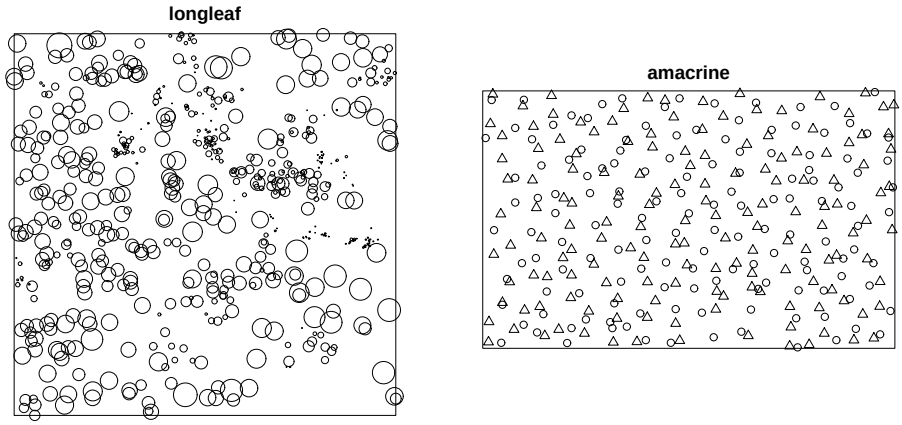
21.1 Marked point patterns

Each point in a spatial point pattern may carry additional information called a 'mark'. For example, points which are classified into two or more different types (on/off, case/control, species, colour, etc) may be regarded as marked points, with a mark which identifies which type they are. Data recording the locations and heights of trees in a forest can be regarded as a marked point pattern where the mark attached to a tree's location is the tree height.

In our current implementation, the mark attached to each point must be a *single* value (which may be numeric, character, complex, logical, or factor). Many of the functions in `spatstat` handle marked point patterns in which the mark attached to each point is either

- a **continuous variate** or "real number". An example is the Longleaf Pines dataset (`longleaf`) in which each tree is marked with its diameter at breast height. The `marks` component must be a `numeric` vector such that `marks[i]` is the mark value associated with the `i`th point. We say the point pattern has *continuous marks*.

- a **categorical variate**. An example is the Amacrine Cells dataset (`amacrine`) in which each cell is identified as either "on" or "off". Such point patterns may be regarded as consisting of points of different "types". The `marks` component must be a `factor` such that `marks[i]` is the label or type of the `i`th point. We call this a *multitype point pattern* and the levels of the factor are the possible types.



Note that, in some other packages, a point pattern dataset consisting of points of two different types (A and B say) is represented by two datasets, one representing the points of type A and another containing the points of type B. In `spatstat` we take a different approach, in which all the points are collected together in one point pattern, and the points are then labelled by the type to which they belong. An advantage of this approach is that it is easy to deal with multitype point patterns with more than 2 types. For example the classic Lansing Woods dataset represents the positions of trees of 6 different species. This is available in `spatstat` as a single dataset, a marked point pattern, with the marks having 6 levels.

21.2 Formulation

A mark variable may be interpreted as an additional coordinate for the point: for example a point process of earthquake epicentre locations (longitude, latitude), with marks giving the

occurrence time of each earthquake, can alternatively be viewed as a point process in space-time with coordinates (longitude, latitude, time).

A marked point process of points in space S with marks belonging to a set M is mathematically defined as a point process in the cartesian product $S \times M$. The space M of possible marks may be 'anything'. In current applications, typically the mark is either a categorical variable (so that the points are grouped into 'types') or a real number. Multivariate marks consisting of several such variables are also common.

A marked point pattern is an unordered set

$$\mathbf{y} = \{(x_1, m_1), \ldots, (x_n, m_n)\}, \quad x_i \in W, \quad m_i \in M$$

where x_i are the locations and m_i are the corresponding marks.

21.3 Methodological issues

21.3.1 Should the data be treated as a marked point process?

In a marked point process the points are random. Treating the data as a point process is inappropriate if the locations are fixed, or if the locations are not part of the 'response'.

Example 16 *Today's maximum temperatures at 25 Australian cities are displayed on a map.*

This is not a point process in any useful sense. The cities are fixed locations. The temperatures are observations of a spatial variable at a fixed set of locations. See the R packages `sp`, `spdep`, `spgwr` for suitable methods.

Example 17 *A mineral exploration dataset records the map coordinates where 15 core samples were drilled, and for each core sample, the assayed concentration of iron in the sample.*

This should *not* be treated as a point process. The core sample locations were chosen by a geologist, and are part of the experimental design. The main interest is in the iron concentration at these locations. This should probably be analysed as a geostatistical dataset. See the R packages `geoR`, `geoRglm` for suitable methods.

21.3.2 Joint vs. conditional analysis

There are more choices for analysis (and more traps) when marks are present. Schematically, if we write X for the points and M for the marks, then a statistical model for the marked point pattern could be formulated in several ways:

- $[X]$ $[M|X]$ — 'conditional on locations' — points X are first generated according to a spatial point process, then marks M are 'assigned' to the points by a random mechanism $[M|X]$;
- $[M]$ $[X|M]$ — 'conditional on marks' or 'split by marks' — marks M are first generated according to some random mechanism $[M]$, then they are placed at certain locations X by point process(es) $[X|M]$;
- $[X, M]$ — 'joint' — marked points are generated according to a marked point process.

These approaches typically lead to different stochastic models and have different inferential interpretations. Correspondingly, there are different null hypotheses that can be tested:

- *random labelling:* given the locations X, the marks are conditionally independent and identically distributed;
- *independence of components:* the sub-processes $\mathbf{X}_m$ of points of each mark m, are independent point processes;
- *complete spatial randomness and independence (CSRI):* the locations $\mathbf{X}$ are a uniform Poisson point process, and the marks are independent and identically distributed. (This implies both random labelling and independence of components).

These null hypotheses are not equivalent.

The properties of random labelling and independence of components are not equivalent. For example, take a point process $\mathbf{X}$ where nearest neighbour distances are always larger than a threshold r, and attach random marks to the points. The resulting marked point process cannot be generated using the independence construction, because if points with different marks are independent, they can come arbitrarily close to one another.

Example 18 (Ant nests data) *Two species of ants build nests in a desert. We want to investigate ecological interaction between the species, and between different nests of the same species. The locations of all nests are mapped, and marked by the species.*

These data can be analysed as a marked point process consisting of two different types of points. The 'mark' attached to each point is its species (a categorical variable). The most natural kind of modelling and analysis is either joint $[X, M]$ or split by species $[M]$ $[X|M]$. We could also treat one of the species as a covariate and analyse the other species conditional on it.

Example 19 *Trees in an orchard are examined and their disease status (infected/not infected) is recorded. We are interested in the spatial characteristics of the disease, such as contagion between neighbouring trees.*

These data probably should *not* be treated as a point process. The response is 'disease status'. We can think of disease status as a label applied to the trees after their locations have been determined. Since we are interested in the spatial correlation of disease status, the tree locations are effectively fixed covariate values. It would probably be best to treat these data as a discrete random field (of disease status values) observed at a finite known set of sites (the trees).

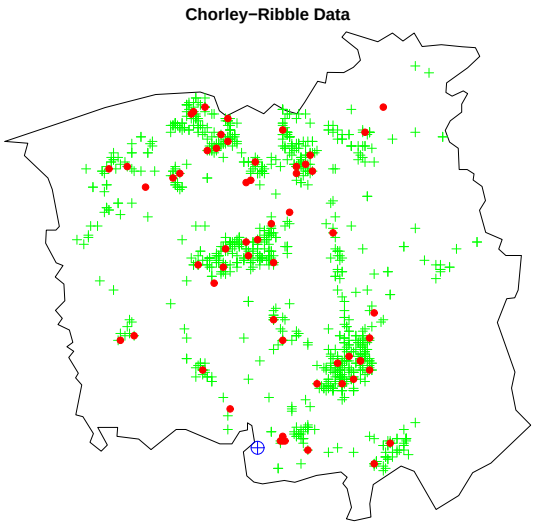
21.3.3 Grey areas

There are some 'grey areas' which permit several alternative choices of analysis. It could be appropriate either to analyse the locations and marks jointly (denoted $[X, M]$), or to analyse the marks conditional on the locations ($[M|X]$) or to analyse the locations given the marks ($[X|M]$).

One grey area occurs when the locations are random, but may be ancillary for the parameters of interest.

Example 20 *Case-control study of cancer [20, 24]. The domicile locations of all new cases of a rare cancer are mapped. To allow for spatial variation in the density of the susceptible population, domicile locations are recorded for a random sample of (matched) controls.*

This can be analysed either as a marked point pattern (where the mark is the case/control label) or, by conditioning on locations, as a random field of case/control values attached to the known domicile locations.



22 Handling marked point pattern data

This section explains how to create a marked point pattern dataset in `spatstat`, and how to manipulate it.

22.1 Creating datasets

In `spatstat` version 1, each point in a point pattern can be marked with a *single* value (i.e. one mark value per point). The marks are stored in a vector, of the same length as the number of points. The marks can be of any atomic type: numeric, integer, character, factor, logical or complex.

A marked point pattern dataset can be created using any of the following tools:

`ppp`	create point pattern dataset
`as.ppp`	convert other data to point pattern
`superimpose`	combine several point patterns into a marked point pattern
`marks`	extract marks from a point pattern
`marks<-`	attach marks to a point pattern
`%mark%`	attach marks to a point pattern
`unmark`	delete marks from a point pattern
`scanpp`	read point pattern data from text file
`clickppp`	create a pattern using point-and-click on the screen

The command `ppp` can be used to create a marked point pattern dataset from raw data. The syntax is

```
> ppp(x, y, ..., marks = m)
```

where `x`, `y` and `m` are vectors of equal length containing the (x, y) coordinates and the corresponding mark values, and `...` are arguments that determine the window for the point pattern.

> Tip: If the marks are intended to be a categorical variable (representing the types in a multitype point pattern),
>
> - ensure that `m` is stored as a `factor` in R.
> - when the point pattern `X` has been created, check that it is multitype using `is.multitype(X)`.
> - check that the factor levels are as you intended, using `levels(m)` or `levels(marks(X))` where `X` is the marked point pattern. If the factor levels are character strings, they will be sorted into alphabetical order by default.
> - be careful when performing equality/inequality comparisons involving a factor. Particular danger occurs when the factor levels are strings that represent integers.

The command `as.ppp` will convert data in another format (for example, a 2-column or 3-column matrix or data frame) to a point pattern object of class `"ppp"`. The third column of a matrix or data frame will be interpreted as containing the marks.

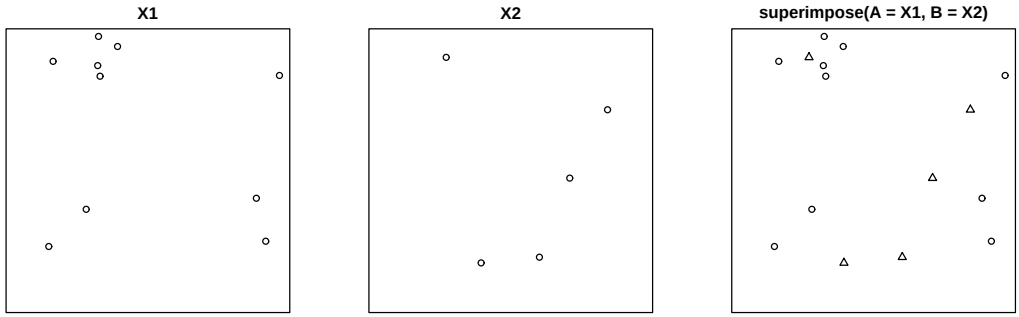
```
> mydata <- data.frame(x = runif(10), y = runif(10), m = sample(letters[1:3],
+     10, replace = TRUE))
> as.ppp(mydata, square(1))
```

```
marked planar point pattern: 10 points
multitype, with levels = a        b        c
window: rectangle = [0, 1] x [0, 1] units
```

If point pattern data are stored in a text file, the command `scanpp` will read the data and create a point pattern object of class `"ppp"`. The argument `multitype=TRUE` will ensure that the mark values are interpreted as a factor.

```
> X <- scanpp("myfile.txt", window = square(1), multitype = TRUE)
```

The command `superimpose` combines several point patterns within the same window. It can be used to create a multitype point pattern, if you have already created separate point patterns containing the points of each type. Suppose `X1` and `X2` are unmarked point patterns Then `superimpose(A=X1, B=X2)` will create a multitype point pattern by attaching the mark A to each point of `X1`, attaching the mark B to each point of `X2`, and combining the points.



Marks can be attached to an existing point pattern `X` using the function `marks<-` as in

```
> marks(X) <- m
```

or using the binary operator `%mark%`,

```
> Y <- X %mark% m
```

These are convenient when you want to assign new marks to a dataset that are computed using another variable, or perhaps to randomise the marks in a dataset.

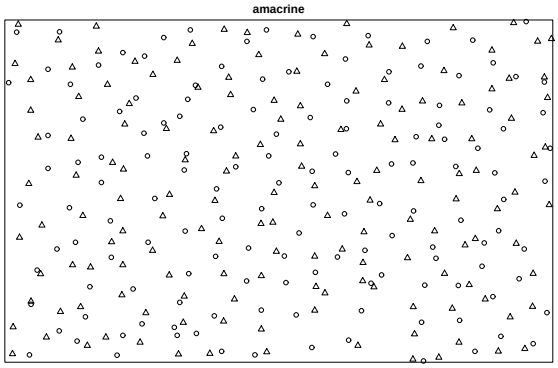
A multitype point pattern can also be created interactively using `clickppp`, using the argument `types` to specify the possible types.

22.2 Inspecting a marked point pattern

Basic tools for inspecting a marked point pattern include the `print`, `plot` and `summary` methods.

```
> data(amacrine)
> amacrine

marked planar point pattern: 294 points
multitype, with levels = off       on
window: rectangle = [0, 1.6012] x [0, 1] units (one unit = 662 microns)

> summary(amacrine)
```

```
Marked planar point pattern: 294 points
Average intensity 184 points per square unit (one unit = 662 microns)
Multitype:
    frequency proportion intensity
off       142      0.483      88.7
on        152      0.517      94.9

Window: rectangle = [0, 1.6012] x [0, 1] units
Window area =  1.60121 square units
Unit of length: 662 microns

> plot(amacrine)

off  on
  1   2
```



You can also convert a marked point pattern into a data frame for closer inspection of the coordinates and mark values:

```
> as.data.frame(amacrine)

         x      y marks
1   0.0224 0.0243    on
2   0.0243 0.1028    on
3   0.1626 0.1477    on
........
```

The marks can be extracted using the function `marks`:

```
> data(longleaf)
> m <- marks(longleaf)
```

Beware the possibility that two points with different marks may occupy the same spatial location. This is not currently detected by `ppp` since, for a marked point pattern, the function `duplicated.ppp` regards two points as identical only when their coordinates **and** mark values are identical. To detect duplication of the spatial locations, use `duplicated(unmark(X))`.

Further tools are presented in the next section.

22.3 Manipulating data

22.3.1 Manipulating marks

The following tools can manipulate the marks in a point pattern:

`marks`	extract marks
`marks<-`	attach marks to a point pattern
`%mark%`	attach marks to a point pattern
`unmark`	remove marks from point pattern

For example, the Lansing Woods data are tree locations marked by diameter at breast height (dbh) in centimetres. To convert the marks from diameters to circular areas,

```
> data(lansing)
> d <- marks(lansing)
> a <- (pi/4) * d^2
> marks(lansing) <- a
```

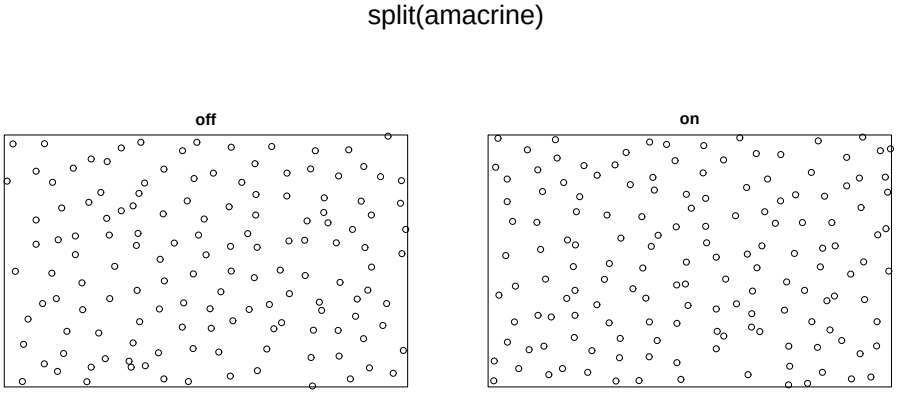
22.3.2 Separating points of different types

A *multitype* point pattern can be separated into the sub-patterns of points of each type, using the `split` command.

```
> data(amacrine)
> Y <- split(amacrine)
```

In fact `split` is a generic function and the commands above invoke the `split` method for the class of point patterns, `split.ppp`. The result `Y` is a list of point patterns, with names that correspond to the type labels. This list also belongs to the class `"splitppp"` which can be plotted automatically:

```
> plot(split(amacrine))
```

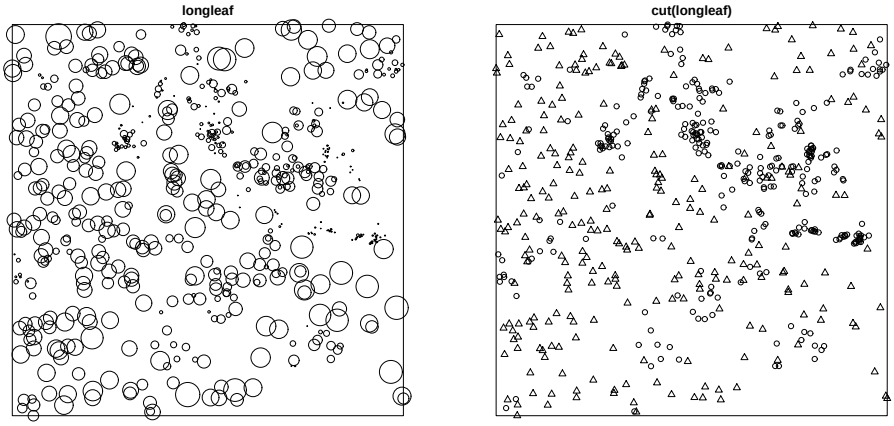


22.3.3 Cutting the numerical scale into bands

For a point pattern with *numeric* marks, the marks can be converted to a factor, using a method for the generic function `cut`. The user specifies a series of cut-points on the numerical scale; all mark values between two cut-points are given the same label.

For example, the Longleaf Pines data are the locations of trees marked with their diameter at breast height, dbh, in centimetres. By convention we define "adult" trees to be those with dbh greater than 30 centimetres. To obtain the bivariate point pattern of adult and juvenile trees,

```
> data(longleaf)
> longleaf

marked planar point pattern: 584 points
marks are numeric, of type 'double'
window: rectangle = [0, 200] x [0, 200] metres

> X <- cut(longleaf, breaks = c(0, 30, 80), labels = c("juvenile",
+     "adult"))
> X

marked planar point pattern: 584 points
multitype, with levels = juvenile        adult
window: rectangle = [0, 200] x [0, 200] metres

> par(mfrow = c(1, 2))
> plot(longleaf)

       0       20       40       60       80
0.000000 1.722522 3.445045 5.167567 6.890090

> plot(X, main = "cut(longleaf)")

juvenile    adult
       1        2

> par(mfrow = c(1, 1))
```



23 Methods 10: exploratory tools for marked point patterns

This section covers some tools for exploratory data analysis of marked point patterns. Most of the tools have been developed for the special case of *multitype* point patterns (i.e. where the marks are categorical).

23.1 Intensity

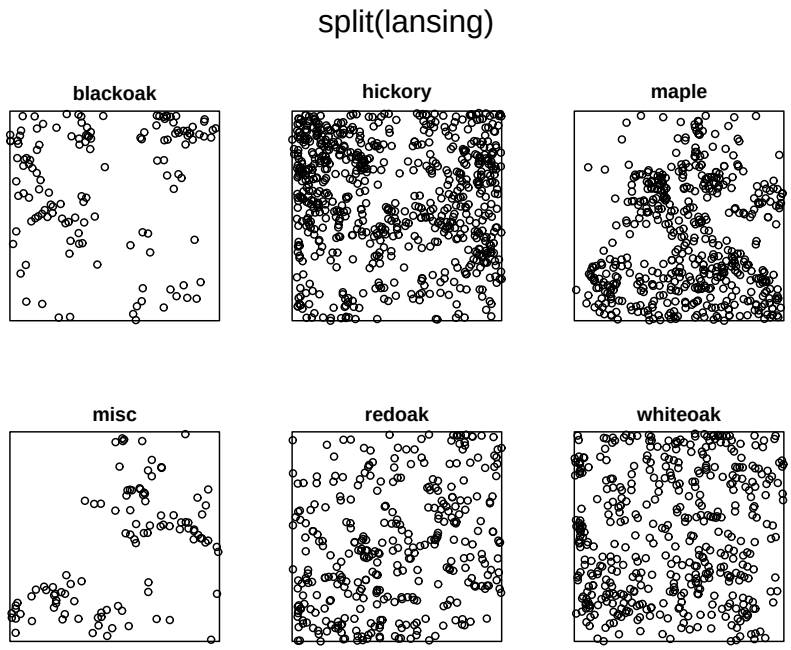
The Lansing Woods data give the locations of 6 species of trees in a forest in Michigan. Elementary estimates of the frequency distribution of species, and the intensity of each species, are available from `summary.ppp`.

```
> data(lansing)
> summary(lansing)
```

```
Marked planar point pattern: 2251 points
Average intensity 2250 points per square unit (one unit = 924 feet)

*Pattern contains duplicated points*
Multitype:
         frequency proportion intensity
blackoak       135     0.0600       135
hickory        703     0.3120       703
maple          514     0.2280       514
misc           105     0.0466       105
redoak         346     0.1540       346
whiteoak       448     0.1990       448

Window: rectangle = [0, 1] x [0, 1] units
Window area =  1 square unit
Unit of length: 924 feet
```

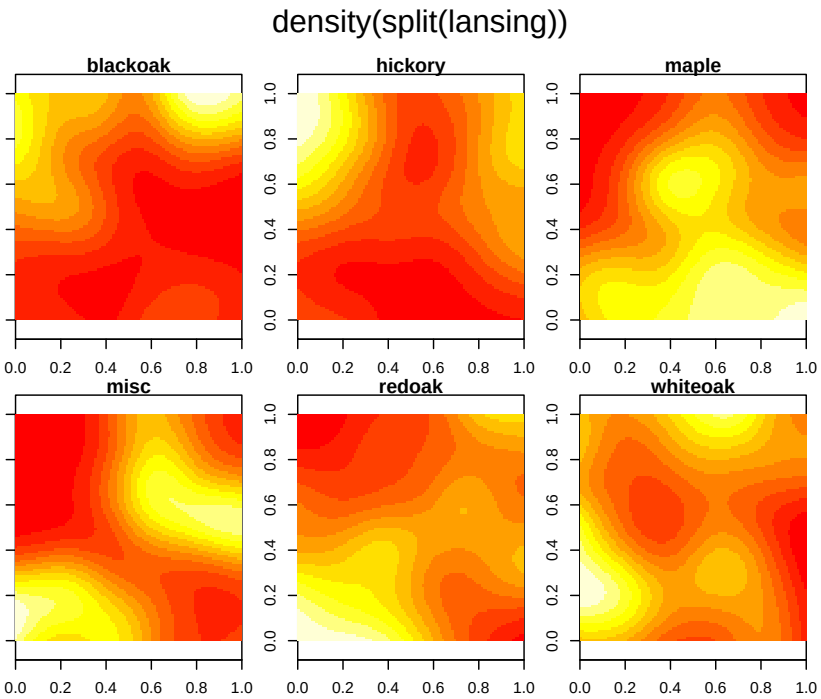
It's sensible to examine the sub-patterns of different types separately, using `split.ppp`.

```
> plot(split(lansing))
```



It would be useful to compute and plot a separate estimate of intensity for each type of tree. This is possible using the functions `density.splitppp` and `plot.listof`. They are invoked simply by typing

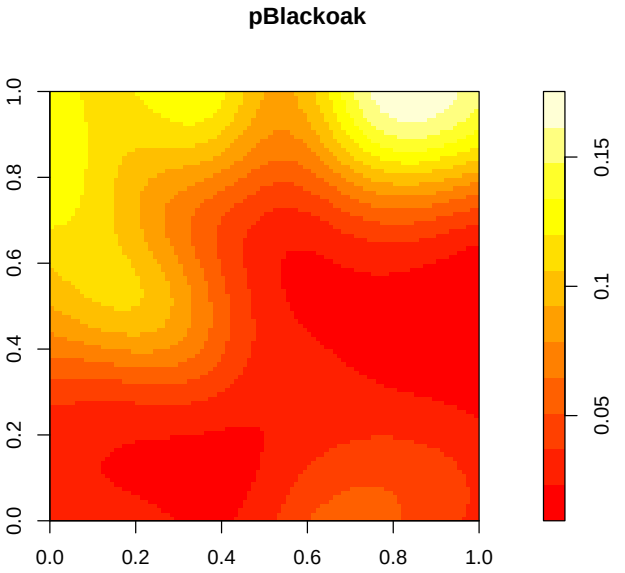
```
> plot(density(split(lansing)), ribbon = FALSE)
```



The relative proportions of intensity can then be computed using `eval.im`:

```
> Y <- density(split(lansing))
> attach(Y)
```

```
> pBlackoak <- eval.im(blackoak/(blackoak + hickory + maple + misc +
+     redoak + whiteoak))
> plot(pBlackoak)
> detach(Y)
```

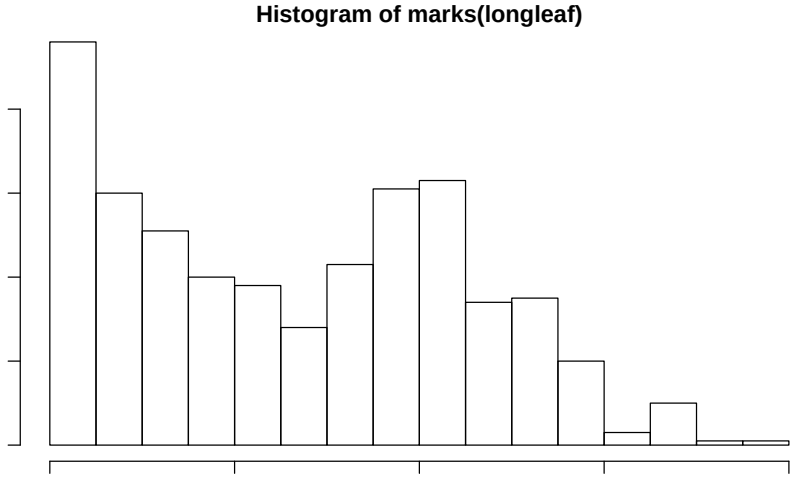


Parametric estimates of intensity can be obtained using `ppm`, fitting a Poisson model with an intensity function that may depend on location and/or on the marks. See below.

23.2 Numeric marks: distribution and trend

For a point pattern with marks that are numeric (real numbers or integers) or logical values, the mark values can be extracted using the `marks` function and inspected using the histogram or kernel density estimate:

```
> data(longleaf)
> hist(marks(longleaf))
```

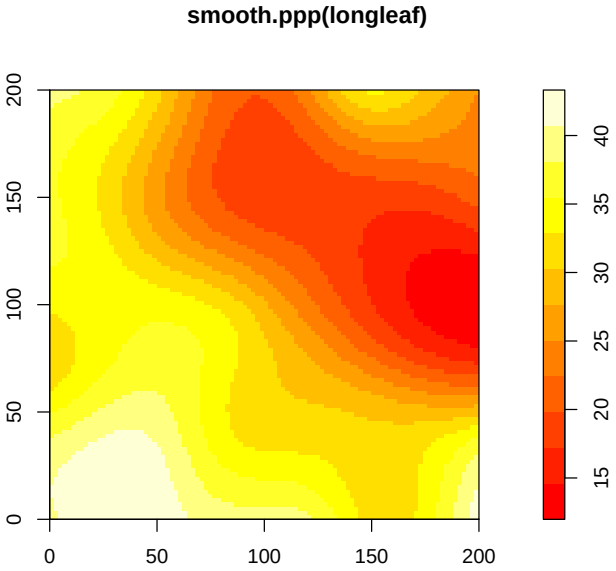


To assess spatial trend in the marks, one way is to form a kernel regression smoother. The smoothed mark value at location $u \in \mathbb{R}^2$ is

$$\widehat{m}(u) = \frac{\sum_i m_i \kappa(u - x_i)}{\sum_i \kappa(u - x_i)}$$

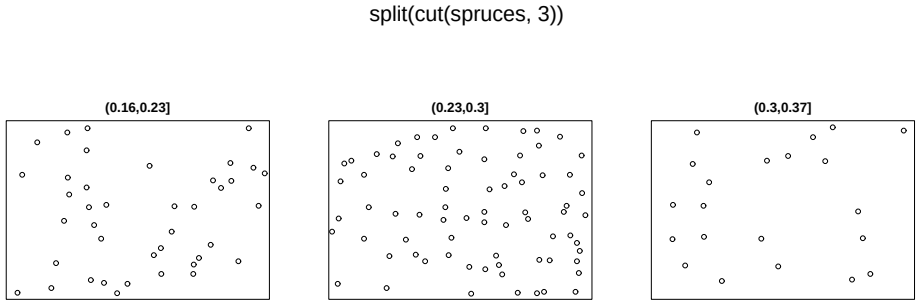
where k is the smoothing kernel, and m_i is the mark value at data point x_i. This is computed by `smooth.ppp`:

```
> plot(smooth.ppp(longleaf))
```



You can also use `cut.ppp` followed by `split.ppp` to look for spatial inhomogeneity of the marks:

```
> data(spruces)
> plot(split(cut(spruces, 3)))
```



23.3 Simple summaries of neighbouring marks

We are often interested in the marks that are attached to the close neighbours of a typical point.

For a multitype point pattern, the function `marktable` compiles a contingency table of the marks of all points within a given radius of each data point:

```
> data(amacrine)
> M <- marktable(amacrine, R = 0.1)
> M[1:10, ]
```

```
     mark
point off on
   1    1  1
   2    2  2
   3    4  3
   4    3  1
   5    4  1
   6    2  3
   7    3  2
   8    1  1
   9    3  1
   10   3  2
```

More general summaries of the marks of neighbours can be obtained using the function `markstat`. For example, to compute the average diameter of the 5 closest neighbours of each tree in the Longleaf Pines dataset,

```
> md <- markstat(longleaf, mean, N = 5)
> md[1:10]
```

```
 [1] 43.40 43.40 48.58 21.70 48.38 53.32 40.28 29.82 24.92 21.70
```

23.4 Summary functions

The summary functions F, G, J and K (and other functions derived from K, such as L and the pair correlation function) have been extended to multitype point patterns.

Assume the multitype point process $\mathbf{X}$ is stationary. Let $\mathbf{X}_j$ denote the sub-pattern of points of type j, with intensity λ_j. Then

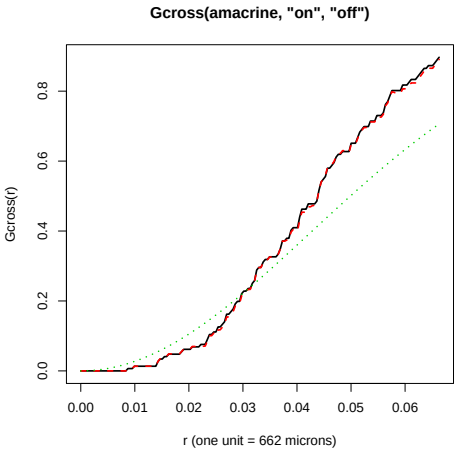
- $F_j(r)$ is the empty space function for $\mathbf{X}_j$
- $G_{ij}(r)$ is the distribution function of the distance from a point of type i to the nearest point of type j
- $K_{ij}(r)$ is $1/\lambda_j$ times the expected number of points of type j within a distance r of a typical point of type i.
- J_{ij} is defined as

$$J_{ij}(r) = \frac{1 - G_{ij}(r)}{1 - F_j(r)}.$$

The functions G_{ij}, K_{ij}, J_{ij} are called "cross-type" or "i-to-j" summary functions. They are computed in `spatstat` by `Gcross`, `Kcross` and `Jcross`.

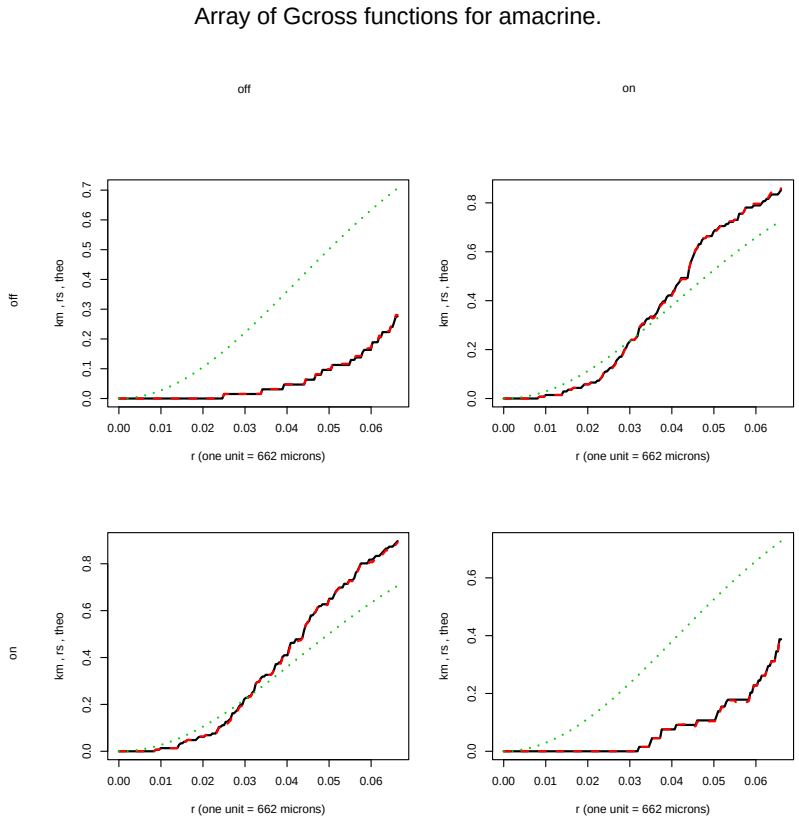
```
> data(amacrine)
> amacrine
```

```
> plot(Gcross(amacrine, "on", "off"))
```



The command `alltypes` enables the user to compute the cross-type summary functions between all pairs of types simultaneously. For example, to compute $G_{ij}(r)$ for all i and j in the amacrine cells data, we would use `alltypes(amacrine, "G")`. The result is automatically displayed as an array of plot panels.

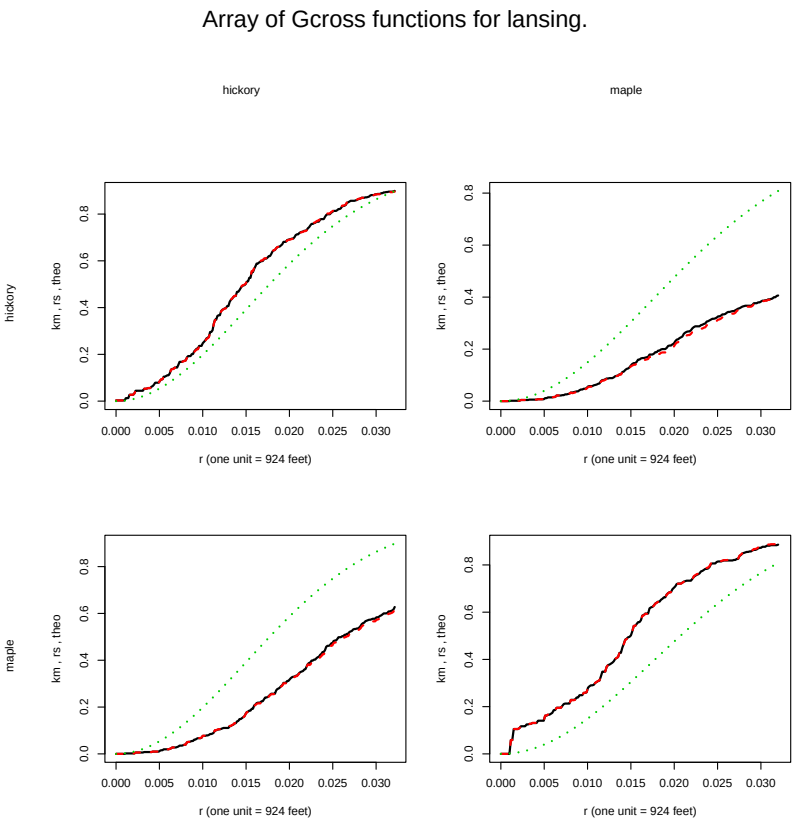
```
> plot(alltypes(amacrine, "G"))
```



The result of `alltypes` is a 'function array' (object of class `"fasp"`) which can be indexed by row and column subscripts. If the point pattern has a large number of possible types, you can compute the array of all possible pairwise G functions, then use the subscript operator to inspect a subset of the array.

```
> data(lansing)
> a <- alltypes(lansing, "G")
```

```
> plot(a[2:3, 2:3])
```



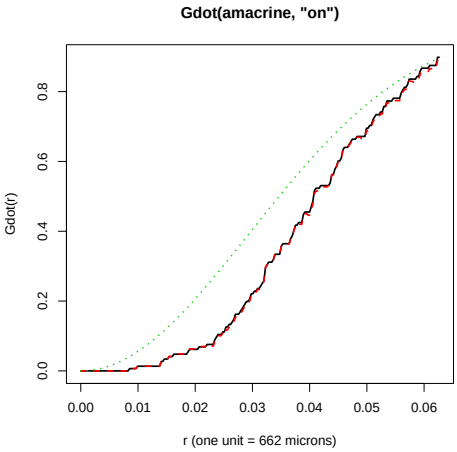
Also defined are the "i-to-any" summaries

- $G_{i\bullet}(r)$, the distribution function of the distance from a point of type i to the nearest other point of any type;
- $K_{i\bullet}(r)$ is $1/\lambda$ times the expected number of points of any type within a distance r of a typical point of type i. Here $\lambda = \sum_j \lambda_j$ is the intensity of the entire process $\mathbf{X}$.
- $J_{i\bullet}$ defined by

$$J_{i\bullet}(r) = \frac{1 - G_{i\bullet}}{1 - F(r)}$$

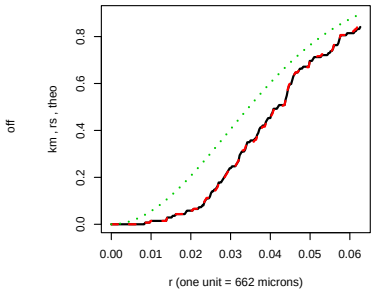
These are computing by `Gdot`, `Kdot` and `Jdot` respectively, or using `alltypes`.

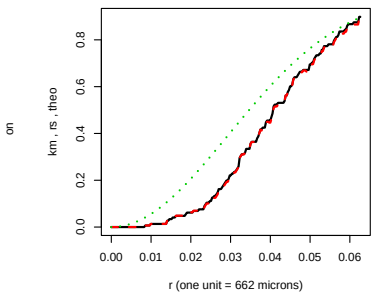
```
> plot(Gdot(amacrine, "on"))
```




```
> plot(alltypes(amacrine, "Gdot"))
```

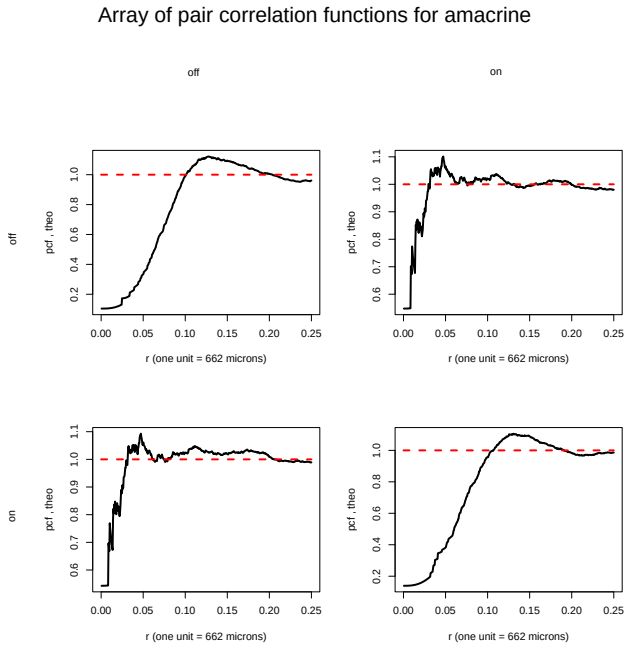
Array of Gdot functions for amacrine.





The pair correlation functions corresponding to the K-functions can also be computed, using `pcf.fasp`.

```
> K <- alltypes(amacrine, "K")
> P <- pcf(K, method = "b", spar = 1)
> plot(P, lwd = 2)
```



23.5 Mark correlation function

The mark correlation function $\rho_f(r)$ of a stationary marked point process $\mathbf{Y}$ is a measure of the dependence between the marks of two points of the process a distance r apart [42]. It is informally defined as

$$\rho_f(r) = \frac{\mathbb{E}[f(M_1, M_2)]}{\mathbb{E}[f(M, M')]}$$

where M_1, M_2 are the marks attached to two points of the process separated by a distance r, while M, M' are independent realisations of the marginal distribution of marks.

Here f is any function $f(m_1, m_2)$ with two arguments which are possible marks of the pattern, and which returns a nonnegative real value. Common choices of f are:

- for continuous real-valued marks, $f(m_1, m_2) = m_1 m_2$;
- for categorical marks (multitype point patterns), $f(m_1, m_2) = \mathbf{1}\{m_1 = m_2\}$;
- for marks taking values in $[0, 2\pi]$, $f(m_1, m_2) = \sin(m_1 - m_2)$.

Note that $\rho_f(r)$ is not a "correlation" in the usual statistical sense. It can take any nonnegative real value. The value 1 suggests "lack of correlation": under random labelling, $\rho_f(r) \equiv 1$. The interpretation of values larger or smaller than 1 depends on the choice of function f.

The mark correlation function is computed in `spatstat` by `markcorr`. It has the syntax

```
> markcorr(X, f)
```

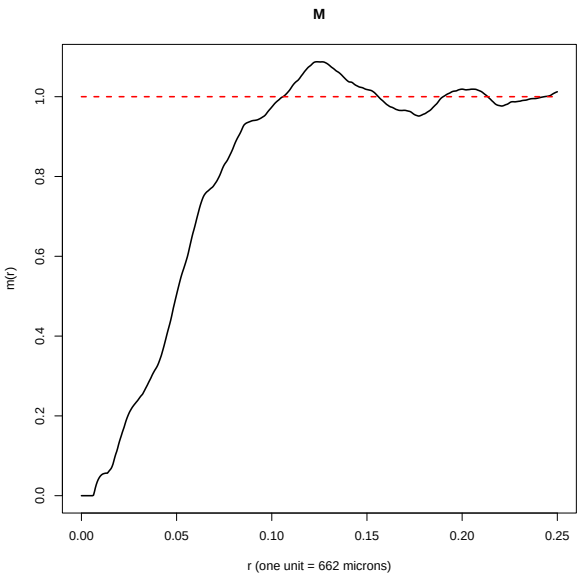
where `X` is a point pattern and `f` is an R language function. For example, for the `amacrine` data, the natural function f is $f(m_1, m_2) = \mathbf{1}\{m_1 = m_2\}$ which we encode as

```
> eqfun <- function(m1, m2) {
+     m1 == m2
+ }
```

Then simply

```
> M <- markcorr(amacrine, eqfun, correction = "translate", method = "density",
+     kernel = "epanechnikov")
```

```
> plot(M)
```



23.6 Randomisation tests

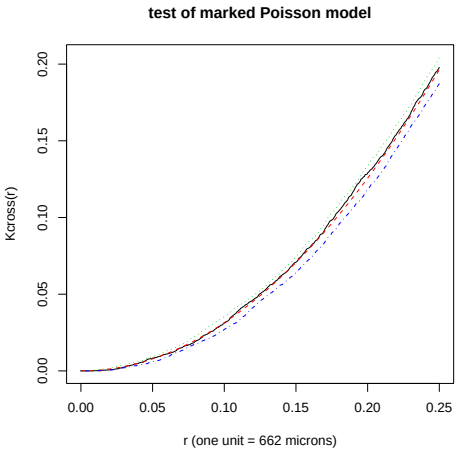
Simulation envelopes of summary functions can be used to test various null hypotheses for marked point patterns.

23.6.1 Poisson null

The null hypothesis of a homogeneous Poisson marked point process can be tested by direct simulation, using `envelope` as before. For example, using the cross-type K function as the test statistic,

```
> data(amacrine)
> E <- envelope(amacrine, Kcross, nsim = 39, i = "on", j = "off")
```

```
> plot(E, main = "test of marked Poisson model")
```



Notice that the arguments `i` and `j` here do not match any of the formal arguments of `envelope`, so they are passed to `Kcross`. This has the effect of calling `Kcross(X, i="on", j="off")` for each of the simulated point patterns `X`. Each simulated pattern is generated by the homogeneous Poisson point process with intensities estimated from the dataset `amacrine`.

23.6.2 Independence of components

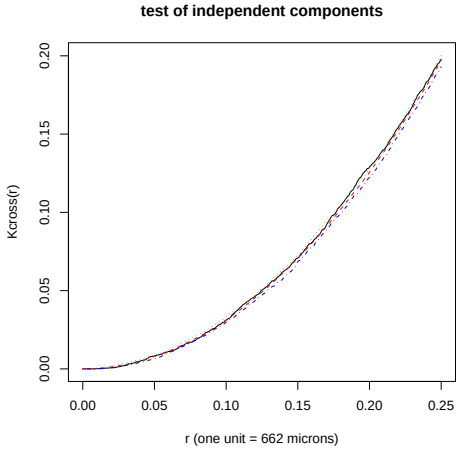
It's also possible to test other null hypotheses by a randomisation test. We discussed two popular null hypotheses:

- *random labelling:* given the locations X, the marks are conditionally independent and identically distributed;
- *independence of components:* the sub-processes $\mathbf{X}_m$ of points of each mark m, are independent point processes.

In a randomisation test of the independence-of-components hypothesis, the simulated patterns `X` are generated from the dataset by splitting the data into sub-patterns of points of one type, and randomly shifting these sub-patterns, independently of each other. The shifting is performed by `rshift`:

```
> E <- envelope(amacrine, Kcross, nsim = 39, i = "on", j = "off",
+     simulate = expression(rshift(amacrine, radius = 0.25)))
```

```
> plot(E, main = "test of independent components")
```



The independence-of-components hypothesis seems to be accepted in this example.

Under the independence hypothesis,

$$\begin{aligned} K_{ij}(r) &= \pi r^2 \\ G_{ij}(r) &= F_j(r) \\ J_{ij}(r) &\equiv 1. \end{aligned}$$

while the "i-to-any" functions have complicated values. Thus, we would normally use K_{ij} or J_{ij} to construct a test statistic for independence of components.

23.6.3 Random labelling

In a randomisation test of the random labelling null hypothesis, the simulated patterns `X` are generated from the dataset by holding the point locations fixed, and randomly resampling the marks, either with replacement (independent random sampling) or without replacement (randomly permuting the marks). The resampling operation is performed by `rlabel`.

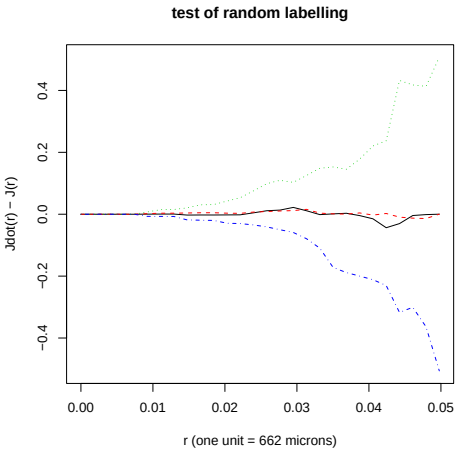
Under random labelling,

$$\begin{aligned} J_{i\bullet}(r) &= J(r) \\ K_{i\bullet}(r) &= K(r) \\ G_{i\bullet}(r) &= G(r) \end{aligned}$$

(where G, K, J are the summary functions for the point process without marks) while the other, cross-type functions have complicated values. Thus, we would normally use something like $K_{i\bullet}(r) - K(r)$ to construct a test statistic for random labelling.

To do this, cook up a little function to evaluate $J_{i\bullet}(r) - J(r)$:

```
> Jdif <- function(X, ..., i) {
+     Jidot <- Jdot(X, ..., i = i)
+     J <- Jest(X, ...)
+     dif <- eval.fv(Jidot - J)
+     return(dif)
+ }
> E <- envelope(amacrine, Jdif, nsim = 39, i = "on", simulate = expression(rlabel(amacrin

> plot(E, main = "test of random labelling")
```



The random labelling hypothesis also seems to be accepted.

24 Methods 11: multitype Poisson models

This section covers multitype Poisson process models: basic properties, simulation, and fitting models to data.

24.1 Theory

24.1.1 Complete spatial randomness and independence

A *uniform Poisson marked point process* in $\mathbb{R}^2$ with marks in $\mathcal{M}$ can be defined in the following equivalent ways.

- randomly marked Poisson process (Poisson $[X]$, iid $[M|X]$): a Poisson point process of locations $\mathbf{X}$ with intensity β is first generated. Then each point x_i is labelled with a random mark m_i, independently of other points, with distribution $\mathbb{P}\{M_i = m\} = p_m$ for $m \in \mathcal{M}$.

- superposition of independent Poisson processes (iid $[M]$, Poisson $[X|M]$): for each possible mark $m \in \mathcal{M}$, a Poisson process $\mathbf{X}_m$ is generated, with intensity β_m. The points of $\mathbf{X}_m$ are tagged with the mark m. Then the processes $\mathbf{X}_m$ with different marks $m \in \mathcal{M}$ are superimposed, to yield a marked point process.

- Poisson marked point process (jointly Poisson $[X, M]$): a Poisson process on $\mathbb{R}^2 \times \mathcal{M}$ is generated, with intensity function $\lambda(u, m) = \beta_m$ at location u and mark m.

These constructions are *equivalent* when $\beta_m = p_m\beta$. See the lovely book by Kingman [28].

Since the established term CSR ('complete spatial randomness') is used to refer to the uniform Poisson point process, I propose that the uniform *marked* Poisson point process should be called 'complete spatial randomness and independence' (CSRI).

24.1.2 Inhomogeneous Poisson marked point processes

A *inhomogeneous* Poisson marked point process $\mathbf{Y}$ with 'joint' intensity $\lambda(u, m)$ for locations u and mark values m is simply defined as an inhomogeneous Poisson point process on $\mathbb{R}^2 \times \mathcal{M}$ with intensity function $\lambda(u, m)$.

Let's restrict attention to the case of categorical marks, where $\mathcal{M}$ is finite. Then the process $\mathbf{Y}$ has the following properties:

- The locations $\mathbf{X}$, obtained by removing the marks, constitute an inhomogeneous Poisson process in $\mathbb{R}^2$ with intensity function

$$\beta(u) = \sum_m \lambda(u, m).$$

- Conditional on the locations $\mathbf{X}$, the marks attached to the points are independent. For a point x_i the conditional distribution of the mark m_i is $\mathbb{P}\{M_i = m\} = \lambda(x_i, m)/\beta(x_i)$.

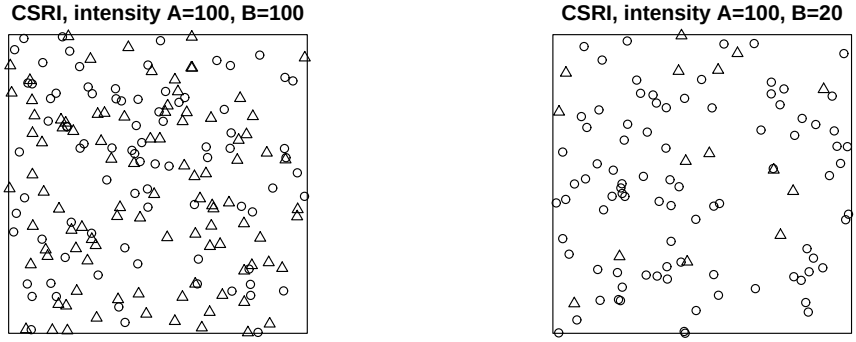
- The sub-process $\mathbf{X}_m$ of points with mark m, is an inhomogeneous Poisson point process with intensity $\beta_m(u) = \lambda(u, m)$.

- The sub-processes $\mathbf{X}_m$ of points with different marks m are independent processes.

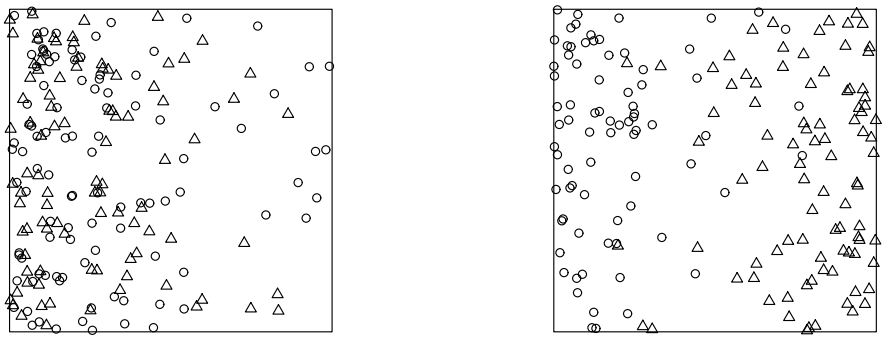
24.2 Simulation

Realisations of Poisson marked point processes can be generated using `rmpoispp`. The first argument of this command specifies the intensity or intensity function $\lambda(u, m)$. It can be a constant, a vector of constants, or an R function.

```
> par(mfrow = c(1, 2))
> Xunif <- rmpoispp(100, types = c("A", "B"), win = square(1))
> plot(Xunif, main = "CSRI, intensity A=100, B=100")
> Xunif <- rmpoispp(c(100, 20), types = c("A", "B"), win = square(1))
> plot(Xunif, main = "CSRI, intensity A=100, B=20")
> par(mfrow = c(1, 1))
```




```
> X1 <- rmpoispp(function(x, y, m) {
+     300 * exp(-3 * x)
+ }, types = c("A", "B"))
> lamb <- function(x, y, m) {
+     ifelse(m == "A", 300 * exp(-4 * x), 300 * exp(-4 * (1 - x)))
+ }
> X2 <- rmpoispp(lamb, types = c("A", "B"))
> par(mfrow = c(1, 2))
> plot(X1, main = "")
> plot(X2, main = "")
> par(mfrow = c(1, 1))
```



24.3 Fitting Poisson models

Poisson marked point process models may be fitted to point pattern data using `ppm`. Currently the methods are only available for multitype point processes (categorical marks).

24.3.1 Probability densities

Let $W \subset \mathbb{R}^2$ be the study region, and $\mathcal{M}$ the (finite) set of possible marks. Then a marked point pattern is a set

$$\mathbf{y} = \{(x_1, m_1), \ldots, (x_n, m_n)\}, \quad x_i \in W, \quad m_i \in \mathcal{M}, \quad n \geq 0$$

of pairs (x_i, m_i) of locations x_i with marks m_i. It can be viewed as a point pattern in the Cartesian product $W \times \mathcal{M}$.

The probability density of a marked point process is a function $f(\mathbf{y})$ defined for all marked point patterns $\mathbf{y}$ including the empty pattern $\emptyset$.

The process with probability density $f(\mathbf{y}) \equiv 1$ is the uniform Poisson marked point process with intensity 1 **for each mark**. That is, for this model, the sub-process of points with mark $m_i = m$ is a uniform Poisson process with intensity 1. If the marks are removed, we obtain a Poisson point process with intensity equal to $|\mathcal{M}|$, the number of possible types.

The uniform Poisson marked point process with intensity $\lambda(u, m) = \beta_m$ has probability density

$$\begin{aligned} f(\mathbf{y}) &= \exp\left(\sum_{m \in \mathcal{M}} (1 - \beta_m)|W|\right) \prod_{i=1}^{n(\mathbf{y})} \beta_{m_i} \\ &= \exp\left(\sum_{m \in \mathcal{M}} (1 - \beta_m)|W|\right) \prod_{m \in \mathcal{M}} \beta_m^{n_m(\mathbf{y})} \end{aligned}$$

where $n_m(\mathbf{y})$ is the number of points in $\mathbf{y}$ having mark value m.

The inhomogeneous Poisson marked point process with intensity function $\lambda(u, m)$, at location $u \in W$ and mark $m \in \mathcal{M}$, has probability density

$$f(\mathbf{y}) = \exp\left(\sum_{m\in\mathcal{M}} \int_W (1 - \lambda(u,m)\,\mathrm{d}u\right) \prod_{i=1}^{n(\mathbf{y})} \lambda(x_i, m_i). \tag{48}$$

24.3.2 Maximum likelihood

For the multitype Poisson process with intensity function $\lambda(u,m)$ at location $u \in W$ and mark $m \in \mathcal{M}$, the loglikelihood is, up to a constant,

$$\log L = \sum_{i=1}^{n} \log \lambda(x_i, m_i) - \sum_{m\in\mathcal{M}} \int_W \lambda(u,m)\,\mathrm{d}u. \tag{49}$$

where m_i is the mark attached to data point x_i. This is formally equivalent to the loglikelihood of a Poisson loglinear regression, so the Berman-Turner algorithm can again be used to maximise the loglikelihood.

24.3.3 Model-fitting in `spatstat`

Poisson marked point process models are fitted to data using `ppm`.

The trend formula in the call to `ppm` may involve the reserved name `marks` as a variable. This refers to the marks of the points. Since the marks are categorical, `marks` is treated as a `factor` variable for modelling purposes.

To fit the homogeneous multitype Poisson process (CSRI), equation (50), we call

```
> ppm(X, ~marks)
```

The formula `~marks` indicates that the trend depends only on the marks, and not on spatial location; since `marks` is a factor, the trend has a separate constant value for each level of `marks`. This is the model (50).

Note that if we had typed

```
> ppm(X, ~1)
```

this would have fitted the special case of CSRI where the intensities β_m are equal, $\beta_m \equiv \alpha$ say, for all possible marks. That model is only appropriate if we believe that all mark values are equally likely.

For the Lansing Woods data, the minimal model that makes sense is (50), so we call

```
> ppm(lansing, ~marks)

Stationary multitype Poisson process
Possible marks:
blackoak hickory maple misc redoak whiteoak

Trend formula: ~marks

Intensities:
beta_blackoak  beta_hickory    beta_maple     beta_misc   beta_redoak
          135           703           514           105           346
beta_whiteoak
          448
```

Since `lansing` is a multitype point pattern (its marks are categorical), the variable `marks` in the formula is a factor. The model has one parameter/coefficient for each level of the factor, i.e. one coefficient for each type of point. In other words, this is the homogeneous Poisson marked point process with intensity β_m for points of mark m.

You'll notice that the parameter estimates $\widehat{\beta}_m$ coincide with those obtained from `summary.ppp` above. That is a consequence of the fact that the maximum likelihood estimates (obtained by `ppm`) are also the method-of-moments estimates (obtained by `summary.ppp`).

A more complicated example is

```
> ppm(lansing, ~marks + x)

Nonstationary multitype Poisson process
Possible marks:
blackoak hickory maple misc redoak whiteoak

Trend formula: ~marks + x

Fitted coefficients for trend formula:
  (Intercept)  markshickory    marksmaple     marksmisc   marksredoak
   4.94294727    1.65008211    1.33694849   -0.25131442    0.94116400
markswhiteoak             x
   1.19951845   -0.07581624
```

This is the marked Poisson process whose intensity function $\lambda((x, y, m))$ at location (x, y) and mark m satisfies

$$\log \lambda((x, y, m)) = \alpha_m + \beta x$$

where $\alpha_1, \ldots, \alpha_6$ and β are parameters. The intensity is loglinear in x, with a different intercept for each mark, but the same slope ("parallel loglinear regression"). In the printout above, the fitted slope parameter β is $\hat{\beta}$ =-0.07581624. As discussed in Section 11.3 on page 61, the fitted coefficients α_m for the categorical mark are interpreted in the light of the 'contrasts' in force. The default is the treatment contrasts, and the first level of the mark is `blackoak`, so in this case the fitted coefficient for `m=blackoak` is 4.942947, while the fitted coefficient for `m=hickory` is $4.942947 + 1.650082 = 6.593029$ and so on.

```
> ppm(lansing, ~marks * x)

Nonstationary multitype Poisson process
Possible marks:
blackoak hickory maple misc redoak whiteoak

Trend formula: ~marks * x

Fitted coefficients for trend formula:
    (Intercept)    markshickory      marksmaple       marksmisc     marksredoak
      5.2378062       1.4424915       0.6795604      -0.8482907       0.6916392
  markswhiteoak               x  markshickory:x    marksmaple:x     marksmisc:x
      1.0901772      -0.7063987       0.4511157       1.3243326       1.2138278
  marksredoak:x markswhiteoak:x
      0.5380413       0.2421379
```

The symbol `*` here is an 'interaction' in the usual sense for linear models. The fitted model is the marked Poisson process with

$$\log \lambda((x, y, m)) = \alpha_m + \beta_m x$$

where $\alpha_1, \ldots, \alpha_6$ and $\beta_1, \ldots, \beta_6$ are parameters. The intensity is loglinear in x with a different slope and intercept for each mark.

The result of `ppm` is again an object of class `"ppm"` representing a fitted point process model. To plot the fitted intensity and conditional intensity of the fitted model, use `plot.ppm`. For a multitype point process you will get a separate plot for each possible mark value.

More complicated examples are:

```
> ppm(lansing, ~marks * polynom(x, y, 2))
> ppm(lansing, ~marks * harmonic(x, y, 2))
```

25 Methods 12: Gibbs models for multitype point patterns

Gibbs point process models (section 18) are also available for marked point processes, and can be fitted to data using `ppm`. Currently the methods are only implemented for *multitype* point processes (categorical marks), so we restrict attention to this case.

25.1 Gibbs models

Much of the theory of Gibbs models described in Section 18 carries over immediately to *multitype* point processes.

25.1.1 Conditional intensity

The conditional intensity $\lambda(u, \mathbf{X})$ of an (unmarked) point process $\mathbf{X}$ at a location u was defined in section 18.5. Roughly speaking $\lambda(u, \mathbf{x})\,\mathrm{d}u$ is the conditional probability of finding a point near u, given that the rest of the point process $\mathbf{X}$ coincides with $\mathbf{x}$.

For a marked point process $\mathbf{Y}$ the conditional intensity is a function $\lambda((u, m), \mathbf{Y})$ giving a value at a location u for each possible mark m. For a *finite* set of marks M, we can interpret $\lambda((u, m), \mathbf{y})\,\mathrm{d}u$ as the conditional probability finding a point *with mark* m near u, given the rest of the marked point process.

The conditional intensity is related to the probability density $f(\mathbf{y})$ by

$$\lambda((u, m), \mathbf{y}) = \frac{f(\mathbf{y} \cup \{u\})}{f(\mathbf{y})}$$

for $(u, m) \notin \mathbf{y}$.

For Poisson processes, the conditional intensity $\lambda((u, m), \mathbf{y})$ coincides with the intensity function $\lambda(u, m)$ and does not depend on the configuration $\mathbf{y}$. For example, the homogeneous Poisson multitype point process or "CSRI" (Section 24.1.1) has conditional intensity

$$\lambda((u, m), \mathbf{y}) = \beta_m \tag{50}$$

where $\beta_m \geq 0$ are constants which can be interpreted in several equivalent ways (section 18.5). The sub-process consisting of points of type m *only* is Poisson with intensity β_m. The process obtained by ignoring the types, and combining all the points, is Poisson with intensity $\beta = \sum_m \beta_m$. The marks attached to the points are i.i.d. with distribution $p_m = \beta_m/\beta$.

25.1.2 Pairwise interactions

A *multitype* pairwise interaction process is a Gibbs process with probability density of the form

$$f(\mathbf{y}) = \alpha \left[\prod_{i=1}^{n(\mathbf{y})} b_{m_i}(x_i) \right] \left[\prod_{i<j} c_{m_i, m_j}(x_i, x_j) \right] \tag{51}$$

where $b_m(u), m \in \mathcal{M}$ are functions determining the 'first order trend' for points of each type, and $c_{m,m'}(u, v), m, m' \in \mathcal{M}$ are functions determining the interaction between a pair of points of given types m and m'. The interaction functions must be symmetric, $c_{m,m'}(u, v) = c_{m,m'}(v, u)$ and $c_{m,m'} \equiv c_{m',m}$. The conditional intensity is

$$\lambda((u, m); \mathbf{y}) = b_m(u) \prod_{i=1}^{n(\mathbf{y})} c_{m,m_i}(u, x_i). \tag{52}$$

25.1.3 Pairwise interactions not depending on marks

The simplest examples of multitype pairwise interaction processes are those in which the interaction term $c_{m,m'}(u,v)$ does not depend on the marks m, m'. For example, we can take any of the interaction functions $c(u,v)$ described in section 18.3 and use it to construct a marked point process.

Such processes can be constructed equivalently as follows [8]:

- an *unmarked* Gibbs process is generated with first order term $b(u) = \sum_{m\in\mathcal{M}} b_m(u)$ and pairwise interaction $c(u,v)$.
- each point x_i of this unmarked process is labelled with a mark m_i with probability distribution $\mathbb{P}\{m_i = m\} = b_i(x_i)/b(x_i)$ independent of other points.

If additionally the intensity functions are constant, $b_m(u) \equiv \beta_m$, then such a point process has the random labelling property.

25.1.4 Mark-dependent pairwise interactions

Various complex kinds of behaviour can be created by postulating a pairwise interaction that does depend on the marks.

A simple example is the *multitype hard core process* in which $\beta_m(u) \equiv \beta$ and

$$c_{m,m'}(u,v) = \begin{cases} 1 & \text{if } ||u-v|| > r_{m,m'} \\ 0 & \text{if } ||u-v|| \le r_{m,m'} \end{cases} \tag{53}$$

where $r_{m,m'} = r_{m',m} > 0$ is the hard core distance for type m with type m'. In this process, two points of type m and m' respectively can never come closer than the distance $r_{m,m'}$.

By setting $r_{m,m'} = \infty$ for a particular pair of marks m, m' we effectively remove the interaction term between points of these types. If there are only two types, say $\mathcal{M} = \{1,2\}$, then setting $r_{1,2} = \infty$ implies that the sub-processes $\mathbf{X}_1$ and $\mathbf{X}_2$, consisting of points of types 1 and 2 respectively, are independent point processes. In other words the process satisfies the independence-of-components property.

The *multitype Strauss process* has pairwise interaction term

$$c_{m,m'}(u,v) = \begin{cases} 1 & \text{if } ||u-v|| > r_{m,m'} \\ \gamma_{m,m'} & \text{if } ||u-v|| \le r_{m,m'} \end{cases} \tag{54}$$

where $r_{m,m'} > 0$ are interaction radii as above, and $\gamma_{m,m'} \ge 0$ are interaction parameters.

In contrast to the unmarked Strauss process, which is well-defined only when its interaction parameter γ is between 0 and 1, the multitype Strauss process allows some of the interaction parameters $\gamma_{m,m'}$ to exceed 1 for $m \ne m'$, provided one of the relevant types has a hard core ($\gamma_{m,m} = 0$ or $\gamma_{m',m'} = 0$).

If there are only two types, say $\mathcal{M} = \{1,2\}$, then setting $\gamma_{1,2} = 1$ implies that the sub-processes $\mathbf{X}_1$ and $\mathbf{X}_2$, consisting of points of types 1 and 2 respectively, are independent Strauss processes.

The *multitype Strauss-hard core process* has pairwise interaction term

$$c_{m,m'}(u,v) = \begin{cases} 0 & \text{if } ||u-v|| < h_{m,m'} \\ \gamma_{m,m'} & \text{if } h_{m,m'} \le ||u-v|| \le r_{m,m'} \\ 1 & \text{if } ||u-v|| > r_{m,m'} \end{cases} \tag{55}$$

where $r_{m,m'} > 0$ are interaction distances and $\gamma_{m,m'} \ge 0$ are interaction parameters as above, and $h_{m,m'}$ are hard core distances satisfying $h_{m,m'} = h_{m',m}$ and $0 < h_{m,m'} < r_{m,m'}$.

25.2 Pseudolikelihood for multitype Gibbs processes

Models can be fitted by maximum pseudolikelihood. For a multitype Gibbs point process with conditional intensity $\lambda((u,m);\mathbf{y})$, the log pseudolikelihood is

$$\log \mathsf{PL} = \sum_{i=1}^{n(\mathbf{y})} \log \lambda((x_i,m_i);\mathbf{y}) - \sum_{m\in\mathcal{M}} \int_W \lambda((u,m);\mathbf{y})\,\mathrm{d}u. \tag{56}$$

The pseudolikelihood can be maximised using an extension of the Berman-Turner device [3].

25.3 Fitting Gibbs models to multitype data

Marked point process models may be fitted to point pattern data using `ppm`. Currently the methods are only available for multitype point processes (categorical marks).

25.3.1 Interactions not depending on marks

The model-fitting function `ppm` expects an argument `interaction` that specifies the interpoint interaction structure of the point process. The default is 'no interaction', corresponding to a Poisson process.

On page 118 there is a list of interpoint interactions for modelling *unmarked* point patterns. These interactions can also be used, without modification, to fit models to *multitype* point patterns.

For example

```
> ppm(lansing, ~marks, Strauss(0.07))
```

fits a multitype version of the Strauss process (section 18.3.2) in which the conditional intensity is

$$\lambda((u,m),\mathbf{y}) = \beta_m \gamma^{t(u,\mathbf{y})}.$$

Here β_m are constants which account for the unequal abundance of the different species of tree. The other quantities are the same as in (42). The interaction between two trees is assumed to be the same for all species, and is controlled by the interaction parameter γ and interaction radius $r = 0.07$. For example, this includes the case $\gamma = 0$ where no two trees (whatever species they belong to) come closer than 0.07 units apart, a 'multitype hard core process'.

25.3.2 Interactions depending on marks

There are two additional interpoint interactions defined in `spatstat` for multitype point patterns:

`MultiStrauss`	the multitype Strauss process
`MultiStraussHard`	multitype hybrid hard core / Strauss process

In these models, the interaction between two points depends on the types of the points as well as their separation. For example, in the multitype Strauss process, for each pair of types i and j there is an interaction radius r_{ij} and interaction parameter γ_{ij}.

To fit the stationary multitype Strauss process to the dataset `betacells`:

```
> data(betacells)
> r <- 30 * matrix(c(1, 2, 2, 1), nrow = 2, ncol = 2)
> ppm(betacells, ~1, MultiStrauss(c("off", "on"), r), rbord = 60)
```

```
Stationary Multitype Strauss process
Possible marks:
off on

First order terms:
    beta_off      beta_on
0.0001373652 0.0001373652

Interaction: Pairwise interaction family
Interaction:         Multitype Strauss process
2 types of points
Possible types:
[1] "off" "on"
Interaction radii:
    off on
off  30 60
on   60 30
Fitted interaction parameters gamma_ij:
       off     on
off 0.0000 0.8303
on  0.8303 0.0000

Relevant coefficients:
markoffxoff  markoffxon   markonxon
-17.2378706  -0.1860184 -17.2138383
```

To fit a nonstationary multitype Strauss process with log-cubic polynomial trend:

```
> ppm(betacells, ~polynom(x, y, 3), MultiStrauss(c("off", "on"),
+     r), rbord = 60)
```

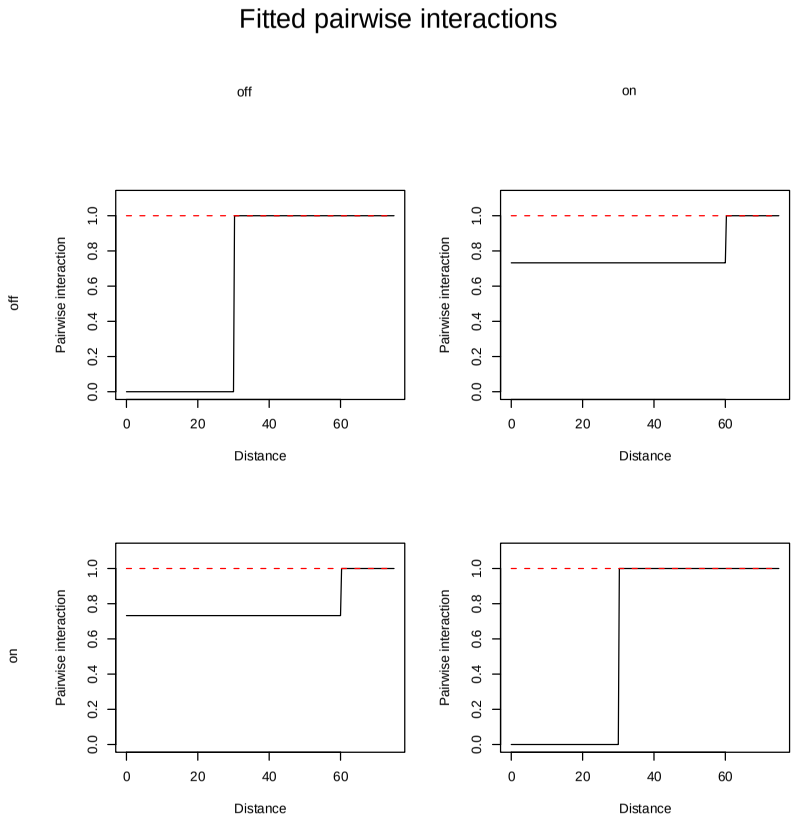
For more detailed explanation and examples of modelling and the interpretation of model formulae for point processes, see [5].

25.3.3 Plotting the fitted interaction

The fitted pairwise interaction in a point process model can be plotted using `fitin`. The value returned by `fitin` is a function array (class `"fasp"`).

```
> model <- ppm(betacells, ~polynom(x, y, 3), MultiStrauss(c("off",
+     "on"), r), rbord = 60)
```

```
> plot(fitin(model))
```

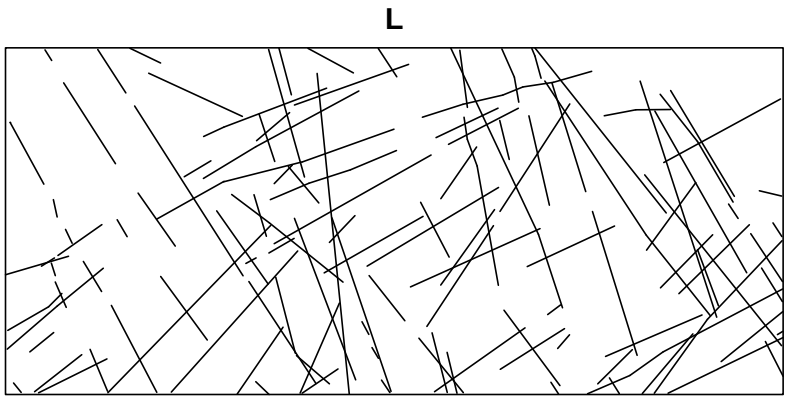


26 Line segment data

`spatstat` also has some facilities for handling spatial patterns of *line segments*.

For example, the `copper` dataset in `spatstat` contains a dataset `copper$Lines` that records the locations of geological faults in a survey region.

```
> data(copper)
> L <- copper$Lines
> L <- rotate(L, pi/2)
> plot(L)
```



A spatial pattern of line segments is represented by an object of class `"psp"`. It consists of a list of line segments (given by the coordinates of their two endpoints), and a window in which the line segments were observed. The line segments may also carry marks.

Objects of class `"psp"` can be created by the function `psp` or obtained by converting other data using the function `as.psp`.

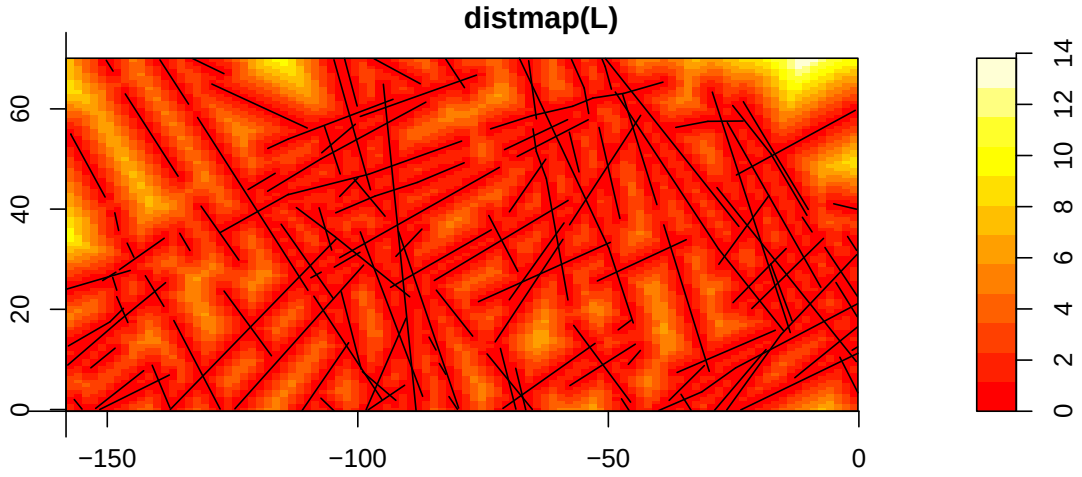
Capabilities available for this class include:

`[.psp`	subset operator (also performs clipping)
`marks.psp`	extract marks
`endpoints.psp`	extract midpoints of line segments
`midpoints.psp`	compute midpoints of line segments
`lengths.psp`	compute lengths of line segments
`angles.psp`	compute angles of orientation for line segments
`rotate.psp`	rotate a line segment pattern
`shift.psp`	shift a line segment pattern
`affine.psp`	apply affine transformation
`pairdist.psp`	distances between line segments
`crossdist.psp`	distances between line segments
`nndist.psp`	closest distances between line segments
`density.psp`	kernel-smoothed intensity image
`crossing.psp`	find intersection points between line segments
`selfcrossing.psp`	find intersection points between line segments
`unitname.psp`	determine units of length
`rescale.psp`	change units of length
`rshift.psp`	apply random shift to each line segment

There are also the usual methods

`plot.psp`	plot a line segment pattern
`print.psp`	print information on a line segment pattern
`summary.psp`	compute summary of a line segment pattern

```
> summary(L)

146 line segments
Lengths:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
 0.09242  6.61400 12.18000 15.02000 19.95000 65.48000
Total length: 2192.57251480451 km
Length per unit area: 0.196937548404655
Angles (radians):
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
0.008107 0.549500 1.747000 1.378000 2.113000 2.912000
Window: polygonal boundary
single connected closed polygon with 4 vertices
enclosing rectangle: [-158.23, -0.19] x [-0.335, 70.11] km
Window area =  11133.3 square km
Unit of length: 1 km

> plot(distmap(L))
> plot(L, add = TRUE)
```



27 Further information on `spatstat`

Help files

For information on a particular command in `spatstat`, consult the online help file by typing `help(`*command*`)`. The help files are detailed and extensive. The complete manual is over 500 pages.

For examples of the use of a particular command, read the examples section in the help file, or type `example(`*command*`)` to see the examples executed.

Quick reference

Type `help(spatstat)` for a quick-reference overview of all the functions available in the package.

For a demonstration of many of the capabilities of `spatstat`, type `demo(spatstat)`.

For a visual display of all the datasets supplied in `spatstat`, type `demo(data)`.

Website

The website `www.spatstat.org` contains information on recent updates to the package, frequently-asked questions, bug fixes, literature and other developments.

Modelling

For examples on fitting point process models, see [5].

Citation

If you use `spatstat` in a research publication, it would be much appreciated if you could cite the paper [4], or mention `spatstat` in the acknowledgements.

In doing so, you will help us to justify the expenditure of time and effort on maintaining and developing the package.

Citation details are also available in the package by typing `citation(package="spatstat")`.

Queries and requests

If you have difficulty in getting the package to do what you want, or if you have a suggestion for additional features that could be added, please contact the package authors, `adrian@maths.uwa.edu.au` and `r.turner@auckland.ac.nz`, or email the R special interest group in spatial and geographical statistics, `r-sig-geo@stat.math.ethz.ch`.

References

[1] A. Baddeley, J. Møller, and A.G. Pakes. Properties of residuals for spatial point processes. *Annals of the Institute of Statistical Mathematics*, 2007. To appear. Accepted for publication 6 July 2007.

[2] A. Baddeley, J. Møller, and R. Waagepetersen. Non- and semiparametric estimation of interaction in inhomogeneous point patterns. *Statistica Neerlandica*, 54(3):329–350, November 2000.

[3] A. Baddeley and R. Turner. Practical maximum pseudolikelihood for spatial point patterns (with discussion). *Australian and New Zealand Journal of Statistics*, 42(3):283–322, 2000.

[4] A. Baddeley and R. Turner. Spatstat: an R package for analyzing spatial point patterns. *Journal of Statistical Software*, 12(6):1–42, 2005. URL: `www.jstatsoft.org`, ISSN: 1548-7660.

[5] A. Baddeley and R. Turner. Modelling spatial point patterns in R. In A. Baddeley, P. Gregori, J. Mateu, R. Stoica, and D. Stoyan, editors, *Case Studies in Spatial Point Pattern Modelling*, number 185 in Lecture Notes in Statistics, pages 23–74. Springer-Verlag, New York, 2006. ISBN: 0-387-28311-0.

[6] A. Baddeley, R. Turner, J. Møller, and M. Hazelton. Residual analysis for spatial point processes (with discussion). *Journal of the Royal Statistical Society, series B*, 67(5):617–666, 2005.

[7] A.J. Baddeley. Spatial sampling and censoring. In O.E. Barndorff-Nielsen, W.S. Kendall, and M.N.M. van Lieshout, editors, *Stochastic Geometry: Likelihood and Computation*, chapter 2, pages 37–78. Chapman and Hall, London, 1998.

[8] A.J. Baddeley and J. Møller. Nearest-neighbour Markov point processes and random sets. *International Statistical Review*, 57:89–121, 1989.

[9] A.J. Baddeley, R.A. Moyeed, C.V. Howard, and A. Boyde. Analysis of a three-dimensional point pattern with replication. *Applied Statistics*, 42(4):641–668, 1993.

[10] A.J. Baddeley and B.W. Silverman. A cautionary example on the use of second-order methods for analyzing point patterns. *Biometrics*, 40:1089–1094, 1984.

[11] A.J. Baddeley and M.N.M. van Lieshout. Area-interaction point processes. *Annals of the Institute of Statistical Mathematics*, 47:601–619, 1995.

[12] M. Bell and G. Grunwald. Mixed models for the analysis of replicated spatial point patterns. *Biostatistics*, 5:633–648, 2004.

[13] M. Berman and T.R. Turner. Approximating point process likelihoods with GLIM. *Applied Statistics*, 41:31–38, 1992.

[14] J. Besag and P.J. Diggle. Simple Monte Carlo tests for spatial pattern. *Applied Statistics*, 26:327–333, 1977.

[15] J.E. Besag and P. Clifford. Generalized Monte Carlo significance tests. *Biometrika*, 76:633–642, 1989.

[16] D.R. Brillinger. Comparative aspects of the study of ordinary time series and of point processes. In P.R. Krishnaiah, editor, *Developments in Statistics*, pages 33–133. Academic Press, 1978.

[17] N.A.C. Cressie. *Statistics for Spatial Data*. John Wiley and Sons, New York, 1991.

[18] D.J. Daley and D. Vere-Jones. *An Introduction to the Theory of Point Processes*. Springer Verlag, New York, 1988.

[19] P.J. Diggle. *Statistical analysis of spatial point patterns*. Academic Press, London, 1983.

[20] P.J. Diggle. A point process modelling approach to raised incidence of a rare phenomenon in the vicinity of a prespecified point. *Journal of the Royal Statistical Society, series A*, 153:349–362, 1990.

[21] P.J. Diggle. *Statistical Analysis of Spatial Point Patterns*. Arnold, second edition, 2003.

[22] P.J. Diggle, N. Lange, and F. M. Benes. Analysis of variance for replicated spatial point patterns in clinical neuroanatomy. *Journal of the American Statistical Association*, 86:618–625, 1991.

[23] P.J. Diggle, J. Mateu, and H.E. Clough. A comparison between parametric and non-parametric approaches to the analysis of replicated spatial point patterns. *Advances in Applied Probability (SGSA)*, 32:331–343, 2000.

[24] P.J. Diggle and B. Rowlingson. A conditional approach to point process modelling of elevated risk. *Journal of the Royal Statistical Society, series A (Statistics in Society)*, 157(3):433–440, 1994.

[25] A.C.A. Hope. A simplified Monte Carlo significance test procedure. *Journal of the Royal Statistical Society, series B*, 30:582–598, 1968.

[26] C.V. Howard, S. Reid, A.J. Baddeley, and A. Boyde. Unbiased estimation of particle density in the tandem-scanning reflected light microscope. *Journal of Microscopy*, 138:203–212, 1985.

[27] F. Huang and Y. Ogata. Improvements of the maximum pseudo-likelihood estimators in various spatial statistical models. *Journal of Computational and Graphical Statistics*, 8(3):510–530, 1999.

[28] J.F.C. Kingman. *Poisson Processes*. Oxford University Press, 1993.

[29] G.M. Laslett. Censoring and edge effects in areal and line transect sampling of rock joint traces. *Mathematical Geology*, 14:125–140, 1982.

[30] P.A.W. Lewis. Recent results in the statistical analysis of univariate point processes. In P.A.W. Lewis, editor, *Stochastic point processes*, pages 1–54. Wiley, New York, 1972.

[31] J.K. Lindsey. *The analysis of stochastic processes using GLIM*. Springer, Berlin, 1992.

[32] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.

[33] J. Møller and R.P. Waagepetersen. *Statistical Inference and Simulation for Spatial Point Processes*. Chapman and Hall/CRC, Boca Raton, 2003.

[34] J. Møller and R.P. Waagepetersen. Modern statistics for spatial point processes. Research Report R-2006-12, Department of Mathematical Sciences, Aalborg University, April 2006. Submitted for publication.

[35] Y. Ogata. Statistical models for earthquake occurrences and residual analysis for point processes. *Journal of the American Statistical Association*, 83:9–27, 1988.

[36] B.D. Ripley. Modelling spatial patterns (with discussion). *Journal of the Royal Statistical Society, series B*, 39:172–212, 1977.

[37] B.D. Ripley. Simulating spatial patterns: dependent samples from a multivariate density. *Applied Statistics*, 28:109–112, 1979.

[38] B.D. Ripley. *Spatial Statistics*. John Wiley and Sons, New York, 1981.

[39] B.D. Ripley. *Statistical Inference for Spatial Processes*. Cambridge University Press, 1988.

[40] A. Särkkä. *Pseudo-likelihood approach for pair potential estimation of Gibbs processes*. Number 22 in Jyväskylä Studies in Computer Science, Economics and Statistics. University of Jyväskylä, 1993.

[41] D. Stoyan and P. Grabarnik. Second-order characteristics for stochastic structures connected with Gibbs point processes. *Mathematische Nachrichten*, 151:95–100, 1991.

[42] D. Stoyan and H. Stoyan. *Fractals, Random Shapes and Point Fields*. John Wiley and Sons, Chichester, 1995.

[43] M.N.M. van Lieshout. *Markov Point Processes and their Applications*. Imperial College Press, 2000.

[44] M.N.M. van Lieshout and A.J. Baddeley. A nonparametric measure of spatial interaction in point patterns. *Statistica Neerlandica*, 50:344–361, 1996.

[45] R. Waagepetersen. An estimating function approach to inference for inhomogeneous Neyman-Scott processes. Submitted for publication, 2006.

Index