

Adrian Baddeley, Ege Rubak and Rolf Turner

Analysing spatial point patterns in R



Contents

I	BASICS	1
1	Introduction	3
1.1	Point patterns	3
1.2	Statistical methodology for point patterns	12
1.3	About this book	17
2	Software Essentials	19
2.1	Introduction to R	19
2.2	Packages for R	34
2.3	Introduction to <code>spatstat</code>	36
2.4	Getting started with <code>spatstat</code>	39
2.5	Core concepts in <code>spatstat</code>	40
2.6	FAQ's	42
3	Collecting and Handling Point Pattern Data	45
3.1	Surveys and experiments	45
3.2	Data handling	48
3.3	Entering point pattern data into <code>spatstat</code>	50
3.4	Data errors and quirks	53
3.5	Pixel images in <code>spatstat</code>	56
3.6	Windows in <code>spatstat</code>	59
3.7	Collections of objects	63
3.8	Interactive data entry	64
3.9	Reading GIS file formats	64
3.10	FAQ's	70
4	Inspecting and Exploring Data	71
4.1	Plotting	71
4.2	Manipulating point patterns and windows	87
4.3	Exploring images	104
4.4	Tessellations	112
4.5	FAQ's	115
5	Point processes	117
5.1	Not included in review	117
II	EXPLORATORY DATA ANALYSIS	119

6 Intensity	121
6.1 Introduction	121
6.2 Estimating homogeneous intensity	123
6.3 Theory	125
6.4 Quadrat counting	127
6.5 Smoothing estimation of intensity function	131
6.6 Investigating dependence of intensity on a covariate	139
6.7 Formal tests of (non-)dependence on a covariate	145
6.8 Hot spots, clusters, and local features	149
6.9 Kernel smoothing of marks	156
6.10 Multitype intensity and relative risk	158
6.11 Intensity in 3D space and space-time	162
6.12 FAQ's	162
7 Correlation	163
7.1 Introduction	163
7.2 Manual methods	164
7.3 The K function	167
7.4 Edge corrections for the K -function*	177
7.5 The class "fv"	184
7.6 The pair correlation function	190
7.7 Standard errors and confidence intervals	194
7.8 Testing statistical significance	196
7.9 Detecting anisotropy	202
7.10 Adjusting for inhomogeneity	208
7.11 Theory*	213
7.12 FAQ's	217
— SUBSEQUENT SOFTWARE UPDATES —	219
8 Shortest distances and empty spaces	221
8.1 Manual methods	221
8.2 Nearest-neighbour function G and empty-space function F	227
8.3 Formal inference and diagnostic plots	234
8.4 Empty space hazard	238
8.5 J -function	241
8.6 Inhomogeneous F , G and J functions	243
8.7 Distance to another spatial pattern	243
8.8 Theory for edge corrections*	246
8.9 Conditioning*	253
8.10 FAQ's	260
— SUBSEQUENT SOFTWARE UPDATES —	262
III STATISTICAL INFERENCE	263

9 Poisson models	265
9.1 Introduction	265
9.2 Poisson point process models	266
9.3 Fitting Poisson models in <code>spatstat</code>	271
9.4 Statistical inference for Poisson models	293
9.5 Theory*	302
9.6 Coarse quadrature approximation*	307
9.7 Pixel approximation*	312
9.8 Conditional logistic regression*	318
9.9 Non-loglinear models	320
9.10 FAQ's	325
10 Hypothesis Tests and Simulation Envelopes	327
10.1 Introduction	327
10.2 Terminology	328
10.3 Testing for a covariate effect in a parametric model	329
10.4 Model selection using AIC	335
10.5 Goodness-of-fit tests for an intensity model	337
10.6 Goodness-of-fit tests of independence between points	340
10.7 Monte Carlo tests	340
10.8 Monte Carlo tests based on summary functions	347
10.9 Envelopes in <code>spatstat</code>	354
10.10 Hahn tests?	360
10.11 Performance	360
10.12 Conclusions	363
10.13 FAQ's	363
11 Validation of Poisson models	367
11.1 Overview of techniques	368
11.2 Goodness-of-fit tests of a fitted model	368
11.3 Relative intensity	371
11.4 Residuals for Poisson processes	372
11.5 Partial residual plots	380
11.6 Added variable plots	383
11.7 Leverage and influence	384
11.8 Validating the independence assumption	396
12 Model-fitting using second moments	403
12.1 Not included in review	403
13 Gibbs models	405
13.1 Not included in review	405
14 Inference for multitype patterns	407
14.1 Not included in review	407

IV ADDITIONAL STRUCTURE	409
15 Inference for multivariate marks	411
15.1 Not included in review	411
16 Replicated point patterns	413
16.1 Not included in review	413
17 Point patterns on a linear network	415
17.1 Not included in review	415
Bibliography	417
Index	437

Part I

BASICS



1

Introduction

1.1 Point patterns

1.1.1 Points

A “spatial point pattern” is a dataset giving the observed spatial locations of things or events. Examples include the locations of trees in a forest, gold deposits mapped in a geological survey, stars in a star cluster, road accidents, earthquake epicentres, mobile phone calls, animal sightings, or cases of a rare disease.

The common characteristic of these examples is not just that they are “point data”, but that the point locations are not determined in advance of the study, and indeed the spatial arrangement of points is the main focus of investigation.

Interest in methods for analysing spatial point pattern data is rapidly expanding across many fields of science, notably in ecology, epidemiology, geoscience, astronomy, econometrics, and crime research.

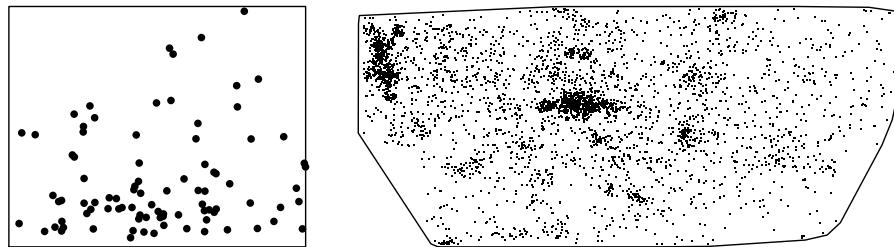


Figure 1.1: Point pattern datasets with spatially-varying density of points. *Left:* enterochromaffin-like cells in histological section of gastric mucosa (T. Bendtsen; see [250, pp. 2, 169]), interior of stomach towards top of picture. *Right:* sky positions of 4215 galaxies in the Shapley Supercluster (M. Drinkwater).

The left panel of Figure 1.1 shows the locations of a particular type of cell observed in an optical microscope section of tissue from the lining of the stomach. The interior of the stomach is toward the top of the picture. There is an obvious gradient in the abundance of these cells. The right panel of the Figure shows the sky positions of galaxies observed in an astronomical survey. There is a dramatic concentration of galaxies in two areas.

Statistical analysis of the spatial arrangement of points can reveal important features: for ex-

ample, a tendency for gold deposits to be found close to a major geological fault, or for cases of a disease to be more prevalent near a pollution source, or for bird nests to maintain a certain minimum separation from each other. Analysis of point pattern data has provided pivotal evidence for important research on everything from the transmission of cholera [304] to the behaviour of serial killers [202, 76] to the large-scale structure of the universe [239].

Figure 1.2 shows the locations of larvae of the waterstrider *Limnoperus (Gerris) rufoscutellatus* (larval stage V) on the surface of a pond, observed in three different photographs. The research question is whether the larvae seem to be exhibiting territorial behaviour. If the larvae are territorial we would expect the spacing between points to be larger than if the points had been strewn completely at random. The human eye is not very good at judging this question: some kind of objective analysis is needed.

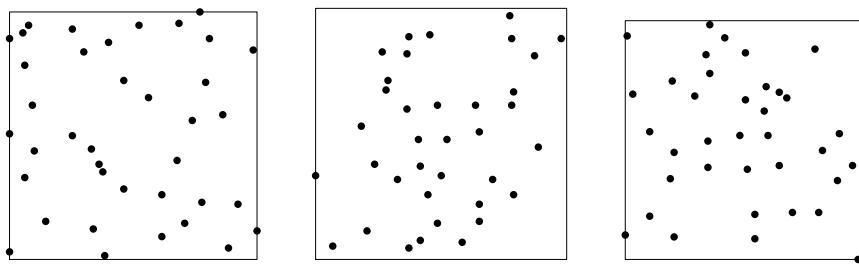


Figure 1.2: Waterstrider larvae recorded from three photographs of the middle of a pond. Each frame is about 48 cm across. Data recorded by Dr Matti Nummelin, University of Helsinki, and kindly contributed by Professor Antti Penttilä.

Figure 1.3 depicts the locations of 204 seedlings and saplings of Japanese black pine (*Pinus Thunbergii*) recorded in a 10×10 metre sampling region within a natural forest stand [256]. In mapping a snapshot of the forest, we hope to understand ecological processes, such as competition for resources (soil nutrients, light, water, growing space), and spatial variation in the landscape, such as variation in soil type or soil fertility. A detailed analysis of these data [258, 259] suggested that *both* these phenomena are present: there is spatial variation in the density of the forest, and also a tendency for trees to avoid growing close together, suggesting competition between neighbouring plants.

As these examples illustrate, the spatial arrangement of points in a point pattern is often a surrogate for unobservable spatial variables (such as soil fertility or pollution exposure), or is the cumulative effect of unrecorded historical events (such as territorial behaviour, forest succession, geological mineralisation history, or cosmological evolution).

There is no simple “drag-and-drop” solution for statistical analysis of spatial point patterns, where we would simply instruct the computer to “analyse” our data. It is a key principle of statistical methodology that the correct way to analyse the data does not simply depend on the format of the data. It depends on how the data were obtained, and of course on the objectives of the analysis.

1.1.2 Points of several types

The points in a point pattern are often classified into different types. For example, trees may be classified into different species. This introduces a new set of scientific questions and requires new kinds of statistical analysis.

Figure 1.4 shows the locations of birch (*Betula celtiberica*) and oak (*Quercus robur*) trees in a secondary wood in Urkiola Natural Park, Basque country, northern Spain. These are part of a more extensive dataset collected and analysed by Laskurain [215]. One important question is whether the

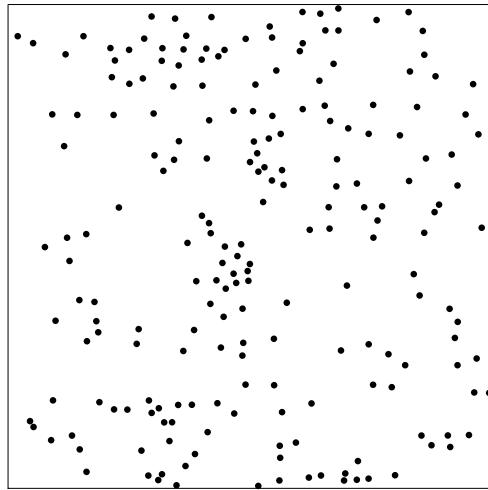


Figure 1.3: Japanese black pine data of Numata [256] showing 204 seedlings and saplings in a 10 metre square sampling region. Data kindly supplied by Professors Y. Ogata and M. Tanemura.

two species have the same spatial distribution, or whether the relative proportions of the two species are spatially varying. In an extreme case a forest can be *segregated* into stands of different species, in the sense that there are regions where one species predominates over the other.

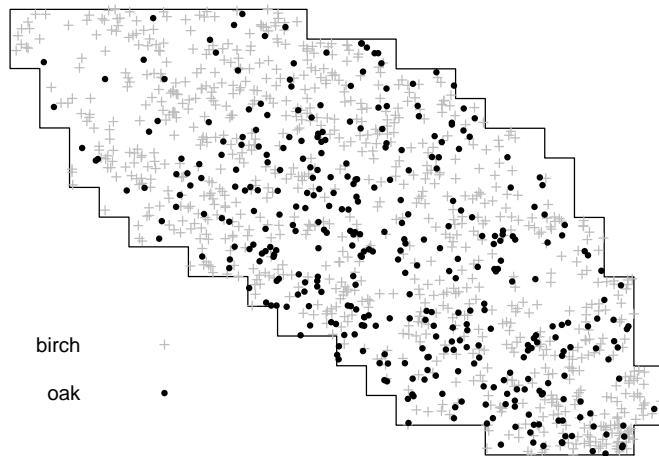


Figure 1.4: Urkiola Woods data: locations of birch and oak trees in a secondary wood in Urkiola Natural Park. Map about 220 metres across. Data collected by N.A. Laskurain, kindly communicated by M. de la Cruz Rot.

Figure 1.5 shows a point pattern of displaced amacrine cells in the retina of a rabbit, with cells categorised as either “on” or “off” according to their response to stimuli. Cells of the *same* type are regularly spaced apart: this is expected, because the on and off cells grow in two separate layers. A key research question is whether the two layers grew separately. This would be contradicted if cells of *opposite* type appeared to be correlated, that is, if the placement of the “on” cells appeared to be affected by the location of the “off” cells, or *vice versa*.

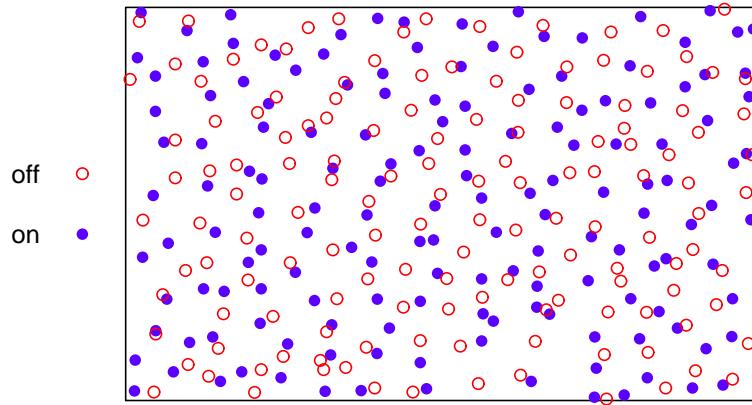


Figure 1.5: Amacrine cells data: centres of 152 “on” cells and 142 “off” cells in a rectangular sampling frame about 1060 by 662 microns from the retina of a rabbit. Data from Professor Austin Hughes, kindly supplied by Professor Peter Diggle.

Figure 1.6 shows the locations of influenza virus proteins on the surface of an infected cell [80]. The protein locations were mapped by immunogold labelling, that is, by growing antibodies to the specific proteins, attaching a gold particle to each antibody, and subsequently imaging the gold particles in electron microscopy. The research problem is to decide whether there is spatial association between the two proteins: this is important for the study of viral replication. While these data are superficially similar in structure to the amacrine cells data, the required analysis is completely different. Whereas the amacrine cells belong to two highly-organised layers, it is appropriate to treat the individual influenza proteins (the individual dots and crosses in Figure 1.6) as individual biochemical entities, each responding to its local environment.

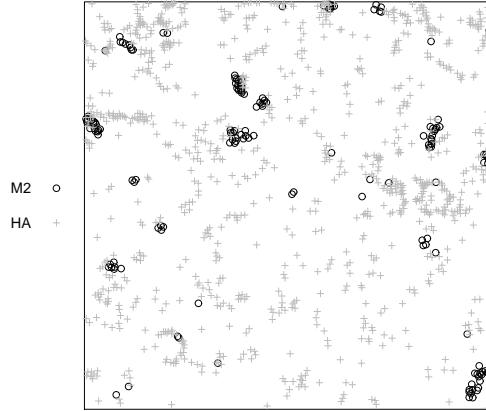


Figure 1.6: Locations of influenza virus proteins M2 and HA, mapped by immunogold labelling, on the surface of an infected cell. Field width is 3331 nanometres. Data kindly supplied by Dr G.P. Leser and Dr R.A. Lamb.

Figure 1.7 shows the spatial locations of nests of two species of ants, *Messor wasmanni* and *Cataglyphis bicolor*, recorded by Professor R.D. Harkness at a site in northern Greece, and described in [177]. The harvester ant *M. wasmanni* collects seeds for food and builds a nest composed mainly of seed husks. *C. bicolor* is a heat-tolerant desert foraging ant which eats dead insects and other arthropods. Interest focuses on whether there is evidence in the data for intra-species competition between *Messor* nests (i.e. competition for resources) and for preferential placement of *Cataglyphis* nests in the vicinity of *Messor* nests. In this example, the role of the two species is not symmetric in the analysis, since one is potential prey for the other.

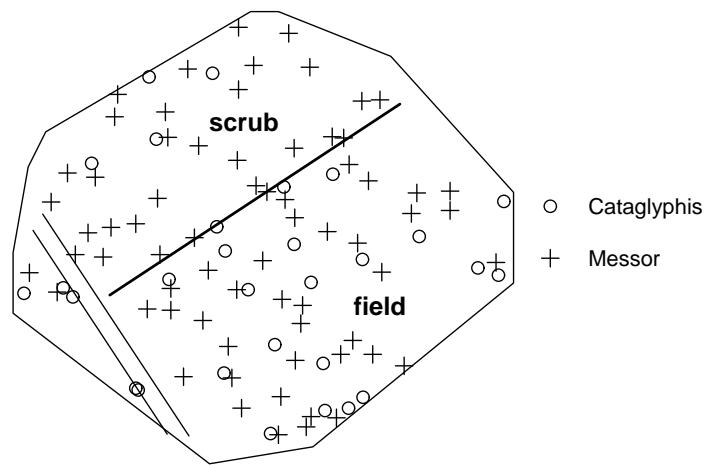


Figure 1.7: Ants' nests of two species (*Messor wasmanni* and *Cataglyphis bicolor*) in a survey region about 425 feet (130 metres) across. Parallel lines show a walking track.

1.1.3 Marked point patterns

The points in a point pattern dataset may carry all kinds of attributes: trees may be labelled by species and diameter; a catalogue of stars may give their sky positions, masses and magnitudes; disease case locations may be linked to clinical records. Such information, attached to each point in the point pattern, is called a **mark** variable.

If the mark attached to each point is a single *categorical* value (such as a species label or a disease status) then we have a multitype point pattern as described above.

Figure 1.8 shows data from a forest survey in which the size of each tree was measured by its diameter at breast height (*dbh*). The locations and diameters of 584 Longleaf Pine (*Pinus palustris*) trees in a 200×200 metre region in southern Georgia (USA), were collected and analysed by Platt, Evans and Rathbun [271]. Each tree is represented in the Figure by a circle, centred at the tree location, with diameter proportional to *dbh*. Space-time models of forest succession have been used to account for the spatial pattern in this study [277]. In the upper right quarter of the survey there appears to be an area of smaller (and therefore probably younger) trees, suggesting that the forest may have been cleared in this area in previous decades.

Figure 1.9 shows a longitudinal plane section through a filter made from bronze powder. The circles are the plane section profiles of bronze particles. The material was produced by sedimentation of bronze powder with varying grain diameter and subsequent sintering [57].

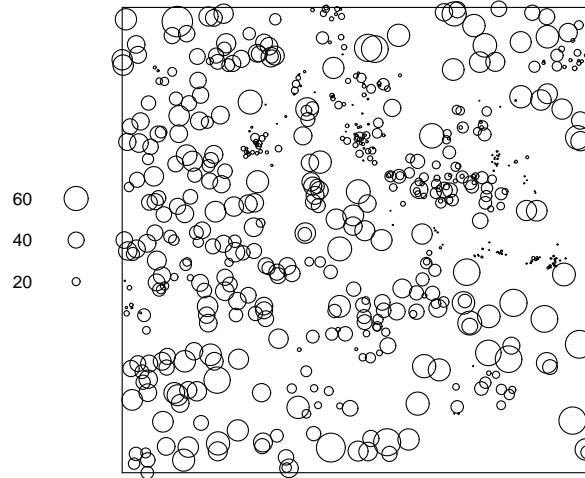


Figure 1.8: Longleaf Pines data: tree locations and diameters in a 200 metre square survey region. Tree diameters are not to scale: legend shows diameters in centimetres.

This example shows that physical objects, which are too large to be treated as ideal points at the scale of interest, can nevertheless be accommodated by recording the location of the centre of the object as a point, and treating the object's size and shape as attributes.

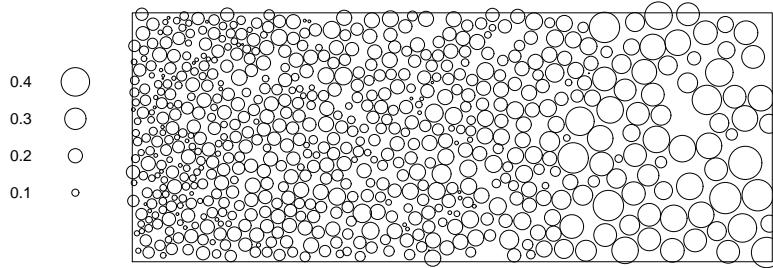


Figure 1.9: Bronze particles in a filter, observed in longitudinal plane section. (R. Bernhardt, Dresden; H. Wendrock; kindly contributed by U. Hahn.)

In principle the mark attached to each point could be any type of information. It is often a multivariate observation: for example, in forest survey data, each tree could be marked by its species, its diameter, *and* a chemical analysis of its leaves.

1.1.4 Covariates

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

One type of covariate is a *spatial function* $Z(u)$ defined at all spatial locations u . Figure 1.10

shows the locations of 3605 trees in a tropical rainforest, with a supplementary grid map of the terrain elevation (altitude). The covariate $Z(u)$ is the altitude at location u . Research questions for investigation include whether the forest density depends on the terrain, and whether, after accounting for the influence of terrain, there is evidence of spatial clustering of the trees.



Figure 1.10: Tropical rainforest data. Locations of *Beilschmiedia pendula* trees (+) and terrain elevation (greyscale) in a 1000×500 metre survey plot in Barro Colorado Island. Part of a larger dataset containing the positions of hundreds of thousands of trees belonging to thousands of species [189, 91, 90].

Another common type of covariate data is a *spatial pattern* such as another point pattern, or a line segment pattern and so on. Figure 1.11 shows the result of an intensive geological survey of mineralisation in a 158×35 kilometre region in Queensland, Australia. It is a map of copper deposits (essentially pointlike at this scale) and geological lineaments (straight lines). Lineaments are linear structures, mostly geological faults, which can easily be observed in aerial surveys. Copper deposits are hard to find, so the main question is whether the faults are ‘predictive’ for copper deposits, for example, whether copper is more likely to be found near faults. This kind of relationship is true for some minerals in some mineral provinces, typically because the mineralisation process involved liquids carrying the target mineral in solution which flowed from the deep earth to the surface through openings in the rock. Common practice is to define $Z(u)$ to be the distance from location u to the *nearest* lineament, and to use the function Z as the spatial covariate.

Figure 1.12 shows the Chorley-Ribble cancer data of Diggle [126] giving the residential locations of new cases of cancer of the larynx (58 cases) and cancer of the lung (978 cases) in the Chorley and South Ribble Health Authority of Lancashire, England, between 1974 and 1983. The location of a disused industrial incinerator is also given. The aim is to assess evidence for an increase in the incidence of laryngeal cancer close to the incinerator. The lung cancer cases serve as a surrogate for the spatially-varying density of the susceptible population. Data analysis in [126, 129, 34] concluded there is significant evidence of an incinerator effect.

1.1.5 Different spaces

In this book the points are usually locations in two-dimensional space, but they could also be locations in one dimension (such as road accidents recorded on a road network) or in three dimensions (such as cells observed in 3D microscopy) or in space-time (such as earthquake epicentre locations and times).

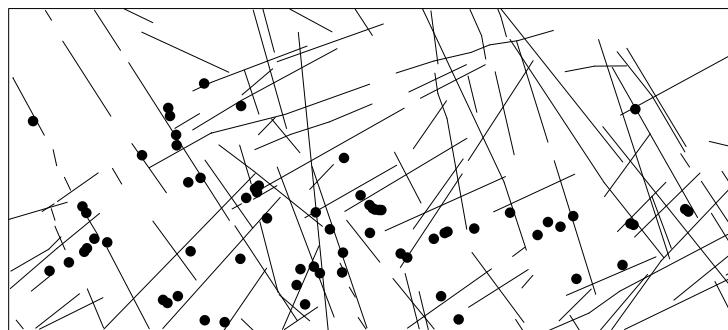


Figure 1.11: Queensland copper data. Copper deposits (●) and geological lineaments (—) in a 158×35 km survey region. Dr. Jonathan Huntington, CSIRO Earth Science and Resource Engineering, Sydney, Australia. Coordinates kindly provided by Dr. Mark Berman and Dr. Andy Green, CSIRO, Sydney, Australia.

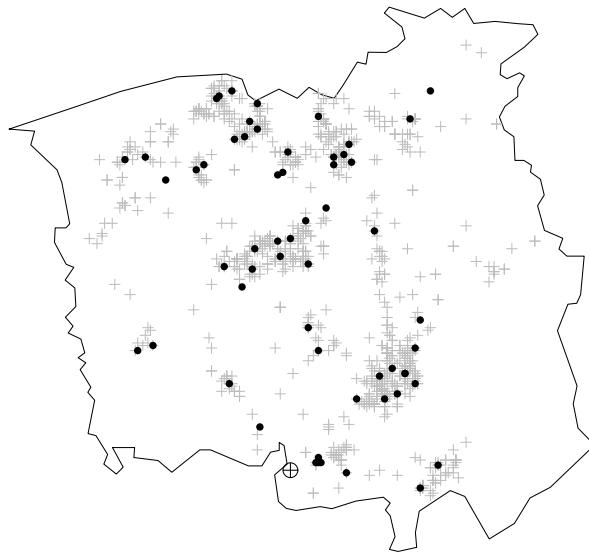


Figure 1.12: Chorley-Ribble data. Spatial locations of cases of cancer of the larynx (●) and cancer of the lung (+) and a disused industrial incinerator (⊕). Survey area about 25 kilometres across.

Figure 1.13 shows the locations of street crimes in the area of the University of Chicago over a two-week period, from a graphic displayed in the university newspaper Chicago Weekly News.

1.1.6 Replicated patterns

The waterstriders data (Figure 1.2) were obtained from photographs taken at three different times and places. They can be regarded as independent repetitions or *replicates* of the same experiment. Replication plays an important role in classical statistics, because it enables us to separate different sources of variability in the experiment. In the case of the waterstriders, replication greatly strengthens the evidence for territorial behaviour [268].

The point pattern of influenza virus proteins shown in Figure 1.6 is one of 41 point patterns

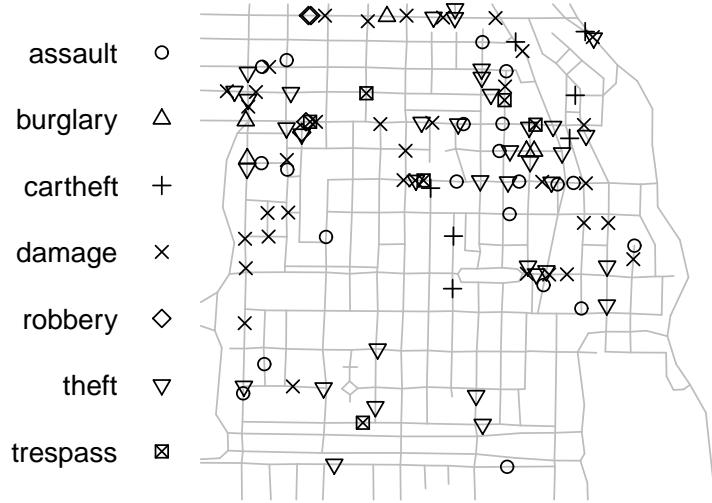


Figure 1.13: Chicago crimes data. Street crimes recorded over a two-week period in the vicinity of the University of Chicago. Survey area 1280 feet (390 metres) across. Chicago Weekly News.

obtained as the ‘responses’ in a designed experiment [80] involving two types of influenza virus, two choices of protein staining, and replication.

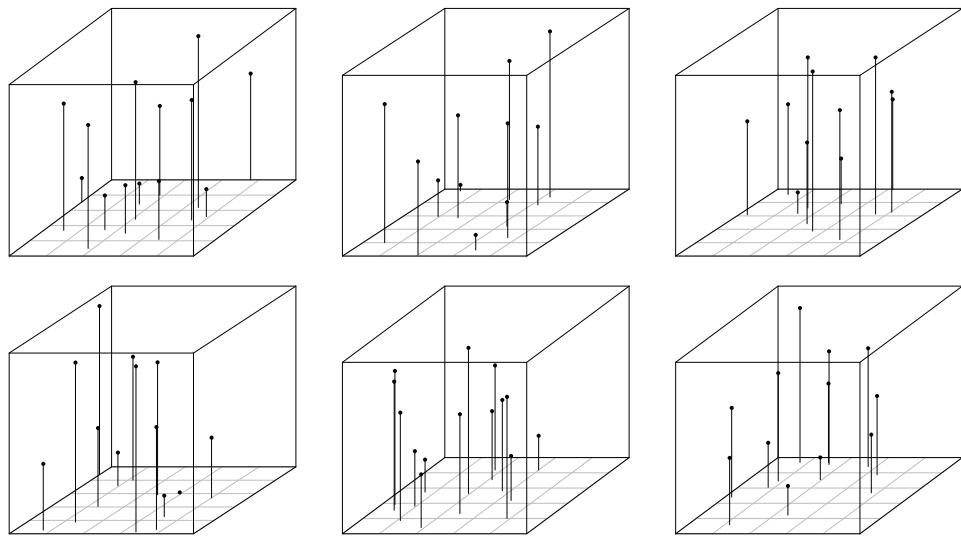


Figure 1.14: Osteocyte lacunae data (subset of full dataset).

Figure 1.14 shows replicated three-dimensional point patterns representing the positions of *osteocyte lacunae*, holes in bone which were occupied by osteocytes (bone-building cells) during life.

Observations were made in several parts of four different skulls of Macaque monkeys using a confocal optical microscope [43].

1.2 Statistical methodology for point patterns

1.2.1 Summary statistics

A time-honoured approach to spatial point pattern data is to calculate a *summary statistic* that is intended to capture an important feature of the pattern.

For the waterstriders data, the feature of interest is the spacing between the waterstrider larvae. An appropriate summary statistic is the average distance from a larva to its *nearest neighbour*, the nearest other larva in the same pattern. The average nearest-neighbour distance is a numerical measure of the typical spacing between larvae.

For the three patterns in Figure 1.2 the average nearest-neighbour distances are 55, 49 and 54 mm respectively. To interpret these values, we need a benchmark or reference value. Since our goal is to decide whether the waterstrider larvae are territorial or not, a suitable benchmark is the average nearest-neighbour distance that would be expected if the waterstriders are *not* territorial, that is, if the points were placed completely at random. This benchmark is slightly different for the three patterns in Figure 1.2, because they are slightly different in the number of points and the size of frame. A simple solution is to normalise the values, dividing the observed distance by the benchmark distance for each pattern. This ratio, called the *Clark-Evans index* [86], should be about 1 if the larvae are completely random, and greater than 1 if the larvae are territorial. The Clark-Evans index¹ values for the waterstrider patterns are 1.304, 1.126 and 1.285 respectively, suggesting that the larvae are territorial.

A summary statistic can be useful, provided it is appropriate to the application, and is defined in a simple way, so that its values can easily be interpreted. However, by reducing a spatial point pattern to a single number, we discard a lot of information. This may weaken the evidence, to the point where it is impossible to exclude other explanations. For example, the Clark-Evans index is very sensitive to spatial inhomogeneity: index values greater than 1 can also be obtained if the points are spread randomly but non-uniformly over the study region. The analysis above does not necessarily support the conclusion that the waterstrider larvae are territorial, until we eliminate the possibility that the waterstriders have a preference for one side of the pond over another.

Summary *functions* are often used instead of numerical summaries. Figure 1.15 shows the estimated *pair correlation functions* for the three waterstrider patterns. For each value of distance r , the pair correlation $g(r)$ is the observed number of pairs of points in the pattern that are about r units apart, divided by the expected number that would be obtained if the points were completely random. Pairs of waterstriders separated by a distance of 3 centimetres (say) are much less common than would be expected if their spatial arrangement was completely random.

Often the main goal is to detect and quantify trends in the density of points. The enterochromaffin-like cells in gastric mucosa (left panel of Figure 1.1) clearly become less dense as we move towards the interior of the stomach (towards the top of the picture). Figure 1.16 shows two ways of quantifying this trend. In the left panel, the micrograph has been divided into equal squares, and the number of points falling in each square has been counted and plotted. The numbers trend downwards as we move upwards. The right panel shows a kernel smoothing estimate of the spatially-varying density of points.

In order to draw a hard-and-fast conclusion from the analysis, researchers often apply a statistical

¹Using Donnelly's edge correction [134]

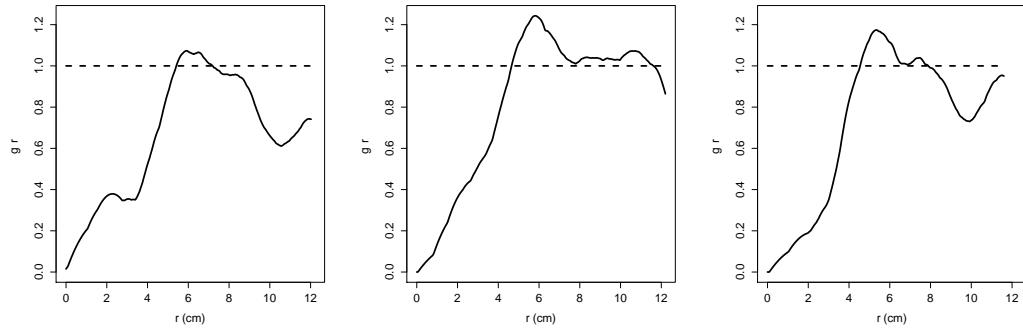


Figure 1.15: Measuring correlation between points. Estimates of the pair correlation function (solid lines) for the three waterstrider patterns. Dashed horizontal line is the expected value if the patterns are completely random.

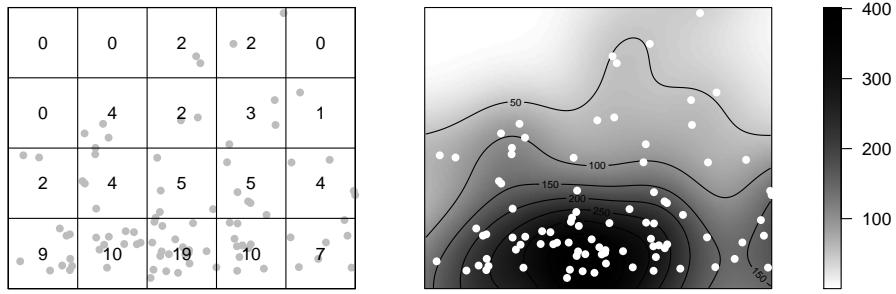


Figure 1.16: Measuring spatial trend. Enterochromaffin-like cells in gastric mucosa (Figure 1.1) showing (Left) counts of points in each square of side length 0.2 units, (Right) kernel-smoothed density of points per unit area.

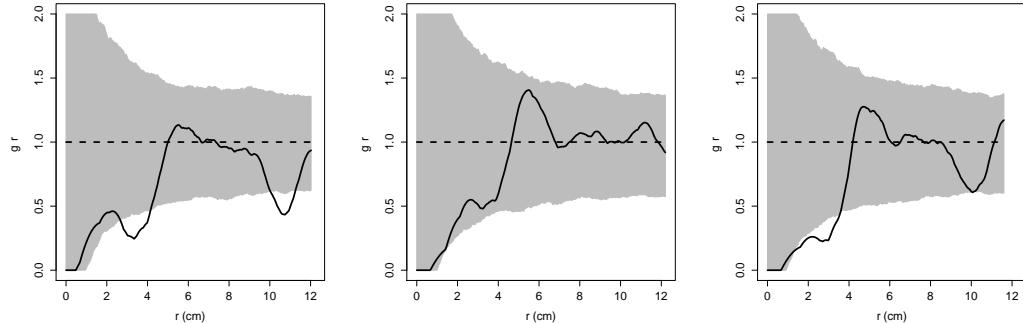


Figure 1.17: Assessing statistical significance of correlation between points. Estimates of the pair correlation function (solid lines) for the three waterstrider patterns. Dashed horizontal line is the expected value if the patterns are completely random. Grey shading shows pointwise 5% significance bands.

hypothesis test. Briefly mention null hypothesis. Figure 1.17 shows a testing technique often applied to summary functions. More explanation.

1.2.2 Statistical modelling and inference

Summary statistics work best in simple situations. In more complex situations it becomes difficult to adjust the summary statistic to ‘control’ for the effects of other variables. For example, the enterochromaffin-like cells (Figure 1.1) have spatially-varying density. The Clark-Evans index cannot be used, unless we can think of a way of taking account of this spatial trend.

Analysing data using a summary statistic or summary function is ultimately unsatisfactory. If, for example, the conclusion from analysis of the waterstriders data is that there is insufficient evidence of territorial behaviour, then we have the lingering doubt that we may have discarded precious evidence by reducing the data to a single number. On the other hand if the conclusion is that there is evidence of territorial behaviour, then we have the lingering doubt that the summary statistic may have been ‘fooled’ by some other aspect of the data, such as the non-uniform density of points.

A more defensible approach to data analysis is to build a *statistical model*. This is a complete description of the dataset, describing not only the averages, trends and relationships in the data but also the variability of the data. It contains all the information necessary to *simulate* the data, i.e. to create computer-generated random outcomes of the model that should be similar to the observed data. For example, when we draw a straight line through a cloud of data points, a *regression model* tells us not only the position of the straight line, but also the scatter of the data points around this line. Given a regression model we can generate a new cloud of points scattered about the same line in the same way.

Statistical modelling is the best way to investigate relationships while taking account of other relationships, and while accounting for variability. By building a statistical model which includes all the variables that influence the data, we are able to *account for* (rather than “adjust for”) the effects of extraneous variables, and draw sound conclusions about the questions of interest.

A statistical model usually involves some *parameters* which control the strength of the relationships, the scale of variability, and so on. For example, a simple linear regression model says that the response variable y is related to the explanatory variable x by $y = \alpha + \beta x + e$ where β is the slope of the line, α is the intercept, and e is a random error with standard deviation σ . The numbers α, β, σ are the parameters of the regression model.

Fitting a model to data means selecting appropriate values of the model parameters so that the model is a good description of the data. For example, fitting a linear regression model means finding the “line of best fit” (choosing the best values for the intercept α and slope β of the line) and also finding the “standard deviation of best fit”, (choosing the best value of σ to describe the scatter of data points around the line).

The best-fit estimate of the model parameters usually turns out to be a sensible summary statistic in its own right.

Statistical modelling may seem like a very complex enterprise. Our correspondents often say “*I’m not interested in modelling my data; I only want to analyse it.*” However, any kind of data analysis or data manipulation is *equivalent* to imposing assumptions. In taking the average of some numbers, we implicitly assume that the numbers all come from the same population. We certainly can’t say something is ‘statistically significant’ unless we assume a model, because the p -value is a probability according to a model.

The purpose of statistical modelling is to make these assumptions or hypotheses explicit. By doing so, we are able to determine the best and most powerful way to analyse data, we can subject the assumptions to criticism, and we are more aware of the potential pitfalls of analysis. In statistical usage, a model is always tentative; it is assumed for the sake of argument. We might even *want* a model to be wrong, that is, we might propose the model in order to refute it, by demonstrating that the data are not consistent with that model. In the famous words of George Box: “All models are wrong, but some are useful.” If you only want to do data analysis without statistical models, your results will be less informative and more vulnerable to critique.

Figure 1.18 shows the results of fitting two models to the tropical rainforest data of Figure 1.10.

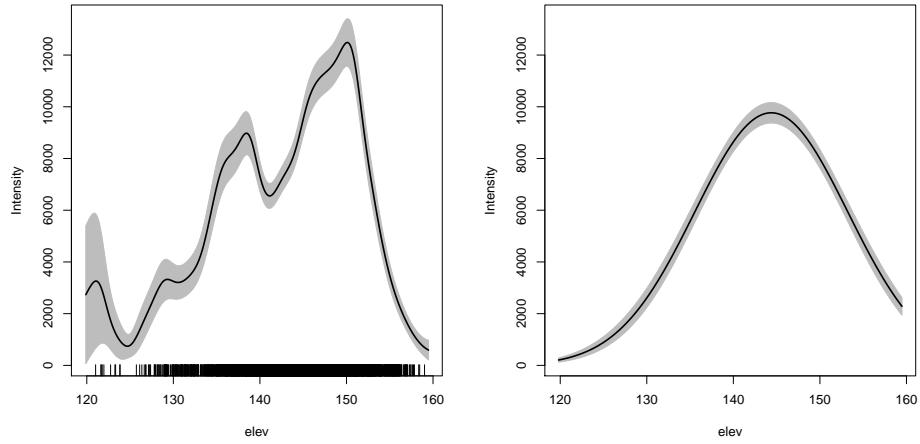


Figure 1.18: Modelling dependence of spatial trend on a covariate. *Beilschmiedia* trees data (Figure 1.10). Estimated mean density of points assuming it is a function of terrain elevation. *Left*: non-parametric estimate assuming smooth function. *Right*: parametric estimate assuming log-quadratic function. Grey shading indicates pointwise 95% confidence interval.

These models assume that the *Beilschmiedia* tree locations are random, but may have a preference for higher or lower altitudes. The left panel is based on the assumption that this habitat preference is a smoothly-varying function of terrain elevation. The right panel is based on a very specific model where the logarithm of forest density is a quadratic function of terrain elevation.

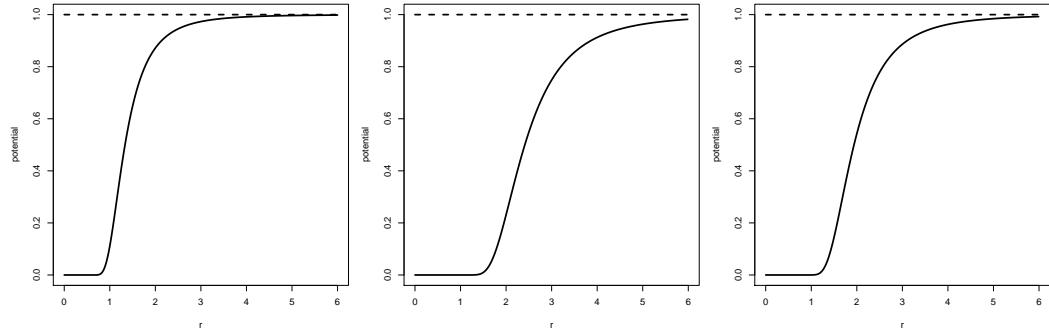


Figure 1.19: Modelling dependence between points. Fitted “soft core” interaction potentials for waterstriders.

Figure 1.19 shows one of the results of fitting a *Gibbs model* to the waterstriders data. Such models were first developed in physics to explain the behaviour of gases. The points represent gas molecules; between each pair of molecules there is a force of repulsion, depending on the distance between them. In our analysis the parameters controlling the strength and scale of the repulsion force have been estimated from data. Each panel of Figure 1.19 shows the interaction probability factor $c(r) = e^{-U(r)}$ for a pair of points separated by a distance r , where $U(r)$ is the potential (total work required to push two points together from infinite distance to distance r). A value of $c(r) \approx 0$ means it is effectively forbidden for two points to be as close as r , while a value of $c(r) \approx 1$ indicates that points separated by the distance r are “indifferent” to each other. These graphs suggest that the

waterstriders do exhibit territorial behaviour (and the model is quite suitable for this application). Figure 1.20 shows simulated realisations of the Gibbs model for the waterstriders data.

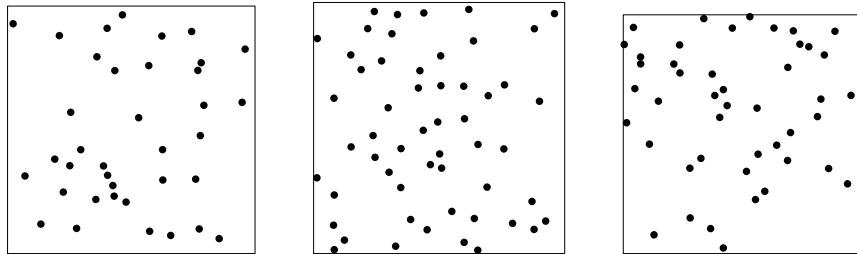


Figure 1.20: Simulated realisations of the fitted soft core model for the waterstriders.

1.2.3 Validation

The main concern with modelling is, of course, that the model could be wrong. Techniques for validating a statistical model make it possible to decide whether the fitted model is a good fit overall, to criticise each assumption of the model in turn, to understand the weaknesses of the analysis, and to detect anomalous data. Statistical modelling is a cyclic process in which tentative models are fitted to the data, subjected to criticism, and used to suggest improvements to the model.

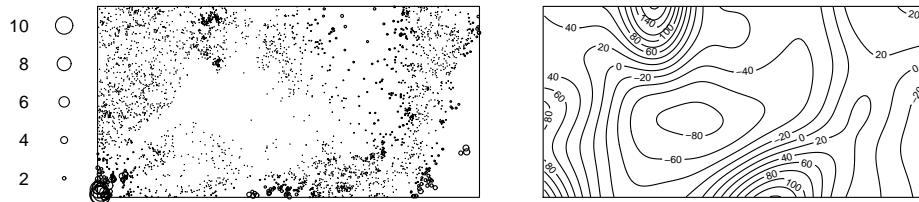


Figure 1.21: Validation of trend model for *Beilschmiedia* trees. *Left:* influence of each point ($\times 1000$). *Right:* smoothed Pearson residual field.

Figure 1.21 shows two tools for model validation, applied to the model in which the *Beilschmiedia* density is a log-quadratic function of terrain elevation. The left panel shows the *influence* of each data point, a value measuring how much the fitted model would change if the point were deleted from the dataset. Circle diameter is proportional to the influence of the point. There is a cluster of relatively large circles at the bottom left of the plot, indicating that these data points have a disproportionate effect on the fitted model. Either these data points are anomalies, or the assumed model is not appropriate for the data.

The right panel of Figure 1.21 is a contour plot of the smoothed *residuals*. Analogous to the residuals from a regression model, the residuals from a point process model are the difference between observed and expected outcomes. The residuals are zero if the model fits perfectly. Substantial deviations from zero suggest that this model does not fit well, and indicate the locations where it does not fit.

1.3 About this book

Methods

There has been a revolution in statistical methodology for spatial point patterns over the last decade. The *ad hoc* methods popular in the 1980's have been augmented by more powerful techniques, systematically organised into a new statistical methodology, that is much closer to mainstream statistical science [250, 154]. This new methodology urgently needs to be communicated to the wider scientific community.

This book describes modern statistical methodology and software for analysing spatial point patterns. It is aimed primarily at scientists, across a broad range of disciplines, who need to analyse their own point pattern data. It provides explanations and practical advice on data analysis for researchers who are results-oriented, together with guidance about the validity and applicability of different methods. Case studies in the book, and online supplementary examples and code, should make it easy for readers to begin analysing their own data.

The reader is assumed to have a general scientific training, including basic statistical methods. Using the book does not require advanced mathematical training, but mathematical formulas are supplied for completeness. The book is not biased toward any particular area of science, and aims to cover a wide range of techniques.

Our book aims to explain the core principles of statistical methodology for spatial point patterns. 'Methodology' is more than just a collection of tools: it is a systematic, principled, coherent approach to analysing data. These principles guide the appropriate choice of tool for each situation, and guide the correct interpretation of the results of analysis. By explaining the core principles we also hope to make the methodology more comprehensible.

In presenting "modern" statistical methodology we shall include many older established techniques. The *ad hoc* methods popular in the 1980's (e.g. Ripley's K -function and its simulation envelopes) still have their place, but now have to share the limelight with newer methods (e.g. parametric models, likelihood devices, estimating equations, model diagnostics) that are also much closer to mainstream statistical science. Many of these newer techniques have not yet been covered in any book.

Rather than following the traditional structure of books on point process statistics, this book is *structured around standard statistical concepts* as much as possible. This is both a pedagogical decision (to make the book more easily comprehensible to people who are trained in basic statistical concepts) and a scientific goal (to bring spatial point process analysis closer to the statistical mainstream).

As statisticians, one of our greatest concerns is that statistical methods should be applied correctly, and that the results should be interpreted correctly. This book will draw attention to common errors and misunderstandings. Examples include confusion over the statistical significance of simulation envelopes, misunderstandings of the assumptions inherent in spatial logistic regression, inappropriate use of techniques, and overemphasis on pseudo-formal methods such as hypothesis testing. We hope the book will serve as an authoritative source of information about statistical methodology for spatial point patterns.

Software

All software featured in the book is free, open-source code written in the R language, and is easily accessible online from the Comprehensive R Archive Network cran.r-project.org in the form of 'contributed packages'. The power of R makes it possible to write a new kind of statistics book: strongly governed by statistical principles, covering statistically advanced techniques, and yet focused on applications.

The main software package for the book is **spatstat**, a contributed package for R written by the authors [32, 18]. Some other R packages are also covered. To the best of our knowledge, **spatstat** is one of the largest software packages available for spatial point pattern analysis, in any language. The result of 20 years' development by experts in spatial statistics, **spatstat** now has over 1500 commands and a 1000-page online manual.

Most of this book is concerned with 2D point patterns. The **spatstat** package was originally implemented for 2D point patterns. However it is being extended progressively to 3D, space-time, and multi-dimensional space-time point patterns.

2

Software Essentials

This chapter introduces the R system (Section 2.1), additional packages for R (Section 2.2), and the `spatstat` package for analysing spatial point patterns (Sections 2.3).

2.1 Introduction to R

The analyses of spatial point pattern data that we present in this book are all conducted in the R software system for statistical computing and graphics [192].

There are myriad reasons for choosing R, as an article in the New York Times explained [323]. It is free software, easy to install and to run on virtually all platforms. Basic commands are easy to learn. R is a powerful computing system, with all the advantages of deep-level computer science, but does not require the user to know a lot of computer science. The basic “out-of-the-box” R system is already highly capable, and is easy to extend, partly thanks to its programming language, also called R. There are thousands of extension packages available, many of extremely high quality, produced by the very large developer/user community.

2.1.1 How to obtain and install R

R is free software with an open-source licence. It can be downloaded from r-project.org. For Mac™ OSX and Windows® you will most likely wish to download a “binary” version of R, pre-compiled and ready to use. The installation on your personal computer is easy and is accomplished by pointing at and clicking on the right icons. Many users of linux operating systems will probably also wish to install a pre-compiled binary version of R. What needs to be done depends on the particular flavour of Linux being run. To get more detail a web search for `install R for xxx` (where `xxx` represents your particular flavour of Linux) will probably set you on your way.

More experienced Linux users may want to download the “source” which is bundled up in a file with a name of the form `R-version.tar.gz` where `version` is the version number, something like `3.0.3`. Such source has to be compiled and installed. This is reasonably simple if you have the necessary compilers and attendant “tools” on your computer.

The **Documentation** page on the R website lists many books, manuals and online tutorials to help you learn to use R. The manual **An Introduction to R** is strongly recommended along with some of the contributed documentation found under Documentation -- Other. A useful web site for information about R is www.biostat.wisc.edu/~kbroman/Rintro.

The classic book of Venables and Ripley [325], is also valuable for understanding how to do statistical analysis in R. It is written for S-PLUS®, a relative of R, and there are some differences between these two systems.

R software is updated regularly, usually four times a year. The updates usually provided in-

creased speed, new facilities, and bug fixes. Users of R are advised to “stay current”; update your version of R frequently — whenever new releases appear or shortly thereafter. It’s worth the mild hassle involved. The `spatstat` package (on which much of this book is based) is also updated frequently, about once every two months, and again it pays to stay current.

2.1.2 Running R

Once R has been installed on a computer, the user can run it to start an *interactive session* of R in which the user will give commands and R will immediately execute them and report the results. How you start the session depends on the computer’s operating system and on your preferences. For new users of R, the simplest advice is to double-click the R icon if it is available, and otherwise (for example on a Linux system) to type the single-letter command R into a system command line (‘shell’ or ‘terminal’) interface.

Basic installations of R on Mac OSX and Windows have a simple point-and-click Graphical User Interface (GUI) which supports many tasks. Additional features are available using RStudio which is a popular “integrated development environment” (IDE). Other GUIs are available as add-ons. These include JGR, RKWard and Rcmdr. The latter is available as a contributed R package that can be obtained from CRAN. The first two can be obtained from their own web sites. The contributed package rpanel provides means by which users can build their own GUI control panels for various functions. It is aimed at “those who know R but are not familiar with the technicalities of the various gui construction systems”.

Experienced users often prefer an IDE which handles the editing, testing and running of R code seamlessly. Besides RStudio the options include, Tinn-R and the ESS (“Emacs Speaks Statistics”) module for the powerful Emacs editor. Tinn-R runs only under Windows.

R can also be run in a *non-interactive session*, when R is given a script of commands to execute, without interacting with the user. There are at least four ways of starting a non-interactive session, shown in Table 2.1. The manual **An Introduction to R** recommends the R CMD BATCH version.

<pre>R CMD BATCH filename Rscript filename R < filename R -f filename</pre>
--

Table 2.1: System commands (Windows command line or Linux shell commands) which will start a non-interactive session in R, where *filename* is the name of the text file containing Rcommand input.

An R session (interactive or non-interactive) starts in a particular folder (“directory”) in the file system, and stores temporary data in that folder. At the end of a session, there is an option to save all the data and the complete history of the session in a *workspace* in the same folder. A new R session, if started in the same folder, “re-loads” the saved workspace and can thereby access the saved data and the session history, effectively taking up where the previous session left off. Use `getwd` to find out the current working directory, and `setwd` to change it.

2.1.3 How R commands are shown in this book

In an R session, users can issue commands to R either by using a graphical point-and-click interface (if they have started their R session in such a way that such an interface is provided) or by typing commands line-by-line to a command interpreter. Typed commands are written in the R language [192]. Simple commands are very easy to learn, and users can gradually extend their skill to exploit progressively more of the powerful features of the language.

When it is waiting for a command, the R interpreter displays this prompt:

```
>
```

R can be used as a glorified calculator, and a typical series of R commands looks like this:

```
> 1+1
> 3 * (10 + 5)
> 2:5
> sqrt(2)
```

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 1+1.

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

```
> 1+1
[1] 2
> 3 * (10 + 5)
[1] 45
> 2:5
[1] 2 3 4 5
> sqrt(2)
[1] 1.414214
```

The annotation [1] indicates the first value in the output. This feature is useful when the output contains many values:

```
> (1:30)^2
[1] 1 4 9 16 25 36 49 64 81 100 121 144 169 196 225 256 289
[18] 324 361 400 441 484 529 576 625 676 729 784 841 900
```

Data can be saved and recalled by “assigning” the data to a name, using the assignment operator <-

```
> x <- 17
> x
[1] 17
```

In an interactive session, R will assume that your command is finished at the end of the line (i.e. when you press ENTER or RETURN) unless the command is obviously unfinished — for example if there is a parenthesis which has not yet been closed, or if the line ended with an arithmetic operator. In this case, R will print the *continuation character* “+” to indicate that it is expecting the previous command to be continued:

```
> folderol <- 1.2
> sin(folderol * folderol * folderol * folderol *
+     folderol * folderol * folderol * folderol *
+     * folderol )
[1] -0.09132148
```

For easier reading in the printed book, we have suppressed this behaviour by changing the continuation character to a blank space. This also makes it easier for users to copy-and-paste code directly from the e-book to the R command line.

2.1.4 Getting help about R commands

For information about a specific function in R, the best resource is the online help. R has online documentation for *every* function that is accessible to the user. The documentation specifies the arguments of the function, explains what they mean, and gives examples of their use. If you know the name of the function, then simply typing `help(functionname)` or `?functionname` will display the online help. For example, `help(glm)` or `?glm` displays the help for the function `glm()`, and `help("[")` shows the help for the subset operator.

If you are using the R command line interpreter, pressing the TAB key will show you the possible completions of a partially-finished command. Typing `gl` and then TAB will give a list of all known functions and objects that begin with `gl`. Typing `glm(` with an open parenthesis, and then TAB, will list all the arguments of the function `glm()`. Command-line completion is not available in some installations of R.

A web browser interface for the help documentation can be started by typing `help.start()`. This also gives access to online manuals, including **An Introduction to R**, and to a search engine for the help files ([Search Engine and Keywords](#)). The search engine can also be accessed directly by the R function `help.search()`.

Other resources for searching include the `RSiteSearch()` function in the base R system, and the function `findFn()` from the contributed package `sos` (see Section 2.2.2 for details on how to install and use contributed packages).

Other sources of information include online manuals, books, tutorials, and the R-help mailing list. Suggestions for books may be found on the R web site (look under Books). Under Manuals on the R web site and then contributed documentation you will find pointers to many freely available online books and tutorials. Under FAQs you can find answers to many questions about R.

It always helps to have a pretty good idea of just what you are looking for, and to have at least an inkling as to what the function you want might be called. If you are feeling lost, it is best to go to the manuals, books, and tutorials. If you go to **An Introduction to R** from the `help.start()` page, and use the web browser's Edit and then Find facility, and search for "linear models" you will be taken immediately to a section called "Linear models" which will enlighten you beyond your wildest dreams.

In order to make use of the R-help mailing list it is necessary to subscribe to this list before posting a question. See [Mailing Lists](#) on the R web site. Please read and follow the R-help posting guide, which explains what information you need to include when posting a question.

2.1.5 Vectors, lists, and indices

The power of R begins with the fact that any dataset can be stored and handled as a single entity, and referred to by a name which we choose, like `x` or `MyTax2014`.

If our data are just a sequence of numbers, they can be stored as a numeric *vector*.

```
> x <- c(3.2, 1.5, 3.6, 9.2, 2.4)
> x
[1] 3.2 1.5 3.6 9.2 2.4
> y <- 1:5
> y
[1] 1 2 3 4 5
```

Here `c` stands for “concatenate” and simply collects all the numbers together into a vector. Many other ways to input data are explained in the standard introductions to R.

Standard ‘calculator’ commands can be applied to an entire vector of numbers at once (called a *vectorised* calculation):

```
> x + 5
[1] 8.2 6.5 8.6 14.2 7.4
> x/y
[1] 3.20 0.75 1.20 2.30 0.48
```

There are also vectors of character strings, vectors of logical values, and so on:

```
> month.name
[1] "January"   "February"  "March"      "April"       "May"
[6] "June"        "July"       "August"     "September"  "October"
[11] "November"   "December"
> x > 3
[1] TRUE FALSE  TRUE  TRUE FALSE
```

To extract elements of a vector, use the subset operator `[]`. For example `x[2]` extracts the second element of `x`:

```
> x[2]
[1] 1.5
> month.name[8]
[1] "August"
```

If `x` is a vector, 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 if `x` is numeric, then `x[x > 3.1]` extracts those elements of `x` which are greater than 3.1.
- a vector of logical values, of *shorter* length than `x`. The entries of `s` will be “recycled” to obtain a logical vector of the same length as `x`, and the corresponding subset will be extracted. For example `x[c(FALSE, TRUE)]` extracts every second element of `x`.

- a vector of names: `x[c("irving", "melvin", "clyde")]` extracts those entries of `x` having the names "irving", "melvin" and "clyde" respectively, if there are any such entries, and yields a missing value NA otherwise.

It is helpful to remember that the R language does not recognise “scalars”; everything is a vector. When we extract a single element from a vector `x` by typing `x[3]`, the result is still a vector, which happens to have length 1.

In the examples above, the entries in the vector were *atomic* values — numbers, logical values, or character strings. A vector containing more complicated data types, or containing a mixture of different types of data, is called a *list*.

```
> b <- list(c("Tuesday", "Friday"), 3:7, c(FALSE,TRUE))
> b
[[1]]
[1] "Tuesday" "Friday"

[[2]]
[1] 3 4 5 6 7

[[3]]
[1] FALSE TRUE
```

The key fact is that lists *are vectors*. Consequently all of the extraction procedures described above are applicable when `x` is a list. The subset operator `[]` applied to a list will always return a list (because a subset of a list is another list). If for example `x` is a list (having length at least 6) then `x[c(2,4,6)]` is a list of length 3, i.e. having 3 components. Notice in particular that if the length of the index vector `s` is 1, then what we get from `x[s]` is a *list* of length 1. For example, `b[2]` will return the list, of length 1, whose first (and only) entry is the vector `3:7`.

If you want to get at the *object* constituting the second component of the list `b` then you need to use *double brackets*: `b[[2]]`.

```
> b[2]
[[1]]
[1] 3 4 5 6 7
> b[[2]]
[1] 3 4 5 6 7
```

A list is like a carton of eggs: the difference between `b[2]` and `b[[2]]` is the difference between cutting down the egg carton so that it only holds one egg, and extracting the egg itself.

Like any vector, a list may have a sequence of *names* to distinguish its elements. Names may be attached when the list is created by giving the elements in `name=value` form:

```
> b <- list(Day="Tuesday", SerialNumbers=3:7,
              Answers=c(FALSE,TRUE))
> names(b)
[1] "Day"           "SerialNumbers" "Answers"
> b
$Day
[1] "Tuesday"

$SerialNumbers
[1] 3 4 5 6 7
```

```
$Answers
[1] FALSE TRUE
```

Alternatively the names may be attached or changed afterward:

```
> names(b) <- c("Day", "SerialNumbers", "Answers")
```

As for any vector, the subset operator can extract elements of a list by name:

```
> b[c("Day", "Answers")]
$Day
[1] "Tuesday"
```

```
$Answers
[1] FALSE TRUE
> b[["Day"]]
[1] "Tuesday"
```

The operator \$ can also be used to extract a *single* named entry, without needing the quotation marks:

```
> b$Day
[1] "Tuesday"
```

Note that the \$ operator does “partial argument matching”, e.g. one could use b\$D to extract the first component of b. (Using b[["D"]] would return NULL.) Partial argument matching saves keystrokes, but involves risks. If the list b also had a component named D then b\$D would extract *that* component and *not* the component named Day. Unless you are very sure about the names of your list you would be well advised to type the complete name of the component that you are interested in.

2.1.6 Matrices and data frames

A matrix is a two-dimensional array of values indexed by row and column number: m[i,j] is the entry on row i and column j.

```
> m <- matrix(1:12, nrow=3, ncol=4)
> m
 [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> m[2,3]
[1] 8
```

A matrix contains atomic values (numbers, character strings, logical values) which are all of the same type.

Notice that, when the vector 1:12 was converted to the matrix m, the sequence of values was copied *column-by-column*. This is the ordering in which matrix data are stored internally in R. To copy the values in row-by-row order, set byrow=TRUE:

```
> matrix(1:12, nrow=3, ncol=4, byrow=TRUE)
```

```
[,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
```

Whole rows or columns can easily be extracted:

```
> m[2,]
[1] 2 5 8 11
> m[,3]
[1] 7 8 9
```

Note the presence of the comma. If a single subset index is used, it will be interpreted using the column-by-column ordering:

```
> m[5]
[1] 5
```

A very important data type in R is a *data frame*. This is a two-dimensional array like a matrix, except that different columns may contain different types of data.

```
> d <- data.frame(i=7:11, month=month.name[7:11],
                     has.r=((1:5) > 2))
> d
  i      month has.r
1 7      July FALSE
2 8     August FALSE
3 9 September TRUE
4 10 October TRUE
5 11 November TRUE
```

We think of the different columns as containing different variables, while the rows correspond to different experimental subjects or experimental units on which these variables have been measured. The columns have *names*, which can be assigned when the data frame is created as above, and extracted or changed using `colnames()`:

```
> colnames(d) <- c("month", "name", "has.r")
> d
  month      name has.r
1      7      July FALSE
2      8     August FALSE
3      9 September TRUE
4      10 October TRUE
5      11 November TRUE
```

A data frame can be indexed using two indices, in the same way as a matrix:

```
> d[2,3]
[1] FALSE
> d[,3]
[1] FALSE FALSE TRUE TRUE TRUE
```

However, because of the heterogeneous structure of the data, a data frame is actually stored as a *list*, in which the successive entries are successive columns of the data frame. We can also extract *columns* by using the subset operators for a list:

```
> d[[3]]
[1] FALSE FALSE TRUE TRUE TRUE
> d[c("month", "name")]
  month      name
1    7       July
2    8     August
3    9 September
4   10 October
5   11 November
> d$month
[1] 7 8 9 10 11
```

2.1.7 Objects, classes and methods in R

R is easy to use because it can accept vague instructions, like `plot(x)`, and take sensible action that is appropriate to the type of data. But how is this possible? Essentially, datasets are tagged as belonging to different *classes* of data, and there are separate *methods* for plotting each class. We need to know a bit about how this works.

In computer science jargon, R is an ‘object-oriented’ language. All individual entities in R are *objects*. A vector, matrix or data frame is an object. Data sets are objects. A data set with some kind of structure on it (e.g. a contingency table, a time series, a point pattern) is treated as a single object.

The famous `sunspots` data set, which comes with R, provides a simple example of the power of object oriented computing. This data set is a time series and essentially consists of monthly counts of sunspot from January 1749 to December 1983. However in addition to these counts the object carries with it the information that the start time is January 1749, the finish time is December 1983 and that the frequency of observation is monthly.

The information contained in `sunspots` is treated and manipulated as if it were a single thing. It can be plotted, summarised, and assigned as the value of another variable.

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

Each object in R is identified as belonging to a particular **class**. For example, the `sunspots` data set has class “`ts`” (time series), which can be discerned using the `class()` function:

```
> class(sunspots)
[1] "ts"
```

The function `plot()` is a so-called **generic** function, for which there may be different **methods** for different classes of data. The method for plotting an object of class “`ts`” is the function `plot.ts()`.

When the user types the command `plot(sunspots)`, the R system recognises that the function `plot()` is generic, inspects the object `sunspots`, identifies that it belongs to class “`ts`”, and looks for a function called `plot.ts()`. This latter function is then called to do the actual work: the system executes `plot.ts(sunspots)`. In R jargon, the call to the generic `plot()` has been “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 customize the plot for an object of any class "*foo*", consult `help(plot.foo)` rather than `help(plot)`.

What we have described here is the simplest arrangement of this kind, called “S3 method dispatch”. There are more intricate arrangements including “S4 method dispatch”, which are also supported in R, but these are not used in `spatstat`.

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 the class "ts":

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

Note that the output of `methods()` depends on what packages are currently loaded.

2.1.8 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. If the function performs a complicated analysis of a data set the returned object will typically belong to a special class. This is a convenient way to handle calculations that yield large or complicated output. The result of the calculations can then be stored in a compact manner. Other methods may then be called upon to display the result in a meaningful way or to effect further processing efficiently and with a minimum of typing.

When you type an R expression on the command line, the result of evaluating the expression is usually printed.

```
> 1+1
[1] 2
> sin(pi/3)
[1] 0.8660254
```

However, just to confuse matters, the result of a function may be tagged as ‘*invisible*’ so that it is not printed.

```
> fun <- function(){invisible(42)}
> fun()
>
```

Invisibility is lost as soon as the value is manipulated in any way, for example by assigning it to an object and then printing the object:

```
> x <- fun()
> x
[1] 42
```

or simply by enclosing it in parentheses:

```
> (fun())
[1] 42
```

You can also use the name `.Last.value` to capture the return value of the last executed command.

```
> fun()
> .Last.value
[1] 42
> x <- .Last.value
> x
[1] 42
```

Graphics functions, such as methods for `plot()`, typically return their results invisibly. Often the result is `NULL` and this is returned invisibly because it is more elegant. However, complicated graphics functions may return a result that specifies how the data were plotted: the positions of histogram bars, the numerical scale, and so on. This is plotted invisibly to avoid deluging the user with unwanted information. When we want to annotate or modify a plot, the return value becomes useful.

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

2.1.9 Factors

In statistics, a "factor" is a categorical variable, that is, a variable whose values are discrete categories (e.g. "red", "green", "blue", "orange" or "animal", "vegetable", "mineral"). The possible categories are called the *levels* of the factor.

In R factors are represented by objects of class "factor". A factor dataset `f` contains values `f[1]`, `f[2]`, ..., `f[n]` which are treated as categorical values.

```
> col <- c("red", "green", "red", "blue",
         "blue", "green", "red")
> col
[1] "red"   "green" "red"   "blue"   "blue"   "green" "red"
> f <- factor(col)
> f
[1] red   green red   blue   blue   green red
Levels: blue green red
```

Factors are superficially similar to vectors of character strings, but their conceptual nature and practical use are very different. The R system treats factors and non-factors very differently.

A factor can be recognised by the way it is printed: there are no quotes around the factor values, and the printout ends with a list of all the possible levels. Alternatively one can use `is.factor()`:

```
> is.factor(col)
[1] FALSE
> is.factor(f)
[1] TRUE
```

To display the possible levels of a factor `f`, type `levels(f)`. To change the order of the levels and the names of the levels use the arguments `levels` and `labels` in the call to `factor()`:

```
> factor(col, levels = c("red", "green", "blue"), labels = c("r", "g", "b"))
[1] r g r b b g r
Levels: r g b
```

A factor can represent a choice among different alternatives, or a division into groups, or a yes/no response. In a clinical trial of pain relief, the treatment administered to each volunteer could be either Aspirin, Paracetamol or Placebo. This information would be recorded as a factor `treat` where `treat[i]` is the treatment administered to subject `i`. The factor `treat` would serve as an explanatory variable in the data analysis. In a study of social disadvantage, each postal district could be classified as Low, Medium or High socioeconomic status. Factors commonly occur when a numeric variable is converted into a grouping variable by dividing the numerical range into a number of groups using the function `cut()`:

```
> x <- c(1,7,4,10,2,14,15,4,9,6)
> cut(x, c(0,5,10,15))
[1] (0,5]  (5,10] (0,5]  (5,10] (0,5]  (10,15] (10,15] (0,5]
[9] (5,10]  (5,10]
Levels: (0,5] (5,10] (10,15]
```

Posts to the `r-help` mailing list reveal there is a lot of confusion about factors in R, and some mistakes seem to recur. It is well worth the effort to gain a better understanding of factors, because they are so important and pervasive.

The key thing to understand is that a factor is represented in the software by *encoding* the categorical values as numbers from 1 to L , where L is the number of levels. The list of all possible levels is also stored, so that the original factor values can be reconstructed when presenting the data to the user. This trick makes computation faster, because it is easier to compare numbers than character strings, and may also save computer memory, because it takes less space to store numbers than long strings.

For example, the factor `f` described above is internally represented by the numerical codes,

```
> as.integer(f)
[1] 3 2 3 1 1 2 3
```

and the factor levels,

```
> levels(f)
[1] "blue"  "green" "red"
```

For example, the first entry in `as.integer(f)` is the number 3, meaning the third level of the factor, which is "red", so the first entry in `f` is the categorical value `red`.

Usually we do not need to think about this internal mechanism. One source of confusion arises when the levels of the factor are actually numbers stored as character strings, such as "1.02" "2.03" "3.04". If `x` is a factor having such levels then one might reasonably expect `as.numeric(x)` to be a numeric vector whose entries were the numbers resulting from converting the character strings to numeric values. Wrong-oh! In this setting `as.numeric(s)` returns the vector of integer indices used in the storage method described above. To get the vector of levels-converted-to-numbers that you really want, you can do `as.numeric(as.character(x))` or more efficiently `as.numeric(levels(x))[x]`.

```
> x <- factor(c("1.02","2.03","2.03",
+               "1.02","3.04","1.02",
+               "2.03","1.02","3.04","2.03"))
> x
[1] 1.02 2.03 2.03 1.02 3.04 1.02 2.03 1.02 3.04 2.03
Levels: 1.02 2.03 3.04
> as.numeric(x)
```

```
[1] 1 2 2 1 3 1 2 1 3 2
> as.numeric(as.character(x))
[1] 1.02 2.03 2.03 1.02 3.04 1.02 2.03 1.02 3.04 2.03
> as.numeric(levels(x))[x]
[1] 1.02 2.03 2.03 1.02 3.04 1.02 2.03 1.02 3.04 2.03
```

For more information about factors, start with the basic manual *An Introduction to R* (section 4) from the R web page www.r-project.org, under `manuals`. Other possibilities include

- http://www.ats.ucla.edu/stat/r/modules/factor_variables.htm
- *Introduction to the R Project for Statistical Computing for Use at the ITC* by David Rossiter, section 4.8, pp. 39 ff., available from the R web page; go to `manuals` and then to `contributed documentation`.
- *Statistics Using R with Biological Examples* by Kim Seefeld and Ernst Linder, pp. 20 ff, available from the R web page; go to `manuals` and then to `contributed documentation`.

2.1.10 Formulas

In statistics we often speak of a variable being “explained by” another variable. A `formula` is a special idiom in the R language which is used to express this relationship. The formula $y \sim x$ essentially means “ y depends on x ”, while $y \sim x + z$ means “ y depends on x and on z ”. A formula is recognizable by the presence of the tilde character “ \sim ”. The variable to the left of the tilde is explained by, predicted by, or dependent on, the variables appearing to the right of the tilde.

Formulas have many potential uses; the most important ones are for *graphics* and for *statistical modelling*. In graphics, the formula $y \sim x$ means that y should be plotted against x . In modelling, $y \sim x$ means that y is the response variable and x is the predictor variable. The precise interpretation depends on the context.

A `formula` can be saved as an object in its own right:

```
> a <- (y ~ x + z)
> a
y ~ x + z
> class(a)
[1] "formula"
```

This object is just a symbolic expression. The variable names x , y , z appearing in the formula are just symbols, and do not have to correspond to any real data. Importantly, the operators $+$, $-$, $*$, $/$, $^$, $:$ are also just symbols, and do not have any particular meaning at this stage.

Typically the user will give a `formula` as one of the arguments to a function that performs graphical display or fits a statistical model:

```
> plot(y ~ x)
> lm(y ~ x)
```

The function will then interpret the formula in a way that is appropriate to the context. It is only at this stage that the variable names x and y will usually be expected to correspond to datasets in the R session, and the operators $+$, $-$, $*$, $/$, $^$, $:$ will be given a particular meaning (which is not the same as their usual meaning in arithmetic).

2.1.10.1 Formulas in graphics

A formula can be used to specify which variables should be plotted against each other in a graphic. Figure 2.1 illustrates how the formula $y \sim x$ can be interpreted in different ways depending on the type of data in x . In this example we have used the standard datasets `stackloss` and `chickwts` supplied with the R system, which are automatically available in any R session. See `?stackloss` and `?chickwts` for information.

The left panel of Figure 2.1 was produced by the command

```
> plot(stack.loss ~ Water.Temp, data=stackloss)
```

in which the variables `stack.loss` and `Water.Temp` are both numeric vectors, which are columns of the data frame `stackloss`. The right panel was produced by the superficially similar command

```
> plot(weight ~ feed, data=chickwts)
```

in which `weight` and `feed` are columns of the data frame `chickwts`. The difference is that `feed` is a factor representing the type of food given to each chicken in the experiment. The sensible way to plot weight against feed type is using a boxplot for each group of observations.

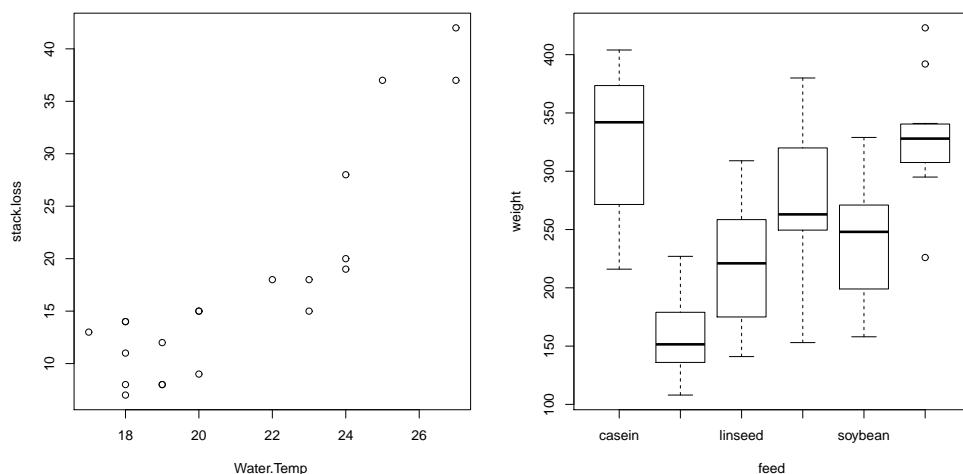


Figure 2.1: Results for plotting y against x depend on the type of data in x . *Left:* x is a numeric vector. *Right:* x is a factor. In both cases, y is numeric.

Notice the argument `data` used in these examples. This tells `plot()` that values for the variables appearing in the formula can be found in correspondingly-named columns of the data frame given by the `data` argument. This is a very handy convention which improves readability and supports well-organised code.

Apart from the generic function `plot()` there are many other graphics functions which accept formula arguments and interpret them in some appropriate way. The help files for these functions explain how the formula will be interpreted in each case. For example,

```
> plot(stack.loss ~ Water.Temp + Air.Flow, data=stackloss)
```

will produce two¹ scatterplots (`stack.loss` against `Water.Temp` and `stack.loss` against `Air.Flow`) while

¹Multiple plots are displayed fleetingly, one after the other, unless you type `par(ask=TRUE)` to pause them, or `par(mfrow=c(1,2))` to display them side-by-side.

```
> library(scatterplot3d)
> with(stackloss,
       scatterplot3d(stack.loss ~ Water.Temp + Air.Flow))
```

produces a three-dimensional scatter plot of the three variables.

2.1.10.2 Formulas in statistical models

A much more important and powerful use of formulas in R is to specify a statistical model.

In the `stackloss` data, suppose we want to fit a simple linear regression of `stack.loss` on `Water.Temp`. This can be done simply by typing

```
> lm(stack.loss ~ Water.Temp, data=stackloss)
```

Here `lm()` is the command to fit a Linear Model. The formula specifies the “systematic” component of the model, which in this case is the linear relationship $y = \alpha + \beta x$, where y is stack loss, x is water temperature, and α, β are parameters (the intercept and slope of the regression line) which are to be estimated from the data. The output from this command is:

```
Call:
lm(formula = stack.loss ~ Water.Temp, data = stackloss)

Coefficients:
(Intercept)  Water.Temp
-41.911      2.817
```

The fitted coefficients are displayed above: $\hat{\alpha} = -41.911$ and $\hat{\beta} = 2.817$.

Note that the coefficients α and β do not appear in the formula

`stack.loss ~ Water.Temp`

Only the names of *variables* appear in the formula: response variables to the left of the tilde, and explanatory variables to the right.

The conciseness of model formulas makes them extremely powerful. Any model, even a very complicated one, can be specified with surprising brevity. Expressions for simple models are very easy to learn and more complex models can be worked up to with a modicum of effort.

The idea of specifying a model by a compact formula syntax is certainly not original or exclusive to R. The particular syntax used in R has its origins in a paper by Wilkinson and Rogers [340]. However the language capabilities of R make formulas very easy to use.

There is a common syntax for interpreting formulas which is used by *almost all*² of the important model-fitting functions in R. We will briefly explain some of the basics of this syntax.

In a model formula, the symbols `+`, `-`, `*`, `/`, `^` and `:` do not have their usual meaning as arithmetic operations. Instead they are operators which combine terms in a model.

The operator “`+`” is used to add terms to a model. To fit a multivariable linear regression of `stack.loss` on `Water.Temp` and `Air.Flow`, we simply type

```
> lm(stack.loss ~ Water.Temp + Air.Flow, data=stackloss)
```

The operator “`-`” is used to remove a term from a model. For example, a model is assumed by default to include a constant term (such as the intercept α in the linear regression described above). If an intercept is not wanted, it can be explicitly removed using `-1`. To fit a proportional regression ($y = \beta x$) instead of a linear regression ($y = \alpha + \beta x$) of `stack.loss` on `Water.Temp`, we type

```
> lm(stack.loss ~ Water.Temp - 1, data=stackloss)
```

²The `nls()` function is one example where the syntax is slightly different.

One can also use `+0` instead of `-1` for the same purpose. The meaning of other model operators `*`, `/`, `^` and `:` is explained in Chapter 9.

The precise interpretation of a model formula also depends on the type of data involved. For example, with the `chickwts` dataset,

```
> lm(weight ~ feed - 1, data=chickwts)
Call:
lm(formula = weight ~ feed - 1, data = chickwts)

Coefficients:
feedcasein    feedhorsebean    feedlinseed    feedmeatmeal
      323.6          160.2          218.8          276.9
feedsoybean   feedsunflower
      246.4          328.9
```

The explanatory variable `feed` is a factor, so by fitting a linear model we are simply estimating the mean weight of chickens in each group. Further explanation is given in Chapter 9.

2.2 Packages for R

2.2.1 Packages and libraries

A **package** is a collection of additional features that can be added to R. It is usually supplied as an archive file with extension `.zip` or `.tar.gz`, compacted from several files containing code in the R language, help files, data sets, vignettes (demonstration examples with explanation) and test examples.

A standard installation of R comes with several essential default packages, such as `stats` (which as we saw above contains the critically useful function `lm()`) and `graphics`. These packages are “loaded” (see below) automatically whenever R is started. In addition a number of recommended packages (e.g. `boot`, `MASS` and `survival`) are installed automatically along with R but are *not* loaded until asked for.

Additional packages can easily be added to an existing installation of R as described below. A package only needs to be installed once (although if the version of R is upgraded, the packages may need to be re-installed). It is probably a good idea to do a complete re-install of your installed packages when you have upgraded R. A facility for doing this in an easy automated manner is provided by the `update.packages()` function.

When a package is installed, it is installed into a directory (“folder”) which is usually referred to as a “library”. That is, a library is a collection of packages. To access the features of a particular package in an R session, the package must be *loaded* in the session, by typing `library(name)` where `name` is the name of the package, with or without quotes. The appropriate metaphor is that the `library()` function “checks the requested package out of the library”. Core packages such as `stats` are automatically “checked out” or loaded at the start of an R session. However, other packages are not, because of the large overhead that would be involved.

2.2.2 Contributed packages

One of the great strengths of R is the large range of user-contributed packages which can be freely downloaded from the R archive site cran.r-project.org under ‘Contributed Packages’. At the

time of writing there are almost 6000 contributed packages, supporting everything from accelerometer data analysis to zooplankton image processing.

The technical quality of the contributed packages is very high. In particular, they have all passed a rigorous automatic checking process which verifies that the code works, the examples can be executed, the documentation is consistent with the code, and so on.

The scientific quality is also generally very high. The vast majority of contributed packages are written by specialists in the relevant field. They are all documented with online help files, most of which are well-written and which contain illuminating examples. The help files are often supported by detailed technical documentation, either installed in the package (usually in the form of a “vignette”) or published in the *Journal of Statistical Software* jstatsoft.org. (For details on how to use vignettes and other information sources see the examples for the *spatstat* package in Section 2.4.2.) There is a large community of enthusiastic users who quickly detect problems and contribute new code.

Instructions on how to install a package are given at cran.r-project.org. Following is a brief explanation.

On a Windows® or Mac™ OSX system , start an R session. In the graphical interface menu bar, pull-down the appropriate menu item for installing packages, and select the package name from the list. If this menu item is not available (typically for internet security reasons), manually download packages from cran.r-project.org under Contributed packages: visit the page for each desired package, and download the Windows or Mac™ OSX binaries. Then install these by starting an R session and using the appropriate menu item to install a local file.

On a Linux system, issue the command `install.packages("name")` in an R session where “`name`” is the name (in quotes) of the desired package. If this is the first time you install a package you may be asked to choose a directory to contain your library of R packages. If this is unsuccessful, manually download packages from cran.r-project.org under Contributed packages: visit the page for each desired package, and download the Linux tar files (ending in `.tar.gz`). Then install these by issuing the command `install.packages("packagename.tar.gz")`.

A package may depend on other packages: the dependencies are stated on the package’s download page. If you are installing a package manually, you will need to download all its dependencies manually, and install them before installing the desired package.

2.2.3 Contributed R packages for spatial point patterns

Amongst the thousands of contributed packages on CRAN, there are hundreds which analyse spatial data. A selective overview of these packages is provided in the Spatial Task View (follow the links to *Task Views — Spatial* on cran.r-project.org).

For spatial point pattern data, useful contributed packages include the following:

<code>adehabitat</code>	habitat selection analysis
<code>ads</code>	spatial point pattern analysis
<code>aspace</code>	‘centrographic’ analysis of point patterns
<code>DCluster</code>	detecting clusters in spatial count data
<code>ecespa</code>	spatial point pattern analysis
<code>MarkedPointProcess</code>	nonparametric analysis of marked spatial point processes
<code>SGCS</code>	spatial graph techniques for detecting clusters
<code>sp</code>	classes and methods for spatial data
<code>sparr</code>	analysis of spatially varying relative risk
<code>spatgraphs</code>	graphs constructed from spatial point patterns
<code>spatialkernel</code>	interpolation and segregation of point patterns
<code>spatalsegregation</code>	segregation of multitype point patterns
<code>splancs</code>	spatial and space-time point pattern analysis

There are also numerous small packages that provide a single tool for spatial point pattern analysis, such as a new statistical technique described in recent literature. The CRAN website search is useful for finding these packages using a keyword or an author's name.

The format in which point pattern data are represented will usually be different in different packages. The `sp` package offers a standard set of spatial data types in R, but many other packages (including `spatstat`) do not conform to these standards. The very useful `maptools` package can convert between different data structures and file formats. Handling data files is discussed at greater length in Chapter 3.

2.3 Introduction to `spatstat`

2.3.1 The `spatstat` package

The analysis of spatial point patterns as discussed in this book is conducted using the `spatstat` package. This is a contributed R package for analysing spatial data, with a major focus on point patterns. Most of the available functionality is for *two-dimensional* spatial point patterns, but this is now being extended to three-dimensional space, space-time, and other domains such as networks.

Both this book and the `spatstat` package are oriented toward producing *complete statistical analyses* of spatial point pattern data. Such analyses include the creation, manipulation and plotting of point patterns; exploratory data analysis (such as producing basic summaries of point patterns, kernel smoothing, the calculation and plotting of Ripley's K -function and other summary functions, and the calculation of quadrat counts and nearest neighbour distances); random generation and simulation of patterns; parametric model-fitting (Poisson models, logistic regression, Gibbs models, Cox models, cluster processes); formal statistical inference (confidence intervals, hypothesis tests, model selection); and informal diagnostics and model validation. In addition to providing the standard tools that would be expected in a package for spatial point pattern analysis, `spatstat` also contains many advanced tools, including results of recent research.

There are over 1500 user-level commands in the `spatstat` package, making it one of the largest contributed packages available for R. The package documentation is extensive: the online help manual has more than 1000 pages, and there are several supporting documents. The package also includes 50 spatial data sets (listed in Appendix ??). There is a dedicated website, www.spatstat.org for `spatstat` that offers information about the package.

Two decades of work by Adrian Baddeley and Rolf Turner, with support from the scientific community, have gone into the creation and development of `spatstat`. The package includes substantial contributions of code from Kasper Klitgaard Berthelsen, Abdollah Jalilian, Marie-Colette van Lieshout, Ege Rubak, Dominic Schuhmacher and Rasmus Waagepetersen. In particular Ege Rubak has recently been playing a significant role and has now joined the `spatstat` developer team. The user community has generously contributed data sets, bug reports and fixes, and code for new purposes.

To obtain `spatstat` simply download it from cran.r-project.org as described in subsection 2.2.2. The package is free open source software, under the GNU Public Licence (GPL) version 2 and above [160]. Several other packages (currently `deldir`, `abind`, `tensor`, `polyclip` and `goftest`) are required by `spatstat`. The packages on which `spatstat` depends also have free open source licences so that there is no barrier to downloading, copying, installing and using the package. The package is copyright, but can be reused under the conditions specified by the GPL.

2.3.2 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 this book. Citations help us to justify the expenditure of time and effort on maintaining and developing the package.

The help files for `spatstat` and the supporting documentation are *copyright*. Please do not copy substantial amounts of text from the `spatstat` help files into a scientific publication, without proper acknowledgement.

2.3.3 Overview of spatstat

As stated above, `spatstat` is designed to *support a complete statistical analysis* of spatial point pattern data sets. It includes implementations of a very wide range of standard techniques for spatial point pattern analysis, together with new techniques which are the result of original research.

data types: The `spatstat` package provides specialised data types for spatial data. They include point patterns, spatial windows, pixel images, line patterns, tessellations, and linear networks.

data: The `spatstat` package provides a large and growing collection of example spatial data sets. They arise from a range of applications including cell biology, wildlife ecology, forestry, epidemiology, geology and astronomy. These data sets have been generously contributed by the scientific community. They are used to demonstrate, test and compare the performance of different methods.

data handling: Facilities in `spatstat` for handling spatial data sets include creation of such data sets and basic data querying and modification (subsetting, splitting/merging, jittering). There are also facilities for printing and summarising objects comprising various kinds of spatial data sets. A wide variety of mathematical calculations and operations can easily be performed using the facilities of `spatstat`. Such calculations include several sorts of distance measurement. The distance transform of a spatial data set can be computed easily. Nearest neighbour calculations (which permit detecting duplicate points), calculations of area and length, and certain specialized types of numerical integration (of functions and measures) are readily available. Available geometrical operations include shifts, rotations, rescaling, affine transformation, intersection, union, and the calculation of convex hulls, morphological operations (dilation, erosion, closing, opening), and raster operations (pixel arithmetic, convolution, thresholding, pixellation). A mechanism is provided enabling the user to conduct some operations in a neighbourhood-wise manner. Tessellations such as the Dirichlet-Voronoi-Thiessen polygons and the Delaunay triangulation can be computed from point patterns.

data generation and sampling: Synthetic data can easily be generated in `spatstat`. Point patterns can be generated according to random sampling designs (independent, stratified, systematic) or more sophisticated point process models (including models fitted to data) or by spatial resampling of existing data. There is an interactive point-and-click interface enabling the user to manually create a point pattern dataset or manually digitise a scanned image. Random patterns of lines and other random sets can also be generated.

plotting: Plotting in R is made convenient through the provision of “*methods*” for the plot function which are designed to produce an appropriate graphic for objects of a given type (“class”). The `spatstat` package follows this paradigm by providing `plot()` methods for each spatial data type. Various alternative plotting procedures, adapted to different statistical purposes are provided. For instance a pixel image can be plotted as a colour image, a contour plot, a perspective view of a surface, a histogram, or a cumulative distribution function. Interactive plotting

(zoom/pan/identify) allows data to be scrutinized closely. Different spatial data sets can be superimposed as “layers” or plotted side-by-side. The `spatstat` package follows the R approach of providing sensible default behaviour while allowing detailed control over the plot. Auxiliary plotting information, for example a colour map, can be saved, examined, edited and re-used. The plot methods for summary functions (such as the K -function and its simulation envelope) are (usually) clever enough to avoid collisions between the curve and the legend, and to generate mathematical notation for the axis labels.

exploratory data analysis: The `spatstat` package supports a very wide range of techniques for exploratory data analysis and nonparametric analysis of spatial point patterns. Such techniques include classical methods such as spatial binning and quadrat counting, kernel estimation of intensity and relative risk, clustering indices (Clark-Evans index), scan statistics, Fry plots, and spatial summary functions (Ripley’s K -function, the pair correlation function, the nearest neighbour distance function, the empty space hazard rate, the J -function and the mark correlation function). The `spatstat` package includes extensions of these techniques (some of which are very recent) to inhomogeneous spatial patterns (e.g. the inhomogeneous K -function), to multi-type point patterns (e.g. the bivariate K -function, mark connection function), to point patterns with *multivariate* marks (e.g. the reverse conditional moment function [293]) as well as local indicators of spatial association (e.g. the local K -function). Standard errors and confidence intervals for summary functions are supported by spatial bootstrap methods. Finally `spatstat` includes modern exploratory and nonparametric techniques such as data sharpening [83], cluster set estimation [7], nearest-neighbour cleaning [75] and kernel estimation of relative intensity [23].

model-fitting: The main distinguishing feature of `spatstat` is its extensive capability for fitting statistical models to point pattern data, and for statistical inference about them. These models are conceptualized in terms of *point processes*, mechanisms which generate random point patterns. The `spatstat` package is able to fit Poisson, Gibbs, Cox, and Neyman-Scott models. Popular ‘alternative’ techniques such as maximum entropy and logistic regression are subsumed in the Poisson model. A very flexible syntax for specifying models is provided, using the powerful features of the R language. Terms in the model specification may represent spatial trend, the influence of spatial covariates, and interaction between points. By evaluating the statistical significance of these model terms the analyst can decide whether there is evidence for spatial trend or evidence for interpoint interaction (after allowing for covariate effects) and so on. The result of fitting a point process model to data is a fitted model object: there are facilities for printing, plotting, predicting, simulating from and updating these objects.

statistical inference: For nonparametric estimation, `spatstat` computes standard errors and confidence intervals calculated using spatial bootstrap methods, and hypothesis tests based on Monte Carlo procedures and simulation envelopes. For testing whether a point process depends on a spatial covariate, `spatstat` supports nonparametric hypothesis tests (Berman, Lawson-Waller, Kolmogorov-Smirnov, Cramér-von Mises etc). In the context of fitted point process models, it is possible to estimate model parameters with standard errors, compute predictions from the model with standard errors, perform significance tests for model terms (likelihood ratio test, score test) and to perform automatic stepwise model selection using likelihood ratios or Akaike’s Information Criterion AIC. At the time of writing, `spatstat` does not yet support Bayesian inference.

simulation, and simulation-based inference Simulation-based inferential techniques play an important role in the analysis of spatial point pattern data. To facilitate such techniques `spatstat` provides a wide range of ways to generate random patterns. There are facilities for generating completely random patterns in observation windows of any shape. As previously noted, systematic and stratified sampling are available. Simulated realisations of a large number of point

process models (created via Markov chain Monte Carlo techniques) are easily produced. In particular it is possible to simulate realisations from *fitted* point process models in a seamless manner. Approximate maximum likelihood inference for point process models, using Monte Carlo simulation, is supported. Spatial dependence between points can be investigated in `spatstat` using Monte Carlo tests including tests based on simulation envelopes.

validation An important part of statistical analysis is checking the validity and assessing the goodness of fit of the models used. Techniques for doing so form a prominent part of “classical” statistical analysis, but have only recently begun to be available in the context of spatial point pattern analysis. The `spatstat` package provides graphical diagnostics for the informal checking of the validity of assumptions in statistical models. Residuals may be calculated and plotted and leverage and influence diagnostics may be calculated. Model-based compensators, residuals of summary functions, partial residuals and added-variable plots are available.

The `spatstat` package is *organised around statistical concepts and terminology*. The name of a command in `spatstat` conforms to the standard statistical nomenclature used in R wherever possible. For example, the `spatstat` command that divides the points of a point pattern into several groups of points is called `split()` because this is the name of the corresponding command in the base R system for dividing a data set of numbers into several groups. This naming convention makes it easier for R users to find and remember command names.

The use of statistical terminology may be initially confusing for scientists from other fields. Unfortunately there is no agreement between scientific fields about terminology. In the book we try to mention equivalent terms for the same concept where relevant, and they are included in the index.

2.4 Getting started with `spatstat`

2.4.1 Installing and running `spatstat`

To start using `spatstat`:

1. Install R on your computer.

Go to r-project.org and follow the installation instructions (or see Section 2.1.1).

2. Install the `spatstat` package in your R system. This of course only needs to be done once. (Well, you *will* need to re-install or update `spatstat` from time to time.)

Start R and type `install.packages("spatstat")`. If that doesn’t work, see Section 2.2.2 or go to cran.r-project.org to learn how to install Contributed Extension Packages.

3. Start R.
4. Type `library(spatstat)` to load the package.
5. Type `beginner` for information.

2.4.2 Information sources for `spatstat`

There is a Quick Reference guide for `spatstat` which summarises its capabilities and gives a comprehensive list of `spatstat` commands. To view the Quick Reference guide, load `spatstat`

and type `help(spatstat)`. Alternatively open the full `spatstat` package manual in a browser, and navigate to the first entry in the manual. You can also download a PDF file of the Quick Reference guide from the website www.spatstat.org.

For information about a particular function in `spatstat`, either follow the links in the Quick Reference guide, or type `help(functionname)`. The keyword search function `help.search()` can be narrowed to the `spatstat` package:

```
> help.search("systematic", package="spatstat")
```

will tell you the `spatstat` function to use for generating “systematic” random point patterns.

Users of `spatstat` are strongly urged to **read the online help files** for the package. Substantial effort has gone into writing clear, detailed and comprehensive explanations of important concepts. Indeed several of the `spatstat` help files have been plagiarised in scientific publications. The `spatstat` online help files also include very practical guidance, warnings, and explanations of common error messages.

For more general guidance about `spatstat`, the best source is this book, especially making use of the index.

The `spatstat` package is constantly evolving. Regular users of `spatstat` can stay up-to-date by typing `latest.news` to read about changes in the most recent version of `spatstat`, or `news(package="spatstat")` to read about changes in all previous versions. The package `news` is also available at www.spatstat.org.

Several introductory “vignettes” are installed with `spatstat`. They include a quickstart guide `Getting Started with Spatstat`, a document about `Handling shapefiles` and a document summarising recent updates. To see a list of *all* available vignettes, type (in the R command line interface) `vignette()`; the list of `spatstat` vignettes is given by `vignette(package="spatstat")`. To read a vignette, type `vignette('name')` where `name` is the name of the document without the file extension, e.g. `vignette("getstart")`. Vignettes can also be accessed using the graphical help interface invoked by `help.start()`.

Several demonstration scripts are also installed with `spatstat`. Type

```
demo(package=spatstat)
```

for a list of all available demonstrations, or `demo(spatstat)` for a general overview.

The forum `R-sig-geo` sometimes handles questions about `spatstat`. General questions about R which are not `spatstat`-specific (and not specific to some other contributed package) can be addressed to the `R-help` forum. Join these or other fora by visiting r-project.org. The `stackexchange.com` forum may be a useful source of information, particularly on issues which are of a statistical nature rather than being R related.

Please email the authors directly if you have identified a bug or problem in `spatstat`. (The authors’ addresses are given in the help files: type `?spatstat` once you have loaded `spatstat`.) You will need to *supply an example of the problem*, and the example should be *reproducible*.

2.5 Core concepts in `spatstat`

2.5.1 Classes of data in `spatstat`

For a new user, the most important thing to know about `spatstat` is how data are organised.

Each spatial dataset is represented in `spatstat` by an object that belongs to a specialised *class*. This makes it possible to accomplish complex and intricate tasks through brief and simple commands, using the object-oriented features of R explained in section 2.1.7.

Table ?? lists the important classes of data defined in `spatstat`. They are also illustrated in Figures 2.2–2.5.

point pattern: A *point pattern* dataset is represented in `spatstat` by an object of class "ppp" (planar point pattern). See Figure 2.2. In order to analyse your own point pattern data, you will need to convert the data into this format: a detailed explanation is given in Chapters 3 and 4, particularly Section 3.3.

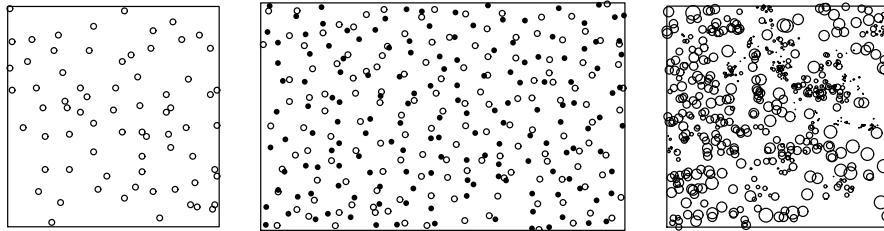


Figure 2.2: Three objects of class "ppp" representing point patterns. *Left*: points of a single type; *Middle*: points of several types; *Right*: points with a continuous numerical mark value.

window: A *window* is a region of space, for example, the spatial region in which a point pattern is recorded, or inside which the points are constrained to lie. A window may be rectangular, polygonal, or represented by true/false values on a pixel grid. See Figure 2.3. Information on how to convert your data into an object of class "owin" is given in Section 3.6.

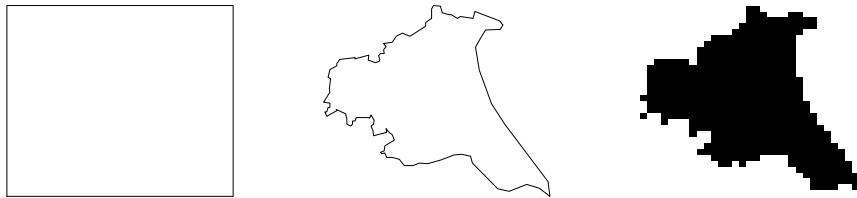


Figure 2.3: Objects of class "owin" representing spatial windows. *Left*: rectangle; *Middle*: polygonal region; *Right*: binary pixel mask.

pixel image: A *pixel image* is an evenly-spaced rectangular grid of spatial locations with values attached to them. It is represented in `spatstat` by an object of class "im". The pixel values may be numeric (real or integer), logical, complex, character, or factor values. See Figure 2.4. Information on how to convert your data into an object of class "im" is given in Section 3.5.

pattern of line segments: A spatial pattern of line segments is represented by an object of class "psp" (planar segment pattern). See the left panel of Figure 2.5. Such objects can be used to represent a pattern of geological faults, roads and so on.

tessellation: A *tessellation* is a division of space into non-overlapping domains. It is represented by an object of class "tess". See the right panel of Figure 2.5. Tessellations are discussed in Chapter 4.

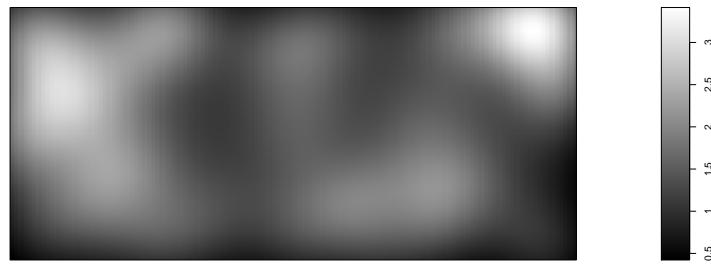


Figure 2.4: Object of class "im" representing a pixel image.

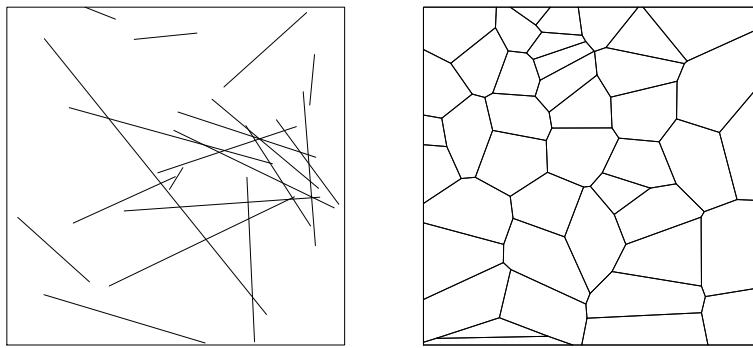


Figure 2.5: *Left:* line segment pattern (object of class "psp"). *Right:* tessellation (object of class "tess").

three-dimensional point pattern: A *three-dimensional point pattern* is represented as the class "pp3" in spatstat.

multidimensional space-time point pattern: A *multidimensional space-time point pattern* is represented as the class "ppx" in spatstat.

arrays of highly-structured data: A so-called hyperframe is used to represent arrays of highly-structured data and has the class "hyperframe" in spatstat.

Some special classes of objects are returned from spatstat functions. Common ones are listed in Table 2.2.

2.6 FAQ's

- What are the connections and differences between *spatstat* and other packages?

See the Spatial Task View on CRAN for an overview of packages for analysing spatial data. Some packages listed there are intended to provide basic data analysis capabilities, file conversion etc.

The *spatstat* package is focused on higher-level statistical analysis of spatial data.

<code>"ppm"</code>	fitted point process model (Poisson or Gibbs type)
<code>"kppm"</code>	fitted point process model (Cox or cluster type)
<code>"fv"</code>	function values (of summary function)
<code>"envelope"</code>	envelope of summary functions
<code>"listof"</code>	list of objects of the same class

Table 2.2: Common classes of objects returned from `spatstat` functions.

It is based on statistical principles, and follows the standard practice and terminology of spatial statistics. It is intended to contain tools for performing all the standard, classical analyses of spatial point pattern data (such as the K -function) as well as a wide selection of modern methods (e.g. residual diagnostics).

The `spatstat` package is not derived from, or dependent on, or superseded by, any other package in spatial statistics. Several packages depend on `spatstat`; see the Reverse Depends: and Reverse Suggests: lists on the CRAN page for `spatstat`.

- *I want to test whether the point pattern is random. Can `spatstat` do that?*

Yes, and much more. The `spatstat` package provides facilities for formal inference (such as hypothesis tests) and informal inference (such as residual plots).

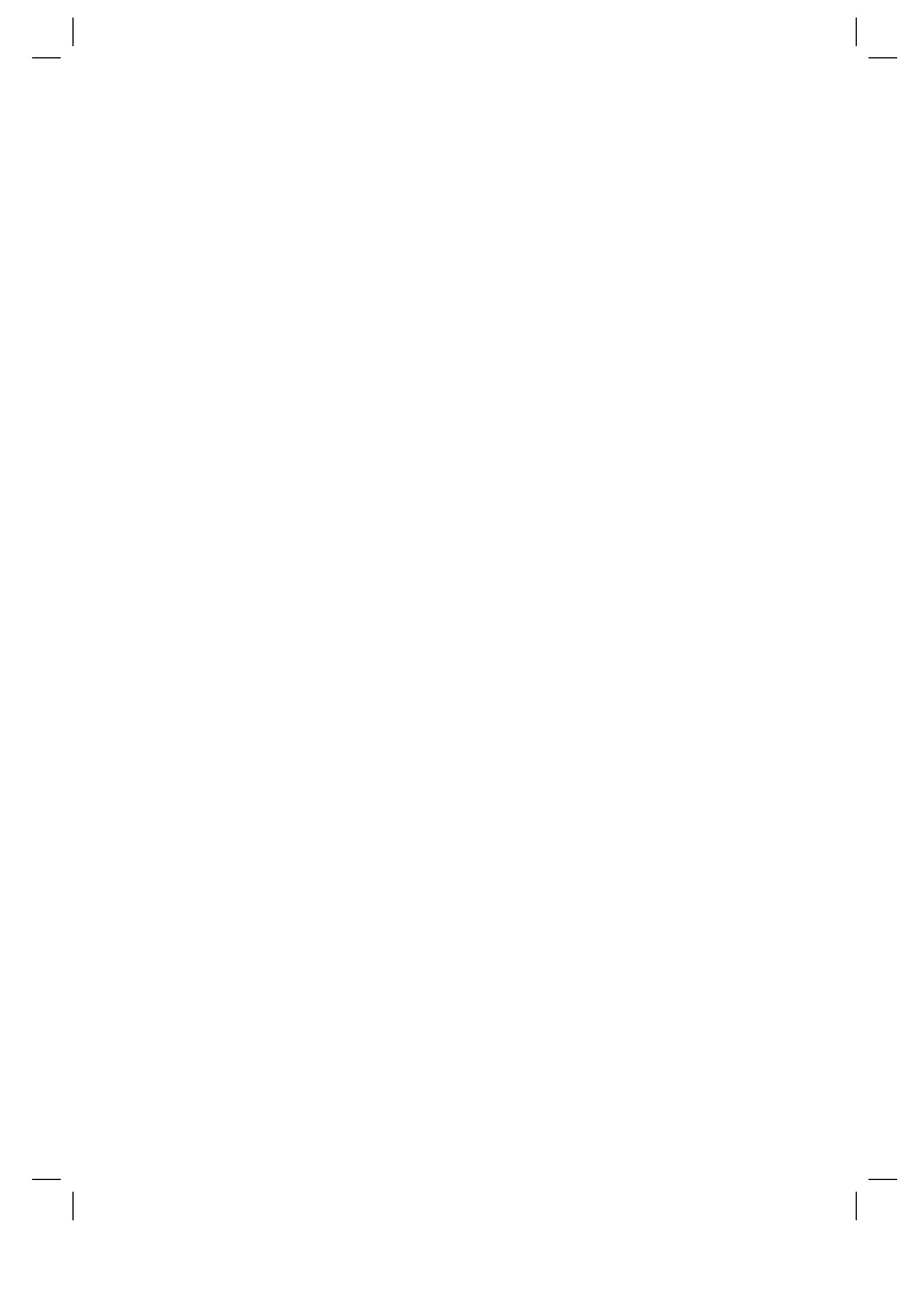
If you want to formally test the hypothesis of Complete Spatial Randomness you can do this using the χ^2 test based on quadrat counts (`quadrat.test()`), the Kolmogorov-Smirnov test based on values of a covariate (`cdf.test()`), graphical Monte Carlo tests based on simulation envelopes of the K function (`envelope()`), non-graphical tests like the Diggle-Cressie-Loosmore-Ford test (`dclf.test()`) or the maximum absolute deviation test (`mad.test()`), or the likelihood ratio test for parametric models (`anova.ppm()`).

If you want to formally test the goodness-of-fit of another point process model, you can again use the χ^2 test (for Poisson models only), or graphical or non-graphical Monte Carlo tests (for Poisson, Gibbs, Cox or cluster process models).

Informal validation of a point process model is equally important. By inspecting various kinds of diagnostic plots, we can assess whether the assumptions underlying our analysis appear to be valid. Tools include the point process residual plots (`diagnose.ppm()`), nonparametric regression (`rhohat()`), partial residual plots (`parres()`) and added variable plots (`addvar()`).

- *What is the practical limit on the number of points in a point pattern?*

This depends on the memory available to R, and on the version of `spatstat`, because the code is continually improving. As a rough guide, on a typical laptop with 2Gb of RAM, it should currently be possible to perform exploratory analysis (such as the K -function) for 1 million points, and model-fitting for about 300,000 points.



3

Collecting and Handling Point Pattern Data

This Chapter provides guidance on collecting spatial data in surveys and experiments (Section 3.1), wrangling the data into files and reading it into R (Sections 3.2 and 3.9), and handling the data in `spatstat` as a point pattern (Section 3.3), a pixel image (Section 3.5) or a spatial window (Section 3.6).

3.1 Surveys and experiments

Every field of research has its own specialised methods for collecting data in surveys, observational studies and experiments. We do not presume to tell researchers how to run their own studies. However, many problems in data analysis occur because of flaws in the initial collection of data. It is all too easy to run an experiment and then to find in retrospect that the data do not contain conclusive information about the research question. In this section we discuss some important aspects of data collection, draw attention to common flaws, and suggest ways of ensuring that the collected data can serve their intended purpose.

3.1.1 Designing an experiment or survey

The most important advice about designing an experiment is to *plan the entire experiment, including the data analysis*. One should think about the entire process that leads from the real things of interest (e.g. trees in a forest with no observers) to the points on the page which represent them, to how you are going to analyse this data, and finally to how this will answer your research question. This exercise helps to clarify what needs to be done and what needs to be recorded in order to reach the scientific goal.

A *pilot experiment* is useful. It provides an opportunity to check and refine the experimental technique, to develop protocols for the experiment, and to find unexpected problems. The data for the pilot experiment should undergo a *pilot data analysis* which checks that the experiment is capable of answering the research question, and provides estimates of variability, enabling optimal choice of sample size.

After running the pilot study, sampling protocols for a study should be specified clearly and precisely so that the data are collected in a consistent manner. For example there should be clear rules for how to treat observed objects (e.g. plants) lying on the border of a quadrat.

Consideration of the entire process, leading from the real world to a pattern of dots on a page, also helps to identify *sources of bias* such as sampling bias and selection effects. In a galaxy survey in radioastronomy, the probability of observing a galaxy depends on its apparent magnitude at radio wavelengths, which in turn depends on its distance and absolute magnitude. Bias of this known type can be handled during the analysis. In geological exploration surveys, uneven survey coverage or

survey effort introduces a sampling bias which cannot be handled during analysis unless we have information about the survey effort.

Ideally, a survey should be designed so that the surveyed items can be *revisited* to cross-check the data or to collect additional data. For example, GPS locations or photographic evidence could be recorded.

3.1.2 What needs to be recorded

In addition to the coordinates of the points themselves, there are usually many other items of information that need to be recorded. First and foremost among these is the *observation window*, that is, the region in which the point pattern was mapped. Recording the boundaries of this window is important, since it is a crucial part of the experimental design: there is information in where the points were *not* observed, as well as the locations where they *were* observed.

Even something as simple as estimating the intensity of the point pattern (average number of points per unit area) depends on the window of observation. Attempting to infer the observation window from the data themselves (e.g. by computing the convex hull of the points) leads to an analysis which has substantially different properties than one based on the true observation window. An analogy may be drawn with the difference between sequential experiments and experiments in which the sample size is fixed *a priori*. Conclusions based upon analyses using an inferred window could be seriously misleading.

Another vital component of information to be recorded consists of *spatial covariates*. A “covariate” or explanatory variable is any quantity that may have an effect on the outcome of the experiment. A “spatial covariate function” is a spatially-varying quantity such as soil moisture content, soil acidity, terrain elevation, terrain gradient, distance to nearest road, ground cover type, region classification (urban, suburban, rural), or bedrock type. More generally, a “spatial covariate” is any kind of spatial data, recruited as covariate information. Examples include a spatial pattern of lines giving the locations of geological faults, or another spatial point pattern.

A covariate may be a quantity whose “effect” or “influence” is the primary focus of the study. For example in the Chorley-Ribble data (page 10) the key question is whether the risk of cancer of the larynx is affected by distance from the industrial incinerator. In order to detect a significant effect, we need a covariate that represents it.

A covariate may be a quantity that is not the primary focus of the study but which we need to “adjust” or “correct” for. In epidemiological studies, measures of exposure to risk are particularly important covariates. The density of the population in which “cases” are observed is clearly important as the denominator used in calculating risk. A measure of sampling or censoring probability can also be important.

Theoretically, the value of a covariate should be observable at any spatial location. In reality, however, the values may only be known at a limited number of locations. For example the values may be supplied only on a coarse grid of locations, or measured only at irregularly scattered sample locations. Some data analysis procedures can handle this situation well, while others will require us to interpolate the covariate values onto a finer grid of locations.

The minimal requirement for covariate data is that, in addition to the covariate values at all points of the point pattern, the **covariate values must be available at some ‘non-data’ or ‘background’ locations**. This is an important methodological issue. **It is not sufficient to record the covariate values at the data points alone.**

For example, the finding that 95% of Kookaburra nests are in Eucalypt trees, is useless until we know what proportion of trees *in the forest* are Eucalypts. In a geological survey, suppose we wish to identify geochemical conditions that predict the occurrence of gold. It is not enough to record the geochemistry of the rocks hosting each gold deposit; this will only determine geochemical conditions that are *consistent* with gold. To *predict* gold deposits, we need to find geochemical

conditions that are more consistent with the presence of gold than with the absence of gold, and that requires information about places where gold is absent.

There are various ways in which a covariate might be stored or presented to a `spatstat` function for analysis. Probably the most useful and effective format is a *pixel image* (section 3.5).

It is good practice to record the *time* at which each observation was made, and to look for any apparent trends over time. An unexpected trend suggests the presence of a *lurking variable* — a quantity which was not measured but which affects the outcome. For instance, experimental measurements may depend on the temperature of the apparatus. If the temperature is changing over time, then plotting the data against time would reveal an unexpected trend in the experimental results, which would help us to recognise that Temperature is a lurking variable.

3.1.3 Risks and good practices

TO BE EDITED

Some miscellaneous items of advice are as follows:

- If something *might* be relevant, record it.
- If you have recorded something don't delete it, even if it appears not to be relevant!
- **Do** record 'zeroes' — e.g. quadrats in which no points appear.
- **Don't** discard data or events. Instead annotate them to say they "should" be discarded, and indicate why.
- In the context of astronomical surveys, don't make fake star catalogues by sub-sampling the data. Instead use weights to represent their (inverse) sampling probabilities, or fit models that take account of sampling probability.
- In the context of traffic data don't delete road accident records that happened at an intersection that no longer exists.
- Record as you go. Use a mobile phone camera if nothing better is to hand.
- Keep backups of original data.
- Record data in a text file. Data that are stored in a compressed or binary format, or in a proprietary format such as a word-processing document, may become unreadable if the format is changed or if the proprietary software is updated.
- When processing, reorganizing or cleaning data, document as you go. Record the sequence of operations applied to the data, and explain variable names etc. This is easy to in R.
- Write scripts for all steps of all data analysis procedures. This makes the analysis completely reproducible and avoids having to confront the frustrating question of "Why am I not getting the same results as I previously got?"
- Specify clearly issues relating to accessibility of the data. Who has permission to access the data and how shall it be accessed?
- Where data are missing, record the "missingness". That is if no value is available for a particular observation, then the record the observation as "NA". Be sure to use proper missing value notation — do not record missing values as supposedly implausible numerical values such as "99" or "-99". Doing the latter can have disastrous results in the analysis. Remember that "missing" is a *completely different concept* from "0"!

3.2 Data handling

3.2.1 Data file formats

If you obtain data files from another source, such as another researcher's website, it is of course important to understand the format of the file or files in which the data are stored and to have access to the software needed to extract the data from the files in question. It is also important to obtain all of the available information about the protocols under which the data were gathered, the range of possible values for each variable, the precision to which the variables were recorded, whether measurements were rounded and if so how, the taxonomic system or nomenclature used and the treatment of missing values. See Chapter 4 for some advice on these matters.

If you have collected data yourself it is, as was mentioned above, good practice to save the original data in a text file, so that it is not dependent on any software. The text file should have a clearly-defined format or structure. Data in a text file can easily be read into R.

For storing the point coordinates and associated mark values (see subsection 3.3.2 for a discussion of marks) we recommend the following file formats.

table format: the data are arranged in rows and columns, with one row for each spatial point. There is a column for each of the *x* and *y* coordinates, and additional columns for mark variables. See Figure 3.1. The first line may be a *header* giving the names of the column variables.

Character strings must be enclosed in quotes unless they consist entirely of letters. Missing values should be entered as NA. The usual file name extension is .txt or .tab (the latter is understood by R to indicate that the file is in table format).

comma-separated values (csv): Most computer software for data handling, including spreadsheet software, allows data to be exported to or imported from a comma-separated values file (extension .csv). This format is slightly more compressed than table format. In it data values are separated by a comma (or other chosen character) rather than by white space. This format has much to recommend it inasmuch as it is accommodated by most software thus making data sharing and transfer very easy. It is often preferred for larger data sets. However errors are more difficult to detect visually than they are in table format.

shapefiles: A shapefile is a popular file format for sharing vector data between geographic information system (GIS) software packages. It was developed and is maintained by the commercial software provider ESRI™. The specification is publicly accessible [142]. Storing a data set in a shapefile will result in a handful of files, with different extensions. These will include at least .shp, .shx and .dbf. The different extensions refer to different information types, e.g. the coordinates and the geographical projection that was used. Reading data from shapefiles is described in section 3.9.

You will also need to store auxiliary data such as the coordinates of the (corners of the) window boundary, covariate data such as a terrain elevation map, and metadata such as ownership, physical scale and technical references. The window boundary and covariate data should also be stored in text files with well-defined formats: we discuss this in section 3.6. Metadata can usually be typed into a plain text file with free format.

3.2.2 Reading data into R

Data in a text file in table format can be read into R using the command `read.table()`. A comma-separated values file can be read into R using `read.csv()`. Set the argument `header=TRUE` if the file has a header line (i.e. if the columns of the file have names).

Easting	Northing	Diameter	Species
176.111	32.105	10.4	"E. regnans"
175.989	31.979	7.6	"E. camaldulensis"
....	

Figure 3.1: Example of text file in table format.

```
> chordata <- read.table("chordatafile.txt", header=TRUE)
> chordata <- read.csv("chordatafile.csv")
```

The resulting object `chordata` is a “data frame” in R. You may need to set various options to get the desired result: type `help(read.csv)` or `help(read.table)` for information.

Check that `chordata` contains the data you expect. Use `colnames(chordata)` to see the names of the columns in the data frame: these may have changed if the original column names contained strange characters. Note that if the file from which the data were read had no names, the columns of the data frame will have the default names `V1`, `V2`, Use `head(chordata)` to see the first few rows of data, and `summary(chordata)` to see a summary of the values in each column of the data frame. See Section 2.1.6 for more on data frames and how to extract single columns etc.

It is important to check that each column of data has the intended class. Note that a column of character strings in the text file will be converted to a factor (categorical variable) by default (unless certain measures — involving "options" — have been taken). Conversion to a factor would probably be appropriate for the `Species` column in Figure 3.1. However, character strings could also represent date-and-time values, or text annotations. In this case `read.table()` or `read.csv()` should be called with `stringsAsFactors=FALSE` to prevent automatic conversion to factors; then each column should be converted to the desired type. Factors are created using `factor()` or `as.factor()`. For more details on factors see Section 2.1.9. Strings representing date-time values are converted using `as.Date()` or `as.POSIXct()`. For more details on handling dates in R see the help of `ISOdate()` and `ISOdatetime()`, or the online resources

- Date-Time Classes by Brian D. Ripley and Kurt Hornik, *R News* Vol. 1/2 June 2001, pp. 8 – 10.
- http://en.wikibooks.org/wiki/R_Programming/Times_and_Dates
- <http://www.stat.berkeley.edu/classes/s133/dates.html>
- <http://www.statmethods.net/input/dates.html>

Note that if a column which is *supposed* to contain numerical data is “corrupted” with alphabetic characters (including punctuation marks) — possibly as a result of typographical errors — then this column will be read in as *character* data (and then by default converted to a factor). Checking on the class of each column serves to detect when such errors have occurred. (The file being read in should then be edited, the errors corrected, and the reading-in of the file repeated.)

A quick and easy way to find out the class of all the columns of your data frame is to use `sapply()`:

```
> sapply(chordata, class)
```

3.3 Entering point pattern data into spatstat

A spatial point pattern in two-dimensional space is stored in `spatstat` as an object of class "ppp" (for 'planar point pattern'). In order to use the capabilities of `spatstat`, a spatial point pattern data set should be converted into an object of this class. This section describes some basic ways to do this. If your data is already in a recognised GIS format consult Section 3.9 for details on the conversion. If you have data in a raw graphical format (such as a digital image) and want to enter the points by pointing and clicking on a graphical device consult Section ??.

3.3.1 Creating a "ppp" object

To create an object of class "ppp" from raw data, use the function `ppp()`. Suppose that the x, y coordinates of the points of the pattern are contained in vectors `x` and `y` (which must of course be of equal length). Then

```
X <- ppp(x, y, other.arguments)
```

will create the point pattern object `X`. The `other.arguments` must determine a window for the pattern. Table 3.1 shows the different options for specifying a window.

If the observation window is a rectangle, it is sufficient to specify the ranges of the x and y coordinates:

```
> chordata <- read.table("chordatafile.txt", header=TRUE)
> east <- chordata$Easting
> north <- chordata$Northing
> X <- ppp(east, north, c(174,178), c(29,33))
```

or more elegantly

```
> X <- with(chordata, ppp(Easting, Northing,
  c(174,178), c(29,33)))
```

<code>ppp(x, y, xrange, yrange)</code>	point pattern in rectangle
<code>ppp(x, y, poly=p)</code>	point pattern in polygonal window
<code>ppp(x, y, mask=m)</code>	point pattern in binary mask window
<code>ppp(x, y, window=w)</code>	point pattern in specified window

Table 3.1: Basic options for creating a point pattern using the creator function `ppp()`.

If the argument `window` is given, then it must be a window object (of class "owin") specifying the window for the point pattern. Otherwise, the additional arguments are passed to the function `owin()` to create a window object. Section 3.6 gives a detailed explanation of these arguments.

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 [285] estimator of the window, given only the point locations. As was remarked in subsection 3.1.2 resorting to this expedient is to be avoided if at all possible since the resulting analysis could be misleading.

3.3.2 Marks

Recall from Chapter 1 that a ‘mark’ is an additional attribute of each point in a point pattern. For example, in addition to recording the locations of trees in a forest, we could also record the species, diameter and height of each tree, a chemical analysis of the leaves of each tree, and so on.

Suppose `x` and `y` are vectors containing the coordinates of the point locations, as before and for simplicity assume that the observation window is a rectangle with extent given by `xrange` and `yrange`. If there are marks attached to the points, store the corresponding marks in a vector `m` with one entry for each point or in data frame `m` with one row for each point and one column for each mark variable. (It is also possible to use a matrix rather than a data frame to store multiple marks, but such a matrix is just converted to a data frame internally by `ppp()` and in general a data frame is preferred.) Then create the marked point pattern by

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

For example, the following code reads raw data from a text file in table format, and creates a point pattern with a column of numeric marks containing the tree diameters:

```
> chordata <- read.table('chordatafile.txt', header=TRUE)
> X <- with(chordata, ppp(Easting, Northing, c(174,178), c(29,33),
  marks=Diameter))
```

An even slicker way to do this is to convert the data frame directly into a point pattern using the conversion operator `as.ppp()`:

```
> chordata <- read.table("chordatafile.txt", header=TRUE)
> X <- as.ppp(chordata, owin(c(174,178), c(29,33)))
```

Notice this requires that the first two columns of `chordata` contain the `x` and `y` coordinates (which they do in this case).

The two steps of reading in data and creating an object of class “`ppp`” can be reduced to one step by using `scanpp()`:

```
> X <- scanpp("chordatafile.txt", owin(c(174,178), c(29,33)))
```

The handling of marks in `spatstat` depends on what type of data they are. Mark values may belong to any of the atomic data types: numeric, integer, character, logical, or complex. They may also be a `factor`, representing categorical values (see below) or a vector of calendar dates or date/time values. Note that marks of mode character are rarely used; character mark vectors should almost always be converted to factors. To check that your data has the intended type use `class(m)` if `m` is a vector and `sapply(m, class)` if `m` is a data frame.

3.3.2.1 Categorical marks

When the mark is a categorical variable, we have what is called a *multitype point pattern*. (Some authors refer to such patterns as “multivariate” patterns, but it should be noted that a different emphasis is placed on the role of the marks when the latter terminology is used.) **The mark values must be stored as a ‘factor’ in R.** The ‘types’ are the different levels of the mark variable.

Here’s an example of an installed data set with categorical marks:

```
> demopat
marked planar point pattern: 112 points
Multitype, with levels = A, B
window: polygonal boundary
enclosing rectangle: [525, 10575] x [450, 7125] furlongs
```

The output (from the `spatstat` function `print.ppp()`) indicates that this is a multitype point pattern. Here is the vector of marks:

```
> marks(demopat)
[1] A B B A B B B A A A B A A B B A A A B B B A A A B B B A A B B B B
[34] B A A B A A B B A A B B B B A B B B B B B A A A B A B A B B B B
[67] B A B B A A B B B B B A B B A A B A B B B A B A B B B B B A A B A
[100] B B B B B A A A B A B A B B A
Levels: A B
```

This output indicates that `marks(demopat)` is a factor with levels A and B (in that order). To stipulate a different ordering of the levels, do something like

```
> marks(demopat) <- factor(marks(demopat), levels=c("B", "A"))
```

Tip: Whenever you create a factor `f`, check that the factor levels are as you intended, using `levels(f)`.

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

3.3.2.2 Multivariate marks

A point pattern has multivariate marks if *several* variables are attached to each point. For example, the `finpines` point pattern is marked by tree diameter and tree height. Note that there is a possibility of terminological confusion here: A point pattern with multivariate marks is not the same as a multivariate point pattern; the latter is synonymous with a *multitype* point pattern as mentioned in Section 3.3.2.1. A pattern with multivariate marks has marks consisting of several variables which are attached to each point:

```
> finpines
marked planar point pattern: 126 points
Mark variables: diameter, height
window: rectangle = [-5, 5] x [-8, 2] metres
```

To create a point pattern with multivariate marks, the mark data should be supplied as a data frame. It is important to check that each column of data has the intended type.

3.3.3 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 `X` were recorded in metres, type

```
> unitname(X) <- "m"
```

to use the standard abbreviation or supply both a singular and plural form if the full version is desired:

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

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

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

Please be aware that the `unitname` applies only to the coordinates, and not to the marks, of a point pattern. The units in which (numeric) marks are recorded are often unrelated to the units in which the coordinates of the points are recorded.

Altering the `unitname` in an existing dataset, while possible, is usually not sensible; it simply alters the name of the unit, without changing the values of the coordinates. To convert the coordinates into a different unit of measurement (e.g. from metres to kilometres) use the command `rescale()` as described in Section 4.2.5.

If you really want to change the coordinates by a linear transformation, producing a data set that is *not equivalent to the original*, use `affine()` or `scalardilate()`.

3.4 Data errors and quirks

Experienced statisticians expect data to have problems that need fixing before a reliable analysis can be performed. Problems can arise in various ways, such as: transcription and recording errors; unclear definitions of variables or units of measurement; unexplained conventions (e.g. recording missing values as 99); errors or omissions in metadata; discretisation of data; bugs in software interfaces and file conversions; software version conflicts; failures of recording equipment; or exigencies of the experiment. Here we discuss various techniques for detecting such problems.

3.4.1 Definition of variables

For the variables recorded in a dataset, we need to know the range of possible values for each variable, the units in which the variables are recorded, and any conventions used for recording special values (such as ‘infinite’ or ‘missing’ values). An unambiguous definition of the variable is also important — for example, for angular coordinates we need to know whether the angle is measured clockwise or anti-clockwise.

If the data are obtained from another source, it is important to obtain this information, usually from supplementary files or metadata. If the data are your own, it is highly recommended to write a separate plain text file containing this information, as discussed in Chapter 3.

Units of measurement are vital. Some important scientific errors (including the loss of a \$300 million spacecraft) have occurred because the units were given incorrectly or misinterpreted. Abbreviations for units can be misinterpreted — for example the symbol “ is used to denote seconds of time, seconds of arc, and inches. In astronomy, Right Ascension is an angular coordinate like longitude, but measured in the opposite direction, and expressed in hours, minutes and seconds of elapsed *time* in a 24-hour clock.

A good way to check for misinterpretation of variables in a dataset is to plot the data (see Section 4.1). Anomalies such as periodic patterns, impossibly dense clusters and large gaps suggest misinterpretation of a variable. If possible, compare your plot with an original graphic of the data — perhaps a figure in the original publication, or an illustration on a website. Superimpose your own plot on the original figure for comparison.

3.4.2 Missing values

Some observations may be missing or unavailable. It is a very common (but *very bad*) practice to encode missing values as strange numbers like 99 or -1. Some people do not distinguish between “missing” and “zero”, and thus record missing values as 0. Errors of this latter sort can be very hard to detect, especially if there are genuine zeroes in the data.

To find out if your data have been infected by this problem the first and best option is to check the available documentation to determine how missing values were recorded.

Otherwise, there are many tricks for guessing such conventions. We recommend a histogram or a stem-and-leaf plot, generated by the R commands `hist()` and `stem()`. Look for frequently-occurring values that seem strange.

```
> x <- c(1.5, 2.1, 4.0, -999, 2.2, 3.8, 0.9, -999, 1.9)
> stem(x)
The decimal point is 3 digit(s) to the right of the |

-1 | 00
-0 |
-0 |
0  | 0000000
```

In R, the symbol `NA` represents a missing value, and the entire system is built to handle missing values. Even when reading a stream of numbers from a text file, R will recognise the string `NA` as denoting a missing value. If you know the convention for representing missing values in your data, we highly recommend that these values be rewritten as `NA` to avoid confusion.

```
> x[x == -999] <- NA
> x
[1] 1.5 2.1 4.0 NA 2.2 3.8 0.9 NA 1.9
```

3.4.3 Data entry checking

There is a number of basic checking techniques that can be applied as a first step in analysing data. In looking for transcription errors (which will sneak in even when the greatest of diligence has been applied) it is often fruitful to plot exploratory graphs, particularly histograms and boxplots, of the data. Transcription errors tend to induce outliers which will be revealed by these graphical methods.

One very basic and easy step in checking over a point pattern for data problems is to print out the coordinate values and marks using `as.data.frame(X)`. You can use `head(as.data.frame(X))` to print only the top few lines. If the number of points is large you might also want to use `page(as.data.frame(X),method="print")` to print the data out on the screen in convenient increments. Visually scanning the data in this way can often reveal obvious errors in data entry.

Another easy (and essential) step is to plot your point pattern (see Section 4.1.2 for details on plotting); it is always wise to do so. Look for unexpected “structure” in the points and for points that lie outside the window. (If points lie outside the window then there is either something wrong with the window or something wrong with the points, or both!) If you have built your point pattern in the recommended manner, points that lie outside the window will already been quarantined for you by `ppp()`. These points are attached, as an “attribute” called `rejects` to the point pattern object that is created. The rejects are plotted (with a warning) along with the legitimate points when point pattern in question is plotted. You can get rid of the rejects via `Y <- as.ppp(X)` (where `X` is the pattern having some points outside of its window). However you are advised not to get rid of them until you understand the reason for their presence in the first place.

If you have concerns or suspicions about an individual point of the pattern you can, after plotting the pattern, identify that point by typing `identify(X)` (see Section 4.1.4) and then clicking on the point in question.

The `ppp` method for the `summary()` function may reveal quirks and anomalies in the data. You may need to determine the specifics of these anomalies by visually (re-) scanning the data as described above. Simply type `summary(X)` to apply the appropriate summary method to `X`.

3.4.4 Duplication

If two entries in a dataset are identical, this may or may not be the result of an error. Duplication of entire lines of a data file may occur because of recording errors or data-entry errors, in which case the duplicated lines will usually be adjacent. Duplication of point coordinates (i.e. having two records refer to the same (x,y) location) may happen for a variety of reasons and is surprisingly common. One of the possible reasons for such duplication is rounding, as discussed in section 3.4.5, but there are others.

Duplication of points is important, because statistical methodology for spatial point processes (as used in this book) is based largely on assumption that processes are *simple*, i.e. that points of the process can never be coincident. When the data have coincident points, *some* statistical procedures designed for simple point processes will be severely affected. For example, the pair correlation function (Chapter 7) will have an infinite value at distance zero. It is strongly advisable to check for duplicated points and to come up with a sensible strategy for dealing with them if they are present.

You can check for duplication of entries in a dataset using the generic function `duplicated()`. If your data are stored as a matrix or a data frame, this will invoke the method `duplicated.data.frame()` which compares *rows* of the array. The result is a logical vector, with one entry for each row of data, that is TRUE if the current row is identical to an *earlier* row.

If `X` is a point pattern, `duplicated(X)` will invoke the method `duplicated.ppp()`. The result is a logical vector, with one entry for each point, that is TRUE if the current point is identical to an earlier point in the sequence. Note that, by default, `duplicated.ppp()` and `duplicated.data.frame()` use different rules for deciding whether values are identical. The rule for data frames is less strict, and thus more likely to declare values to be identical. See `help(duplicated.ppp)` for options to make the two methods consistent.

For a *marked* point pattern, two points are declared to be identical when their coordinates *and* their marks are identical. Two points at the same location but with different marks are not considered duplicates. To check for duplication of point coordinates only, use `duplicated(unmark(X))` or `duplicated(X, rule="unmark")`.

To discard duplicate points, type `Y <- unique(X)`. This is equivalent to `Y <- X[!duplicated(X)]` and thus retains a data point if it is not identical to any earlier points in the sequence. The function `unique()` is generic; the method for point patterns takes account of the marks of the points as well as their spatial coordinates. To ignore the marks when deciding whether points are identical, type `Y <- unique(X, rule="unmark")`. Note that if several marked points share the same spatial location, this command extracts the first of these points in the sequence.

To count the number of coincident points, use `multiplicity(X)`. This returns a vector of integers, with one entry for each point of `X`, giving the number of points that are identical to the point in question (including itself). The function `multiplicity()` is generic. The method for point patterns again takes account of the marks. To ignore marks when computing multiplicity, use `multiplicity(unmark(X))`.

A handy syntax to use when checking for duplication is `any(duplicated(X))` which will reveal if any duplication occurs. Applying `which(multiplicity(X) > 1)` will allow you to locate where the duplication has occurred and perhaps help you to determine how to account for it.

What to *do* about duplicated points is often unclear; it depends on the context and on the objectives of the analysis. An alternative to deleting duplicate points is to perturb the coordinates slightly

using `rjitter()`. Another alternative is to make the points of the pattern unique using `unique()`, and to attach the multiplicities of the points to the pattern as marks. This can be done by something like:

```
> dup <- duplicated(X)
> marks(X) <- cbind(marx=marks(X), mul=multiplicity(X))
> Y <- X[!dup]
```

Data with multiplicities require different analysis techniques, depending on the objective.

3.4.5 Rounding

It can be important to determine whether the coordinates of the point pattern data have been rounded. The summary method for "ppp" objects attempts to do this (using `rounding.ppp()`) but may not always be correct. It is a good idea to look at the data and at a plot of the data and to use your judgment in addition to using `summary()` when deciding about rounding. The presence of rounding can sometimes have an appreciable effect on the assessment of interaction between points, particularly at short distances. Rounding brings us back to the topic of *duplication*; if coordinates are rounded to, say, the nearest metre then points within a metre of each other will tend to be recorded as being the same point.

3.5 Pixel images in spatstat

Pixel images form a very important and useful class of objects in the context of the analysis of spatial data. In `spatstat` such images are stored and presented as objects of class "im". Such an object consists essentially of a rectangular grid of (usually) very small regions ("pixels") in two dimensional space and a set of scalar values associated with 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 no value is defined at that location.

A pixel image may be thought of as representing a spatial function $Z(u)$. The value of $Z(u)$ is the value associated with the pixel in which u lies (i.e. $Z(u)$ is a "step function", constant over individual pixels). This idea is useful in many different contexts. Using a pixel image to represent a function allows the function values to consist of observed or experimental data (such as a map of terrain elevation) or of computed values (such as a kernel estimate of point process intensity). Observed values may be directly obtained from a camera (for instance in the form of a satellite image).

A pixel image can be created directly from raw data in `spatstat` by the function `im()`; one form of the syntax is `A <- im(mat,xcol,yrow)`. (See `help(im)` for other forms.) Here `mat` is (or is interpreted as) a matrix whose entries are interpreted as the values associated with the appropriate pixels. It is an irritating source of confusion that the question of which pixel is associated with which matrix entry has a less than obvious answer due to conflicting conventions for Cartesian coordinates and for matrix indexing. In Cartesian coordinates the first entry increases horizontally from left to right and the second entry increases vertically upwards. In matrix indices the first entry increases vertically downwards and the second entry increases horizontally from left to right. These are just conventions, but the fact that they are inconsistent is enough to make a saint swear.

The reader may have noticed the somewhat idiosyncratic names of the last two arguments of `im()`, namely `xcol` and `yrow`. They are given these names to remind the user of the conflict in conventions. 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)**.

The value associated with the pixel whose centre is $(x[j], y[i])$ is **mat[i, j]**. Note the switch in order of **i** and **j**. This may best be summarised by a toy example as illustrated in Figure 3.2. Here the value 6 is located at the second **x**-coordinate and third **y**-coordinate in the pixel image $((x[2], y[3]))$ while it is at matrix location 3,2 (third row and second column).

3	6	9
2	5	8
1	4	7

$$\begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

Figure 3.2: A toy example of an image with 9 pixels with the values 1-9 (left) and the corresponding matrix containing the data (right).

The matrix **mat** associated with most images that are realistic and useful is rather large, usually at least 128×128 . Consequently keying in the entries of such a matrix “by hand” is not feasible. Instead **mat** might be read in from a file or produced by some systematic coding procedure. Most of the time pixel images are read in from data files which have been created by other means (e.g. from photographic images) or are produced by converting objects of other classes to objects of class “**im**” (this may in many cases be done with **as.im()**).

If the data is already in a recognised GIS format it may be possible to convert it directly to an “**im**” object using the **maptools** package as described in Section 3.9.2.

If the spatial data at hand is given as a mathematical function (i.e. described by an explicit formula) it may be converted to an image:

```
> f <- function(x,y){15*(cos(2*pi*sqrt((x-3)^2+(y-3)^2)))^2}
> A <- as.im(f, W=square(6))
```

Note the mandatory observation window argument **W**; an image is always confined to a spatial region, which in this case must be given by the user since it cannot be inferred from a general function. A word of caution is appropriate here: You may not want to convert a spatial function to an image (effectively a discretisation of the function) since **spatstat** has a separate class “**funxy**” for spatial functions as described in . The image **A** is plotted in Figure 3.3.

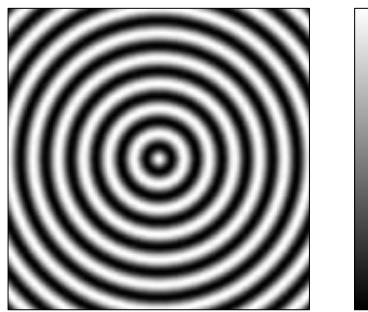


Figure 3.3: A function converted to a pixel image.

3.5.1 Factor valued images

For some strange reason, R does not allow matrices with categorical (factor) values, and many operations that create factors in R will convert a matrix to a vector. We will assume a file with values 1,2,3 representing three different values of a factor has been read into a matrix `mat` in R (here we great `mat` artificially).

```
> vec <- rep(1:3, each = 400)
> mat <- matrix(vec, nrow=40, ncol=30)
> f <- factor(mat)
> is.factor(f)
[1] TRUE
> is.matrix(f)
[1] FALSE
```

Although `mat` was a matrix, `f` is a vector, with factor values.

To create a pixel image with categorical values, leave the pixel values as a vector, and let the `im()` function reshape it:

```
> factorim <- im(f,
+                  xcol=seq(0,1,length=30),
+                  yrow=seq(0,1,length=40))
> factorim
factor-valued pixel image
factor levels:
[1] "1" "2" "3"
40 x 30 pixel array (ny, nx)
enclosing rectangle: [-0.017241, 1.0172] x [-0.012821, 1.0128] units
```

The image `factorim` is plotted in Figure 3.4.

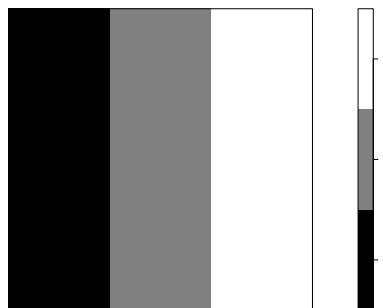


Figure 3.4: The image `factorim` created in the text.

Alternatively one can just assign a `dim` attribute to a factor. This effectively turns it into a matrix without its losing its factor characteristics:

```
> dim(f) <- c(40,30)
> factorim <- im(f)
```

In the foregoing construction the `xcol` and `yrow` vectors will be the sequences `1:30` and `1:40` respectively, so `factorim` has a completely different physical size than in the former construction.

A third alternative is to create an integer-valued matrix, and assign a `levels` attribute to it. This will be interpreted as a matrix with categorical values.

3.6 Windows in spatstat

Most commands in `spatstat` are designed to work with point pattern objects (objects of class "ppp"). Such objects necessarily involve a window and hence commands used to create them require specification of a window, where the presence (and absence) of points has been observed. Such regions are stored as objects of class "owin" ("observation window").

An "owin" object belongs to one of three types: rectangles, polygonal regions, and binary pixel masks. See Figure 3.5.

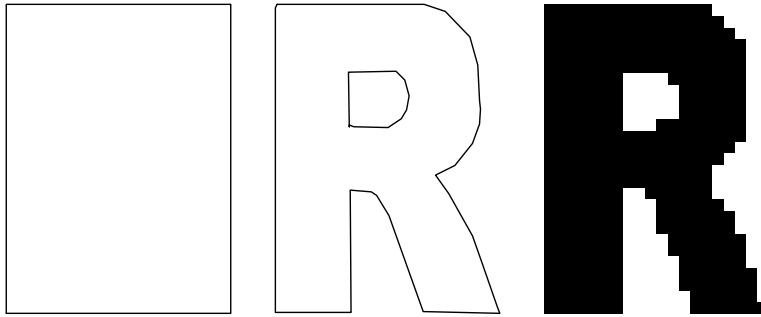


Figure 3.5: Types of windows. *Left:* rectangle; *Middle:* polygonal; *Right:* binary mask.

The details involved in creating windows of course depend on which type of window is to be created:

<code>owin(xrange, xrange)</code>	rectangle
<code>owin(poly=p)</code>	polygonal region
<code>owin(mask=m)</code>	binary pixel mask

Table 3.2: Options for creating a window using the creator function `owin()`.

3.6.1 Rectangular window

Rectangular windows are obviously the simplest to deal with. To create a rectangular window, type `owin(xrange, xrange)` 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
> square(c(1,3))
window: rectangle = [1, 3] x [1, 3] units
```

3.6.2 Polygonal window

Prior to the inception of **spatstat** almost all statistical analysis of spatial point patterns was done in rectangular windows (although non-rectangular regions were used in the GIS context much earlier). However in real life rectangular windows are the exception rather than the rule. Polygonal windows are much more realistic, and most windows can be reasonably well approximated by polygons. Polygonal windows of arbitrary shape and topology are handled by **spatstat** with consummate ease. The boundary of such a window may consist of one or more closed polygonal curves, which do not intersect themselves or each other. Moreover the window may have ‘holes’. To create a polygonal window type `owin(poly=p, xrange, yrange)` or just `owin(poly=p)`.

The argument `poly=p` indicates that the window is polygonal and its boundary is given by the data set `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 matrix or data frame with two columns, or a list with components `x` and `y`, giving the coordinates of the vertices of the window boundary, **traversed anticlockwise**. For example, the triangle in the left panel of Figure 3.6 with corners $(0,0)$, $(10,0)$ and $(0,10)$ is created by

```
> Z <- owin(poly=list(x=c(0,10,0), y=c(0,0,10)))
```

Note that the polygons specified by `p` should **not** be closed, i.e. the last vertex should **not** equal the first vertex. The same convention is used in the standard R plotting function `polygon()`. Actually, the latest and greatest **spatstat** happily copes with a polygon being closed. Note however that if you “close” the polygon you may trap yourself into thinking that there are, e.g. 8 vertices when in fact there are only 7. (The window object created would contain a list object with 7 entries for each of `x` and `y`.)

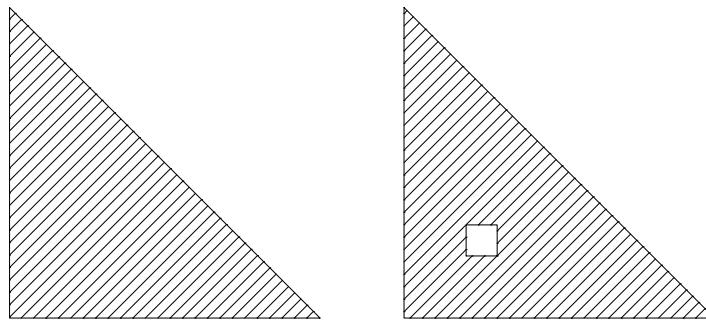


Figure 3.6: Triangles created in the text. Left: Triangle `Z`. Right: Triangle with a square hole `ZH`. Plotted with line shading (`hatch=TRUE`).

If a region boundary is a single polygon, with the vertices saved in a text file `bdry.tab` in table format, then the corresponding window can be created by

```
> bd <- read.table("bdry.tab")
> W <- owin(poly=bd)
```

If the window boundary consists of several separate polygons, then p should be a list, each of whose components p[[i]] is a matrix or data frame or 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 the triangle with a square hole displayed in the right panel of Figure 3.6.

```
> ZH <- 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))))
```

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

Often a study region is some geographical region which may have a very complicated boundary, e.g. if part of the boundary is a coastline which could be arbitrarily intricate. While such a boundary can certainly be represented as a polygon, the number of edges and vertices of that polygon might well be extremely large, making the task of keying in the coordinates of the vertices by hand completely impractical. At least sometimes the data determining the boundary will be supplied or will be available in some sort of file which can, with some effort, be read into R after which they can be passed to `owin()` in order to create a window object.

The task of reading these data from the file into R will vary in nature depending on the nature of the file (and on the nature of the window — whether it consists of discontiguous parts and whether it has an holes). It may in some instances be possible to specify (to those supplying the data) the structure of the file. If so, one fairly simple suggestion for the structure is that the file should have three columns, the first two of which consist of the x coordinates and the y coordinates of the polygon vertices. The third column would consist of the values of an indicator variable, running from 1 to n, where n is the number of individual polygons making up the window.

As was previously stated, it is a good idea to store data, including data specifying observation windows, as text files. If you are *storing* a window (that has been obtained by other means) as a text file, then we recommend that you use the structure described above.

To take a simple example, suppose that the file is a plain text file, called say `parrot.txt`. If the contents of this file are columns x and y (holding the coordinates of polygon vertices) and a column `ind` holding integer values 1, 2 and 3 that indicate with which of three possible polygons a coordinate pair is associated, then in R we can do:

```
> DF <- read.table("parrot.txt", header=TRUE)
> PY <- split(DF, f=DF$ind)
> W <- owin(poly=PY)
```

The resulting "owin" object is shown in Figure 3.7

Note that one of the polygons (say the third) constitutes a hole in the another polygon. To indicate this the coordinates of the vertices corresponding to index 3 would have been listed in *clockwise* order in `parrot.txt`. Vertices corresponding to the other two indices would have been listed in *anticlockwise* order.

It might be of interest under some circumstances to extract the individual polygons which make up a window consisting of several polygons. These individual polygons are (un-named) components of a list called `bdry` which is in turn a component of the "owin" object under consideration. (Note: It is not inconceivable that the internal structure of "owin" objects could change in the future, so the forgoing advice should perhaps be taken with a small grain of salt.)

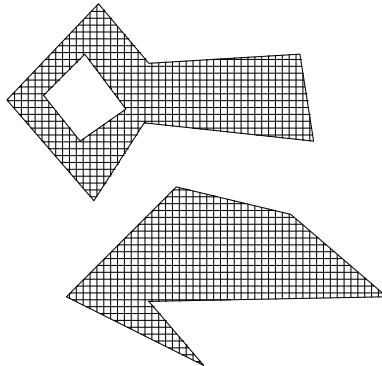


Figure 3.7: Polygonal window with vertices read in from a file.

3.6.3 Circular and elliptical windows

Circular (or disc-shaped) and elliptical windows are prototypical examples of windows which are not actually polygons but can be approximated by polygons to whatever accuracy is desired. The `spatstat` functions to effect such an approximation are `disc()` and `ellipse()`. For example, to make a circular window of radius 3 centered at the origin do:

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

As indicated above, a “circular” or “elliptical” window is actually a polygon with a (usually) large number of edges. By reducing the number of edges, in `disc()` one can create a regular polygon with any desired number of vertices. For example, to create a regular hexagon one can use `disc(npoly=6)`. To create an equilateral triangle one can use `disc(npoly=3)`.

3.6.4 Binary mask

Data which are obtained from digital recording devices are often naturally associated with windows that take the form of “binary masks”. In some GIS applications the study region of interest will be given as a binary pixel image. (Examples include objects of class `"SpatialGridDataFrame"` read in from a shapefile.) Windows in the form of binary masks also arise in terms of subsets (e.g. level sets) of pixel images. Sometimes it is inconvenient, or overly arduous, to approximate a complicated window by a polygon, in which case a fruitful approach is to use a binary mask as a computational approximation to the window. A binary mask consists of a rectangular array of pixels each with an associated logical value, either TRUE or FALSE. Pixels whose associated value is TRUE are interpreted as being part of, or inside, the window. Those whose associated value is FALSE are interpreted as being outside the window.

The corresponding object is a window object of type `mask`. Such an object is essentially a pixel image (see section 3.5) whose values are *logical* but is of class `"owin"` rather than `"im"`. The reader should be aware that there are some internal differences between the structures of objects of class `"im"` and those of class `"owin"`.

To create a window of type `mask` directly from raw data one can use the command `owin(mask=m, xrange, yrange)` where `m` is (or is interpreted as) a matrix with logical entries determining which pixels are inside the window and which are outside. Another possible syntax is `owin(mask=m, xy=xy)` where `xy` is list with components `x` and `y` giving equispaced `x` and `y` coordinates of the pixel centres. The same confusing conflict between the conventions for Cartesian coordinates and for matrix indexing arises here as it does in the context of pixel images. In detail,

the rectangle with dimensions `xrange`, `yrange` is divided into equal rectangular pixels. The matrix entry `m[i, j]` is TRUE if the point (`xx[j], yy[i]`) (sic) belongs to the window, where `xx`, `yy` are vectors of the coordinates of pixel centres, equally spaced over `xrange` and `yrange` respectively. The length of `xx` is `ncol(m)` while the length of `yy` is `nrow(m)`.

As for pixel images the matrix (here denoted by `m`) containing the pixel values is usually far too large to be keyed in by hand, and should be read in from a file which has been created by some other application. A safe strategy for handling this situation is to dump the data from the external application into a text file, and read the text file into R using `scan()`. Next reformat the scanned-in data as a matrix, and finally use `owin()` to create the window object.

When saving such a window as a text file the pixel values are best stored in so-called “column major” order, which is to say that the first column of an $m \times n$ matrix should constitute the first m entries of the text file, the second column should constitute the next m entries, and so on. Column major order is the order in which (by default) R packs values into a matrix. If `W` is a window of type `mask`, it can be stored as a text file in a manner something like `write(as.matrix(W), file="W.txt")`, which will automatically store the pixel values column by column, i.e. in column major order.

One can get the window back via

```
M <- scan("W.txt", what=logical())
W.chk <- owin(M, xrange=xr, yrange=yr)
```

where `xr` and `yr` are the `xrange` and `yrange` of the original `W`. One should then obtain TRUE from `all.equal(W, W.chk)`. When storing a mask-type window as a text file, it is probably best to store `xrange` and `yrange` as “metadata”.

Rectangles and polygonal windows can be converted to binary masks using `as.mask()`. For example the window in the right hand panel in Figure 3.5 was created by `as.mask(letterR, eps=0.1)`. See the online help for `as.mask()` for details about the `eps` argument.

3.7 Collections of objects

3.7.1 The "listof" class

A `listof` object is a list of similar objects. In `spatstat` we use a `listof` object to store several different spatial objects of the same class, for example, several point patterns, or to store the results of several different types of analysis applied to the same spatial dataset. Various functions in `spatstat` produce objects of class “`listof`”.

The class “`listof`” is actually defined in the base R system, in the `stats` package, where it is used to represent a list of alternative statistical models fitted to the same dataset. Relatively little use is made of the “`listof`” class elsewhere. The `spatstat` package has “hijacked” this class for its own purposes and has added a substantial number of methods for it, most notably a plot method.

You can make a “`listof`” object using `listof(entry1, entry2, ...)` or `as.listof(xxx)` where `xxx` is a list of objects.

```
> P <- listof(A=cells, B=japanesepines, C=redwood)
```

3.7.2 The "hyperframe" class

3.8 Interactive data entry

3.9 Reading GIS file formats

3.9.1 GIS file formats

There is a wide variety of software packages for handling spatial data, especially Geographical Information Systems (GIS). These packages use many different file formats to represent spatial data. Common formats include *shapefiles*, *NetCDF* and *GRIB*.

Typically *spatstat* does not support these formats directly: this would not be good software design. Instead, we rely on specialised R packages which exist for handling different spatial data file formats. The following packages can be useful:

<code>maptools</code>	Tools for reading and handling spatial objects
<code>shapefiles</code>	Read and write ESRI™ shapefiles
<code>RArcInfo</code>	interface to ArcInfo system and data format
<code>rgdal</code>	interface to GDAL geographical data analysis system
<code>GeoXp</code>	interactive spatial exploratory data analysis
<code>sp</code>	spatial data classes and methods

For our purposes the most useful file-handling package is *maptools*. It handles a large number of different file formats, and contains interface code for exchanging spatial objects between different R packages.

When a file is read by *maptools*, the data are represented in R using the data structures defined in the package *sp*. The *sp* package [64] supports a standard set of spatial data types in R. These standard data types can be handled by many other packages, so it is useful to convert your spatial data into one of the data types supported by *sp*.

The *maptools* package also contains code for converting *sp* data types to the data structures supported by *spatstat*. Our recommended strategy for converting spatial data from a standard GIS format into *spatstat* is therefore:

1. using the facilities of *maptools*, read the data and store the data in one of the standard formats supported by *sp*;
2. convert the *sp* data type into one of the data types supported by *spatstat* typically using *maptools*.

Using the *sp* data types as an intermediate stage is also useful if you plan to employ other R packages for spatial data analysis, which often use the *sp* data types.

3.9.2 Read shapefiles using maptools

A shapefile [142] represents a list of spatial objects — a list of points, a list of lines, or a list of polygonal regions — and each object in the list may have additional variables attached to it. A data set stored in shapefile format is actually stored in a collection of text files, for example *chordata.shp*, *chordata.prj*, *chordata.sbn*, *chordata.dbf*, which all have the same base name *chordata* but different file extensions. To refer to this collection, always use the file name with the extension *shp*.

The `maptools` package contains facilities for reading and writing files in shapefile format. The typical procedure is:

```
> library(maptools)
> x <- readShapeSpatial("chordata.shp")
> class(x)
```

The class of the resulting object `x` may be "`SpatialPoints`" indicating a point pattern, "`SpatialLines`" indicating a list of polygonal lines, or "`SpatialPolygons`" indicating a list of polygons. It may also be "`SpatialPointsDataFrame`", "`SpatialLinesDataFrame`" or "`SpatialPolygonsDataFrame`" indicating that, in addition to the spatial objects, there is a data frame of additional variables. The classes "`SpatialPixelsDataFrame`" and "`SpatialGridDataFrame`" represent pixel image data.

Here are some examples, using the example shapefiles supplied in the `maptools` package itself.

```
> library(maptools)
> oldfolder <- getwd()
> setwd(system.file("shapes", package="maptools"))
> baltim <- readShapeSpatial("baltim.shp")
> columbus <- readShapeSpatial("columbus.shp")
> fylk <- readShapeSpatial("fylk-val.shp")
> setwd(oldfolder)
```

Then `class(baltim)` returns "`SpatialPointsDataFrame`", while `class(columbus)` returns "`SpatialPolygonsDataFrame`" and `class(fylk)` returns "`SpatialLinesDataFrame`".

3.9.3 Converting sp data to spatstat format

To convert the data set to an object in the `spatstat` package, the subsequent procedure depends on the type of data, as explained below.

3.9.3.1 Objects of class "SpatialPoints"

An object `x` of class "`SpatialPoints`" represents a spatial point pattern. Use `as(x, "ppp")` or `as.ppp(x)` to convert it to a spatial point pattern in `spatstat`.

The window for the point pattern will initially be taken from the bounding box of the points. You will probably wish to change this window, usually by taking another data set to provide the window information. Use `[.ppp` to change the window: if `X` is a point pattern object of class "`ppp`" and `W` is a window object of class "`owin`", type

```
> X <- X[W]
```

3.9.3.2 Objects of class "SpatialPointsDataFrame "

An object `x` of class "`SpatialPointsDataFrame`" represents a pattern of points with additional variables ('marks') attached to each point. It includes an object of class "`SpatialPoints`" giving the point locations, and a data frame containing the additional variables attached to the points.

Use `as(x, "ppp")` or `as.ppp(x)` to convert an object `x` of class "`SpatialPointsDataFrame`" to a spatial point pattern in `spatstat`. In this conversion, the data frame of additional variables in `x` will become the `marks` of the point pattern `z`.

```
> y <- as(x, "ppp")
```

Before the conversion you can extract the data frame of auxiliary data by `df <- x@data` or `df <- slot(x, "data")`. After the conversion you can extract these data by `df <- marks(y)`.

For example:

```
> balt <- as(baltim, "ppp")
> bdata <- slot(baltim, "data")
```

3.9.3.3 Objects of class "SpatialLines"

A “line segment” is the straight line between two points in the plane. In the `spatstat` package, an object of class “`psp`” (“planar segment pattern”) represents a pattern of line segments, which may or may not be connected to each other (like matches which have fallen at random on the ground). In the `sp` package, an object of class “`SpatialLines`” represents a **list of lists of connected curves**, each curve consisting of a sequence of straight line segments that are joined together (like several pieces of a broken bicycle chain.) These two data types do not correspond exactly: see Figure 3.8.

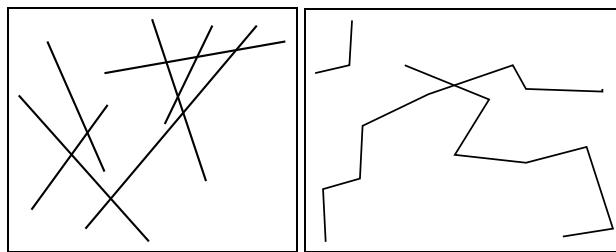


Figure 3.8: Objects of class “`psp`” (Left) and “`SpatialLines`” (Right).

The list-of-lists hierarchy in a “`SpatialLines`” object is useful when representing internal divisions in a country. For example, if `USA` is an object of class “`SpatialLines`” representing the borders of the United States of America, then `USA@lines` might be a list of length 52, with `USA@lines[[i]]` representing the borders of the *i*-th State. The borders of each State consist of several different curved lines. Thus `USA@lines[[i]]@Lines[[j]]` would represent the *j*th piece of the boundary of the *i*-th State.

If `x` is an object of class “`SpatialLines`”, there are at least two different things that you might want to do:

1. Collect together all the line segments (all the segments that make up all the connected curves) and store them as a single object of class “`psp`”.

To do this, use `as(x, "psp")` or `as.psp(x)` to convert it to a spatial line segment pattern.

2. Convert each connected curve to an object of class “`psp`”, keeping different connected curves separate.

To do this, type something like the following:

```
> out <- lapply(x@lines,
                  function(z) { lapply(z@Lines, as.psp) })
```

In either case, the window for the spatial line segment pattern can be specified as an argument `window` to the function `as.psp()`.

In the second case, the result will be a **list of lists** of objects of class “`psp`”. Each one of these

objects represents a connected curve, although the `spatstat` package does not know that. The list structure will reflect the list structure of the original "SpatialLines" object `x`. If that's not what you want, then use `curvelist <- do.call("c", out)` or

```
> curvegroup <- lapply(out,
  function(z) { do.call("superimposePSP", z)})
```

to collapse the list-of-lists-of-"`psp`"'s into a list-of-"`psp`"'s. In the first case, `curvelist[[i]]` is a "`psp`" object representing the `i`-th connected curve. In the second case, `curvegroup[[i]]` is a "`psp`" object containing all the line segments in the `i`-th group of connected curves (for example the `i`-th State in the USA example).

3.9.3.4 Objects of class "SpatialLinesDataFrame"

An object `x` of class "SpatialLinesDataFrame" is a "SpatialLines" object with additional data. The additional data is stored as a data frame `x@data` with one row for each entry in `x@lines`, that is, one row for each group of connected curves.

In the `spatstat` package, an object of class "`psp`" (representing a collection of line segments) may have a data frame of marks. Note that each *line segment* in a "`psp`" object may have different mark values.

If `x` is an object of class "SpatialLinesDataFrame", there are two things that you might want to do:

1. collect together all the line segments that make up all the connected lines, and store them as a single object of class "`psp`".

To do this, use `as(x, "psp")` or `as.psp(x)` to convert it to a marked spatial line segment pattern.

2. keep each connected curve separate, and convert each connected curve to an object of class "`psp`". To do this, type something like the following:

```
> out <- lapply(x@lines,
  function(z) { lapply(z@Lines, as.psp) })
> dat <- x@data
> for(i in seq(nrow(dat)))
  out[[i]] <- lapply(out[[i]], "marks<-",
  value=dat[i, , drop=FALSE])
```

The result is a list-of-lists-of-"`psp`"'s. See the previous subsection for explanation on how to change this using `c()` or `superimposePSP()`.

In either case, the mark variables attached to a particular *group of connected lines* in the "SpatialLinesDataFrame" object, will be duplicated and attached to each *line segment* in the resulting "`psp`" object.

3.9.3.5 Objects of class "SpatialPolygons"

First some terminology. A *polygon* is a closed curve that is composed of straight line segments. You can draw a polygon without lifting your pen from the paper. A *polygonal region* is a region in space whose boundary is composed of straight line segments. A polygonal region may consist of several unconnected pieces, and each piece may have holes. The boundary of a polygonal region consists of one or more polygons. To draw the boundary of a polygonal region, you may need to lift and drop the pen several times. See Figure 3.9.

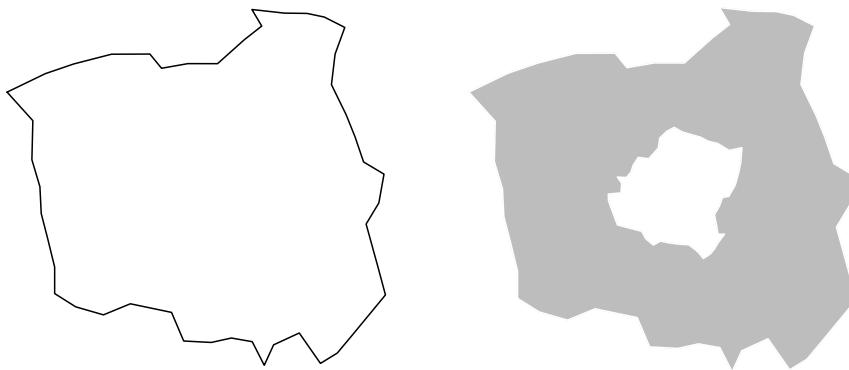


Figure 3.9: Distinction between a polygon (*Left*) and a polygonal region (*Right*).

An object of class "owin" in `spatstat`, if it is polygonal, represents a **single polygonal region**. It is a region of space that is delimited by boundaries made of lines. It may consist of several pieces.

An object `x` of class "SpatialPolygons" represents a **list of polygons**. For example, a single object of class "SpatialPolygons" could store information about every State in the United States of America (or the United States of Malaysia). Each State would be a separate polygonal region (and it might contain holes such as lakes).

There are two things that you might want to do with an object of class "SpatialPolygons":

1. combine all the polygonal regions together into a single polygonal region, and convert this to a single object of class "owin".

For example, you could combine all the States of the USA together and obtain a single object that represents the territory of the USA.

To do this, use `as(x, "owin")` or `as.owin(x)`. The result is a single window (object of class "owin") in the `spatstat` package.

2. keep the different polygonal regions separate; convert each one of the polygonal regions to an object of class "owin".

For example, you could keep the States of the USA separate, and convert each State to an object of class "owin".

To do this, type the following:

```
> regions <- slot(x, "polygons")
> regions <- lapply(regions,
+                     function(x) { SpatialPolygons(list(x)) })
> windows <- lapply(regions, as.owin)
```

The result is a list of objects of class "owin". Often it would make sense to convert this to a tessellation object, by typing

```
> te <- tess(tiles=windows)
```

During the conversion process, the geometry of the polygons will be automatically “repaired” if needed. Polygon data from shapefiles often contain geometrical inconsistencies such as self-intersecting boundaries and overlapping pieces. For example, these can arise from small errors in curve-tracing. Geometrical inconsistencies are tolerated in an object of class "SpatialPolygons" which is a list of lists of polygonal curves. However, they are not tolerated in an object of class "owin", because an "owin" must specify a well-defined region of space. These data inconsistencies must be repaired to prevent technical problems. The `spatstat` package uses polygon-clipping code to automatically convert polygonal lines into valid polygon boundaries. The repair process changes the number of vertices in each polygon, and the number of polygons (if you chose option 1). To disable the repair process, set `spatstat.options(fixpolygons=FALSE)`.

3.9.3.6 Objects of class "SpatialPolygonsDataFrame"

An object `x` of class "SpatialPolygonsDataFrame" represents a list of polygonal regions, with additional variables attached to each region. It includes an object of class "SpatialPolygons" giving the spatial regions, and a data frame containing the additional variables attached to the regions. The regions are extracted by

```
> y <- as(x, "SpatialPolygons")
```

and you then proceed as above to convert the curves to `spatstat` format.

The data frame of auxiliary data is extracted by `df <- x@data` or `df <- slot(x, "data")`.

For example:

```
> cp <- as(columbus, "SpatialPolygons")
> cregions <- slot(cp, "polygons")
> cregions <- lapply(cregions,
                      function(x) { SpatialPolygons(list(x)) })
> cwindows <- lapply(cregions, as.owin)
```

There is currently no facility in `spatstat` for attaching marks to an "owin" object directly. However, `spatstat` supports objects called **hyperframes**, which are like data frames except that the entries can be any type of object. Thus we can represent the `columbus` data in `spatstat` as follows:

```
> ch <- hyperframe(window=cwindows)
> ch <- cbind.hyperframe(ch, columbus@data)
```

Then `ch` is a hyperframe containing a column of "owin" objects followed by the columns of auxiliary data.

3.9.3.7 Objects of class "SpatialGridDataFrame" and "SpatialPixelsDataFrame"

An object `x` of class "SpatialGridDataFrame" represents a pixel image on a rectangular grid. It includes a "SpatialGrid" object `slot(x, "grid")` defining the full rectangular grid of pixels, and a data frame `slot(x, "data")` containing the pixel values (which may include NA values).

The command `as(x, "im")` converts `x` to a pixel image of class "im", taking the pixel values from the *first column* of the data frame. If the data frame has multiple columns, these would currently have to be converted to separate pixel images in `spatstat`. For example

```
> y <- as(x, "im")
> ylist <- lapply(slot(x, "data"),
                  function(z, y) { y[,] <- z; y }, y=y)
```

An object `x` of class "SpatialPixelsDataFrame" represents a *subset* of a pixel image. To convert this to a `spatstat` object, it should first be converted to a "SpatialGridDataFrame" by `as(x, "SpatialGridDataFrame")`, then handled as described above.

3.10 FAQ's

- Why doesn't `spatstat` use the same classes as `sp`?
- Can/should I record points lying just outside the sampling quadrat?
-

4

Inspecting and Exploring Data

This Chapter covers techniques for inspecting, manipulating and exploring spatial data. Section 4.1 describes methods for plotting point patterns, pixel images, and some other classes of object. Section 4.2 gives tools and techniques for manipulating point pattern data. Section 4.3 gives methods for plotting, manipulating and exploring pixel image data. Section ?? describes the class of observation windows in `spatstat`. Section 4.4 describes the class of tessellations in `spatstat`. Section ?? discusses exploratory methods for investigating relationships in spatial data.

4.1 Plotting

Graphics are essential in statistical science. In spatial statistics, plotting the original spatial data is useful for checking errors in the data record, for motivating the appropriate analysis, and for investigating anomalies. Plotting transformations of the data, and results of the analysis, is essential for exploratory and formal statistical analysis. For publication and communication, plots of spatial data are attractive and can have great impact. Most of all, graphics are fun.

4.1.1 Overview of plot commands

The generic `plot()` function in R is intended to “generate a sensible graphical display” of the data, whatever that means for the particular dataset. In `spatstat`, every class of spatial data has a method for the generic `plot()` function. Thus, even if you can’t remember any other commands in `spatstat`, it is worth trying `plot(X)`, where X is the name of your dataset, to obtain some kind of reasonable plot. The display is generated by `plot.foo()`, the `plot()` method for class “`foo`”; for information on how to modify the display, consult the help for `plot.foo()`. The `plot()` methods for point patterns and other spatial objects in `spatstat` are discussed in this section, and in Section 4.3.

Additionally, R and its packages provide many other functions for generating graphical displays. Some of these functions are also generic. For example, `hist()`, `contour()`, `image()` and `persp()` are generic functions that display a histogram, contour plot, colour image, and surface perspective plot of the data. Only some classes of objects will have methods for these special plots. For pixel images, these and other plotting tools are presented in Section 4.3.

In `spatstat` there are some special classes of object whose main purpose is simply to indicate that the data should be plotted in a particular way. An object of class “`listof`” is a list of spatial objects of similar kind; `plot.listof()` plots these objects side-by-side. An object of class `layered` is a list of spatial objects, possibly of different kinds, that occupy the same spatial region; `plot.layered()` plots these objects on top of one another. Simple facilities are provided for selecting only some of the objects for plotting, and for modifying the plot in different ways.

Interactive display of point patterns is discussed in Section 4.1.4.

4.1.2 Plotting a point pattern

The `plot()` method for point patterns, `plot.ppp()`, displays the observation window for the pattern and plots the spatial locations of the points, using equal physical scales for the x and y coordinates. If the points carry marks, then the points are represented by different symbols according to their mark value, and the mapping from mark values to symbols is displayed in a legend. This mapping is also returned (invisibly) as the result of the `plot()` command.

Default behaviour

The precise behaviour depends on the nature of the marks, and on arguments given to the `plot()` function by the user. The *default* behaviour is illustrated in Figures 4.1 and 4.2.

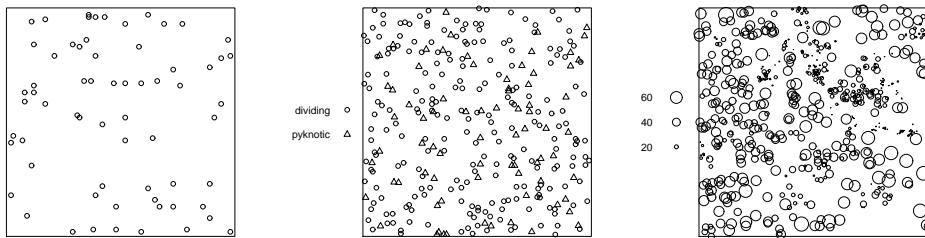


Figure 4.1: Default behaviour of `plot.ppp()` for different kinds of data. *Left:* unmarked point pattern; *Middle:* multitype point pattern; *Right:* point pattern with numeric marks.

If the pattern is unmarked (has no marks) or if the argument `use.marks` is set equal to FALSE, then only the spatial locations are plotted. By default the plotting symbol is an open circle. The plotting symbol is controlled by the argument `pch`. Good options, depending of course on taste and on context, are `pch=16` or `pch=3`. Use the character expansion factor `cex` to change the size of the symbols. For very large patterns, one can also set `pch=". "` to represent each point as a tiny dot.

Notice that, in the help file for `plot.ppp()`, the list of arguments does not include `pch` and `cex`. However it does include “...” which stands for “other unrecognised arguments”. This mechanism allows such unrecognised arguments to be passed through the function `plot.ppp()` to some other function which does recognise them. The help for `plot.ppp()` states that such arguments are passed to the low-level R graphics function `points()`. Looking up the help file for `points()`, we find that `pch` and `cex` are recognised parameters. Possible values for `pch` are explained in `help(points)` and there is a beautiful demonstration of them all in `example(points)`.

If the marks of the pattern are categorical (of class "factor") then each level of the factor is plotted as a different symbol. That is, a different value of `pch` is used to plot points corresponding to each factor level. A legend is plotted to the left of the main plot indicating which symbol corresponds to each mark. The default is to use the plotting characters $1, \dots, k$ where k is the number of levels of the factor. A different choice may be specified using the `chars` argument which should be a vector of length equal to the number of levels. The entries of `chars` should be values which are acceptable as values of `pch`.

If the marks are real numbers, then points whose marks are positive numbers are plotted as circles of diameter proportional to the mark value. Points whose marks are negative numbers are plotted as squares of side length proportional to the (absolute value of the) mark. Again a legend

is plotted to the left indicating the relationship between the mark value and the size of the plotted shape.

If the marks are dates then these dates are converted to seconds of elapsed time since some “origin” and plotted as circles in the same manner as are positive real valued marks.

If the marks are multivariate (i.e. if the `marks` component of the pattern is a data frame) then a separate plot is produced for each column of marks, as shown in Figure 4.2. The argument `which.marks` may be used to specify that plots are to be produced only for a subset of the columns of the `marks` data frame. For example, a single column of marks may be picked out for use.

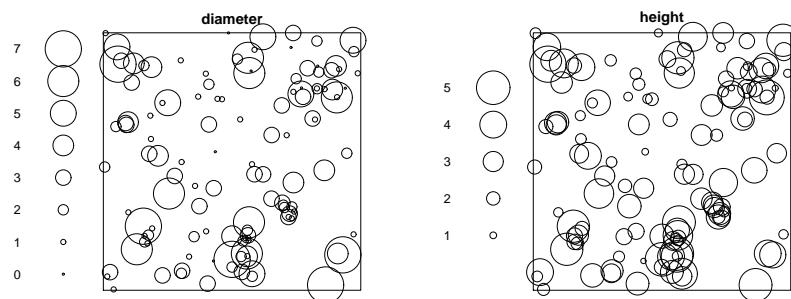


Figure 4.2: Default behaviour of `plot.ppp()` for a point pattern with several mark variables: a separate plot panel is displayed for each mark variable.

The default behaviour of `plot.ppp()` can be modified by giving additional graphics parameters, as we discuss below.

Symbol maps

The return value from `plot.ppp()` is an object of class "symbolmap" representing the mapping from marks to symbols. This object can be examined, manipulated, and used for other plots.

```
> a <- plot(amacrine)
> a
Symbol map for discrete inputs:
[1] "off" "on"
chars: [1] 1 2
> (plot(longleaf))
Symbol map for real numbers in [2, 75.9]
shape: circles
size: function (x, scal = 0.172252245134423)
{
  scal * x/2
}
<environment: 0x51cd2a8>
```

A symbol map specifies the possible inputs (mark values that can be mapped), and the graphical parameter values to which these inputs will be mapped. The inputs may be a vector of categorical values or a range of real numbers. The graphical parameter names may be any of the graphics

arguments `shape`, `pch`, `chars`, `size`, `cex`, `col`, `cols`, `fg`, `bg`, `lty`, `lwd`, `border`, `fill` and `etch`. Each graphical parameter may be either a constant value, a vector of values corresponding to the inputs, or a function that will be applied to the input values.

The default symbol map used by `plot.ppp()` was effectively described above. This default can be modified by specifying graphics parameters in the call to `plot.ppp()`. For example, the default behaviour for a point pattern with positive numeric marks is to display circles of diameter proportional to the marks. If we wish to encode the mark values of the Longleaf Pines data as colours rather than sizes, one possibility would be

```
> A <- colourmap(heat.colors(128),
+                  range=range(marks(longleaf)))
> plot(longleaf, pch=21, bg=A, cex=1)
```

The effect of `cex=1` is that all symbols will be the same size; plot symbol 21 is a filled circle; parameter `bg` is the background or interior colour for plot symbols 21 to 25; and the object `A` is a colour map, explained in Section 4.3. A greyscale equivalent is shown in Figure 4.3.

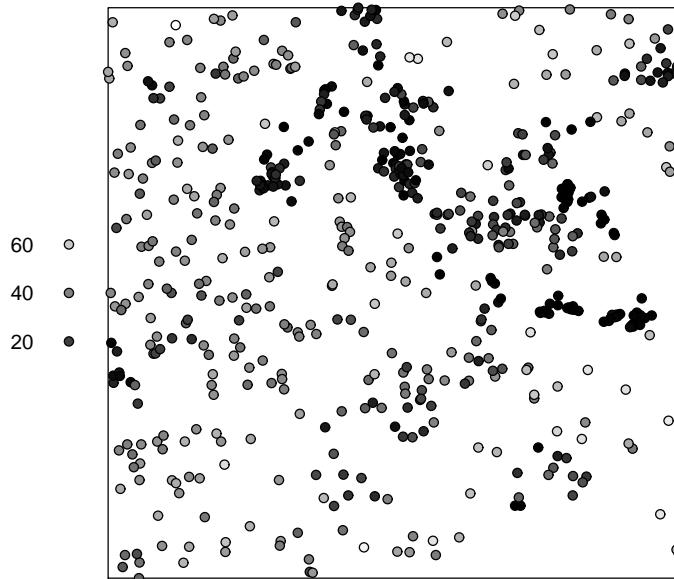


Figure 4.3: Longleaf Pines data with marks encoded as greyscale values.

If the argument `symp` is given in the call to `plot.ppp()`, then this determines the symbol map used in the plot of the point pattern, overriding the default behaviour described above. A typical use of `symp` is to re-use a previous symbol map. For example, the following commands display the Longleaf Pines data, and next to it, the subset of trees with diameter less than 30 cm, *using the same scale* for the marks in both plots. See Figure 4.4.

```
> juveniles <- subset(longleaf, marks <= 30)
> a <- plot(longleaf)
> plot(juveniles, symp=a)
```

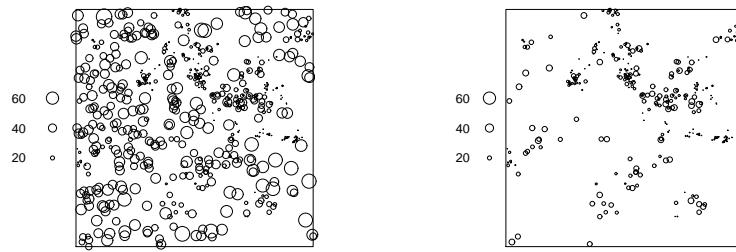


Figure 4.4: Illustrating the re-use of a symbol map. Full dataset of Longleaf Pines data (*Left*) and subset consisting of juvenile trees (*Right*) plotted with the same symbol map.

A symbol map can also be created directly using the function `symbolmap`.

```
> g1 <- symbolmap(inputs=letters[1:10], pch=11:20)
> g2 <- symbolmap(range=c(0,100), size=function(x) {x/50})
```

A symbol map can be modified using `update.symbolmap()`, a method for the generic function `update()`:

```
> g3 <- update(g2, col="red")
> g4 <- update(g3,
+               col=function(x) ifelse(x < 30, "red", "black"))
> g4
Symbol map for real numbers in [0, 100]
col: function(x) ifelse(x < 30, "red", "black")
size: function(x) {x/50}
shape: circles
```

A `symbolmap` can be printed and plotted in its own right: the legend accompanying the plot of a point pattern is produced by `plot.symbolmap()`.

4.1.2.1 Plotting a window

In some cases it may be useful to plot an object of class "owin" in its own right. For instance the dataset `murchison` includes an "owin" object outlining the greenstone in the survey area, which can be plotted using `plot.owin()`:

```
> plot(murchison$greenstone, main = "")
```

For polygonal (and rectangular) windows the plotting is done by the R function `polygon()` and arguments acceptable to this function can be given for finer control as explained in the help for `plot.owin()`. For example, to paint the interior of the polygon in red, use the argument `col="red"`. To draw the polygon edges in green, use `border="green"`. To suppress the drawing of polygon edges, use `border=NA`. It is also possible to use transparent colours (fourth argument to functions such as `rgb()` and `hsv()`), which can be convenient when plotting overlapping polygons:



Figure 4.5: Plotting an "owin" object.

```
> plot(square(c(-1,1)), main = "")  
> plot(ellipse(1,.5), col = rgb(0,0,0,.2), add = TRUE)  
> plot(ellipse(.5,1), col = rgb(0,0,0,.2), add = TRUE)
```

The resulting overlapping ellipses are shown in Figure 4.6.

4.1.3 Plotting an image

Just as a dog tilts its head from side to side to understand what it is seeing, so also a data analyst should try several different views of the same data.

Table 4.1 shows some of the most useful commands for displaying a pixel image (object of class "im"). Figure 4.7 shows two examples.

plot.im() and colour maps

In keeping with the design of R, the `plot()` method for images, `plot.im()`, gives the most “natural” display of the data in most cases. Pixel values are rendered as colours or shades of grey, and displayed as a block of colour or greyscale on the plot region. Colours are rendered as greyscales in this book. The mapping from pixel values to colours or greyscales — the *colour map* — is displayed in a “ribbon” to the right of the image. The return value from `plot.im()` is an object of class "colourmap" representing the colour map used in the plot.

The observer’s perception of details in a colour image depends greatly on the colour map used [68] so it is well worth experimenting with several colour maps for the same image data. The colour

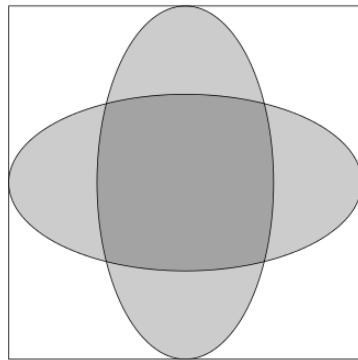


Figure 4.6: Overlapping "owin" objects plotted with semi-transparent colors.

<code>plot(X)</code>	plot colour image
<code>image(X)</code>	plot colour image
<code>contour(X)</code>	draw contour map
<code>persp(X)</code>	display perspective view of surface
<code>iplot(X)</code>	launch interactive plot panel
<code>textureplot(X)</code>	plot values as textures
<code>plot(transect.im(X))</code>	plot pixel values along a line transect

Table 4.1: Useful commands for plotting a pixel image. The standard functions `plot()`, `image()`, `contour()` and `persp()` are generic, with methods for pixel images provided by `spatstat`. Additionally `iplot()` is generic.

map used in `plot.im()` is controlled by the argument `col`, which can be either an explicit colour map object (of class "colourmap") or a vector of values interpretable as colours.

Values which specify colours in R can be strings containing the names of colours (like "magenta") or strings of hexadecimal codes (like "#FF00FF") or integers in the range 1 to 8 indexing the standard palette of pen colours (see `palette()`). There are many facilities available in R for generating smooth gradations of colour, for use in a colour map. Simple choices are the functions `rainbow()`, `heat.colors()`, `terrain.colors()`, `topo.colors()` and `cm.colors()`. For example `rainbow(128, end=5/6)` generates a vector of 128 colour values looking something like the visible rainbow. See `help(colours)` or the package `RColorBrewer` for further options.

In the command `plot(X, col=V)`, if `V` is a vector of colours, then the range of pixel values in `X` will be mapped linearly to the colours in `V`. This is convenient for a single use, but if we plot two different images `X` and `Y` using `plot(X, col=V)` and `plot(Y, col=V)`, comparison between the plots is not possible, because the same colour represents different pixel values in the two images.

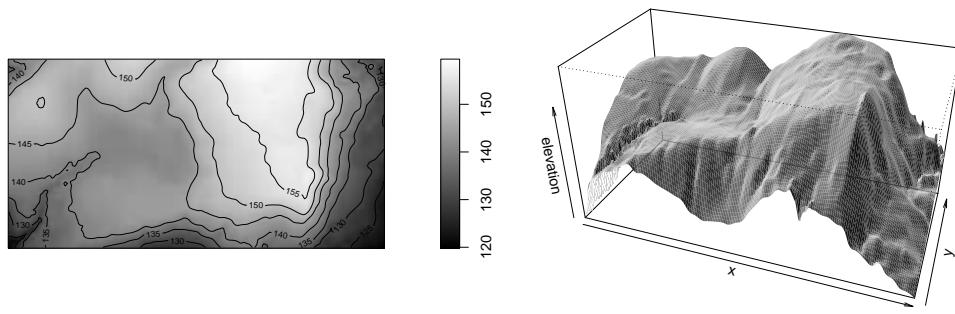


Figure 4.7: Different displays of the same image data. *Left:* image plot with contour plot superimposed. *Right:* perspective plot.

A colourmap object is a *fixed* correspondence between numerical pixel values and colours. If two images are plotted using the same colourmap object, we can be confident that a given colour represents the same pixel value in both plots.

The user can make a colourmap object by hand using the function `colourmap()`:

```
> g <- colourmap(rainbow(128), range=c(0,100))
> h <- colourmap(c("green", "yellow", "red"),
+                 inputs=c("Low", "Medium", "High"))
```

Here `g` associates colours with numerical values, and `h` associates colours with categorical values. A colourmap object also behaves as a function, so that `g(50)` returns the colour associated with the pixel value 50:

```
> g(50)
[1] "#00FFFFFF"
> h("Medium")
[1] "yellow"
```

Colour map objects can be modified by the `spatstat` functions `interp.colourmap()`, `tweak.colourmap()`, `complementarycolour()` and `to.grey()`.

To ensure that two images `X` and `Y` are plotted using a common colour map which embraces the range of values in both images, the usual idiom is

```
> ra <- range(X, Y)
> cm <- colourmap(rainbow(128), range=ra)
> plot(X, col=cm)
> plot(Y, col=cm)
```

One can also use `plot.listof()` to plot several images side-by-side with the same colour map, as explained below.

Other plotting functions for images

In addition to `plot.im()`, several other graphics commands for pixel image data are listed in Table 4.1.

The `spatstat` interactive plotting function `iplot()` also applies to pixel images. It is effectively an interactive version of `plot.im()`, and is useful for zooming in to view fine details.

The R generic functions `contour()` and `persp()` require the pixel values to be numerical,

and treat them as the elevation (altitude) of a surface: `contour()` produces a contour map of this surface, and `persp()` produces a perspective view.

For a pixel image the command `contour(X)` is dispatched to `contour.im()` which ultimately calls `contour.default()`. Arguments to control the contour map can be found in the help files for `contour.im()` and `contour.default()`. Useful ones include `nlevels` for the number of contour lines, `drawlabels` to specify whether the contour lines should be labelled, and `col`, `lty`, `lwd` for the colour, type, and width of contour lines.

The method `persp.im()` displays a perspective view of the surface, with equal scales for the *x* and *y* coordinates but a different scale for the *z* coordinate. The ratio between the *z* and *x*, *y* scales is the argument `expand`. Plotting is performed by `persp.default()`. This command is very flexible and can produce some beautiful pictures: see `demo(persp)`.

The angle from which the surface is viewed can be quite influential; it is controlled by the arguments `theta` and `phi`. The default behaviour of `persp()` is to draw the pixel edges in black, producing a wireframe representation of the surface. It is usually better to set `border=NA` to suppress the wireframe and `shade=0.7` (for example) to fill the pixel surfaces with shades of grey calculated as if the sun were shining on the surface. For shades of red instead of grey, set `col="red"`.

The `spatstat` method `persp.im()` has some additional features allowing the surface colours to be controlled by spatial data. If the argument `colin` (“colour input”) is given, it should be a pixel image, and the surface will be coloured according to the values of `colin`. Thus `persp(X, colin=Y)` looks as if we had printed `plot(Y)` on a sheet of paper and draped it over the wireframe surface in `persp(X)`. The colour map is controlled by the argument `colmap`. If the argument `colmap` is given and `colin` is missing, then the surface colours are determined by the *surface elevation* according to the specified colour map: try `colmap=terrain.colors(128)`. If the argument `shade` is given, the colours described above will be transformed to darker shades of the same colour according to the terrain lighting model. If `apron=TRUE`, the surface will be surrounded by vertical sides so that it looks more like a solid. For a realistic display of the *Beilschmiedia* terrain data, complete with snow-capped mountains, try

```
> persp(bei.extra$elev, expand=6,
        theta=-30, phi=20,
        colmap=terrain.colors(128),
        shade=0.2,
        apron=TRUE, main="", box=FALSE)
```

The return value of `persp` is a matrix representing the projection from 3D to 2D space. This can be used to add other graphics to the perspective plot, typically using the function `trans3d()`. For example, Figure 4.8 shows the locations of the *Beilschmiedia* trees as dots on the terrain. It was produced by

```
> mat <- persp(bei.extra$elev,
                 border=NA, apron=TRUE, shade=0.2,
                 expand=5, box=FALSE, theta=-30, phi=45)
> xy <- trans3d(bei$x, bei$y, bei.extra$elev[bei], pmat=mat)
> points(xy, pch=". ", cex=2)
```

A problem with this technique is that all points are plotted, even if they should be obscured by other parts of the terrain. Here we chose a large enough `phi` and small enough `expand` so that all surface facets are visible.

The function `transect.im()` extracts the pixel values of an image along a line in two-dimensional space — by default, the diagonal line from bottom left to top right of the image frame. The result is a function table (class “fv”) which can be plotted directly, to produce a “profile” or “vertical section” of the corresponding surface. Figure 4.9 shows the result of `with(bei.extra, plot(transect.im(elev)))`.

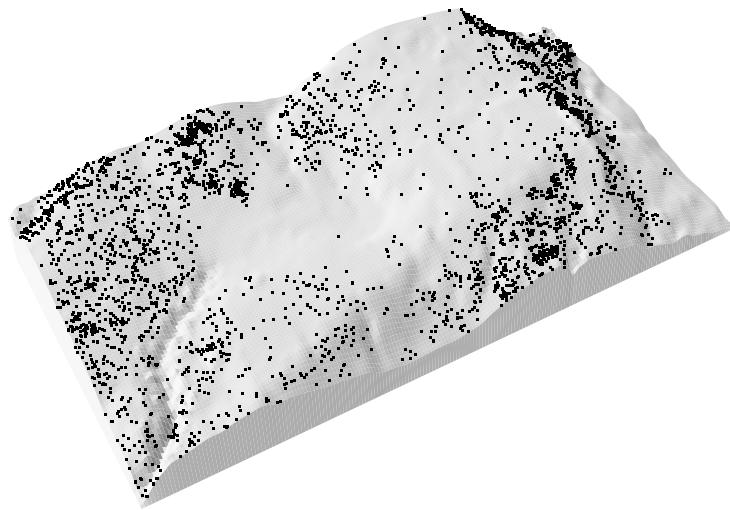


Figure 4.8: Perspective view of rainforest terrain with *Beilschmiedia* tree locations superimposed.

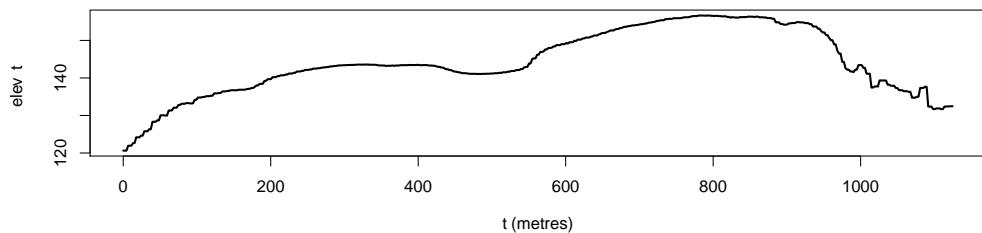


Figure 4.9: Profile of terrain elevation along a line transect of the *Beilschmiedia* data. Transect runs from bottom left to top right of study area.

The R generic function `cut()` transforms numerical data into categorical values by dividing the range of numbers into bands, and categorising the numbers according to the band in which they fall. The method for images, `cut.im()`, applies this to the pixel values of an image. This can be very useful for visualising pixel images which have continuous gradual changes in pixel value.

For images with categorical (factor) values, `plot.im()` is usually the best option if colour display is available. Each level of the factor will be displayed as a different colour; careful choice of the colour map may be needed. An alternative is `textureplot()`, which finds the regions associated with each level of the factor, and fills each region with a different geometric pattern. Regions associated with levels of the factor can also be extracted with `split()` or `solutionset()` (see below) and plotted separately.

Distribution of pixel values

The function `hist.im()`, a method for the generic `hist()`, displays a histogram of the pixel values in an image with numerical values, or a bar plot of the pixel values in an image with categorical values. The function `spatialcdf()` computes the cumulative distribution function of the pixel

values in an image with numerical values. The result is a function (of class "stepfun") which can be plotted immediately.

```
> with(bei.extra, hist(grad, freq=FALSE))
> with(bei.extra, plot(spatialcdf(grad, normalise=TRUE)))
```

Inspection of such displays (shown in Figure 4.10) may suggest an appropriate transformation of the pixel values; see section 4.3.3.

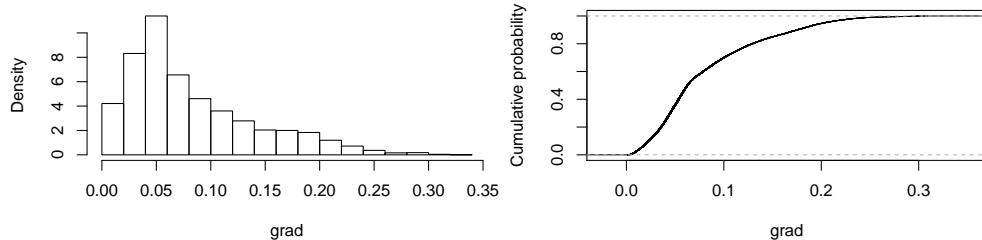


Figure 4.10: Histogram (*Left*) and cumulative distribution function (*Right*) of terrain elevation in the *Beilschmiedia* data.

4.1.4 Interactive plotting

Interactive display, in which the user is able to point the mouse at the display and click to extract information or change the display, is also supported in a modest way.

Mouse-click interaction

Virtually all installations of R support a simple but reliable form of interaction which responds only to mouse clicks (and not to mouse-down, mouse-up and other signals) in the recommended package `graphics`. The function `locator()` records the spatial locations of the mouse when the left mouse button is clicked.

The R generic function `identify()` allows the user to select data points in an existing plot, and read information about them. Methods for `identify()` are provided in `spatstat`, including `identify.ppp()`. After typing

```
> plot(amacrine)
> identify(amacrine)
```

the user can click on any point of the `amacrine` pattern. Information about the data point will be printed, including its serial number and its mark value.

The `spatstat` package also provides interactive functions based on `locator()`, including `clickbox()`, `clickpoly()` and `clickppp()`, which can be used to draw a rectangle, a polygonal window or a point pattern, respectively. A common use of `clickbox()` is to select a sub-region of a spatial data set.

Elaborate interaction

Additionally `spatstat` has an experimental interactive plotting function `iplot()` which depends on the `rpanel` package and is less stable. The function `iplot()` is generic, with methods for objects of classes "ppp" and "layered" as well as a default method that works for classes "im", "psp" and "owin". When the user types `iplot(X)`, a new pop-up window is launched. See Figure 4.11. The spatial data set X is displayed in the middle of the window using the appropriate `plot()` method. The

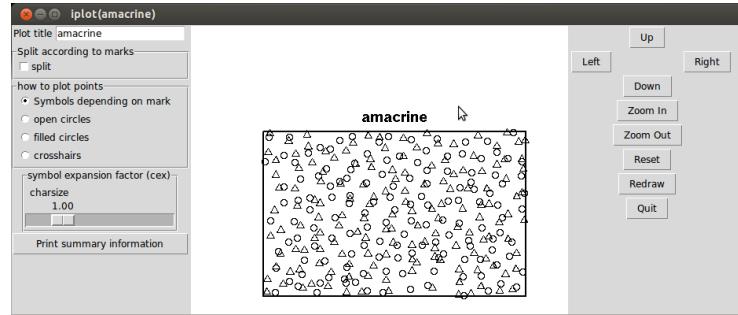


Figure 4.11: Screenshot of the interactive graphics window started by `iplot()` for a point pattern.

left side of the window contains buttons and sliders allowing the user to change the plot parameters. The right side of the window contains navigation controls for zooming (changing magnification), panning (shifting the field of view relative to the data), redrawing and exiting. If the user clicks in the area where the point pattern is displayed, the field of view will be re-centred at the point that was clicked.

Zooming in to a spatial point pattern can be extremely useful for investigating point patterns with dense concentrations of points, and for checking whether the spatial coordinates were discretised.

4.1.5 Plotting several objects

4.1.5.1 Layered plots

Layering is a simple mechanism for controlling a high-level plot that is composed of several successive plots, for example, a background and a foreground plot. The layering mechanism makes it easier to issue the plot command, to switch on or off the plotting of each individual layer, to control the plotting arguments that are passed to each layer, and to zoom in.

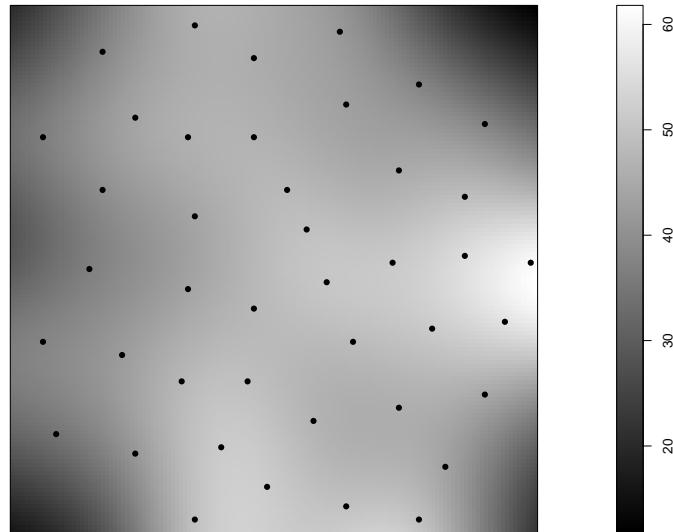


Figure 4.12: Plot of a layered object

The command

```
> X <- layered(...)
```

simply saves the objects . . . as a list, of class "layered". Each individual object should belong to some class that has a method for the generic function `plot()`.

The layered object `X` can then be plotted by the method `plot.layered()`. Thus, you only need to type a single `plot()` command to produce the multi-layered plot. Individual layers of the plot can be switched on or off, or manipulated, using arguments to `plot.layered()`, or by selecting a subset using "[.layered"].

Default values of the plotting arguments for each layer can be set, by calling

```
> X <- layered(..., plotargs=p)
```

or equivalently

```
> X <- layered(...)
> layerplotargs(X) <- p
```

where `p` should be a list, with one entry for each layer. Each entry of `p` should be a list of arguments in the form `name=value`, which are recognised by the `plot()` method for the relevant layer.

For example, Figure 4.12 was generated by the following code:

```
> X <- layered(density(cells), cells)
> layerplotargs(X)[[2]] <- list(pch=16)
> plot(X, main="")
```

There is also an `iplot()` method for layered objects, which allows the user to switch each layer on and off at the click of a mouse, to zoom in and out, and so on.

Various other objects can be converted to layered objects by `as.layered()`.

4.1.5.2 Plotting "listof" objects

The `plot()` method for `listof` objects (see Section 3.7.1), `plot.listof()`, displays the entries of the list side-by-side, or in a rectangular array. If the entries are all spatial objects (such as point patterns, line segment patterns, pixel images), then `plot.listof()` will try to plot them using compatible scaling and alignment. If the entries are not all spatial objects, `plot.listof()` will still try to plot them appropriately, but scaling and alignment cannot always be chosen well.

In the following example, all the entries of the list are spatial objects.

```
> X <- swedishpines
> QC <- quadratcount(X)
> QCI <- as.im(X, dimyx=5)
> DI <- density(X)
> L <- listof(X, QC, QCI, DI)
```

Figure 4.13 shows a plot of the resulting object `L`.

For a `listof` object whose entries are all spatial objects, setting `equal.scales=TRUE` will ensure that all entries are plotted using the same physical scale for the coordinates. This might be undesirable if the objects have very different spatial sizes. Setting `valign=TRUE` will ensure that exactly the same `y` coordinate system is used for all plots on a given row of the array of plots. Setting `halign=TRUE` will ensure that exactly the same `x` coordinate system is used for all plots in a given column of the array. Alignment options are undesirable if the objects are not in the same spatial location. Try the examples

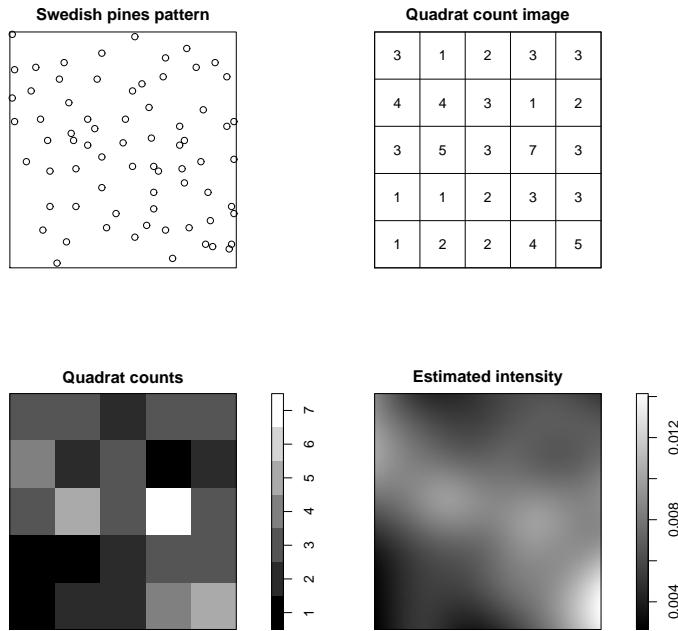


Figure 4.13: Various plots associated with the `swedishpines` data, stored in the "listof" object L described in the text.

```
> P <- listof(A=cells, B=japanesepines, C=redwood)
> plot(P, equal.scales=TRUE)
> plot(P, equal.scales=TRUE, valign=TRUE)
```

In Figure 4.13 we needed to set `valign=TRUE`.

4.1.5.3 Plotting several images

Table 4.2 lists commands which are useful for comparing several pixel images that are defined on the same spatial region.

Layered plots were discussed in Section 4.1.5.1. If X, Y, Z are pixel images defined on the same region, then `plot(layered(X, Y, Z))` would plot the images on top of one another, so that only Z would be visible. However `iplot(layered(X, Y, Z))` starts an interactive graphics panel which gives the option of switching off each of the layers X, Y, Z. This makes it possible to flip quickly between the different images.

Several pixel images X, Y, Z, ... can be assembled into an object of class "listof" by typing `L <- listof(X, Y, Z, ...)` as discussed in Section 3.7.1. Then `plot(L)` invokes `plot.listof()`, which displays the images side-by-side or in a rectangular array. The argument `equal.ribbon=TRUE` will ensure that the same colour scale is used for all images. The arguments `equal.scales=TRUE`, `halign=TRUE` and `valign=TRUE` may or may not be needed, as explained in Section 3.7.1.

<code>plot(listof(X,Y,...))</code>	plot images side-by-side
<code>iplot(layered(X,Y,...))</code>	interactively switch between images
<code>rgbim(X,Y,Z)</code>	RGB colour composite image
<code>hsvim(X,Y,Z)</code>	HSV colour composite image
<code>pairs(X,Y,...)</code>	scatterplot of corresponding pixel values

Table 4.2: Useful commands for plotting or comparing several pixel images X, Y, ...

For a `listof` object whose entries are all pixel images, by default `plot.listof()` calls the function `image.listof()` to display the images. (Note that `equal.ribbon` is an argument of `image.listof()`). In other cases, `plot.listof()` calls the generic `plot()` function to display each entry in the list. This can be overridden by the argument `plotcommand`. For example, setting `plotcommand="hist"` would generate an array of histograms, and `plotcommand="contour"` an array of contour plots, of each entry in the list.

Given three images `X, Y, Z` with numerical pixel values, defined in the same spatial region, the function `rgbim()` interprets the three images as the brightness values of the Red, Green and Blue channels in an RGB colour space. The result is a pixel image whose pixel values are colour values (character strings of hexadecimal colour codes) which can be plotted directly by `plot.im()`. This is useful for detecting and locating differences between two very similar images, such as year-to-year changes in a remotely sensed image. For example `plot(rgbim(X,Y,0))` will show areas of disagreement between `X` and `Y` as patches of red or green, while the areas of agreement will be yellow. Similarly the function `hsvim()` interprets three image inputs as the Hue, Saturation and Value channels of a colour. This is useful for combining image datasets that are fundamentally different.

To study relationships between variables in a dataset, the generic R function `pairs()` can be useful. It displays an array of scatterplots for each pair of variables in the dataset. In `spatstat` we provide a method for images, `pairs.im()`, which extracts the values of corresponding pixels in several images, and optionally generates the array of scatterplots using `pairs.default()`. There is also a `pairs.listof()` for handling a list of pixel images. Figure 4.14 shows the result of the command

```
> pairs(density(split(lansing)[c(2,3,5)]))
```

This divides the Lansing Woods data into 6 sub-patterns of different tree species, extracts species 2, 3 and 5 which are the most common, computes the kernel smoothed intensity estimate for each species, and then displays scatterplots of the intensity estimates for each pair of species, plotted against each other.

The plot suggests that hickory and maple trees are strongly segregated from one another (since a high density of hickories is strongly associated with a low density of maples). There appears to be some segregation of hickory and red oak as well, although this association is not as strong. The relationship between red oak and maple exhibits no clear structure.

The scatterplots in each panel can be replaced with other kinds of plots, by specifying the argument `panel` (or `upper.panel` and `lower.panel`). The argument should be a function, as described in the help for `pairs.default()`. In `spatstat` we provide panel functions `panel.contour()`, `panel.image()` which draw contour maps and image displays respectively. It is also possible to

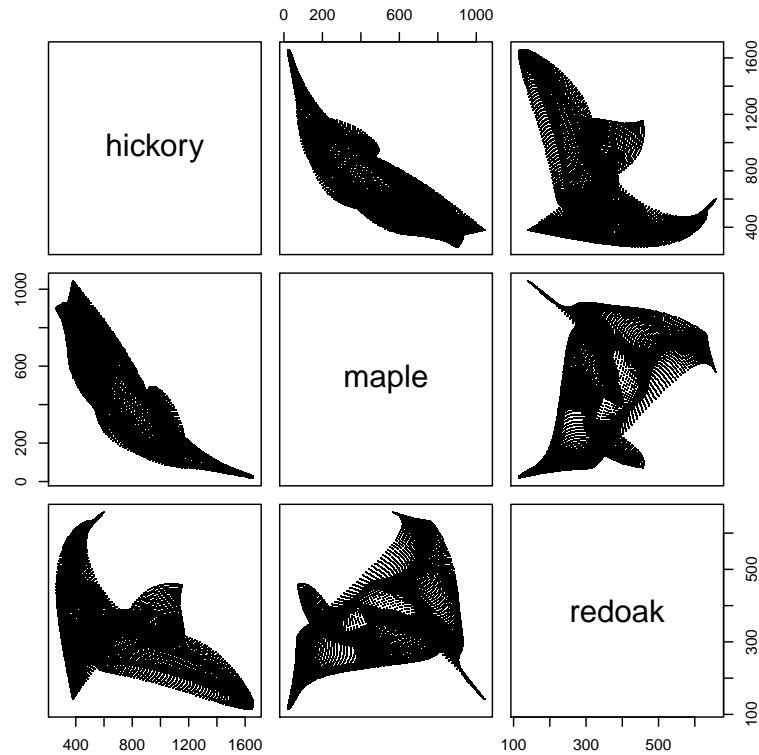


Figure 4.14: Scatterplot pairs display for the densities of each tree species in Lansing Woods.

fill the diagonal panels with another plot specified by the argument `diag.plot`: `spatstat` provides the panel function `panel.histogram()` to draw a histogram of the pixel values.

The return value of `pairs.im()` is a data frame containing the corresponding pixel values for each of the images. This is useful for calculating summary statistics. For example, to compute the sample correlation between the density estimates for the three dominant species in Lansing Woods,

```
> L <- density(split(lansing)[c(2,3,5)])
> df <- pairs(L, plot=FALSE)
> co <- cor(df)
> round(co, 2)
      hickory  maple  redoak
hickory    1.00 -0.85 -0.51
maple     -0.85  1.00   0.33
redoak    -0.51   0.33   1.00
```

4.2 Manipulating point patterns and windows

We now show how to manipulate point pattern data, for example, how to extract or change the spatial coordinates or the marks, extract a subset of the point pattern, split a pattern into several subsets, or combine several point patterns to make a single pattern.

4.2.1 Basic operations on a point pattern

A point pattern is represented in `spatstat` by an object of the class "ppp". This object consists effectively of the coordinates of the points, the window or study region in which the points were observed, and optionally the mark values associated with each of the points.

Extracting data from a point pattern

Although it is possible to directly access the internal structure of a point pattern object, this is not advisable. *Modifying* the internal components is dangerous, because the data can become internally inconsistent. The internal format of objects in a package can change from one version of the package to another.

It is much safer and neater to use facilities defined in the package to extract and modify or manipulate the data in a point pattern object. Table 4.3 lists the most important functions for extracting data from a point pattern. These are all generic functions with methods for class "ppp". For help on extracting the coordinates of a point pattern, see the help for `coords.ppp()`.

<code>npoints(X)</code>	number of points in X
<code>marks(X)</code>	marks of X
<code>coords(X)</code>	coordinates of points in X
<code>as.data.frame(X)</code>	coordinates and marks of X
<code>Window(X)</code>	window of X
<code>Frame(X)</code>	containing rectangle of X
<code>boundingbox(X)</code>	bounding box of X

Table 4.3: Important functions for extracting data from a point pattern.

The bounding box of a spatial object in `spatstat` is the smallest rectangle with sides parallel with the coordinate axes that encloses all of the “essential points” of the object. For a point pattern object the essential points are of course the points of the pattern. For a window the relevant points are the vertices of the window (meaning the outer corners of the TRUE pixels for a mask while the meaning is obvious for windows of type `polygonal` and `rectangle`). The bounding box of the point pattern is always contained in the bounding box of the window. The “Frame” of a point pattern, extracted by `Frame(X)`, is a rectangle that contains the observation window, `Window(X)`. This is usually equal to the bounding box of the window, but may be larger in some circumstances. See Figure 4.15. The Frame is used in many calculations which require a rectangle containing X, for example when setting the plot region for `plot(X)`, or when converting X to a pixel image by `pixellate(X)`.

Note that the `ripras()` function mentioned in Section 3.3, called with `shape="rectangle"`, returns a slightly larger rectangle than does `boundingbox()`. The `ripras()` function attempts to accommodate the negative bias which is present when the window is “estimated” by the bounding box. Note that if the window were taken to be the bounding box then there would always be points of the pattern on all four edges of the window. This would not happen for a real experiment with a proper sampling region, except perhaps in settings where there was large rounding error in the recorded coordinates.

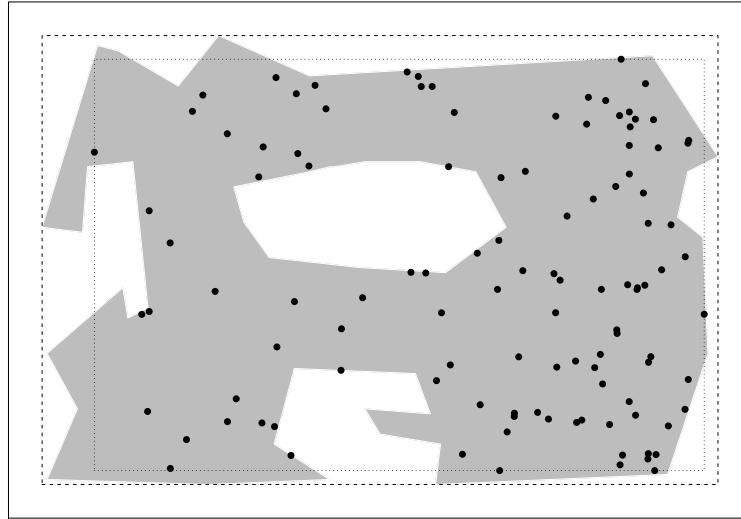


Figure 4.15: Point pattern dataset showing the bounding box of the points (dotted lines), bounding box of the Window (dashed lines), the Window (grey shading), and the Frame (solid lines).

Changing data in a point pattern

<code>marks(X) <- value</code>	change the marks of X
<code>coords(X) <- value</code>	change the coordinates of X
<code>Window(X) <- value</code>	change the window of X
<code>Frame(X) <- value</code>	change the containing rectangle of X

Table 4.4: Important commands for altering the data in a point pattern.

Table 4.4 lists the most important commands for altering data in a point pattern. For example,

```
> X <- redwood
> marks(X) <- nndist(X)
```

will attach a mark to each point in the `redwood` dataset, with mark value equal to the distance to the nearest other point.

The side-effect of `marks(X) <- m` is that the dataset `X` is changed. If we had written

```
> marks(redwood) <- nndist(redwood)
```

then the standard `redwood` dataset would have been changed, which is not advisable if you wish to preserve your sanity.

It is useful to understand the mechanics of these commands. In R syntax, an assignment such as `marks(X) <- m` is translated into a call to the assignment function "`marks<-`". This is a generic function, with a method for point patterns, "`marks<- .ppp`". Consult the help for "`marks<- .ppp`" for information about assigning marks to a point pattern.

Note that R is clever enough to handle multiple layers of assignment in one expression. For example

```
> levels(marks(Y)) <- c("case", "control")
```

is equivalent to

```
> m <- marks(Y)
> levels(m) <- c("case", "control")
> marks(Y) <- m
```

Similarly `marks(X)[3] <- 5` would set the mark value of the third data point in `X` to be equal to 5, leaving other mark values unchanged. It is syntactically expanded to something like

```
> m <- marks(X)
> m[3] <- 5
> marks(X) <- m
```

In commands like `Window(X) <- W`, if the new window `W` is smaller than the current `Window(X)`, points falling outside `W` will be discarded. Similarly for `Frame(X) <- B`, all data outside the box `B` will be removed, and the window will be intersected with `B`.

4.2.2 Conversion of window types

As indicated above it is possible to change the observation window of a point pattern. One reason for doing this may be that another representation of the window is wanted. As described in Section 3.6 there are three different window formats in `spatstat`: rectangles, polygonal regions, and binary pixel masks. Table 4.5 lists some useful tools for converting between different window formats.

The use of these functions is fairly straightforward and the reader should find the help files easy to understand. However a few comments are perhaps in order.

When applied to a window object of type `rectangle`, the function `as.polygonal()` just changes the internal structure of that object to that of a window object of type `polygonal`. There is no real change in the window. When applied to window of type `mask` it creates a polygon all of whose edges are either horizontal or vertical, these edges being edges of the outer pixels comprising the mask-type window.

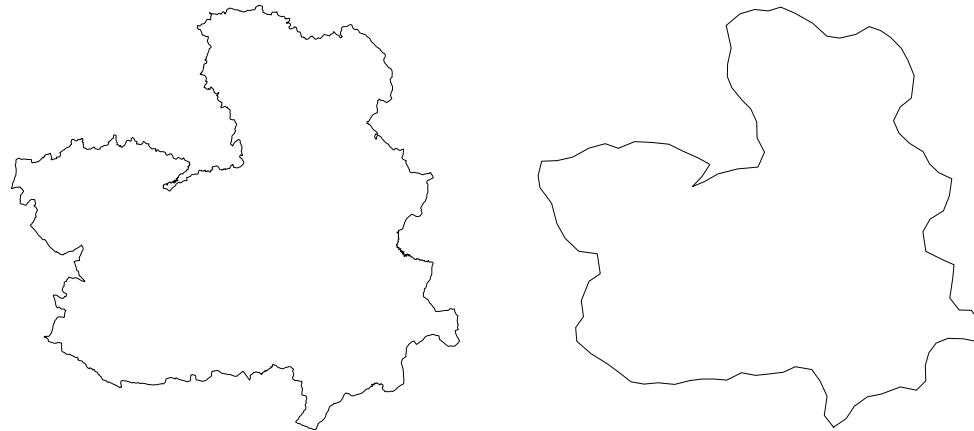
The `simplify.owin()` function converts a complicated polygon into one with fewer edges:

```
> W <- Window(clmfires)
> U <- simplify.owin(W,10) # Gives a polygon with about 200 edges.
```

The resulting windows are shown in Figure 4.16.

<code>as.polygonal()</code>	convert a window to a polygonal window
<code>as.mask()</code>	convert a window to a binary image mask window
<code>as.rectangle()</code>	extract the bounding rectangle of a window
<code>as.matrix.owin()</code>	convert a window to a logical matrix
<code>pixellate.owin()</code>	convert window to pixel image
<code>simplify.owin()</code>	approximate window by a polygon
<code>raster.xy()</code>	raster coordinates of a pixel mask
<code>is.rectangle()</code>	determine if a window is a rectangle
<code>is.polygonal()</code>	determine if a window is polygonal
<code>is.mask()</code>	determine if a window is a binary pixel mask

Table 4.5: Tools for window format testing and conversion.

Figure 4.16: *Left:* complicated polygonal window W ; *Right:* simplified window U created in the text.

4.2.3 Extracting subsets of a point pattern

In `spatstat` the tools for extracting subsets from a point pattern dataset are called "`[.ppp`" and `subset.ppp()` and these are methods for the R generic commands "`[`" and `subset()`. This supports a versatile and easy-to-use system of subset extraction.

Subset defined by an index

Subsets of a point pattern can be specified by indices using "`[`", in the same manner as for subsets of a vector. That is, if X is a point pattern and s is a vector of positive integers, of negative integers, or of logical values, then $X[s]$ is also a point pattern, consisting of those points of X selected by s . For example we could take s to be the vector of positive integers $1:10$.

```
> 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

```

This produces a point pattern with 10 points. These points are the first 10 points of `bei`. The window of the resulting pattern is the same as that of `bei`.

Note that, although a point pattern should (in principle) be treated as an unordered set, the coordinates are obviously stored in a particular order, and can be addressed using that order. If the points of `bei` had written down in some other (equally arbitrary but different) order, then `bei[1:10]` would consist of a different set of 10 points.

If we took `s` to be the vector of negative integers `-(1:10)`

```
> bei[-(1:10)]
```

we would obtain a point pattern consisting of all *except* the first 10 points of `bei`.

The index can be a vector of logical values with one entry for each point of the pattern, with value `TRUE` if the corresponding point is to be retained. For example,

```
> swedishpines[nndist(swedishpines) > 10]
```

extracts all points of the `swedishpines` pattern for which the nearest other point lies further than 10 units (1 metre) away; and

```
> longleaf[marks(longleaf) >= 42]
```

gives a (marked) point pattern consisting of all those points from `longleaf` where the mark value (the diameter at breast height of the tree) is at least 42 centimetres. The latter example could be done more elegantly using the `subset()` function, explained below. A logical vector shorter than the required length will be ‘recycled’, so that

```
> longleaf[c(FALSE,TRUE)]
```

would extract *every second point* in the sequence.

Tip: You need to put quotes around the subset operator when asking for online help about it. The generic subset operator is "`[`"; the right hand square bracket that you type when using this operator is really just punctuation. The help file for this operator has to be summoned by typing `help("[")`, otherwise an error will result. Likewise the subset method for point patterns is called "`[.ppp`"; the help file is summoned by typing `help("[.ppp")`.

Subset defined by a window

In `spatstat` the indexing operator "`[.ppp`" can also extract a subset of a point pattern corresponding to 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

```

The window of the resulting pattern is `W`.

Subset defined by an expression

The method `subset.ppp()` can be used to extract a subset of a point pattern defined by an expression involving the names of spatial coordinates `x`, `y`, the `marks`, and the names of individual columns of marks, if there are several columns. For example

```
> subset(cells, x > 0.5 & y < 0.4)
```

extracts the subset of the `cells` data consisting of all points with `x`-coordinate greater than 0.5 and `y`-coordinate less than 0.4. Note the single `&`, the vector-wise “and” operator, so that the result of evaluating the expression is a logical vector of length equal to the number of points. Similarly

```
> subset(longleaf, marks >= 42)
```

extracts the trees from the Longleaf Pines data with mark value greater than or equal to 42. The `finpines` dataset has two columns of marks, named `diameter` and `height`, so

```
> subset(finpines, diameter > 2 & height < 4)
```

extracts the trees with diameter more than 2 centimetres and height less than 4 metres.

The argument `select` can be used to retain only some of the columns of marks. It is another expression, involving the name `marks` or the names of columns of marks. For example

```
> subset(finpines, diameter > 2, select=height)
```

retains only the `height` column of marks. In the expression `select`, the variable names are interpreted as column numbers in the data frame of marks, so

```
> subset(nbfires, year == 1999, select=cause:fnl.size)
```

is meaningful; it extracts the subset of the New Brunswick fires data where the year was 1999, and retains only the columns `cause`, `ign.src` and `fnl.size`. We could also do

```
> subset(finpines, select = -height)
```

to remove the `height` column from the marks.

4.2.4 Manipulating marks

The commands `m <- marks(X)` for extracting marks, and `marks(X) <- m` for assigning marks, were discussed in Section 4.2.1 above.

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

```
> 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. It is frequently convenient to make use of this facility when plotting. E.g.

```
> plot(unmark(anemones))
> radii <- rexp(npoints(redwood), rate=10)
> plot(redwood %mark% radii)
```

The last line above plots the `redwood` data as a marked point pattern with random numeric marks.

A common task is to attach marks to a point pattern where the marks depend on another spatial dataset. If `X` is a point pattern and `Z` is a pixel image, then `Z[X]` gives the values of the image at the points of `X`, so `X %mark% Z[X]` attaches these values to the points.

```
> elev <- bei.extra$elev
> Y <- bei %mark% elev[bei]
```

To attach an *additional* column of marks to a dataset which already has marks, one would typically use `cbind()` or `data.frame()`.

```
> X <- amacrine
> marks(X) <- data.frame(type=marks(X), nn=nndist(amacrine))
> Y <- finpines
> vol <- with(marks(Y), (100 * pi/12) * height * diameter^2)
> marks(Y) <- cbind(marks(Y), volume=vol)
```

The function `cut.ppp()` is a method for the generic function `cut()` briefly mentioned in Section 2.1.9. For a point pattern with real-valued marks, will divide the range of mark values into several discrete bands, yielding a point pattern with categorical marks:

```
> Y <- cut(longleaf, breaks=c(0,5, 20, Inf))
> Y
marked planar point pattern: 584 points
Multitype, with levels = (0,5], (5,20], (20,Inf]
window: rectangle = [0, 200] x [0, 200] metres
```

If `breaks` is a single number, it determines the number of bands:

```
> Y <- cut(longleaf, breaks=3)
> Y
marked planar point pattern: 584 points
Multitype, with levels = (1.93,26.6], (26.6,51.3], (51.3,76]
window: rectangle = [0, 200] x [0, 200] metres
```

Used in this context, `cut.ppp()` operates on the marks rather than the spatial coordinates.

If `W` is a window (class "owin"), then `cut(X, W)` will attach a logical-valued mark to each point, equal to TRUE if the point lies inside `W`, and FALSE otherwise. If `A` is a tessellation (class "tess", discussed in Section 4.4) then `cut(X, A)` will attach a factor-valued mark to each point, indicating which tile of the tessellation contains the point.

4.2.5 Converting to another unit of length

To convert the spatial coordinates to a different unit of length, use the function `rescale()`:

```
> Y <- rescale(X, s)
```

The spatial coordinates in the data set `X` will be re-expressed in terms of a new unit of length that is `s` times the current unit of length given in `X`.

The `spatstat` generic function `rescale()` is designed so that **the rescaled object is equivalent to the original**. For example if `X` is a data set giving coordinates in metres, then `rescale(X, 1000)` will divide the coordinate values by 1000 to obtain coordinates in kilometres, and the unit name will be changed from `metres` to `1000 metres`. The resulting object is equivalent to the original; the values have just been converted to another unit of measurement.

If the argument `unitname` is given, it will be taken as the new name of the unit of length. It should be a valid name for the unit of length, as described in Section 3.3.3. In the example above, `rescale(X, 1000, "km")` will divide the original coordinate values (in metres) by 1000 to obtain coordinate values in kilometres, and the unit name will be changed to `km` (rather than to `1000 metres`).

The unit name of a `spatstat` object may also be a multiple of a standard “base” unit. For example, the Lansing Woods trees were originally observed in a square of physical side length 924 feet, then the coordinates were scaled to a unit square, so the `lansing` data set in `spatstat` has `unitname(lansing) = 924` feet. For such “composite” units, the command `rescale(X)` with no further arguments will convert `X` to the base unit:

```
> rescale(lansing)
marked planar point pattern: 2251 points
Multitype, with levels =
[1] blackoak hickory maple misc redoak whiteoak
window: rectangle = [0, 924] x [0, 924] feet
```

Note that rescaling the point pattern has no impact upon the marks. Any alteration of the marks must be done “by hand”. For example, if your marks represent tree diameter in inches, and you want to change from inches to centimetres, then you could do something like `marks(X) <- marks(X)/2.54`.

Sometimes spatial data are stored or presented as a list of related spatial objects. If rescaling is applied to one of the objects in such a list then the *same* rescaling almost certainly should be applied to *all* of the objects in the list so as to maintain consistency. For example, to convert the `murchison` data from having units of metres to having units of kilometres one could do:

```
> murch2 <- lapply(murchison, rescale, s=1000, unitname="km")
> murch2 <- as.listof(murch2)
```

Since `murchison` was a list of class “`listof`”, we have used `as.listof()` to ensure that the transformed data also belong to this class.

4.2.6 Geometrical transformations

Many geometrical transformations of spatial data are supported in `spatstat`. Suppose that the window of your point pattern is a nice well-behaved rectangle, but its edges are skew-whiff with respect to the coordinate axes. It may be more convenient to have the edges parallel with the axes; this can be accomplished by rotating the point pattern object. Rotation and other commonly used geometrical transformations are listed in Table 4.6.

By default, rotation is performed about the *origin* in the spatial coordinate system. To rotate about another centre of rotation, you can specify the argument `centre`, which may be either a pair of coordinates, or a string indicating a special location. For example

```
> rotate(chorley, pi/2, centre="centroid")
```

applies a 90-degree anticlockwise rotation to the Chorley-Ribble data around the centroid of the window.

The command `scalardilate(X, f)` multiplies all the spatial coordinates of the object `X` by the factor `f` without changing any other information. It produces a dataset which is not equivalent to the original.

Table 4.7 lists some commonly used geometrical operations that only apply to windows (i.e. there is no method for “`ppp`” objects).

Further, `spatstat` supports the morphological window operations listed in Table 4.8.

4.2.7 Random perturbations of a point pattern

The `spatstat` package provides several functions for making random changes to a spatial point pattern.

<code>shift(X)</code>	translate (shift)
<code>rotate(X)</code>	rotate
<code>reflect(X)</code>	reflect about the origin
<code>flipxy(X)</code>	swap x and y coordinates
<code>scalardilate(X)</code>	expand or contract by a scale factor
<code>affine(X)</code>	general affine transformation
<code>convexhull(X)</code>	convex hull

Table 4.6: Common geometrical transformations for both point patterns and windows supported by `spatstat` (for point patterns the transformation applies to both the points and the observation window). I.e. the class of X can be either "`ppp`" or "`owin`".

<code>intersect.owin()</code>	intersection of two windows
<code>union.owin()</code>	union of two windows
<code>setminus.owin()</code>	set difference of two windows
<code>complement.owin()</code>	swap inside and outside
<code>inside.owin()</code>	determine whether a point is inside a window
<code>is.subset.owin()</code>	determine whether one window contains another
<code>is.convex()</code>	test whether a window is convex
<code>centroid.owin()</code>	compute centroid (centre of mass) of window
<code>incircle()</code>	find largest circle inside window
<code>trim.rectangle()</code>	Cut off side(s) of a rectangle

Table 4.7: Geometrical operations on windows (objects of class "`owin`").

The function `rjitter()` displaces each point of the pattern by a small random distance in a random direction, independently of other points. If the original dataset contained some duplicate points, then jittering will separate these duplicates, making them visible in a plot of the data. If jittering the data causes a drastic change in the results of analysis, then this shows that the results were highly sensitive to the precise locations.

The function `rshift()` applies the *same* random shift to every point in a pattern, or to all points in a specified subset. In a multitype point pattern, all points of *the same type* will be subjected to the same shift, while the shifts applied to different types of points are independent. This can be used in a Monte Carlo test of the hypothesis that the different types of points are independent; see Chapter 10.

The function `rlabel()` randomly assigns new mark values to the points in a pattern, by randomly permuting or resampling the original marks. This can be used in a Monte Carlo test of the hypothesis that the marks are independent of the points.

The function `quadratresample()` performs a block resampling procedure in which the win-

<code>dilation.owin()</code>	morphological dilation
<code>erosion.owin()</code>	morphological erosion
<code>opening.owin()</code>	morphological opening
<code>closing.owin()</code>	morphological closing
<code>border()</code>	create a border region around a window

Table 4.8: Morphological operations on windows.

dow of the point pattern is divided into rectangles, and these rectangles are randomly resampled to generate a new point pattern.

The function `rthin()` randomly deletes some of the points in a point pattern, according to specified probability rules.

4.2.8 Generating random point patterns

Apart from randomly modifying an existing point pattern as described above it can also be useful to randomly generate entire patterns from scratch. Table 4.9 contains a list of useful `spatstat` functions for generating random point patterns. They all (except `simulate.ppm()`) start with a lower case “r”, which is the R convention for random generators.

4.2.9 Splitting and combining point patterns

It is sometimes useful to split a point pattern data set into several sub-patterns, and perform calculations separately on each sub-pattern.

4.2.9.1 Splitting a point pattern into sub-patterns

The R generic function `split()` is used to divide data into subsets: `split(x, f)` divides the dataset `x` into subsets determined by the grouping `f`.

In `spatstat` `split.ppp()` enables the user to divide a point pattern `x` into sub-patterns using a variety of criteria.

If `f` is a factor, of length equal to the number of points in `x`, then `split(x, f)` separates the data points into groups according to the values of `f`. The result is a list of point patterns, each with the same window as `x`. The names of the list entries are the levels of `f`.

If `f` is a factor-valued image, then `split(x, f)` splits the `window` of `x` into sub-windows according to the pixel values of `f`. The result is a list of point patterns obtained by restricting `x` to each of these sub-windows.

If `f` is a tessellation, then `split(x, f)` causes the window of `x` to be sub-divided by the tiles of the tessellation. The result is a list of point patterns obtained by intersecting `x` with the tiles of the tessellation.

For a multitype point pattern `x`, the factor `f` defaults to `marks(x)`, so that `split(x)` separates the data into the patterns of points of each type. The resulting point patterns all have the same window as `x`.

The result of `split.ppp()` also belongs to the classes “listof” and “splitppp” so

<code>runifpoint</code>	generate n independent uniform random points
<code>rpoint</code>	generate n independent random points
<code>rmppoint</code>	generate n independent multitype random points
<code>rpoispp</code>	simulate the (in)homogeneous Poisson point process
<code>rmipoispp</code>	simulate the (in)homogeneous multitype Poisson point process
<code>runifdisc</code>	generate n independent uniform random points in disc
<code>rstrat</code>	stratified random sample of points
<code>rsyst</code>	systematic random sample of points
<code>rMaternI</code>	simulate the Matérn Model I inhibition process
<code>rMaternII</code>	simulate the Matérn Model II inhibition process
<code>rSSI</code>	simulate Simple Sequential Inhibition process
<code>rStrauss</code>	simulate Strauss process (perfect simulation)
<code>rHardcore</code>	simulate Hard Core process (perfect simulation)
<code>rDiggleGratton</code>	simulate Diggle-Gratton process (perfect simulation)
<code>rDGS</code>	simulate Diggle-Gates-Stibbard process (perfect simulation)
<code>rNeymanScott</code>	simulate a general Neyman-Scott process
<code>rPoissonCluster</code>	simulate a general Poisson cluster process
<code>rMatClust</code>	simulate the Matérn Cluster process
<code>rThomas</code>	simulate the Thomas process
<code>rGaussPoisson</code>	simulate the Gauss-Poisson cluster process
<code>rCauchy</code>	simulate Neyman-Scott Cauchy cluster process
<code>rVarGamma</code>	simulate Neyman-Scott Variance Gamma cluster process
<code>rcell</code>	simulate the Baddeley-Silverman cell process
<code>rmh</code>	simulate Gibbs point process using Metropolis-Hastings
<code>simulate.ppm</code>	simulate Gibbs point process using Metropolis-Hastings
<code>runifpointOnLines</code>	generate n random points along specified line segments
<code>rpoisppOnLines</code>	generate Poisson random points along specified line segments

Table 4.9: Random point pattern generators in **spatstat**.

that it can easily be printed and plotted. Figure 4.17 shows the result of the command `plot(split(amacrine))`.

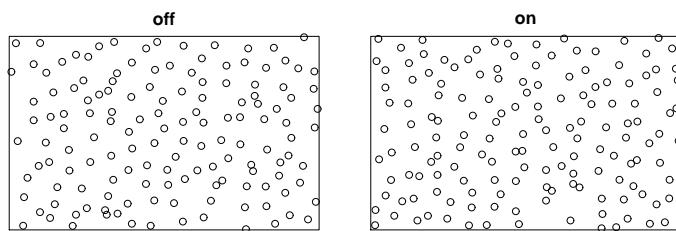


Figure 4.17: The result of splitting the multitype point pattern `amacrine` into sub-patterns of points of each type, generated by `split(amacrine)`.

You can use the R command `lapply()` to perform any desired operation on each element of

the list. For example, to apply adaptive estimation of intensity (see Section ??) separately to each species of tree in the Lansing Woods data, do:

```
> V <- split(lansing)
> A <- lapply(V, adaptive.density)
> plot(as.listof(A))
```

A neater way to operate on sub-patterns is to use `by.ppp()`, a method for the R function `by()`. The call `by(X, INDICES=f, FUN=g)` is essentially equivalent to `lapply(split(X,f), g)`. The result above can be produced more briefly:

```
> A <- by(lansing, FUN=adaptive.density)
> plot(A)
```

Tip: Don't give your data objects names which consist entirely of capital letters. Names of this sort tend to be used as argument names in R functions like `lapply()` and `by()`. Errors can result if an object has a name that conflicts with such an argument name.

The individual patterns which form the components of the list returned by `split.ppp()` can be extracted and saved as individual patterns instead of list components. For example,

```
> S <- split(chorley)
> cases <- S[["larynx"]]
> controls <- S[["lung"]]
```

Note that if the marks of X consist of a single factor, then by default `split(X)` splits X into *unmarked* subpatterns, each corresponding to a level of these marks. In the foregoing example, `cases` and `controls` are unmarked point patterns. See `help(split.ppp)` for options to control the handling of marks.

4.2.9.2 Un-splitting

There is an R generic function `split<-()` which undoes the splitting operation, in the sense that it recombines the data, replacing each entry in its original position. A method "`split<-.ppp`" is provided in `spatstat`. This can be useful when we want to apply a transformation separately to each subset of the data. For example, in the `ants` data, suppose we want to apply jittering to the `Messor` nests only. This can be done simply by

```
> X <- ants
> u <- split(X)
> u$Messor <- rjitter(u$Messor)
> split(X) <- u
```

The resulting pattern `X` is identical to `ants` except that the points with type `Messor` have been slightly displaced. (Note we have avoided over-writing the standard dataset `ants`.) The use of "`split<-`" is delicate because the transformed data must be compatible with the original data: for example the number of data points in each sub-pattern must not be changed.

4.2.9.3 Combining point patterns

The `spatstat` function `superimpose()` combines any number of point patterns into a single pattern. Continuing the example above we could try to do the un-splitting by

```
> X <- superimpose(u$Messor, u$Cataglyphis)
```

However, the resulting pattern is unmarked. To keep track of which point came from which pattern, use argument names for the point patterns in the call to `superimpose()`. The names will then become marks for the combined pattern (in addition to any existing marks).

```
> X <- superimpose(Cataglyphis=u$Cataglyphis, Messor=u$Messor)
```

In the combined pattern the points are stored in the order in which the arguments were given. This means that in the pattern `X` created above the Messor points are listed last while they are listed first in the original `ants` pattern. Thus, a bit of caution is needed (the problem would of course be avoided in this case by reversing the arguments to `superimpose()` above).

One advantage of `superimpose()` is that it also handles the case when the number of points is changed. So if we randomly thin out the (jittered) Messor nests with a retention probability of 0.5 we can still combine the patterns:

```
> u$Messor <- rthin(u$Messor, 0.5)
> X <- superimpose(Messor=u$Messor, Cataglyphis=u$Cataglyphis)
```

The window for the combined pattern is taken to be the union of the windows of the patterns being superimposed, unless given explicitly by the argument `W`. Thus combining a (random) pattern in the square $[0, 2] \times [0, 2]$ with a (random) pattern in the square $[1, 3] \times [1, 3]$ gives a (random) pattern in a polygonal region (the union of the two overlapping squares).

```
> X <- runifpoint(50, square(c(0,2)))
> Y <- runifpoint(50, square(c(1,3)))
> superimpose(X,Y)
planar point pattern: 100 points
window: polygonal boundary
enclosing rectangle: [0, 3] x [0, 3] units
```

Unless we specify another region (e.g. the bounding square $[0, 3] \times [0, 3]$):

```
> superimpose(X, Y, W=square(3))
planar point pattern: 100 points
window: rectangle = [0, 3] x [0, 3] units
```

4.2.10 Basic summaries of point patterns and windows

Table 4.10 lists commands that are commonly used to obtain basic summary information about a point pattern. These are all generic functions so it is the method for class "ppp" that should be consulted for further information.

The R generic function `print()` can be invoked by typing `print(X)` or just `X`, the name of the object. For simple data types, this command causes all the data to be printed, but for datasets belonging to a class, the `print()` method displays only minimal information, for the sake of efficiency. The method `print.ppp()` displays the number of points, the type of marks (if any), and a brief description of the window (this window information is also printed by `print.owin()` if one types `Window(X)`).

The R generic function `summary()` yields a basic statistical summary of the data, typically including the mean and range of each variable. The method `summary.ppp()` displays detailed information about the density of points, the distribution of the mark values, and the geometry of the window (again the summary of the window geometry can be obtained by `summary.owin()` if one types `summary(Window(X))`).

print(X)	Print basic information
X	Print basic information
summary(X)	Print detailed summary
npoints(X)	Number of points
coords(X)	Extract spatial coordinates
intensity(X)	Average density of points per unit area
density(X)	Spatially-varying density of points
istat(X)	Interactive data analysis
pairdist(X)	Distances between all pairs of points
nndist(X)	Distances to nearest neighbours
nnwhich(X)	Identify nearest neighbours
distmap(X)	Distance from each pixel to nearest data point

Table 4.10: Basic summaries of a point pattern X.

```

> summary(chorley)
Marked planar point pattern: 1036 points
Average intensity 3.29 points per square km

*Pattern contains duplicated points*

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

Multitype:
      frequency proportion intensity
larynx        58       0.056       0.184
lung         978       0.944       3.100

Window: polygonal boundary
single connected closed polygon with 131 vertices
enclosing rectangle: [343.45, 366.45] x [410.41, 431.79] km
Window area = 315.155 square km
Unit of length: 1 km

```

A special trick used by some methods for `summary()` (including `summary.ppp()` and `summary.owin()`) is that, instead of printing the summary information, the `summary()` method only computes the summary information and returns it in a list, belonging to a special class. If the user types `summary(X)` then the `print()` method for this special class is then invoked, and this generates the printed output. For a point pattern, the result of `summary(X)` is an object of the special class "summary.ppp" and the print method for this class is `print.summary.ppp()`. Confused yet? The advantage is that if the user types `A <- summary(X)` then the summary information is stored in `A` for examination and re-use.

```

> A <- summary(chorley)
> names(A)

```

```
[1] "is.marked"      "n"           "window"          "intensity"
[5] "nduplicated"    "rounding"     "multiple.marks" "is.numeric"
[9] "marknames"      "marktype"    "is.multitype"   "marks"
> B <- summary(Window(chorley))
> names(B)
[1] "xrange"        "yrange"       "type"          "area"         "units"
[6] "npoly"         "areas"        "nvertices"    "nhole"
```

The command `intensity(X)` computes the average number of points per unit area, while `density(X)` computes a pixel image giving a spatially-varying density of points per unit area; these are discussed in Chapter 6.

The command `istat()` starts an interactive interface in which the user can select different summary operations to be applied to the data. A screenshot of the interface is shown in Figure 4.18.

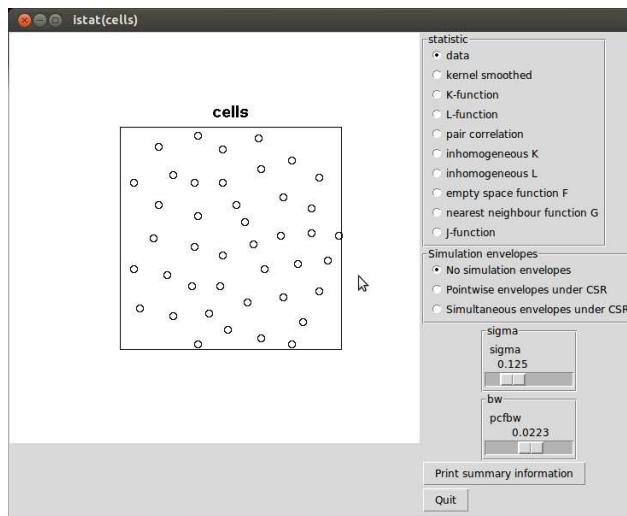


Figure 4.18: Screenshot of the interactive graphics window started by `istat()`.

The command `pairdist(X)` returns a matrix containing the distances between all pairs of points in `X`. The command `nndist(X)` returns a numeric vector with one entry for each point of `X`, giving the distance from this point to the nearest neighbour (the nearest other point in `X`). Related to this, `nnwhich(X)` returns an integer vector giving for each point of `X` the index of the nearest neighbour. For example if `m <- nnwhich(X)` and `m[3] = 5`, then the nearest neighbour of the point `X[3]` is the point `X[5]`. A useful graphic related to `nndist()` is the *Stienen diagram* (see Section).

Table 4.11 lists some additional commands useful for summarising the marks in a point pattern. If the marks are categorical, then `table(marks(X))` or `barplot(table(marks(X)))` are useful summaries of the distribution of marks. If the marks are numeric, then `hist(marks(X))` or `plot(ecdf(marks(X)))` or `plot(density(marks(X)))` are useful summaries.

For numeric marks, `Smooth(X)` or `markmean(X)` will produce an image showing the spatially-varying average (Nadaraya-Watson smoother) of the mark value, and `markvar(X)` produces an image of the spatially-varying variance of the marks. These are discussed in Chapter 6.

The functions `marktable()`, `markstat()` and `applynbd()` perform calculations on the points and marks in a “neighbourhood” of each data point, where the “neighbourhood” of a point can be defined as the `N` nearest neighbours, or all data points within a distance `R`.

Table 4.12 lists the basic `spatstat` commands for calculating common summaries of a window such as the area or perimeter.

<code>marks(X)</code>	Extract marks
<code>Smooth(X)</code>	Spatially-varying average mark
<code>markmean(X)</code>	Spatially-varying average mark
<code>markvar(X)</code>	Spatially-varying variance of marks
<code>marktable(X, ...)</code>	Tabulate marks of neighbours
<code>markstat(X, ...)</code>	Summarise marks of neighbours
<code>applynbd(X, ...)</code>	Apply operation on each neighbourhood

Table 4.11: Basic summaries of the marks of a point pattern.

<code>area()</code>	compute window's area
<code>diameter()</code>	compute window's diameter
<code>perimeter()</code>	compute window's perimeter length
<code>deltametric()</code>	measure discrepancy between two windows
<code>dilated.areas()</code>	compute areas of dilated windows
<code>eroded.areas()</code>	compute areas of eroded windows
<code>distmap.owin()</code>	distance transform image
<code>distfun.owin()</code>	distance transform function

Table 4.12: Basic summaries of a window.

See `help(spatstat)` for an aide-memoire list of operations that can be applied to or carried out on windows or point pattern objects.

4.2.11 Examples

We will now illustrate some of the ideas that were discussed in the previous material by applying them to an example data set. We will use one of the point pattern data sets that is installed with the `spatstat` package, namely the NZ trees data set `nztrees`. This represents the positions of 86 trees in a forest plot measuring 153 by 95 feet. The data are plotted in the left panel of Figure 4.19.

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

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

We have chosen a value of 10 for the “bandwidth” (smoothing parameter). The issue of the choice of smoothing parameter will be discussed elsewhere. (See section 6.5.1.) We have also chosen to display the result as a contour plot. The result is shown in the right panel of Figure 4.19.

A striking feature of the intensity estimate is that it has a steep slope at the top right-hand corner

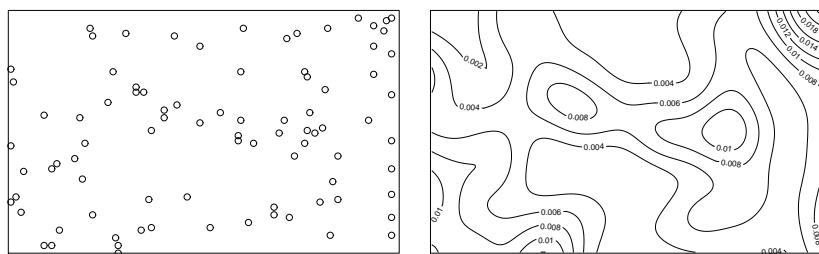


Figure 4.19: *Left:* New Zealand trees data. *Right:* contour plot of kernel estimate of intensity.

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 feature of the data sticks out like a sore thumb if we plot a histogram of the x coordinates of the trees, `hist(coords(nztrees)$x)`, shown in Figure 4.20

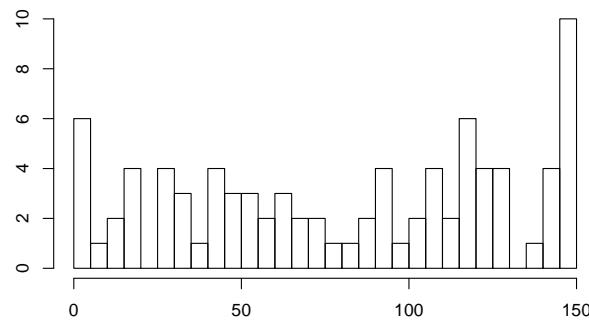


Figure 4.20: Histogram of x coordinates of New Zealand trees.

Consequently we might want to exclude the right-hand boundary from the study region, so as 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 <- Window(nztrees)
> chopped <- trim.rectangle(win, xmargin=c(0,5), ymargin=0)
> chopped
window: rectangle = [0, 148] x [0, 95] feet
```

Of course `chopped` is not a point pattern, but rather an `owin` object which is of type `rectangle` and hence simply a rectangle in the plane. We can however get the point pattern that we want by using the subset operator "`[.ppp`", to 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
```

```
Coordinates are integers
i.e. rounded to the nearest 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)
```

The results are shown in Figure 4.21.

Removing the right margin seems to have produced a much more uniform pattern, although there is still a suggestion of variation in forest density.

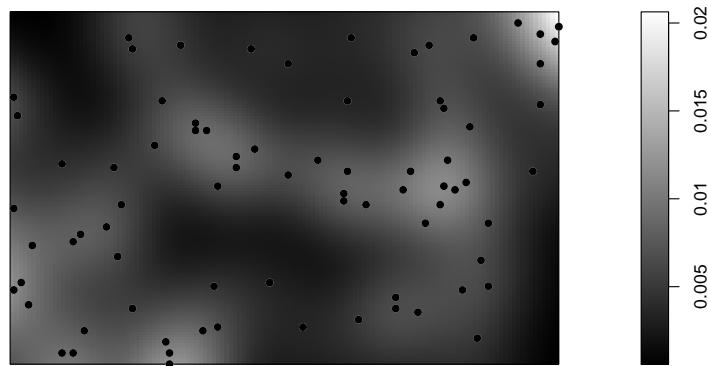


Figure 4.21: New Zealand trees pattern with right-hand border removed, and kernel estimate of intensity.

4.3 Exploring images

Pixel images were introduced in Section 3.5. This section explains how to plot pixel images in various ways, extract subsets of a pixel image, and manipulate image data.

4.3.1 Summarising an image

For basic information about an image Z , use the following:

Z	print basic information
print(Z)	print basic information
summary(Z)	print detailed information
dim(Z)	pixel raster dimensions (y, x)
nrow(Z)	number of rows (y coordinate)
ncol(Z)	number of columns (x coordinate)
range(Z)	range of pixel values
max(Z)	maximum of pixel values
min(Z)	minimum of pixel values
mean(Z)	mean of pixel values
median(Z)	median of pixel values
quantile(Z, ...)	quantiles of pixel values
sum(Z)	sum of pixel values
integral.im(Z)	sum of pixel values times pixel area

To compute other numerical summaries of pixel values that are not on this list, see below.

4.3.2 Extracting pixel values and subimages

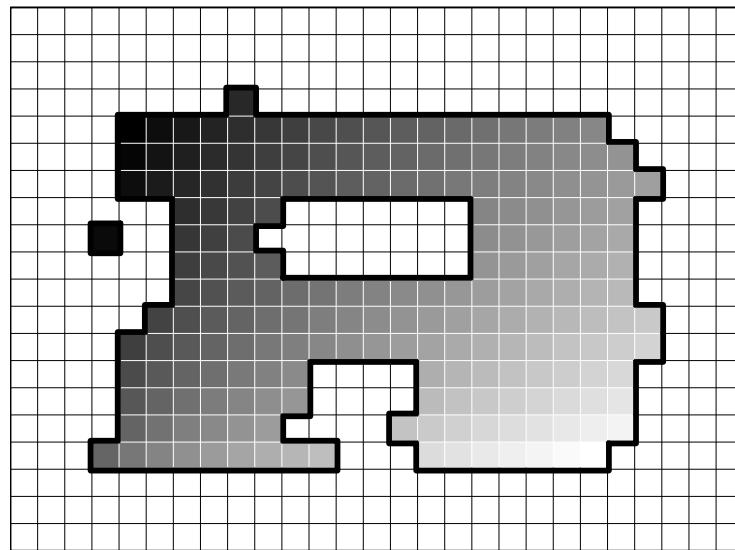


Figure 4.22: Anatomy of a pixel image in `spatstat`.

Figure 4.22 shows the anatomy of a pixel image X in `spatstat`. Small squares represent pixels. The spatial coordinates of a pixel are the coordinates of the centre of the square. The outermost rectangle is `Frame(X)`. Every pixel inside the frame has a pixel value. White squares represent pixels with value `NA`, indicating that the pixel lies outside the observation window. Thick black polygons delineate `Window(X)`.

If you just want to get your hands on the pixel values of an image, use `as.matrix.im()`, a method for the generic `as.matrix()`. The resulting matrix contains the pixel values for all pixels in the frame, arranged according to the idiosyncratic convention of `spatstat`: columns correspond to increasing values of the x coordinate, and rows to increasing values of the y coordinate. If you need to refer to the spatial locations of the pixels, use the functions `raster.x()` and `raster.y()`, which make matrices containing the corresponding pixel coordinates. Note that the

result of `as.matrix.im()` may contain NA values, which come from pixels lying outside the window where the image is defined.

Subsets of the pixel values can be extracted using the indexing operator `[`. Two common variants of the syntax are:

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

where the subset to be extracted is determined by the index argument S. This argument can take many different forms, which is what makes this subset operator method so powerful and which makes it worthwhile to learn its syntax.

- 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 (see below), and a numeric vector otherwise.
- 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 NA values (corresponding to pixels outside the observation window) should be omitted (`drop=TRUE`) or retained (`drop=FALSE`).

If P is a point pattern, `X[P]` is a vector with one entry for each point in P, but may contain NA values, while `X[P, drop=TRUE]` is generally a shorter vector.

If S is a window or logical-valued image, `X[S]` can be thought of as the restriction of the image X to the region S. It is obtained by setting all pixels outside S to have the value NA. This changes the Window, but not the Frame or the pixel coordinates.

The argument `tight` determines whether the original image frame will be retained (`tight=FALSE`) or replaced by the smallest possible rectangle (`tight=TRUE`). The same effect can be achieved using an idiom like

```
> Y <- X[S]
> Frame(Y) <- boundingbox(Y)
```

If S is a window or logical-valued image, `X[S, drop=TRUE]` consists only of pixel values inside the region S. If S is a non-rectangular window then the result must be a vector containing the values for the selected pixels. However if S is a rectangle, the result *could* be returned as a pixel image. The logical argument `rescue` determines whether this will happen: the default is `rescue=TRUE`. Thus, if R is a rectangle, `X[R]` and `X[R, drop=TRUE]` both return the image which is the restriction of X to the region R, but `X[R, drop=TRUE, rescue=FALSE]` yields the vector of pixel values inside this rectangle.

Like many methods for `[`, the method `[.im` can also be called without an index, in the forms `X[,]` and `X[]`. The form `X[,]` is equivalent to `as.matrix(X)`: all the pixel values are returned, and the matrix structure is preserved. The form `X[]` generally means “all relevant values”; for an image X the expression `X[]` represents all *defined* pixel values, that is, all pixel values of X which are not NA, returned as a vector.

The most common uses of `[.im` are summarised in Table 4.13. See `help("[.im")` for full details.

A very handy technique is to use the subset index operator to look up the value of a pixel image at a single point:

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

COMMAND	FORMAT	DESCRIPTION	NA's?
<code>as.matrix(X)</code>	matrix	all pixel values	Yes
<code>X[,]</code>	matrix	all pixel values	Yes
<code>X[]</code>	vector	all defined pixel values	No
<code>X[P]</code>	vector	values at all points of P	Yes
<code>X[P, drop=TRUE]</code>	vector	values defined at points of P	No
<code>X[W]</code>	image	image restricted to W	Yes
<code>X[W, drop=TRUE]</code>	vector	pixel values inside W	No
<code>X[R]</code>	image	image restricted to R	No
<code>X[R, drop=TRUE, rescue=FALSE]</code>	vector	pixel values inside R	No

Table 4.13: Typical commands for extracting subsets of a pixel image X indexed by a point pattern P, a non-rectangular window W, or a rectangular window R.

```
[1] 147.08
```

As we have previously noted (page ??), this sort of subsetting can even be performed interactively, using the R function `locator()` to click on a point in the window:

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

To display a subregion of an image we could the index argument to be a window object defining the subregion:

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

The subset can also be chosen interactively:

```
> plot(elev)
> S <- clickpoly(add=TRUE)
> plot(elev[S, drop=FALSE, tight=TRUE])
```

The result is shown in Figure 4.23.

4.3.3 Calculations with pixel values

Summarising pixel values

As we indicated earlier Z [] simply returns all of the (defined) pixel values of Z, which you can then do as you like with. For instance you could apply standard statistical summaries such as `mean()`, `sd()`, `median()` and so on. You can reproduce the effect of applying `hist.im()` by using `hist(Z [])` to create a histogram of the pixel values. (In fact the function `hist.im()` is just a slight tweak of this construction.) Similarly `plot(density(Z []))` can be used to produce a kernel estimate of the probability density function of the pixel values.



Figure 4.23: Interactive selection of a subset of an image. *Left:* Full image, with polygon selected using `clickpoly()`. *Right:* Selected subset of image.

Pixelwise calculation

The function `eval.im()` performs pixel-by-pixel calculations on an image, or on several images, and returns a new image.

For instance if `Z` is an image with numerical values, we can add 10 to each pixel value by typing

```
> Y <- eval.im(Z + 10)
```

The result is a new image `Y`, where the pixel value is equal to 10 plus the corresponding pixel value in `Z`. If `A` and `B` are images, we can add their corresponding pixel values by

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

The argument of `eval.im()` can be any R language expression in which the ‘variables’ are the names of pixel images. It may also involve the names of constants and functions.

```
> eval.im(sqrt(Z))
> eval.im(sin(pi * Z))
> eval.im(Z > 3)
```

The expression can involve several images:

```
> eval.im(log(X) + Y - 3)
```

The main restriction is that the expression must operate ‘pixel-by-pixel’, in the sense that the output has as many pixel values as the input:

```
> W <- eval.im(max(0,Z)) # Throws an error.
> W <- eval.im(pmax(0,Z)) # Works.
```

The first expression throws an error because it gives a single value, whereas the second expression gives the same number of pixel values as are contained in `Z`.

```
> eval.im(if(X < 3) 3 else 1) ## Throws an error.
> eval.im(ifelse(X < Y, 3, 1)) ## Works.
```

The expression must handle NA pixel values. This is automatically true for most expressions in R, but not for some standard functions such as `mean()`. To centre the pixel values around their mean,

```
> eval.im(X - mean(X, na.rm=TRUE))
```

If an expression produces infinite or NaN ('not a number') values, these will be converted to NA. For example we can safely take the logarithm

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

of any numeric-valued image Z. If any of the pixel values of Z are zero or negative, a warning is issued, and the relevant pixels of logZ get the pixel value NA.

The pixel grids of the image objects are permitted to differ. "Resampling" of the pixel grids is effected if necessary to produce a common pixellation over which the arithmetic is carried out.

The expression passed to eval.im() must involve the *names* of pixel images. The following will fail, because cells is a point pattern, not an image:

```
> eval.im(density(cells) - 3) ## Throws an error
```

The argument envir can be used to supply pixel images which need to be evaluated:

```
> eval.im(Z - 3, list(Z=density(cells)))
```

It was suggested in section 4.1.3 that after inspecting hist(Z) one might wish to consider appropriate transformations of the pixel values. These transformations can be effected using eval.im(). We have already referred to the possibility of doing a logarithm transformation. Similarly a Box-Cox transformation can be carried out using Y <- eval.im((Z^pwr-1)/pwr) for some chosen value of pwr.

Another very useful transformation is the probability integral transformation (aka "histogram equalisation") which can be effected using the ecdf() function:

```
> g <- ecdf(Z)
> Y <- eval.im(g(Z))
```

Plots of the resulting image Y often reveal better detail than do plots of the original. In interpreting these plots remember that, e.g., a value of 0.5 means that the corresponding pixel value of Z was the median (0.5 or 50% quantile) of the pixel values of Z.

4.3.4 Manipulating images

Table 4.14 lists commands which change the spatial coordinates of the pixels but do not alter the pixel values. These are all generic functions defined in spatstat, with methods for class "im".

Table 4.15 shows commands which operate on the pixel values of an image Z. In the expression Z[P], the argument P can be a point pattern or a list(x,y). The result is the vector of pixel values at the pixels *nearest to* the specified points. The alternative interp.im(Z,P) performs bilinear interpolation from the nearest pixels onto the specified points.

The use of methods for the generic function cut() to effect discretization of numerical values was discussed in section 4.2.4. There is a method for cut() the "im" class of objects. For example, given an image Z with numeric 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. This may serve to filter out extraneous detail that appears when the raw numeric values are plotted and thus reveal interesting features which would otherwise be obscured by the blur of the detail.

The method split.im() enables the user to divide a pixel image into sub-images using various criteria. The options are similar to those for split.ppp() discussed in Section 4.2.9.1. The method by.im() is more elegant for some purposes: the call by(X, INDICES=f, FUN=g) is essentially equivalent to lapply(split(X,f), g).

The command levelset(Z, h) or levelset(Z, h, "<=") finds the region where the

<code>rescale(Z, s)</code>	convert to different unit of length
<code>scalardilate(Z, f)</code>	multiply coordinates by f
<code>shift(Z, v)</code>	shift pixel grid
<code>rotate(Z, a)</code>	rotate pixel grid
<code>reflect(Z, f)</code>	reflect pixel grid about origin
<code>flipxy(Z, f)</code>	swap x and y coordinates
<code>affine(Z, m, v)</code>	affine transformation of coordinates

Table 4.14: Commands which change the spatial coordinates of an image Z but do not affect the pixel values.

<code>Z[P]</code>	pixel value nearest to points
<code>interp.im(Z, P)</code>	interpolate to points
<code>scaletointerval(Z, lo, hi)</code>	scale pixel values to range
<code>cut(Z, ...)</code>	divide image into sub-images
<code>split(Z, f)</code>	divide image into sub-images
<code>by(Z, f, fun)</code>	apply <code>fun()</code> to subsets
<code>levelset(Z, h)</code>	region where $Z \leq h$
<code>solutionset(sin(Z) > 0)</code>	region where statement is true

Table 4.15: Commands for manipulating pixel values of an image Z.

pixel value is less than or equal to h, and returns this region as a window. The command `solutionset(sin(Z) > 0)` uses `eval.im()` to evaluate the expression in parentheses; this should result in an image with logical values. The region where the value is TRUE is returned as a window.

An example of the use of the `levelset()` function can be based on the data set `bei.extra$elev` which is a pixel image containing altitude (elevation) values for a study region. If we wished to restrict the studying region where altitude exceeds 145 we could form a window consisting of those pixels in the image whose values satisfy this constraint. This can be done in the following manner:

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

More complicated constraints can be accommodated by the `solutionset()` function. The data set `bei.extra$grad` accompanying the point pattern `bei` is a pixel image of the slope (gradient) of the terrain. To restrict the study area to those locations where altitude is below 140 and slope exceeds 0.1, we can form a window as follows:

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

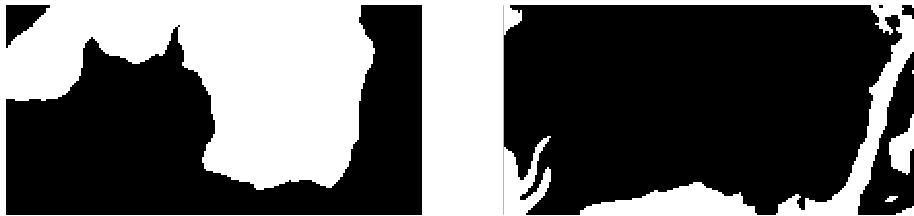


Figure 4.24: Binary masks W and V created in the text.

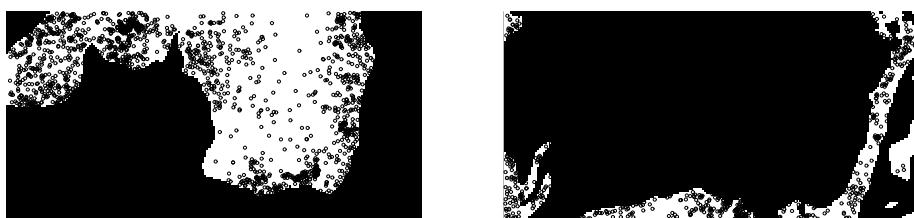


Figure 4.25: The point pattern `bei` in the restricted study regions.

4.3.4.1 Functions that return a pixel image

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

<code>as.im</code>	converts other data to a pixel image
<code>density.ppp</code>	kernel smoothing of point pattern
<code>density.psp</code>	kernel smoothing of line segment pattern
<code>pixellate.ppp</code>	approximate point pattern by pixel image
<code>pixellate.psp</code>	approximate line segment pattern by pixel image
<code>pixellate.owin</code>	approximate window by pixel image
<code>distmap.owin</code>	distance function of window
<code>distmap.ppp</code>	distance function of point pattern
<code>distmap.psp</code>	distance function of line segment pattern
<code>setcov</code>	geometric covariance function of a window
<code>connected</code>	identify connected components of a window
<code>predict.ppm</code>	fitted intensity of a point process model
<code>[.im</code>	subset of an image (or look up pixel values)
<code>shift.im</code>	vector shift of image domain
<code>rescale.im</code>	rescaling of image domain
<code>eval.im</code>	evaluate any expression involving images
<code>cut.im</code>	convert numeric image to factor image
<code>split.im</code>	divide pixel image into sub-images
<code>by.im</code>	apply function to subsets of pixel image
<code>interp.im</code>	spatial interpolation of image
<code>blur</code>	spatial blurring and extrapolation of image
<code>Smooth</code>	spatial blurring and extrapolation of image

4.4 Tessellations

A *tessellation* is a division of a window into non-overlapping regions. These regions are called “tiles” (Latin *tessera*) although their shape is arbitrary, so they may not look anything like ceramic tiles.

Tessellations are often used to represent administrative or political divisions, e.g. the subdivision of a country into states or provinces. They can be completely artificial, e.g. the rectangular quadrats which we use in quadrat counting. Tessellations can be observational data, e.g. the classification of a survey region into different rock types. A tessellation can be computed from data, e.g. the Dirichlet tessellation defined by a set of points.

Note that the tiles of a tessellation must be non-overlapping. The union of the tiles is the window associated with the tessellation: it is this window which is divided up by the tiles. Note that this window could be irregular or disconnected so that there could be apparent “gaps” between the tiles, but these are not part of the window.

Tessellations have several uses in `spatstat`. A good way to conduct an initial exploration and rudimentary analysis of a spatial object is to cut that object up into pieces, calculate particular quantities for each piece and then examine in some way the relationship amongst these calculated quantities. Cutting an object up into pieces is effectively a “tessellation”.

4.4.1 Creating a tessellation

In `spatstat` tessellations are represented as objects of class "`tess`". Currently `spatstat` supports three kinds of tessellations:

- **rectangular tessellations** in which the tiles are rectangles with sides parallel to the coordinate axes;
- **tile lists**, tessellations consisting of a list of windows, usually polygonal windows;
- **pixellated tessellations**, in which space is divided into pixels and each tile occupies a subset of the pixel grid.

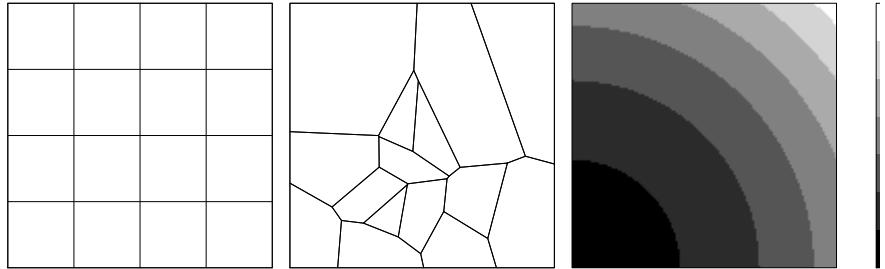


Figure 4.26: Types of tessellations. *Left*: rectangular. *Middle*: tile list. *Right*: pixellated.

All three types of tessellation can be created by the command `tess()`. To create a rectangular tessellation do:

```
> tess(xgrid=xg, ygrid=yg)
```

where `xg` and `yg` are vectors of coordinates of vertical and horizontal lines determining a grid of rectangles. Alternatively, if you want to subdivide a window `W` (usually but not necessarily rectangular) by rectangles of equal size, you can use `quadrats()`. Type:

```
> quadrats(W, nx, ny)
```

where `nx`, `ny` are the numbers of rectangles in the *x* and *y* directions, respectively. A common use of this command is to create quadrats for a quadrat-counting method.

To create a tessellation from a list of windows,

```
> tess(tiles=z)
```

where `z` is a list of objects of class "owin". The windows should not be overlapping; currently `spatstat` does not check this. This command is commonly used when the study region is divided into administrative regions (states, départements, postcode regions, counties) and the boundaries of each sub-region are provided by GIS data files. The window that is tessellated by the list `z` is by default the union of the tiles, i.e. of the windows which constitute the components of `z`.

By default, the window of the tessellation will be computed as the union of the tiles. For efficiency, the `tess()` function has an argument `window` which specifies the window. The `tess()` function does not check whether this argument equals the union of the tiles; if it does not then the data will be internally consistent.

To create a tessellation from a pixel image do:

```
> tess(image=Z)
```

where `Z` is a pixel image with factor values. Each level of the factor represents a different tile of the tessellation. The pixels that have a particular value of the factor constitute a tile. This command is often used to separate the landcover types in a landcover image (a pixel image in which each pixel is labelled by the type of vegetation or land use at that location) into different regions. The tiles produced will rarely if ever look *anything* like what one would normally think of as being tiles. In particular individual tiles can (often will) be topologically disconnected.

The function `as.tess()` can also be used to convert other types of data to a tessellation.

4.4.2 Computed tessellations

There are two commands which directly compute tessellations from point patterns.

The command `dirichlet(X)` computes the *Dirichlet* or *Voronoi* (or if you insist, *Thiessen*) tessellation of the point pattern `X`. The tile associated with a given point of the pattern `X` is the region of space which is closer to that point than to any other point of `X`. The Dirichlet tiles are polygons. The command `dirichlet(X)` computes these polygons and intersects them with the window of `X`.

The command `delaunay(X)` computes the *Delaunay triangulation* of the point pattern `X`. The object returned is a tessellation (i.e. is of class "tess" but is not a tessellation of the window of `X`). It is usually conceptualised as a network or graph defined in terms of the Dirichlet tessellation of `X`, formed by joining some of the points of `X` by straight lines. Two points of `X` are joined if their Dirichlet tiles share a common edge. The resulting network forms a set of non-overlapping triangles. These triangles cover the *convex hull* of `X` rather than the entire window of `X`. Thus a Delaunay triangulation constitutes a tessellation of the convex hull of `X`. However this is not how such a triangulation is usually thought of.

4.4.3 Operations involving a tessellation

Tessellations are spatial objects and hence one often wishes to plot them. There is a "tess" method for `plot()`. There is also a method for `print()` which provides the essential information about a

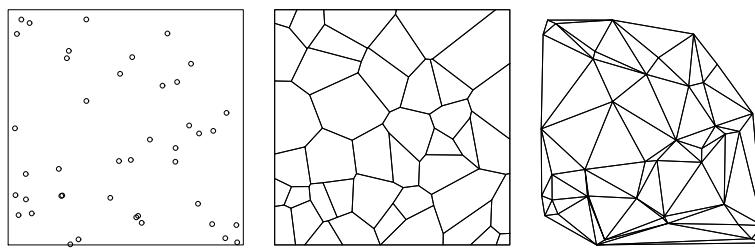


Figure 4.27: A point pattern X , its Dirichlet tessellation $\text{dirichlet}(X)$, and its Delaunay triangulation $\text{delaunay}(X)$.

"tess" object without making a mess all over your computer screen. In addition there is a method for "[" for subsetting tessellations. This is used to select out a subset of the tiles which make up the tessellation in question.

Use the function `tiles()` to extract a list of the tiles in a tessellation. The result is a list of windows ("owin" objects). This can be handy if, for example, you want to compute some characteristic of the tiles in a tessellation, such as their areas or diameters:

```
> X <- runifpoint(10)
> V <- dirichlet(X)
> U <- tiles(V)
> unlist(lapply(U, area))
    1         2         3         4         5         6
0.13781493 0.11003783 0.09565677 0.05094797 0.12997292 0.09999957
    7         8         9        10
0.04589386 0.06755349 0.16461867 0.09750400
```

Tessellations are very handy for classifying the points of a pattern, so that the resulting classification can be used by `split.ppp()`, `cut.ppp()` and `by.ppp()`. If X is a point pattern and V is a tessellation (of the window of X) then

- `cut(X, V)` attaches marks to the points of X identifying which tile of V each point falls into;
- `split(X, V)` divides the point pattern into sub-patterns according to the tiles of V , and returns a list of the sub-patterns;
- `by(X, V, FUN)` divides the point pattern into sub-patterns according to the tiles of V , applies the function `FUN()` to each sub-pattern, and returns the results as a list.

If we superimpose plots of two tessellations on the same spatial domain (e.g. by setting `add=TRUE`) what we see is another tessellation. The "intersection" (or "overlay" or "common refinement") of two tessellations X and Y is the tessellation whose tiles are the intersections between tiles of X and tiles of Y . The command `intersect.tess()` computes the intersection of two tessellations. The intersection of tessellations is illustrated in Figure 4.30.

Other operations for tessellations include:

<code>bdist.tess()</code>	compute distance from tile to window boundary
<code>chop.tess()</code>	divide tessellation along a line
<code>rpoislinetess()</code>	generate tessellation based on random lines

These operations are documented in the online help for `spatstat`.

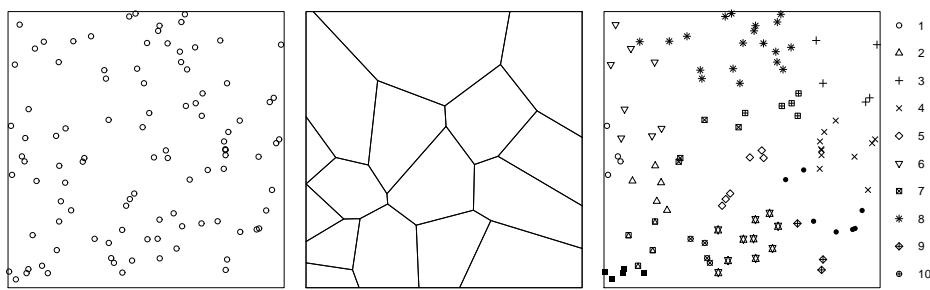


Figure 4.28: A point pattern X , a tessellation Z , and the result of $\text{cut}(X, Z)$.

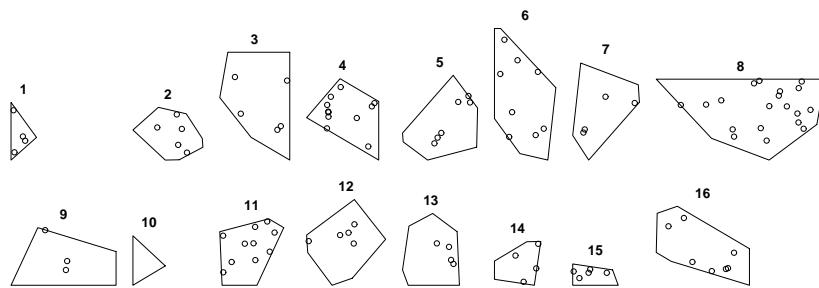


Figure 4.29: The result of $\text{split}(Z, X)$.

4.5 FAQ's

- How do I remove the white space around the plot of a point pattern?

Graphics code in `spatstat` uses the base R graphics system (the package `graphics`). General properties of a plot such as the text margin are controlled by the graphics parameters managed by the function `par`. To reduce the white space, change the parameter `mar`. Typically, `par(mar=rep(0.5, 4))` is adequate, if there are no annotations or titles outside the window.

In `spatstat`, spatial objects such as point patterns and pixel images are always plotted using equal scales on the x and y axes. White space is inevitable if the space available for plotting does not have the same *shape* (aspect ratio) as the spatial object to be plotted. If you are plotting on a computer screen, then you can change the white space by resizing the plot window using the mouse. If you are plotting to a file using `postscript()` or `pdf()`, try specifying the `width` and `height` parameters to have roughly the same ratio as the width and height of the spatial object to be plotted.

- My plot looks good on the computer screen, but when I send the same plot commands to a PDF or Postscript file, it looks different. For example the text is too large, or the plot symbols are too small.

This is usually related to discrepancies between the size parameters (`width` and

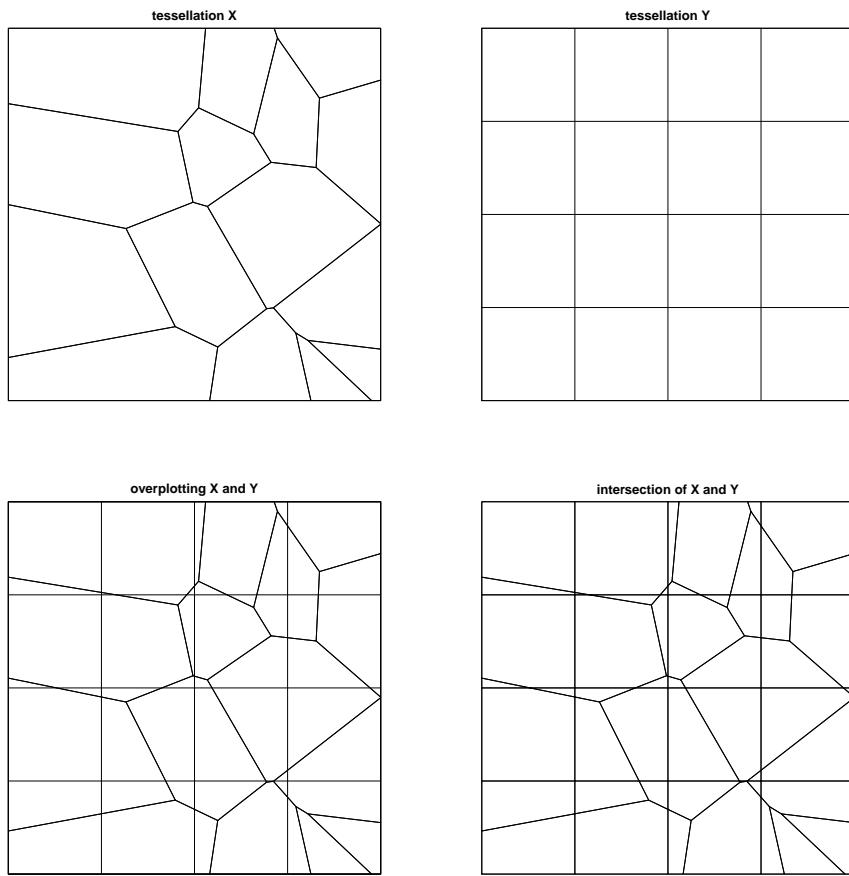


Figure 4.30: Intersection of tessellations.

`height`) for the graphics window and for the output file. Draw the plot on the computer screen, and use a ruler to measure the physical size of the window in *inches*. Then pass these parameters as the `width` and `height` for `postscript()` or `pdf()`.

- How can I see the internal structure of a "ppp" object, and extract some of the components?

To extract components of a "ppp" object, use the functions `coords()`, `marks()`, `as.data.frame()`, `as.owin()` or `npoints()`.

We strongly advise against manipulating the internal structure of objects that belong to a class. This can cause the data to become internally inconsistent, leading to errors that are very hard to fix. You Have Been Warned.

The internal data in an object could also be quite voluminous, filling your screen with a mess. To see a compact description of the internal structure of an object `x`, type `str(x)`.

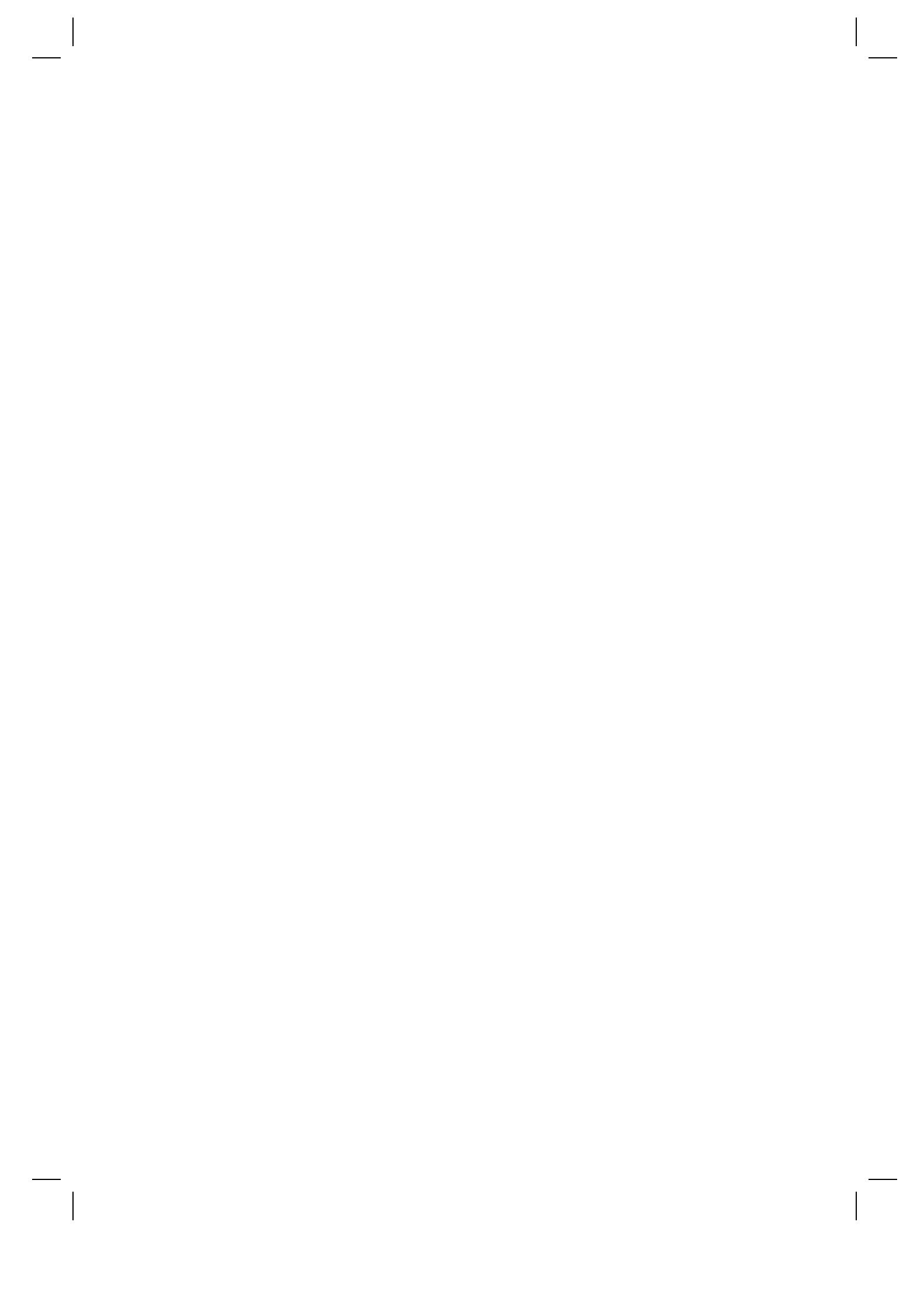
To strip away the class structure entirely, use `y <- unclass(x)`, which can then be printed to reveal the entire data structure. If this is too complicated, use `names(y)` to determine the names of the components of `y`, and inspect individual components `y[[i]]`.

5

Point processes

This chapter is not part of the review draft.

5.1 Not included in review



Part II

EXPLORATORY DATA ANALYSIS



6

Intensity

Part II covers methods for exploratory data analysis, starting in this Chapter with the fundamental concept of the intensity of a point process.

6.1 Introduction

Dividing the total number of points by the area of the survey region gives the average density of points per unit area. For example, a wildlife ecologist who maps bird nests in a survey might choose to report the result as a density of nests per hectare. In different contexts, the average number of points per unit area could be a measure of abundance (for example, the abundance of virus particles on a cell surface), density (of light-sensitive cells in the retina), productivity (of crops), risk (of crimes), intensity (of a lightning storm), or prospectivity (of undiscovered mineral deposits). The standard generic term is *intensity*.

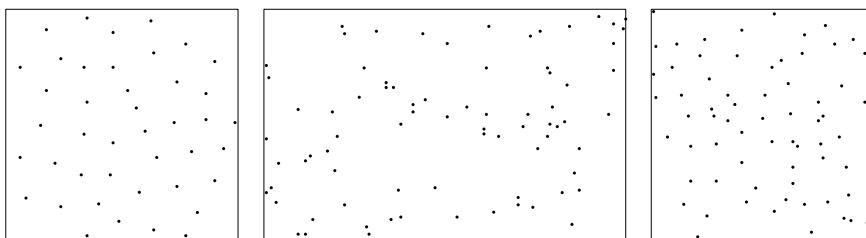


Figure 6.1: Classic point pattern datasets with (roughly) homogeneous intensity. *Left:* biological cell centres in histological section [281, 283]. *Middle:* trees in a New Zealand forest, excluding a five-foot border [236, 283]. *Right:* Swedish Pine saplings [315, 281].

Investigation of the intensity of a point pattern is one of the first and most important steps in data analysis. The intensity is a basic descriptive characteristic of a point process, an average ('expectation' or 'first moment') analogous to the average of a population of numbers. Compared to other properties of a point process, the intensity requires relatively few modelling assumptions.

This chapter deals with *exploratory* investigation of intensity using *nonparametric* tools (i.e. avoiding restrictive model assumptions). Parametric modelling of intensity is covered in Chapter 9.

In dividing the total number of points by the total area of the survey plot, the ecologist has effectively assumed that the spread of points is 'uniform' or 'homogeneous', as illustrated in Figure 6.1. Homogeneity may be a tentative, working assumption for some kinds of analysis. Homogeneity may be assumed if there is theoretical justification: for example, much of modern cosmology assumes that the universe is homogeneous on sufficiently large scales.

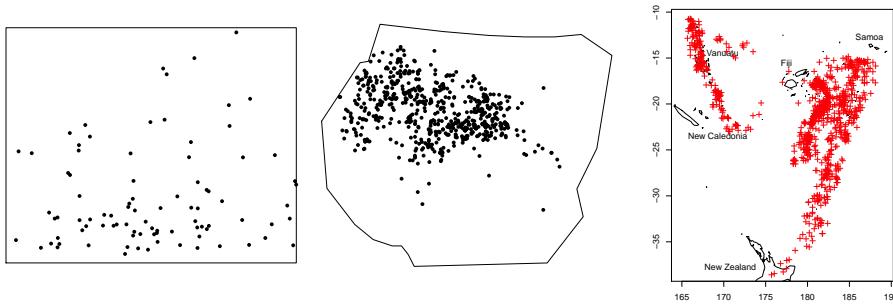


Figure 6.2: Point pattern datasets with inhomogeneous intensity. *Left*: enterochromaffin-like cells in histological section of gastric mucosa (Dr Thomas Bendtsen; see [250, pp. 2, 169]), interior of stomach towards top of picture. *Middle*: gorilla nesting sites in Kagwene Gorilla Sanctuary, Cameroon (Wildlife Conservation Society Takamanda–Mone Landscape Project, WCS-TMLP; Dr N Funwi-Gabga; see [152]). *Right*: epicentres of 1000 earthquakes of Richter magnitude 4.0 or greater in the South Pacific showing concentrations along two plate boundaries (R package datasets; Harvard PRIM-H project; Dr John Woodhouse).

However, this assumption would be inappropriate for other point patterns, where the intensity is spatially varying ('nonuniform', 'inhomogeneous' or 'heterogeneous' intensity) as illustrated in Figure 6.2.

Sometimes the most important scientific question is whether the intensity is homogeneous or not. Inhomogeneity of the intensity can reflect spatial variation in abundance (of a bird population), fertility (of a natural forest), or risk (of tornadoes). It can reflect preference (of wild animals for certain types of habitat), avoidance or segregation (between different species of tree). It can reflect dependence on spatially-varying external factors: for example, the likelihood of finding a gold deposit often depends on proximity to geological faults. When the intensity is spatially-varying, it is effectively a function of spatial location, and we can use statistical methods to estimate this function from data.

Dividing the number of points by the area of the survey region also assumes that the area is the appropriate 'denominator'. This would not be appropriate in spatial epidemiology, for example, where the number of disease cases reported to a health authority would usually be expressed as a ratio relative to the *total population* in the same region, rather than the area. The ratio of cases to population is an estimate of the *disease risk per head of population*. That calculation effectively assumes that this risk is constant. This is equivalent to assuming that the spatially-varying intensity of disease cases (cases per square kilometre) is proportional to the spatially-varying population density (people per square kilometre).

To compare the *relative* spatial distribution of two different kinds of points, we can study the ratio of their intensities. For example a spatial case-control dataset gives the spatial locations of a set of disease cases, and of a separate set of controls (notionally a sample from the population at risk of the disease). If the disease risk does not depend on spatial location, then we would expect the intensities of these two point processes to be proportional.

The points in a point pattern may carry different 'weights'. For example, astronomers are interested in the spatial distribution of *mass* in the universe, so the analysis of a spatial pattern of galaxies might take into account the estimated mass of each galaxy. The appropriate definition of intensity is then the average *total mass* of galaxies per unit volume of space. This reflects a different choice for the 'numerator' of the intensity. Similarly there are cases where each data point represents one or more events that occurred at the same spatial location, such as multiple disease cases at the same residence. It is then appropriate to weight each residential location by the number of cases, so that

the intensity would be the average *total number of cases* per unit area, not the number of affected residences per unit area.

We noted above that the investigation of intensity is an important step in analysing a point pattern. This remains true even when the intensity is not a major focus of statistical analysis. Spatial variation in intensity can easily be confused with other pattern characteristics such as clustering [51]. In order to establish convincingly that a point pattern is clustered, we need to eliminate alternative explanations, including spatial variation in the intensity. If the intensity is found to be spatially-varying, some of the standard tools for investigating clustering can be *adjusted* for this effect, provided we have estimated the intensity faithfully. This is similar to the principle that, in a controlled experiment, a test for the presence of interaction between two factors should involve either adjustment for the main effects, or confirmation that there is no main effect.

Section 6.2 discusses the estimation of intensity assuming that the point process is homogeneous. General theory of intensity is presented in Section 6.3. Quadrat counting methods are introduced in Section 6.4 and kernel smoothing methods in Section 6.5. Methods for investigating whether intensity depends on a spatial covariate are developed in Sections 6.6 and 6.7. Parametric models for intensity are mentioned briefly in Section ???. Techniques for detecting and locating ‘hot spots’, ‘clusters’ and ‘features’ are discussed in Section 6.8. Section 6.9 presents methods for smoothing the mark values. Section 6.10 discusses smoothing for multitype point processes, including the important problem of relative risk. Finally Section 6.11 discusses higher dimensions and space-time.

6.2 Estimating homogeneous intensity

Analysis of a point pattern often starts by tentatively assuming homogeneity. In order to assess or validate this assumption, we must define exactly what is meant by homogeneous intensity. Figure 6.3 shows synthetic patterns generated at random by a mechanism that is spatially homogeneous. But a single point pattern such as those in Figure 6.3 cannot be perfectly homogeneous, since a point cannot be spread uniformly over the plane like butter. The only way to make rigorous sense of the assumption of ‘homogeneity’ is by thinking statistically. The outcome of the ecologist’s survey of bird nests is one of many possible outcomes that could have been obtained, depending on random events in the history of the forest, its colonization by birds, and the selection of the survey plot. It is the *average over all possible outcomes* which is homogeneous.

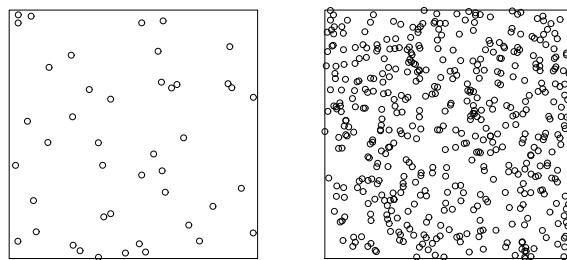


Figure 6.3: Simulated point patterns with homogeneous intensity. *Left:* 50 points per unit area. *Right:* 500 points per unit area.

Accordingly a point process \mathbf{X} is defined to have *homogeneous intensity* if, for any sub-region

B of two-dimensional space, the *expected* number of points of \mathbf{X} falling in B is proportional to the area of B :

$$\mathbb{E}[n(\mathbf{X}_B)] = \lambda \text{area}(B) \quad (6.1)$$

where λ is a constant called the *intensity*.

The intensity λ is the expected number of points per unit area. It has dimensions length⁻² so it depends on the unit of measurement. For example, 300 points per square kilometre is equivalent to 0.0003 points per square metre.

In basic statistics, the sample mean of a dataset is an unbiased estimate of the population mean. Similarly for point processes, the empirical density of points,

$$\bar{\lambda} = \frac{n(\mathbf{x})}{\text{area}(W)} \quad (6.2)$$

is an unbiased estimate of the true intensity λ , assuming the point process has homogeneous intensity. Here \mathbf{x} is the point pattern dataset, observed in a window W , and $n(\mathbf{x})$ is the number of points in \mathbf{x} .

The estimate $\bar{\lambda}$ could be computed by hand using the functions `npoints()` and `area.owin()`, or using the special command `intensity.ppp()`, a method for the generic function `intensity()`.

For example, the Swedish Pines pattern shown in the right panel of Figure 6.1 is available in `spatstat` as the dataset `swedishpines`. The intensity estimate $\bar{\lambda}$ could be computed by hand as `npoints(swedishpines)/area.owin(as.owin(swedishpines))`, or just by typing `intensity(swedishpines)`, which both return the value 0.0074, meaning that the estimated intensity is $\bar{\lambda} = 0.0074$ points per square unit. However, the units of length for the `swedishpines` dataset are unconventional: `unitname(swedishpines)` returns “0.1 metres”, so the estimated intensity is 0.0074 trees per square decimetre, equivalent to 0.74 trees per square metre or 7400 trees per hectare. One could also type `summary(swedishpines)` which gives a longer summary of the data including both the intensity and the unit of length.

The `rescale()` command can be used to convert the coordinates to a standard unit of length, before calculating the intensity. If the call to `rescale()` does not specify the scaling factor, the default is to convert a composite unit like “0.1 metres” to “metres”. For example `intensity(rescale(swedishpines))` gives 0.74, the intensity in points per square metre.

To quote a standard error for our estimate of the intensity, we would need to make additional assumptions. If we provisionally assume that the point process is *Poisson*, then the observed total number of points $n(\mathbf{x})$ is a realisation of a Poisson random variable with mean $\lambda \text{area}(W)$. For the Poisson distribution, the variance is equal to the mean, so $\bar{\lambda}$ has variance $\text{var}\bar{\lambda} = \text{var}[n(\mathbf{X}_W)]/\text{area}(W)^2 = \lambda/\text{area}(W)$. An estimate of the standard error of $\bar{\lambda}$ is $\sqrt{\bar{\lambda}/\text{area}(W)}$:

```
> X <- rescale(swedishpines)
> W <- as.owin(X)
> lam <- intensity(X)
> sdX <- sqrt(lam/area.owin(W))
> sdX
[1] 0.08777239
```

The estimated standard error is 0.0878 points per square metre. Standard errors and confidence intervals can also be obtained using resampling methods () or by fitting parametric models ().

If the point pattern is multitype (with a vector of marks that is a `factor` classifying each point into one of several possible types), then `intensity.ppp()` will return a vector of intensities, one for each possible type.

```
> unitname(amacrine)
```

```

662 microns
> X <- rescale(amacrine, 1000/662)
> unitname(X) <- "mm"
> intensity(X)
      off          on
202.3599 216.6106

```

To estimate the intensity of the pattern of all points regardless of their type, use either `sum(intensity(X))` or `intensity(unmark(X))`.

If the point pattern has weights, use the argument `weights` to `intensity.ppp()`. For example, in the `finpines` dataset each point is the location of a pine sapling, and has two marks, giving the tree's diameter (in centimetres) and height (in metres).

```

> finpines
marked planar point pattern: 126 points
Mark variables: diameter, height
window: rectangle = [-5, 5] x [-8, 2] metres

```

A rough estimate of the sapling's volume is obtained by pretending that it is a cone (since a cone of height h with base diameter d has volume $\pi h d^2 / 12$):

```

> height <- marks(finpines)$height
> diameter <- marks(finpines)$diameter
> volume <- (pi/12) * height * (diameter/100)^2

```

or more elegantly

```

> volume <- with(marks(finpines),
                  (pi/12) * height * (diameter/100)^2)

```

We use the estimated tree volumes as weights:

```

> intensity(finpines, weights=volume)
[1] 0.001273693

```

There is an estimated average *total volume* of 0.00127 cubic metres per square metre of forest floor.

6.3 Theory

We need a technical definition of ‘intensity’ which applies even when the point pattern is not homogeneous. Examples of inhomogeneous patterns are shown in Figure 6.4.

The left and middle panels of Figure 6.4 can be described by saying that the intensity value λ is spatially varying. At a spatial location u , the intensity is $\lambda(u)$, where $\lambda(u)$ is some function of location. More precisely, in a small neighbourhood of the location u , with small area a , the expected number of points is approximately equal to $\lambda(u)a$. For any region B , we may divide B into small subregions and add up the expected number of points in each subregion, giving

$$\mathbb{E}[n(\mathbf{X}_B)] = \int_B \lambda(u) du \quad (6.3)$$

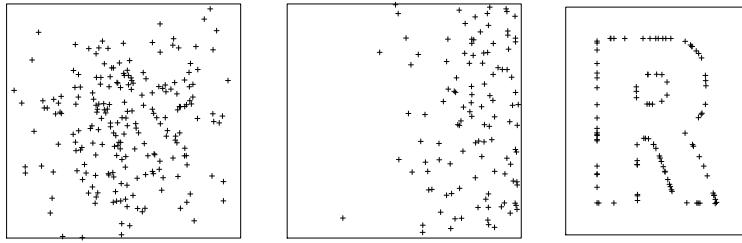


Figure 6.4: Simulated point patterns with different intensities. *Left:* diffuse intensity, higher at centre of plot. *Middle:* diffuse intensity, higher at right-hand edge of plot. *Right:* singular intensity concentrated on edges of the letter R.

for all regions B . We can think of $\lambda(u)$ as an undulating surface whose height represents the intensity. From this standpoint, equation (6.3) states that the expected number of points falling in some region of interest is equal to the volume under the surface. If (6.3) holds, then $\lambda(u)$ is called the *intensity function* of the point process. The values of $\lambda(u)$ are intensity values (points per unit area) with dimensions length $^{-2}$.

For example, the middle panel of Figure 6.4 is a realisation of a point process in the unit square with intensity function

$$\lambda(x, y) = 300x^2$$

where x and y are the Cartesian coordinates.

However, the right panel of Figure 6.4 is a realisation of a (simulated) point process which *does not have an intensity function*. The points are concentrated along the edges of the letter R. Point processes of this kind are common in seismology (since many earthquakes occur exactly at a tectonic plate boundary). To handle such cases we need a more general language.

For any point process \mathbf{X} , the intensity characteristics are completely described if we know the expected number of points

$$\Lambda(B) = \mathbb{E}[n(\mathbf{X}_B)] \quad (6.4)$$

falling in any given set B . If the point process has homogeneous intensity, then $\Lambda(B) = \lambda \text{area}(B)$ for all B , where the constant $\lambda \geq 0$ is the intensity value. A compact way to say this is that “ Λ is proportional to area”.

Notice that Λ is a set function — a function whose argument is a *set* or *region* B , and which returns a numerical value $\Lambda(B)$. In the same way, “area” is a set function which, when applied to any region B , yields the area of the region. We call Λ is the *mean measure* of the point process \mathbf{X} . This is mostly a mathematical concept, but it enables us to speak about the intensity of any¹ point process.

In some other texts, Λ is called the *intensity measure*. This can lead to confusion, because the values $\Lambda(B)$ are not intensities (numbers of points per unit area); they are mean numbers of points.

The mean measure Λ is called *diffuse* if it has an intensity function $\lambda(u)$ in the sense of (6.3). It is called *singular* if the points are confined to a set with zero area, such as the linear boundary in the right panel of Figure 6.4.

¹The only requirement is that the expectation in (6.4) should be finite whenever the region B is bounded (contained in a rectangle of finite size).

6.4 Quadrat counting

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

6.4.1 Quadrat counts

A simple way to check for inhomogeneity is to check whether regions of equal area contain roughly equal numbers of points (as they must do if the point process is homogeneous).

In *quadrat counting*, the observation window W is divided into subregions B_1, \dots, B_m of equal area, called quadrats.² We count the numbers of points falling in each quadrat, $n_j = n(\mathbf{x}_{B_j})$ for $j = 1, \dots, m$. Since these counts are unbiased estimators of the corresponding expected values $\mathbb{E}[n(\mathbf{X}_{B_j})]$, they should be equal ‘on average’ if the intensity is homogeneous. In particular, any apparent spatial trend in the counts n_j suggests that the intensity is inhomogeneous.

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

```
> Q3 <- quadratcount(swedishpines, nx=3, ny=3)
> Q3
      x
y      [0,32] (32,64] (64,96]
(66.7,100]     8       6       7
(33.3,66.7]    8       11      9
[0,33.3]        5       6       11
```

The value returned by `quadratcount()` is an object belonging to the special class “`quadratcount`”. We used the `print()` method for this class to obtain the text output above, and the `plot()` method to get the display in the left panel of Figure 6.5. Note that the `plot()` and `print()` methods display the counts in the same spatial arrangement.

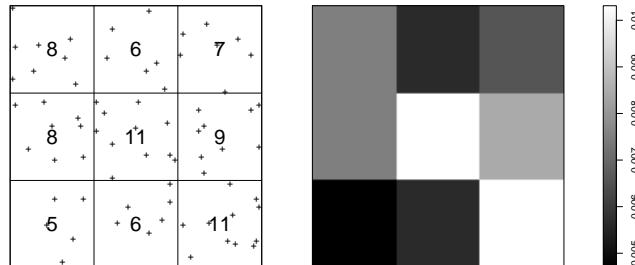


Figure 6.5: Quadrat counting for Swedish Pines data. *Left:* quadrat counts. *Right:* intensity estimates.

The arguments `nx`, `ny` to `quadratcount()` specify that the quadrats should be an `nx` by `ny` grid

²A ‘quadrat’ (German for ‘square’) originally meant a rectangular wooden frame used for random sampling in the field. It now refers to a spatial sampling region of any shape. A quadrat is very different from a ‘quadrant’, which is one-quarter of the infinite Euclidean plane.

of rectangles. This ensures that the quadrats have equal area, provided the observation window is a rectangle. Other options are discussed below.

In choosing the size of quadrats, there is a tradeoff between bias and variability: choosing larger quadrats reduces the coefficient of variability (variance divided by mean) of the counts n_j but also obliterates the spatial variation in intensity within each quadrat.

If the quadrat counts are divided by the areas of the corresponding quadrats, we obtain the average intensity in each quadrat, which is a simple estimate of the intensity function. The method `intensity.quadratcount()` calculates these intensity estimates from a "quadratcount" object.

```
> intensity(Q3)
x
y      [0,32]   (32,64]   (64,96]
(66.7,100] 0.0075000 0.0056250 0.0065625
(33.3,66.7] 0.0075000 0.0103125 0.0084375
[0,33.3]    0.0046875 0.0056250 0.0103125
```

Use `intensity(, image=TRUE)` to obtain a pixel image that is an estimate of the intensity function. Setting `L3 <- intensity(Q3, image=TRUE)` and calling `plot(L3)` gives the display in the right panel of Figure 6.5.

In general, the quadrats could have unequal sizes and shapes. The alternative argument `tess` to `quadratcount()` allows the quadrats to be any tessellation of the window. For example, hexagonal tiles can be created using `hextess()`:

```
> H <- hextess(swedishpines, 10)
> hQ <- quadratcount(swedishpines, tess=H)
```

The counts are plotted in the left panel of Figure 6.6. Since the quadrat areas are not all equal, the counts of points in these quadrats should not be compared directly. Under the hypothesis of homogeneity, equation (6.1), the expected count in each quadrat is *proportional to the area* of the quadrat. The average intensity (6.2) in each quadrat is an unbiased estimator of the homogeneous intensity λ . For exploratory purposes, we can plot the average intensity in each quadrat. The right panel of Figure 6.6 shows a plot of `intensity(hQ, image=TRUE)`.

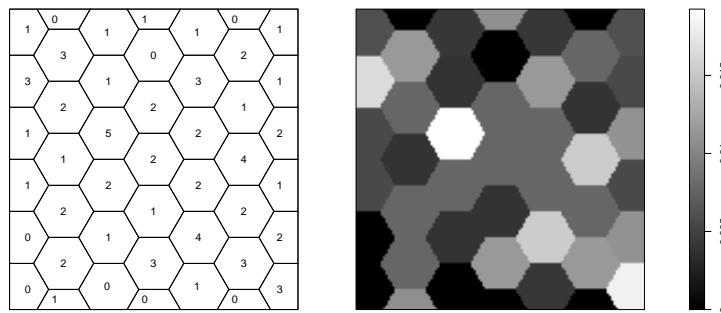


Figure 6.6: Quadrat counting with hexagonal quadrats. *Left:* quadrat counts. *Right:* intensity estimates.

6.4.2 Quadrat counting test of homogeneity

Historically the `swedishpines` dataset has been analysed under the assumption that it has homogeneous intensity [316, 283]. However, the quadrat counts suggest that the intensity may be slightly elevated along a diagonal swath from top left to bottom right of the plot.

One way to assess this is to conduct a formal statistical test. To apply such a test we assume, provisionally, that the point process is Poisson. The null hypothesis is that the intensity is homogeneous while the alternative hypothesis is that it is inhomogeneous in some unspecified fashion. Under the null hypothesis, the point process is a homogeneous Poisson process (exhibits Complete Spatial Randomness, CSR).

The window W is divided into quadrats B_1, \dots, B_m . We count the numbers of points falling in each quadrat, $n_j = n(\mathbf{x}_{B_j})$ for $j = 1, \dots, m$. Under the null hypothesis, the n_j are realisations of independent Poisson random variables with expected values $\mu_j = \lambda a_j$ where λ is the unknown intensity and a_j is the area of B_j . The Pearson χ^2 test of goodness-of-fit can be used. Given the total number of points $n = \sum_j n_j$, and the total window area $a = \sum_j a_j$, the estimated intensity is $\bar{\lambda} = n/a$, and the expected count in quadrat B_j is $e_j = \bar{\lambda} a_j = n a_j / a$. The test statistic is

$$\chi^2 = \sum_j \frac{(\text{observed} - \text{expected})^2}{\text{expected}} = \sum_j \frac{(n_j - e_j)^2}{e_j} = \sum_j \frac{(n_j - \bar{\lambda} a_j)^2}{\bar{\lambda} a_j}. \quad (6.5)$$

If the quadrats all have equal area, then the n_j are independent with *equal* expected value under the null hypothesis. The test statistic reduces to

$$\chi^2 = \sum_j \frac{(n_j - n/m)^2}{n/m}. \quad (6.6)$$

Under the null hypothesis, the distribution of the test statistic is approximately a χ^2 distribution with $m - 1$ degrees of freedom. The approximation is traditionally deemed to be acceptable when the expected counts e_j are greater than 5 for all quadrats.

The quadrat counting test is performed in `spatstat` by `quadrat.test()`.

```
> tS <- quadrat.test(swedishpines, 3,3)
> tS
    Chi-squared test of CSR using quadrat counts
    Pearson X2 statistic

  data: swedishpines
  X2 = 4.6761, df = 8, p-value = 0.4169
  alternative hypothesis: two.sided

  Quadrats: 3 by 3 grid of tiles
```

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 Swedish Pines data. The *p*-value can also be extracted by

```
> tS$p.value
[1] 0.4168565
```

The return value of `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

8 7.9 0.04	6 7.9 -0.67	7 7.9 -0.32
8 7.9 0.04	11 7.9 1.1	9 7.9 0.4
5 7.9 -1	6 7.9 -0.67	11 7.9 1.1

Figure 6.7: Quadrat counting test of CSR for Swedish Pines data.

Pearson residuals. Figure 6.7 shows the result of `plot(swedishpines); plot(tS, add=TRUE)`. In each quadrat the observed counts n_j are displayed at top left; expected counts e_j at top right; and Pearson residuals $r_j = (n_j - e_j)/\sqrt{e_j}$ at bottom.

Other arguments to `quadrat.test()` make it possible to conduct a one-sided test, and to compute the p -value using Monte Carlo simulation instead of the χ^2 approximation.

```
> quadrat.test(swedishpines, 5, alternative="regular",
   method="MonteCarlo")
Conditional Monte Carlo test of CSR using quadrat counts
Pearson X2 statistic

data: swedishpines
X2 = 18.0845, p-value = 0.198
alternative hypothesis: regular

Quadrats: 5 by 5 grid of tiles
```

The function `quadrat.test()` is generic, with methods for "ppp" objects (which we have used above), but also for "splitppp" and "quadratcount" objects. It is possible to perform a χ^2 test using previously-computed counts, for example the counts Q3 above:

```
> quadrat.test(Q3)
Chi-squared test of CSR using quadrat counts
Pearson X2 statistic

data:
X2 = 4.6761, df = 8, p-value = 0.4169
alternative hypothesis: two.sided

Quadrats: 3 by 3 grid of tiles
```

The results of several quadrat tests can also be pooled. For example, suppose an ecologist has recorded the spatial pattern of trees in three separate plots in the same forest. The data from each plot have been subjected to a quadrat counting test as described above. Then an overall test of uniform intensity is performed by applying `pool.quadrat.test()` to the three test results:

```
> test1 <- quadrat.test(X1, 3)
> test2 <- quadrat.test(X2, 3)
> test3 <- quadrat.test(X3, 5)
> pool(test1, test2, test3)
```

6.4.3 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 (6.6) 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 is optimal when the quadrats are neither very large nor 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 is also an implicit choice of spatial scale, because the data points are aggregated at this scale.

6.5 Smoothing estimation of intensity function

6.5.1 Kernel estimation

If the point process has an intensity function $\lambda(u)$, this function can be estimated nonparametrically by *kernel estimation*.

Our favorite analogy is to imagine placing one square of chocolate on each data point. Using a hair dryer we apply heat to the chocolate so that it melts slightly. The result is an undulating surface of chocolate; the height of the surface represents the estimated intensity function of the point process.

6.5.1.1 Kernel estimators

The usual *kernel estimators* of the intensity function [124, 63] are the uncorrected estimate

$$\tilde{\lambda}^{(0)}(u) = \sum_{i=1}^n \kappa(u - x_i), \quad (6.7)$$

the uniformly corrected estimate

$$\tilde{\lambda}^{(U)}(u) = \frac{1}{e(u)} \sum_{i=1}^n \kappa(u - x_i), \quad (6.8)$$

and Diggle's corrected estimate

$$\tilde{\lambda}^{(D)}(u) = \sum_{i=1}^n \frac{1}{e(x_i)} \kappa(u - x_i), \quad (6.9)$$

for any spatial location u inside the window W , where $\kappa(u)$ is the *kernel function* (the shape of one melted square of chocolate) and

$$e(u) = \int_W \kappa(u - v) dv \quad (6.10)$$

is a correction for bias due to edge effects. Outside the window W , the estimated intensity is zero.

For a data point at location x_i , the function $f(u) = \kappa(u - x_i)$ represents the melted square of chocolate that was originally placed at x_i . The kernel κ must be a probability density, that is, $\kappa(u) \geq 0$ for all locations u , and $\int_{\mathbb{R}^2} \kappa(u) du = 1$. A common choice is the isotropic Gaussian (Normal distribution) probability density. The standard deviation of the kernel is the *smoothing bandwidth*: a larger bandwidth gives more smoothing. The choice of bandwidth involves a tradeoff between bias and variance: as bandwidth increases, typically the bias increases and variance decreases.

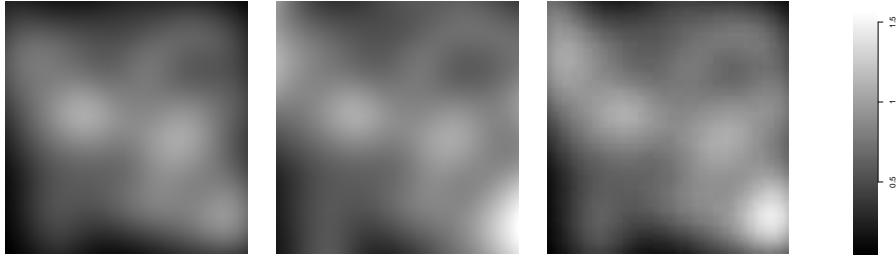


Figure 6.8: Kernel estimates of intensity for Swedish Pines using different edge corrections. *Left:* raw estimate. *Middle:* uniformly corrected estimate. *Right:* Diggle's correction estimate. Isotropic Gaussian kernel with standard deviation 10 decimetres. Density values were multiplied by 100.

Figure 6.8 shows the different kernel estimates (6.7)–(6.9) for the Swedish Pines data using the same kernel. Note that the “raw” or “uncorrected” estimate (6.7) decreases close to the boundary of the observation window. This is an *edge effect*. Trees lying just outside the window were not observed, and did not contribute to the estimate (6.7), so that locations u closer to the boundary of the window receive fewer contributions in the sum (6.7). The raw estimate therefore has a strong negative bias at locations close to the boundary, due to edge effects. The raw estimate should be used only in those rare situations where there are no edge effects — for example, in mapping the density of an isolated stand of trees, where every tree in the stand is represented in the data.

The uniform correction (6.8) and Diggle's correction (6.9) are designed to compensate for the edge effect arising when a point process is observed inside a window. The uniformly corrected estimator (6.8) is unbiased when the true intensity is homogeneous. Diggle's corrected estimator (6.9) has better performance overall (smaller mean square error) and is normalised so that the integral of $\tilde{\lambda}^{(D)}(u)$ over the window is exactly equal to the observed number of points.

Kernel estimators of the intensity function are slightly biased in general, because they smooth out details in the intensity function. To understand their statistical properties we can use *Campbell's formula*. Suppose $f(u)$ is a function of spatial location u , and we consider the random sum

$$\sum_i f(x_i)$$

of the values of f at each of the points x_i in a point process \mathbf{X} . Campbell's formula states that this

sum has expected value

$$\mathbb{E}\left[\sum_i f(x_i)\right] = \int_{\mathbb{R}^2} f(u) \lambda(u) du \quad (6.11)$$

where $\lambda(u)$ is the intensity function of \mathbf{X} .

To apply Campbell's formula to kernel estimation, suppose we fix a spatial location v , and let $f(u) = \kappa(v - u)$ if u is inside the window W , and $f(u) = 0$ if it is outside. Then

$$\sum_i f(x_i) = \sum_i \kappa(v - x_i) = \tilde{\lambda}^{(0)}(v)$$

where the sum is over all points x_i in the window W . By (6.11)

$$\mathbb{E}[\tilde{\lambda}^{(0)}(v)] = \mathbb{E}\left[\sum_i f(x_i)\right] = \int f(u) du = \int_W \kappa(v - u) \lambda(u) du.$$

The expected value of the estimate $\tilde{\lambda}^{(0)}(v)$ is not equal to the true intensity value $\lambda(v)$. Even if the true intensity is constant, say $\lambda(v) \equiv \beta$ for all locations v , we get

$$\mathbb{E}[\tilde{\lambda}^{(0)}(v)] = \int_W \kappa(v - u) \beta du = \beta \int_W \kappa(v - u) du = \beta e(v)$$

where $e(v)$ was defined in (6.10). This motivates the definition of the corrected estimator $\tilde{\lambda}^{(U)}(u)$ which is then unbiased at least when the intensity is constant. In general, $\tilde{\lambda}^{(U)}(v)$ is an unbiased estimator of

$$\lambda^*(v) = \frac{1}{e(v)} \int_W \kappa(v - u) \lambda(u) du,$$

a smoothed version of the true intensity function $\lambda(v)$.

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

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

The smoothing bandwidth is specified by the argument `sigma`. This may be a single numerical value (specifying the bandwidth of an isotropic Gaussian kernel in the same units as the point pattern), or a pair of numerical values (specifying different standard deviations in the x and y directions), or a function which performs automatic bandwidth selection (see below).

By default, the uniformly corrected kernel estimator (6.8) is calculated. For Diggle's corrected estimator (6.9), specify `diggle=TRUE`. For the uncorrected estimator (6.7), set `edge=FALSE`.

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. The method for `plot()` was used to produce Figure 6.8. The methods for `persp()` and `contour()` produced Figure 6.9.

The range of values of the kernel estimate of intensity for this dataset and bandwidth (computed by `range(den)`) is roughly from 0.002 to 0.016 points per square decimetre. In Figure 6.8 the text labels on the colour map ribbon show a different range of values, because we used the argument `ribayscale` to `plot.im()` to specify a scaling factor for the text labels.

6.5.1.2 Bandwidth selection

Different values of the bandwidth `sigma` may lead to over- or under-smoothing, as shown in Figure 6.10.

If the bandwidth `sigma` is not specified in the call to `density.ppp()`, the default is to take

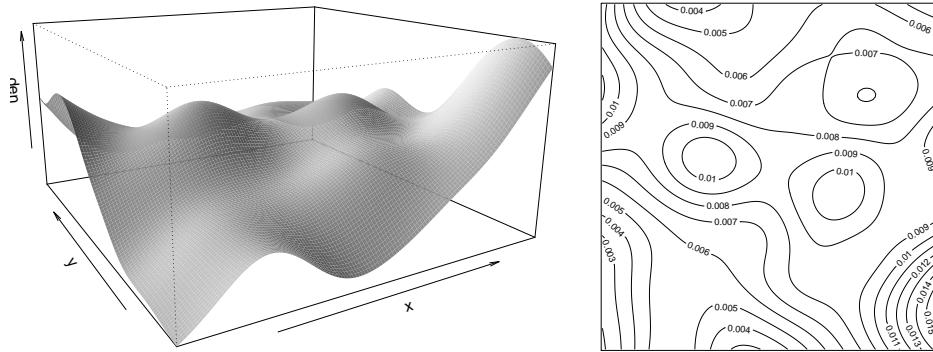


Figure 6.9: Perspective plot (*Left*) and contour plot (*Right*) for the kernel estimate of intensity. Plots were generated using `persp.im()` and `contour.im()` respectively.

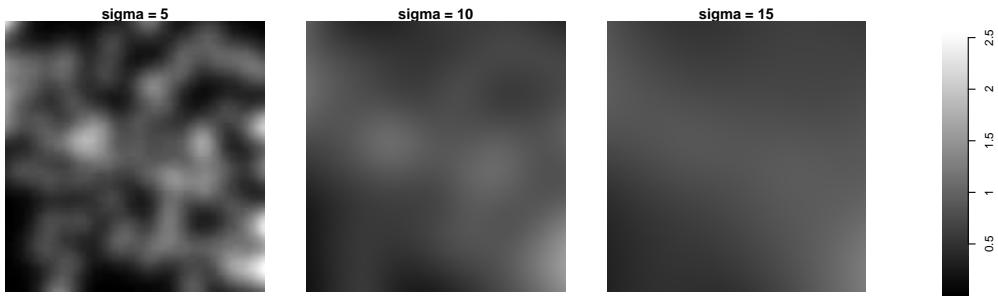


Figure 6.10: Density estimates with different smoothing bandwidths.

`sigma` equal to one-eighth of the shortest side length of the enclosing rectangle. This is a very rough rule of thumb which may be unsatisfactory in many cases.

Several algorithms are available for automatically selecting the bandwidth `sigma` by minimising a measure of error. They include `bw.diggle()` for Diggle and Berman's [124, 55] mean square error cross-validation method and `bw.ppl()` for the likelihood cross-validation method [230, Sect. 5.3].

```
> b <- bw.ppl(swedishpines)
> b

sigma
41.46591
```

These commands return a numerical value, the optimised bandwidth, which also belongs to the special class "`bw.optim`". The `plot()` method for this class shows the objective function for the optimisation. Figure 6.11 shows the results of plotting the likelihood cross-validation value using `plot(b)` and zooming in using `plot(b, xlim=c(30,60))`. The first plot suggests that any smoothing bandwidth greater than about 20 decimetres would be adequate. This is what might be expected for a homogeneous point pattern.

Bandwidth selection may also be based on a fast rule-of-thumb. Examples include `bw.scott()` for Scott's rule of thumb for bandwidth selection in multidimensional smoothing [296, p. 152], and

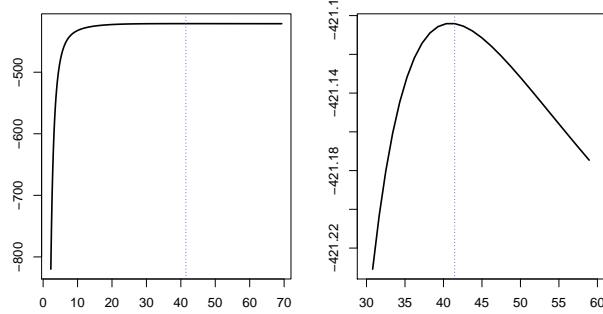


Figure 6.11: Likelihood cross-validation criterion for smoothing bandwidth plotted against bandwidth σ . Right panel is zoomed in to the range $30 \leq \sigma \leq 60$.

`bw.frac()` for a fast bandwidth selection rule based on the window geometry (explained in the `spatstat` help file).

Bandwidth selection commands can be invoked in either of the following ways:

```
> D <- density(swedishpines, sigma=bw.diggle(swedishpines))
> D <- density(swedishpines, sigma=bw.diggle)
```

Different bandwidth selection methods can disagree substantially. For the Swedish Pines data, `bw.diggle()` gives 5.72, while `bw.ppl()` gives 43.8 and `bw.scott()` gives (14.1, 13.2).

All bandwidth selection rules give unsatisfactory results in some cases, because they are based on assumptions about the dependence between points, which may be inappropriate. Likelihood cross-validation `bw.ppl()` assumes an inhomogeneous Poisson process; `bw.diggle()` assumes a Cox process, which is more clustered (positively correlated) than a Poisson process; both these assumptions are probably inappropriate for the Swedish Pines data which are somewhat more regular (negatively correlated) than a Poisson process. It is often convenient to be able to adjust the automatically-selected bandwidth by specifying the argument `adjust`, a numeric value which multiplies the selected bandwidth:

```
> D <- density(swedishpines, sigma=bw.diggle, adjust=2)
```

This is equivalent to selecting the bandwidth by `b <- bw.diggle(swedishpines)` then computing `D <- density(swedishpines, sigma=2*b)`.

6.5.1.3 Estimation of intensity at the data points

It is sometimes required to estimate the intensity values $\lambda(x_i)$ at the data points x_i themselves. For example, the estimated intensity values at the data points can be used as weights in some analysis procedures. However, the estimates $\tilde{\lambda}^{(U)}(x_i)$ and $\tilde{\lambda}^{(D)}(x_i)$ have a large positive bias, because of the term $\kappa(u - x_i) = \kappa(x_i - x_i) = \kappa(0)$ appearing in the sum in (6.8)–(6.9). To deal with this problem it is advisable to use a *leave-one-out estimator* in which the value of $\lambda(x_i)$ is estimated using all of the data points *except* x_i :

$$\tilde{\lambda}_{-i}^{(U)}(x_i) = \frac{1}{e(x_i)} \sum_{j \neq i} \kappa(x_i - x_j) \quad (6.12)$$

$$\tilde{\lambda}_{-i}^{(D)}(x_i) = \sum_{j \neq i} \frac{1}{e(x_j)} \kappa(x_i - x_j). \quad (6.13)$$

Typically the leave-one-out estimates have a slight negative bias.

To compute intensity estimates at the data points, invoke `density.ppp()` with the argument `at="points"`. The result is a numeric vector of density values for each data point. The default is to compute the leave-one-out estimates; this can be suppressed by setting `leaveoneout=FALSE`.

```
> dX <- density(swedishpines, sigma=10, at="points")
> dX[1:5]
[1] 0.003749609 0.007880300 0.006397353 0.006143837 0.003938111
```

6.5.1.4 Computation

The `spatstat` package uses different algorithms to compute the intensity estimates at data points and on a pixel grid. The intensity estimates at the data points are computed to high precision using the formulae (6.8)–(6.9) or (6.12)–(6.13) in double precision arithmetic. For the intensity estimates on a pixel grid, exact calculation would be too slow, so the pixel values are computed by spatially discretising the point pattern and convolving using the Fast Fourier Transform [95]. Thus, the following are approximately but not exactly equal:

```
> den <- density(swedishpines, sigma=10)
> denXpixel <- den[swedishpines]
> denXpixel[1:5]
[1] 0.009142232 0.010829191 0.009039346 0.010103363 0.005993841
> denXexact <- density(swedishpines, sigma=10, at="points",
  leaveoneout=FALSE)
> denXexact[1:5]
[1] 0.009211079 0.010836106 0.009145110 0.009050688 0.006037687
```

6.5.1.5 Weighted kernel estimators

If the data points x_i have numerical weights w_i , we can use weighted versions of the kernel estimators described above. The contribution from a point x_i to the estimator is simply multiplied by the weight w_i , so that (for example) the raw intensity estimator $\hat{\lambda}^{(0)}(u) = \sum_i \kappa(u - x_i)$ becomes $\hat{\lambda}^{(0,w)}(u) = \sum_i w_i \kappa(u - x_i)$. Using the chocolate analogy (page 131) the data point x_i is represented by w_i units of chocolate rather than one unit.

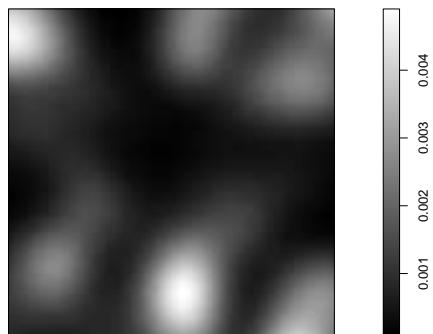


Figure 6.12: Volume-weighted intensity for Finnish Pines data. The argument `weights` to `density.ppp()` specifies weights for the density calculation. Figure 6.12 was generated by

Weighted kernel estimators are natural if the weight of a point represents its multiplicity (e.g. number of disease cases at the same residence) or physical mass (e.g. mass of galaxy) or economic value (e.g. total endowment of a mineral deposit).

For example, in a forest inventory we could take the ‘weight’ of each tree to be its estimated volume. The volume-weighted intensity is the average *standing volume of wood* per unit area of forest. Figure 6.12 shows this quantity for the Finnish Pines data. The scale is in metres (cubic metres of wood per square metre of forest).

The argument `weights` to `density.ppp()`

```
> vols <- with(marks(finpine),  
+ (pi/12) * height * (diameter/100)^2)  
> Dvol <- density(finpine, weights=vols, sigma=bw.ppl)
```

6.5.2 Spatially adaptive smoothing

The fixed-bandwidth kernel estimators described above have several weaknesses. A fixed smoothing bandwidth is unsatisfactory if the true intensity varies greatly across the spatial domain, because it is likely to cause over-smoothing in the high-intensity areas and under-smoothing in the low intensity areas. Kernel estimation is unsatisfactory when there is a sharp boundary between areas of high and low intensity, because this boundary will be smoothed out. These problems militate against the use of kernel estimation in seismology, for example.

The `spatstat` package offers several adaptive estimators of intensity. In `adaptive.density()`, a specified fraction f of the points in the point pattern are selected at random, and used to construct a Dirichlet tessellation. A quadrat counting estimator of the intensity is based on this tessellation. This process is repeated $nrep$ times and the results are averaged. Estimators of this type have been used in statistical seismology and perform well when there is an abrupt change in intensity.

```
> aden <- adaptive.density(swedishpine, f=0.1, nrep=30)
```

The value returned by `adaptive.density()` is another pixel image (object of class "im").

Another strategy is to measure the distance R from a fixed point u to the nearest data point x_i , and to compute the area $A = \pi R^2$ of the corresponding disc. For a homogeneous Poisson process with intensity λ , the random area A is negative exponential (λ) distributed, and the maximum likelihood estimate of λ based on R is $\hat{\lambda} = 1/(\pi R^2)$. Similarly we could use the distance R_k to the k -th nearest data point (for $k \geq 1$) and set $\hat{\lambda}_k = k/(\pi R_k^2)$. This can be calculated rapidly for all points u in a pixel grid. The `spatstat` function `nndensity()` computes this intensity estimate.

```
> nden <- nndensity(swedishpine, k=10)
```

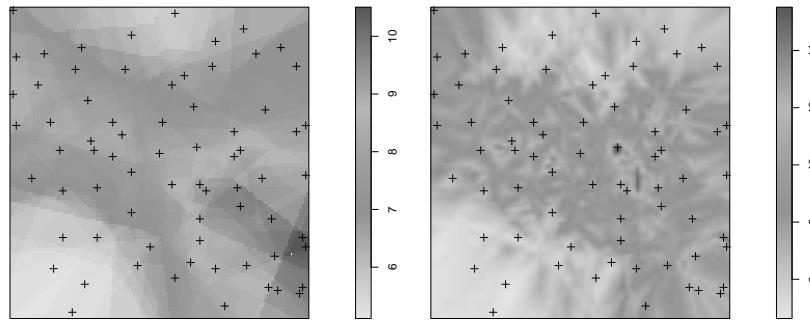


Figure 6.13: Adaptive density estimate (*Left*) and 10th nearest neighbour density estimate (*Right*) for the Swedish Pines data. Density values multiplied by 1000.

6.5.3 Projections, transformations, change of coordinates

In Section 6.2 and 6.3 we saw that the intensity depends on the unit of length. In fact, changing the spatial coordinate system in any way — through a change of units, a change of scale, a geometric

transformation or a geographic projection — affects the intensity. Intensity is the expected number of points per unit *area*, so any geometric transformation which changes the value of area also changes the value and the very *meaning* of the intensity.

The intensity function of a point process after a spatial transformation has been applied is related to the intensity function of the original point process, through the general principle of ‘change of coordinates’.

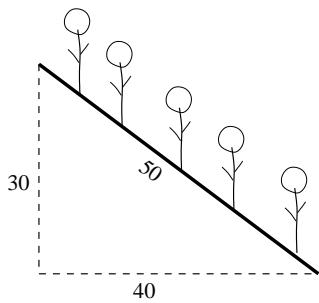


Figure 6.14: Illustration of change of coordinates for intensity.

We motivate this with an example. Imagine a hillside, covered in trees: see Figure 6.14. Walking straight up the hill involves a walking distance of 50 metres and an increase in altitude of 30 metres, so using trigonometry, the horizontal distance travelled is 40 metres. Suppose the hillside is 100 metres wide. The total surface area of the hillside is $50 \times 100 = 5000$ square metres or half a hectare. If there are 800 trees on the hillside, then the estimated intensity is $800/0.5 = 1600$ trees per hectare of hillside surface. However, if we photograph the hillside from directly above in a survey aircraft, or map the tree locations using a GPS device, the hillside is represented by a rectangle only $40 \times 100 = 4000$ square metres or 0.4 hectare in area on the map, and the estimated intensity is $800/0.4 = 2000$ trees per hectare of map area.

Which of these calculations is ‘right’? Actually, both are correct. The “forest density” could be defined either as a density per hectare of soil (which might be more useful for understanding soil ecology) or as a density per hectare of map area (perhaps more useful for understanding competition in the forest canopy).

The key is that the two measures of intensity are inter-related. Map area is equal to hillside surface area multiplied by the cosine of the slope angle, which is $40/50 = 0.8$ in the example above. Therefore the intensity per unit area of map is equal to the intensity per unit area of hillside, divided by the cosine of the slope angle.

The general principle is the following. Suppose that a point process \mathbf{X} has intensity function $\lambda(u)$. We now apply a geographic projection, a geometrical transformation, or a change of the coordinate system, so that the points x_i are mapped to new coordinate positions $y_i = f(x_i)$. The transformed point process $\mathbf{Y} = f(\mathbf{X})$ has intensity function

$$\lambda_{\mathbf{Y}}(u) = J(u) \lambda_{\mathbf{X}}(f^{-1}(u)) \quad (6.14)$$

where f^{-1} is the inverse mapping (that is, $f^{-1}(u)$ is the point mapped onto u), and $J(u)$ is the *Jacobian* (determinant of the derivative matrix) of the inverse mapping.

For example, invoking this general principle, we can apply the same simple trigonometry to real terrain data where the slope is spatially-varying. The tropical rainforest point pattern dataset `bei` comes with an extra set of covariate data `bei.extra`, which contains a pixel image of terrain slope `bei.extra$grad` (for ‘gradient’). The command `density(bei)` computes the estimated intensity relative to map area. To convert this to an estimate of the intensity relative to terrain surface area, we need the cosine of the slope angle. The covariate `grad` is given as the number of metres of elevation increase for every metre on the map, so that a `grad` value of 1 corresponds to a 45 degree slope. That is, `grad` is the tangent of the slope angle. Recalling our high school trigonometry, $\cos^2(x) = 1/(1 + \tan^2(x))$, so the conversion is:

```
> grad <- bei.extra$grad
> dens.map <- density(bei, W=grad)
> dens.ter <- eval.im(dens.map * sqrt(1+grad^2))
```

The two estimates are not very different in this case, because the maximum value of `grad` is only 0.33, so that the maximum inflation factor is only $\sqrt{1 + 0.33^2} = 1.05$.

An alternative to the calculation above is to introduce the Jacobian weight into the kernel smoother. That is, we smooth the point pattern in the projected space, but weight each data point by the Jacobian term at that point:

```
> dens.ter2 <- density(bei, weights=sqrt(1+grad[bei]^2))
```

This is justified by Campbell's formula (page 132). An advantage of this approach is that we only need to know the Jacobian values at the data points.

6.6 Investigating dependence of intensity on a covariate

6.6.1 Spatial covariates

Often we want to know how the intensity of points depends on the values of a covariate. For example, the tropical rainforest point pattern dataset `bei` comes with an extra set of covariate data `bei.extra`, which contains a pixel image of terrain elevation `bei.extra$elev` and a pixel image of terrain slope `bei.extra$grad` (for 'gradient'). It is of interest to determine whether the trees prefer steep or flat terrain, and whether they prefer a particular altitude. Other applications include spatial epidemiology (e.g. disease risk as a function of environmental exposure), spatial ecology (e.g. habitat preferences of organisms [235]) exploration geology (e.g. prospectivity of mineral deposits predicted from survey data [65]) and seismology.

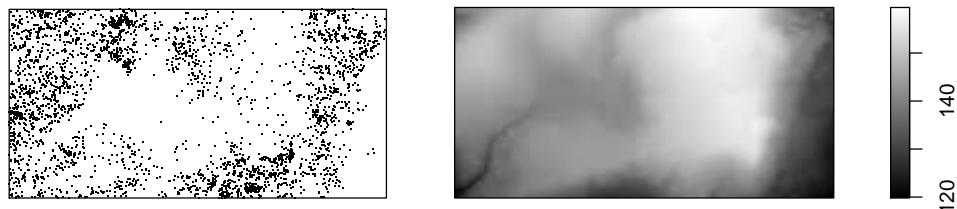


Figure 6.15: *Beilschmiedia* data. Trees (left) and terrain elevation (right) in a 1000 by 500 metre research plot.

6.6.2 Quadrats determined by a covariate

In quadrat counting methods, any choice of quadrats is permissible. From a theoretical viewpoint, the quadrats do not have to be rectangles of equal area, and could be regions of any shape.

Quadrat counting is more useful if we choose the quadrats in a meaningful way. One way to do this is to define the quadrats using covariate information.

For the tropical rainforest data `bei`, it might be useful to split the study region into several sub-regions according to the terrain elevation:

```
> elev <- bei.extra$elev
> b <- quantile(elev, probs=(0:4)/4)
> Zcut <- cut(elev, breaks=b, labels=1:4)
> V <- tess(image=Zcut)
```

The call to `quantile()` gave us the quartiles of the elevation values, so the four tiles in the tessellation `V` have equal area (ignoring discretisation effects). In other words, we have divided the study region into four zones of equal area according to the terrain elevation. The resulting tessellation is shown in Figure 6.16.

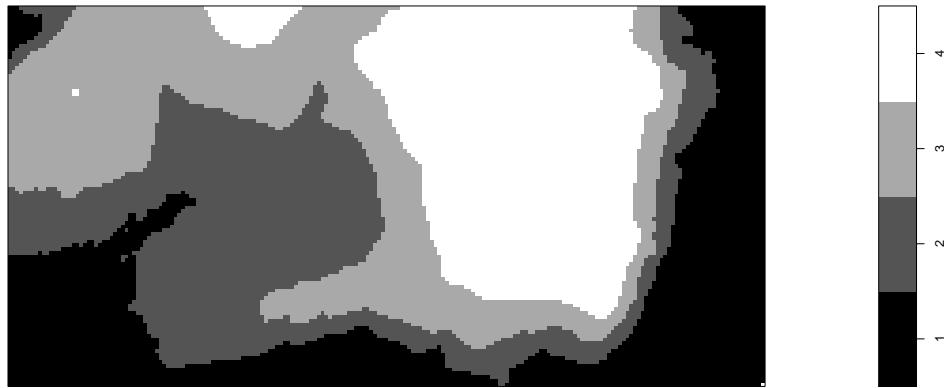


Figure 6.16: Equal-area tessellation corresponding to the quartiles of terrain elevation in the *Beilschmiedia* dataset.

We can now use this tessellation to study the point pattern `bei`. We could invoke the commands `split()`, `cut()` or `by()` to divide the points according to this tessellation and manipulate the sub-patterns.

The command `quadratcount()` also works with tessellations:

```
> qb <- quadratcount(bi, tess=V)
> qb
tile
  1    2    3    4
714  883 1344  663
```

The output shows the number of trees in each region. Since the four regions have equal area, the counts should be approximately equal if there is a uniform density of trees. Obviously they are not equal; there appears to be a strong preference for higher elevations (dropping off for the highest elevations).

In the following calculation we divide the range of elevations into intervals of equal width, estimate the average intensity for each interval, and plot the result as a bar chart.

```
> b5 <- seq(0, 5 * ceiling(max(elev)/5), by=5)
> Zcut5 <- cut(elev, breaks=b5, include.lowest=TRUE)
> Q5 <- quadratcount(bi, tess=tess(image=Zcut5))
> lam5 <- intensity(Q5)
```

A bar plot of `lam5` is shown in Figure 6.17.

6.6.3 Relative distribution estimate

In the previous analysis we were effectively assuming that the intensity of the point process is a *function* of the covariate Z . At any spatial location u , let $\lambda(u)$ be the intensity of the point process,

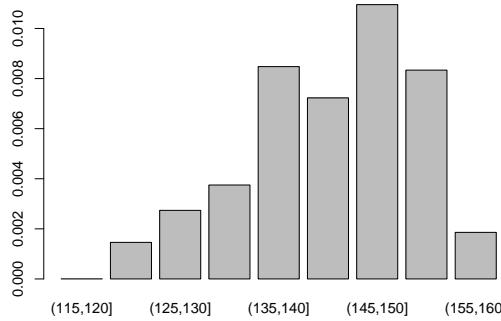


Figure 6.17: Estimates of intensity of *Beilschmiedia* trees in each 5-metre band of terrain elevation.

and $Z(u)$ the value of the covariate. Then we are assuming

$$\lambda(u) = \rho(Z(u)) \quad (6.15)$$

where ρ is a function that we want to investigate, telling us how the intensity of points depends on the value of the covariate.

In ecological applications where the points are the locations of individual organisms, ρ is a *resource selection function* [235] reflecting preference for particular environmental conditions Z . In geological applications where the points are the locations of valuable mineral deposits, ρ is an index of the *prospectivity* [65] or predicted frequency of undiscovered deposits as a function of geological and geochemical covariates Z .

Estimation of ρ

Nonparametric estimation of ρ is closely connected to estimation of a probability density from biased sample data [195, 139] and to the estimation of relative densities [171]. Under regularity conditions, ρ is proportional to the ratio of two probability densities, the numerator being the density of covariate values at the points of the point process, while the denominator is the density of covariate values at random locations in space. Kernel smoothing can be used to estimate the function ρ as a relative density [23, 164].

For a numerical covariate Z , the important tool is the *spatial distribution function*, the cumulative distribution function of the covariate value $Z(U)$ at a random point U uniformly distributed in W :

$$G(z) = \frac{1}{|W|} \int_W \mathbf{1}\{Z(u) \leq z\} du. \quad (6.16)$$

Equivalently $G(z) = |W_z|/|W|$ where $W_z = \{u \in W : Z(u) \leq z\}$ is the level set consisting of all locations in W where the covariate value is less than or equal to z . In practice $G(z)$ would often be estimated by evaluating the covariate at a fine grid of pixel locations, and forming the c.d.f.

For a numerical covariate Z , the analogues of Jones' estimators [195] are the “ratio” form

$$\hat{\rho}_R(z) = \frac{1}{|W|G'(z)} \sum_i \kappa(Z(x_i) - z) \quad (6.17)$$

and the “reweighted” form

$$\hat{\rho}_W(z) = \sum_i \frac{1}{|W|G'(Z(x_i))} \kappa(Z(x_i) - z) \quad (6.18)$$

while the analogue of El Barmi and Simonoff's "transformation" estimator is

$$\hat{\rho}_T(z) = \frac{1}{|W|} \sum_i \kappa(G(Z(x_i)) - G(z)) \quad (6.19)$$

where x_1, \dots, x_n are the data points, $Z(x_i)$ are the observed values of the covariate Z at the data points, $|W|$ is the area of the observation window W , and κ is a *one-dimensional* smoothing kernel — smoothing is conducted on the observed values $Z(x_i)$ rather than in the window W . The derivative $G'(z)$ is usually approximated by smoothing G .

The estimators (6.17)–(6.19) were developed in [23]. An estimator similar to (6.17) was proposed in [164].

Implementation in spatstat

In **spatstat** these estimators are computed by the command `rhohat()`.

```
> rh <- rhohat(bei, elev)
```

Extra arguments to `rhohat()` control the choice of estimator and the spatial resolution. As an alternative to kernel smoothing, local likelihood smoothing [231] can also be used.

The result of `rhohat()` is an object belonging to the special class "rhohat" which represents the estimated function ρ together with additional information. The `print()` method (not shown here) gives detailed information about the smoothing technique and the results that have been calculated.

The `plot()` method (shown in the left panel of Figure 6.18) generates a plot of the estimated function $\hat{\rho}(z)$ against covariate values z , together with 95% confidence bands assuming an inhomogeneous Poisson point process. The plot indicates that the *Beilschmiedia* trees are relatively likely to be found at elevations between 135 and 155 metres.

There is also a method for the generic function `predict()`. This computes the predicted intensity $\hat{\lambda}(u) = \hat{\rho}(Z(u))$ at each spatial location u , and returns it as a pixel image. The result of `predict(rh)` is plotted in the right panel of Figure 6.18.

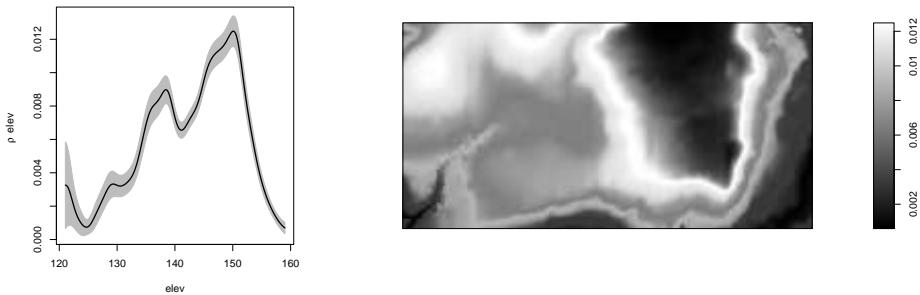


Figure 6.18: Intensity as a function of terrain elevation for the *Beilschmiedia* data. *Left:* estimated function $\hat{\rho}(z)$ giving forest density as a function of terrain elevation. Solid lines show function estimate; grey shading is pointwise 95% confidence band. *Right:* predicted forest density $\hat{\lambda}(u) = \hat{\rho}(Z(u))$ at each spatial location u .

An object of class "rhohat" can be converted to a function in the R language by `as.function()`. To obtain the predicted intensity at terrain elevation 130 metres:

```
> rhf <- as.function(rh)
> rhf(130)
```

```
[1] 0.003250479
```

The estimated function values can also be extracted as a data frame using `as.data.frame()`.

Validation

This analysis assumes that the intensity at a location u depends *only* on the covariate value $Z(u)$. If this is not true, $\hat{\rho}(z)$ is still meaningful: it is effectively an estimate of the average intensity $\lambda(u)$ over all locations u where $Z(u) = z$.

To validate the assumption (6.15) we can compare the predicted intensity $\hat{\rho}(Z(u))$ assuming (6.15) with a (spatial) kernel estimate $\hat{\lambda}(u)$ which does not assume (6.15).

```
> pred <- predict(rh)
> kden <- density(bei, 50)
```

A scatterplot of the two estimates at corresponding pixels can be generated by `pairs(pred, kden)`; the scatterplot should concentrate around the diagonal under the assumption. The difference of estimates at corresponding pixels can be plotted as an image using `plot(eval.im(kden - pred))`; the difference should be roughly equal to 0 everywhere. For the *Beilschmiedia* data with terrain elevation as the covariate, these graphics suggest that (6.15) is a reasonable approximation. By contrast, a similar exercise performed for the terrain *slope* covariate suggests that forest density is not simply a function of terrain slope.

Baseline

In some circumstances there is a natural ‘baseline’ intensity function $B(u)$ such that the *relative* intensity $\lambda(u)/B(u)$ can be assumed to depend only on the covariate Z . That is, we assume

$$\lambda(u) = \rho(Z(u))B(u) \quad (6.20)$$

instead of (6.15). The interpretation of the function $\rho(z)$ is different in this case: values of $\rho(z)$ are dimensionless, and the value $\rho(z) = 1$ corresponds to the baseline intensity. The same estimators (6.17)–(6.19) can be used, provided we replace $|W|$ and $G(z)$ by their B -weighted counterparts

$$\begin{aligned} W_B &= \int_W B(u) du \\ G_B(z) &= \frac{1}{W_B} \int_W \mathbf{1}\{Z(u) \leq z\} B(u) du \end{aligned}$$

respectively. An alternative is to use *Horvitz-Thompson* weighting, in which the contribution from each data point x_i to the estimators (6.17)–(6.19) is weighted by a factor $1/B(x_i)$. Then the denominators $|W|$ and $G(z)$ are unchanged.

To compute the modified estimators in `spatstat`, the baseline $B(u)$ should be given as the argument `baseline` to `rhohat()`. The baseline may be either a `function(x, y)` or a pixel image. Horvitz-Thompson weighting is selected by setting `horvitz=TRUE`.

Two covariates

If there are two numerical covariates $Z_1(u), Z_2(u)$ assumed to determine the intensity together, that is $\lambda(u) = \rho(Z_1(u), Z_2(u))$, then similar techniques can be applied [23] to estimate the function $\rho(z_1, z_2)$. These are implemented in the `spatstat` function `rho2hat()`. For example

```
> with(besi.extra, rho2hat(besi, grad, elev))
```

would compute an estimate of the intensity of *Beilschmiedia* trees as a function jointly of the terrain slope and terrain elevation.

6.6.4 Distance map

One particularly important kind of spatial covariate is a distance function. The dataset `copper` gives the locations of copper deposits in a survey region, and also the location of geological lineaments (which are mostly geological faults). It is conjectured that copper is more likely to be deposited close to a fault. Figure 6.19 shows the southern half of this dataset, rotated by 90 degrees:

```
> X <- rotate(copper$SouthPoints, pi/2)
> L <- rotate(copper$SouthLines, pi/2)
```

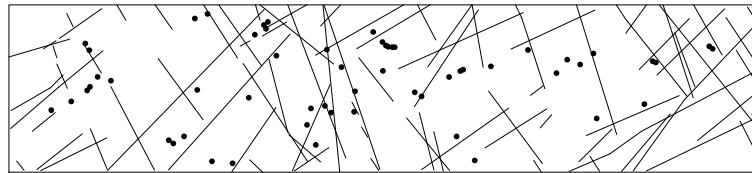


Figure 6.19: Queensland copper data, southern half.

To apply the methods described above, the covariate information contained in the map of geological faults L must be converted into a covariate that is a function $Z(u)$ of spatial location u . A natural choice is the *distance function*

$$Z(u) = \text{distance from } u \text{ to } L$$

This can be computed by the command `distmap()`, which returns a pixel image containing the values $Z(u)$ at a fine grid of pixels u . Figure 6.20 shows a contour plot of `Z <- distmap(L)`.

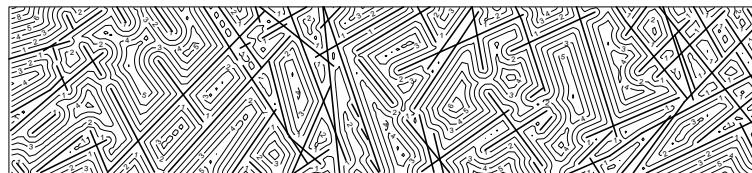


Figure 6.20: Distance map of geological lineaments in Queensland copper data.

Having created this covariate image we can now apply the other techniques such as relative distributions. Figure 6.21 shows the nonparametric estimate `rhohat(X, Z)` of the intensity of copper deposits as a function of distance to the nearest lineament.

A slightly more sophisticated version of `distmap()` is the command `distfun()`. Whereas `distmap()` returns a pixel image at a certain spatial resolution, `distfun()` returns a function with arguments `(x, y)` that can be evaluated at any spatial location.

```
> f <- distfun(L)
> f
Distance function for line segment pattern
planar line segment pattern: 90 line segments
window: rectangle = [-158.233, -0.19] x [-0.335, 35] km
> f(-42, 10)
```

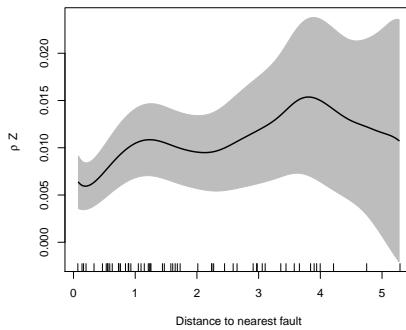


Figure 6.21: Nonparametric estimate (6.17) of intensity of copper deposits as a function of distance to the nearest lineament.

```
[1] 2.387029
```

In most commands in `spatstat` where a spatial covariate is required, a "distfun" can be used instead of a pixel image. This increases the precision of many calculations. It is usually advisable to call `distfun()` rather than `distmap()`, unless you really need a pixel image.

6.7 Formal tests of (non-)dependence on a covariate

It is often important to decide whether or not the intensity depends on a nominated spatial covariate Z . A formal hypothesis test may be useful (even in exploratory analysis).

We assume the point process is Poisson. The null hypothesis is that the intensity does not depend on Z , and is therefore constant, so the null hypothesis is Complete Spatial Randomness (CSR). The alternative hypothesis is that the intensity does depend on Z in some unspecified way.

Since Z can be any covariate of interest, including something artificial such as a Cartesian coordinate, the tests described in this section also serve as general tests of homogeneity.

6.7.1 Quadrat counting test of homogeneity

Our exploration of the tropical rainforest pattern `bei` in Section 6.6.2 above suggested that the density of trees depends on terrain elevation.

To test this formally we can apply the χ^2 quadrat counting test (Section 6.4.2 based on quadrats defined by ranges of values of the covariate).

The command `quadrat.test()` accepts a tessellation and uses the tiles of the tessellation as the quadrats:

```
> Z <- bei.extra$elev
> b <- quantile(Z, probs=(0:4)/4)
> Zcut <- cut(Z, breaks=b, labels=1:4)
> V <- tess(image=Zcut)
> quadrat.test(bei, tess=V)
Chi-squared test of CSR using quadrat counts
Pearson X2 statistic
```

```

data: bei
X2 = 319.8648, df = 3, p-value < 2.2e-16
alternative hypothesis: two.sided

Quadrats: 4 tiles (levels of a pixel image)

```

The test could also be performed using the previously-computed quadrat counts qb as `quadrat.test(qb)`.

Because of the large counts in these regions, we can probably ignore concerns about independence, and conclude that the trees are not uniform in their intensity.

6.7.2 Kolmogorov-Smirnov test of CSR

For testing whether a point process depends on a specified covariate, a more powerful test is a spatial application of the Kolmogorov-Smirnov (K-S) test [204, 302] in which we compare the observed and expected distributions of the values of the covariate.

The principle is that, if the point process is completely random, then the data points are effectively a random sample of spatial locations in the window, so the values of the covariate at the data points, $z_i = Z(x_i)$, should be a random sample of the values of the covariate at *all* spatial locations in the window.

To perform the K-S test, the covariate Z is evaluated at each of the data points, yielding $z_i = Z(x_i)$, and we compile the cumulative distribution function $\hat{F}(z)$ of these values. Then Z is evaluated at ‘every’ spatial location u in the window (for example, evaluated at the centre of every pixel in a grid) and we form the cumulative distribution function $F_0(z)$ of these values. The K-S test statistic is the maximum vertical separation between the graphs of $\hat{F}(z)$ and $F_0(z)$:

$$D = \max_z |\hat{F}(z) - F_0(z)|. \quad (6.21)$$

A test is possible because the null distribution of D depends only on the number of points, assuming the point process is Poisson and F_0 is a continuous function. As far as we are aware, this spatial version of the K-S test was first mentioned explicitly by Berman [54].

In `spatstat` this spatial version of the Kolmogorov-Smirnov test is performed by `cdf.test()`. The same function also allows us to replace the Kolmogorov-Smirnov statistic (6.21) by other measures of discrepancy such as the Cramér-Von Mises [103, 326, 109] or Anderson-Darling [10, 11, 238] statistics.

The function `cdf.test()` is generic. The method for point patterns, `cdf.test.ppp()`, performs a test of CSR. If X is the data point pattern, then

```
> cdf.test(X, covariate, test="ks")
```

performs the Kolmogorov-Smirnov test of CSR, where `covariate` is the spatial covariate that will be used. Here `covariate` can be a pixel image, a `function(x,y)` in the R language, or one of the strings “x” or “y” indicating one of the Cartesian coordinates. The argument `test` is either “ks”, “cvm” or “ad” for the Kolmogorov-Smirnov, Cramér-Von Mises or Anderson-Darling test respectively.

For the *Beilschmiedia* data:

```

> elev <- bei.extra$elev
> cdf.test(bei, elev)
    Spatial Kolmogorov-Smirnov test of CSR in two dimensions

data: covariate 'elev' evaluated at points of 'bei'

```

```
and transformed to uniform distribution under CSR
D = 0.1063, p-value < 2.2e-16
alternative hypothesis: two-sided
```

The result of `cdf.test()` is an object of class "htest" (the standard R class for hypothesis tests) and also of class "cdftest" 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 (shown in Figure 6.22) displays the observed and expected distribution functions. The Kolmogorov-Smirnov test statistic is the maximum vertical separation between these functions.

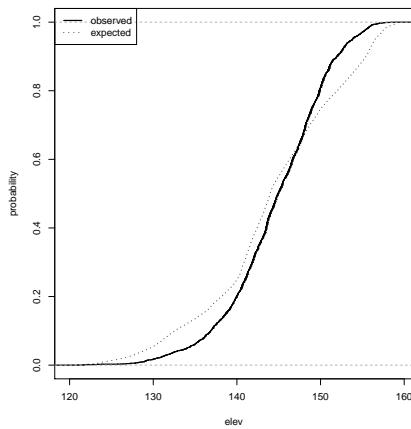


Figure 6.22: Plot of `cdf.test(besi, elev)`, illustrating the Kolmogorov-Smirnov test for dependence of the intensity of *Beilschmiedia* trees on terrain elevation.

The plot shows the two cumulative distribution functions $\hat{F}(z)$ and $F_0(z)$ plotted against z . Alternatively, selecting `style="PP"` in the plot command gives a P-P plot, showing $\hat{F}(z)$ plotted against $F_0(z)$; selecting `style="QQ"` gives a Q-Q plot, showing empirical quantiles $\hat{F}^{-1}(p)$ against theoretical quantiles $F_0^{-1}(p)$ for each probability $0 < p < 1$.

A warning message from `cdf.test()` about tied values (which does not arise in this example) indicates that several values z_i are equal. Typically this occurs because the covariate values have been rounded, so that several data points give the same observed value of the covariate. If the discretisation is severe, or if the covariate is an intrinsically discrete-valued variable or a factor, the Kolmogorov-Smirnov test is not supported by theory (and is ineffective because of tied values), and the χ^2 test based on quadrat counts would be preferable.

Choosing `covariate="x"` means the covariate $Z(x,y) = x$ is the x coordinate, so we are simply comparing the observed and expected distributions of the x coordinate. This is a useful strategy for testing the null hypothesis of homogeneity against the alternative of a large-scale spatial trend, especially when there are no covariate data. A warning message about tied values typically occurs if the coordinates in the dataset have been rounded.

```
> cdf.test(swedishpines, "x")
Spatial Kolmogorov-Smirnov test of CSR in two dimensions

data: covariate 'x' evaluated at points of 'swedishpines'
and transformed to uniform distribution under CSR
D = 0.0884, p-value = 0.604
alternative hypothesis: two-sided
```

6.7.3 Berman's tests

Berman [54] proposed two tests for the dependence of a point process on a spatial covariate, which have better performance than the K-S test against certain alternatives.

In Berman's Z_1 test, the test statistic is a standardised version of the *sum* of the observed values of the covariate at the data points:

$$Z_1 = \frac{\sum_i Z(x_i) - nM_1}{\sqrt{nM_2}} \quad (6.22)$$

where

$$M_k = \frac{1}{|W|} \int_W Z(u)^k du, \quad k = 1, 2.$$

Under the null hypothesis, Z_1 is asymptotically normal with mean 0 and variance 1. This test is optimal (uniformly most powerful) against the alternative hypothesis that the intensity is a loglinear function of Z . Closely-related tests were proposed independently by Waller et al [330] and Lawson [220], so this test is often termed the “Lawson-Waller” test in epidemiological literature.

In Berman's Z_2 test, the observed values z_i are first transformed to $u_i = F_0(z_i)$ where

$$F_0(z) = \frac{1}{|W|} \int_W \mathbf{1}\{Z(u) \leq z\} dz$$

is the spatial cdf of the covariate Z . Under the null hypothesis, these transformed values would be uniformly distributed. The test statistic is the standardised sum of these transformed values,

$$Z_2 = \sqrt{\frac{12}{n}} \left(\sum_i u_i - \frac{1}{2} \right).$$

Again Z_2 is asymptotically normal under the null hypothesis.

Berman's tests are performed in `spatstat` by the command `berman.test()`. The default is the Z_1 test.

```
> elev <- bei.extra$elev
> B <- berman.test(bei, elev)
> B
      Berman Z1 test of CSR in two dimensions

data: covariate 'elev' evaluated at points of 'bei'
Z1 = -0.7292, p-value = 0.4659
alternative hypothesis: two-sided
```

The result of `berman.test()` is another hypothesis test object of class "htest"; the text above was generated by the `print()` method. The result also belongs to the special class "bermantest" for which there is a `plot()` method. The plot is identical to a plot of the result of `cdf.test()` except for the addition of two vertical lines which show the mean values of the empirical and null distributions.

In the example above, the *p*-value from the Berman-Lawson-Waller Z_1 test is 0.466 suggesting no evidence that *Beilschmiedia* density depends on terrain elevation, whereas the Kolmogorov-Smirnov test gave a *p*-value close to zero, suggesting strong evidence. The discrepancy between test outcomes is explained by Figure 6.22. The deviation between the two distribution curves is roughly symmetrical, so the two distributions have approximately equal means — that is, the average terrain elevation of a *Beilschmiedia* tree is roughly equal to the average terrain elevation in the study region. The Z_1 test statistic is a comparison between the two mean values, and is not sensitive to the type of deviation shown in Figure 6.22.

For a more searching investigation of the dependence of intensity on a covariate, the analyst would typically choose a class of parametric models for the intensity, and test hypotheses about the model parameters, as explained in Chapter 9.

When the covariate Z is the distance to a spatial pattern, another useful diagnostic is Foxall's J -function [148], available using `Jfox()`.

6.8 Hot spots, clusters, and local features

It is good practice to examine any spatial point pattern dataset for anomalies. Their presence can skew the results of statistical analysis if not handled appropriately.

Often it is desired to detect anomalies in the intensity of a point process. A *hot spot* in a point process is a zone of elevated intensity. In archaeology, it often happens that ancient artefacts such as pottery fragments are much more abundant in one area than another. In seismology, earthquake epicentres are typically highly concentrated along a plate boundary. Old minefields can sometimes be recognised by their elevated density of metal fragments.

In spatial epidemiology one of the important goals is to detect spatial “clusters” of disease cases. A cluster is a group of cases, lying close together, which are more numerous or more dense than expected, relative to the background pattern of such cases. A confirmed cluster (that is, one which is judged statistically significant) could be explained by a common source of infection or toxicity, by contagion in the local population, or by observer effects such as increased vigilance by the health services. A source of toxicity would lead to an *elevated intensity* of disease cases near the source, while contagion would cause *positive correlation* in disease status between neighbouring people. It can be difficult, contentious and even impossible [51] to distinguish between these two effects. Thus, a hot spot of disease cases is only one possible kind of disease cluster — although it can be difficult to distinguish it from other kinds.

[118, 208, 221, 295, 344, 60, 78, 110, 115, 153]

Detecting and locating hot spots is an ill-defined or open-ended problem. The optimal choice of detection method will depend on the kind of anomaly we are looking for: this choice depends heavily on prior information and subject knowledge.

Exploratory techniques for investigating localised features in a point pattern include scan statistics, model-based clustering, cluster set estimation, nearest-neighbour cleaning, and data sharpening.

The simplest place to start is with a kernel estimate of the intensity function. Zones of elevated intensity are often recognisable in an image plot of the kernel estimate.

The left panel of Figure 6.23 shows the California redwood seedlings and saplings data (Ripley's [281] subset of a dataset of Strauss [316]). Clusters of points, clearly visible in the plot, are thought to have arisen by dispersal of seeds from a parent tree which has later died [316]. Depending on the purpose of analysis, it might be appropriate to locate the positions of these clusters. The right panel of Figure 6.23 shows a kernel estimate of intensity:

```
> denRed <- density(redwood, bw.ppl, ns=16)
```

(The argument `ns` is passed to `bw.ppl()`.) We selected the bandwidth by likelihood cross-validation (`bw.ppl()`) as in our experience this tends to select more appropriate values than other methods when the pattern consists *predominantly* of tight clusters. However to detect a single tight cluster in the midst of random noise, mean square cross validation (`bw.diggle()`) often seems to work best.

To determine whether such a zone is ‘significant’ we might use a Monte Carlo test:

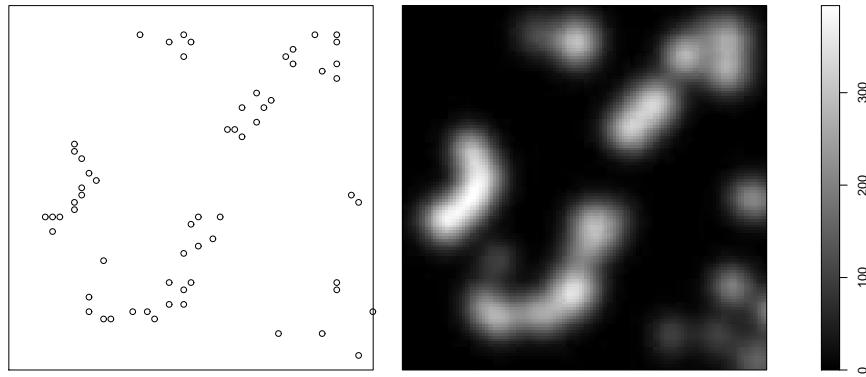


Figure 6.23: California redwood seedlings and saplings (Ripley's subset). *Left*: point pattern scaled to unit square. *Right*: kernel estimate of density, with bandwidth selected by likelihood cross-validation.

```

> obsmax <- max(denRed)
> simmax <- numeric(99)
> lamRed <- intensity(redwood)
> winRed <- as.owin(redwood)
> for(i in 1:99) {
  Xsim <- rpoispp(lamRed, win=winRed)
  denXsim <- density(Xsim, bw.ppl, ns=16)
  simmax[i] <- max(denXsim)
}
> pval <- (1+sum(simmax > obsmax))/100

> pval
[1] 0.01

```

A similar idea is the *scan test* [207, 208]. At each spatial location u , we draw a circle centred at u with a chosen radius r , denoted $b(u, r)$, and count the numbers of points inside and outside the circle, $n_{in} = n(\mathbf{x}_{b(u,r)})$ and $n_{out} = n(\mathbf{x}_{W \setminus b(u,r)})$. Assuming the point process is Poisson, we test the null hypothesis that the intensity is homogeneous, against the alternative that the intensity is different inside and outside the circle. The likelihood ratio test statistic is

$$\Gamma(u, r) = 2n_{in} \log(n_{in}/A_{in}) + 2n_{out} \log(n_{out}/A_{out}) - 2n(\mathbf{x}) \log(n(\mathbf{x})/\text{area}(W))$$

where $A_{in} = \text{area}(W \cap b(u, r))$ and $A_{out} = \text{area}(W \setminus b(u, r))$ are the areas of the regions inside and outside the circle.

Figure 6.24 shows an image of $\Gamma(u, r)$ as a function of the centre location u , for a fixed value of r . This was computed by

```
> LR <- scanLRTS(redwood, r = 2 * bw.ppl(redwood))
```

The radius r was chosen so that the circle is comparable in size to the Gaussian kernel with

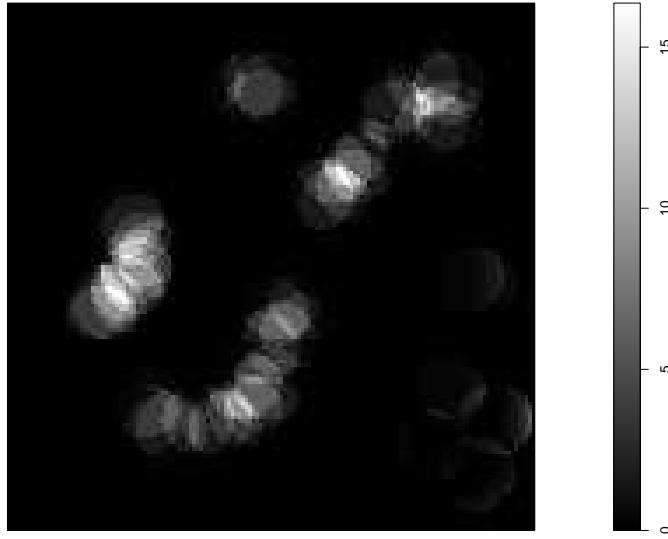


Figure 6.24: Image of likelihood ratio test statistic $\Gamma(u, r)$ for fixed r , computed for the redwood data.

bandwidth selected by likelihood cross-validation.³ For fixed values of u and r , the null distribution of $\Gamma(u, r)$ is approximately χ^2 with 1 degree of freedom. The corresponding p -values can be computed by hand:

```
> pvals <- eval.im(pchisq(LR, df=1, lower.tail=FALSE))
```

Figure 6.25 shows the resulting p -values, and the set of locations where the p -value is less than 0.01.

For fixed u and r , the likelihood ratio test is theoretically optimal for detecting a difference in intensity inside and outside the circle $b(u, r)$.

The *scan statistic* Γ^* is the maximum value of $\Gamma(u, r)$ over all locations u in the window, and optionally over a range of different values of r . The null distribution of Γ^* is more complicated than in the case of fixed r [8, 231]. The command `scan.test()` performs a Monte Carlo test.

Sometimes it is suspected that the spatial domain can be divided into two distinct regions of low and high intensity respectively. An artificial example is shown in the left panel of Figure 6.26.

Specialised estimators of the high-intensity region are available for this situation. The nonparametric maximum likelihood estimator is the Allard-Fraley [7] cluster set. The Dirichlet tessellation induced by the point pattern is computed; the k smallest tiles (ranked by tile area) are identified, where k is determined by maximising a likelihood criterion analogous to the likelihood ratio test statistic described above; the union of these k smallest tiles is the cluster set.

The right panel of Figure 6.26 shows the result of `clusterset(X, what="domain")` where X is the point pattern in the left panel of the Figure.

The set estimate tends to be slightly smaller than the true set (because the Dirichlet tiles associated with the points closest to the boundary of the true set often have relatively large area) apart from

³The bandwidth of a kernel is defined as the standard deviation, or in higher dimensions the root-mean-square length (square root of the expected sum of squared coordinate values), of the distribution represented by the kernel. The expected sum of squares of an isotropic Gaussian vector with mean zero and standard deviation σ is $2\sigma^2$ so the bandwidth is $\sqrt{2}\sigma$. The expected squared length of a random vector uniformly distributed in a disc of radius r is $r^2/2$ so the bandwidth is $r/\sqrt{2}$. The bandwidths match if $r = 2\sigma$.

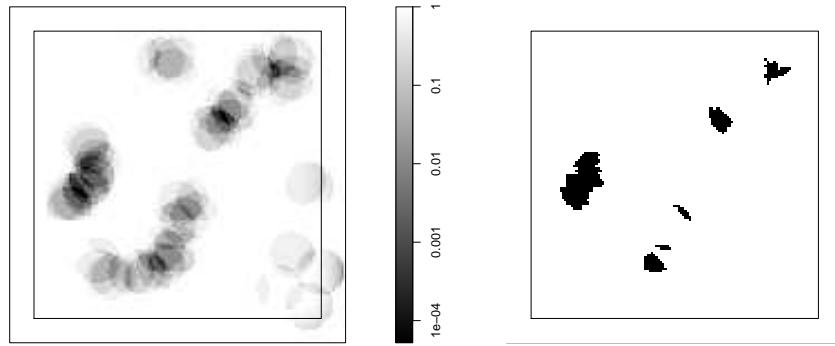


Figure 6.25: Scan test with fixed circle radius. *Left*: p -values of likelihood ratio test statistic, with logarithmic colour scale. *Right*: locations where $p < 0.01$.

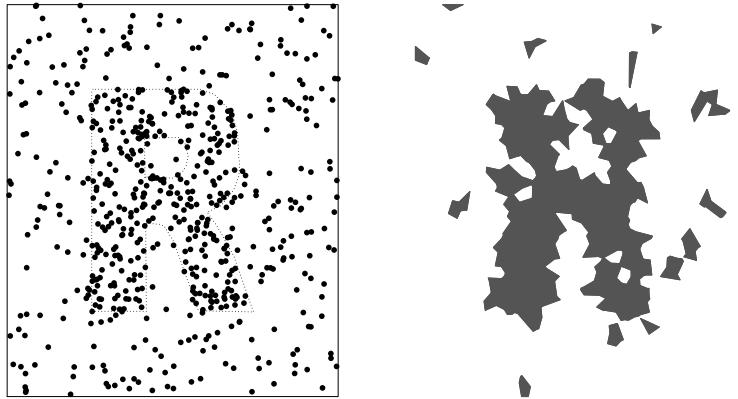


Figure 6.26: Artificial example of high-intensity region. *Left*: simulated data from Poisson process with intensity 100 inside the letter R, and intensity 20 outside. *Right*: Nonparametric (Allard-Fraley) estimate of high-intensity region.

including small isolated fragments outside the true set (because of random noise). These artefacts can be mitigated by constraining the set estimate to be a connected set; Allard and Fraley propose a more complex optimization algorithm [7] which is not yet implemented in `spatstat`.

If it is a requirement that the estimated set be connected, a fast alternative to constrained optimization of the likelihood, canvassed in the the GIS literature under the heading of ‘regionalisation’ [166, 165] is to apply hierarchical clustering methods to the tiles, after constructing an appropriate measure of dissimilarity between tiles.

In large datasets, computation of the Dirichlet tessellation, the tile areas, and the union of tiles, can be expensive. There is a `fast=TRUE` option to `clusterset()` which uses discretisation to accelerate the calculations.

A quick and useful alternative is to compute the distance from each point to its nearest neighbour. *Nearest-neighbour cleaning* [75] groups the points into two classes — ‘feature’ and ‘noise’ — on the basis of their nearest-neighbour distances. In a homogeneous Poisson process of intensity λ , the

distance D_k from a typical point to its k th-nearest neighbour is such that the disc area $A_k = \pi D_k^2$ has a gamma distribution with rate λ and shape k . If we suspect that there are two regions with different intensities, one strategy is to calculate A_k for each data point, and estimate the two intensities λ_1, λ_2 by fitting a two-component mixture model. The model states that each A_k value is drawn either from the $\text{Gamma}(\lambda_1, k)$ distribution (with probability p) or from the $\text{Gamma}(\lambda_2, k)$ distribution (with probability $1 - p$). Using the E-M algorithm, we estimate the unknown parameters λ_1, λ_2, p . From the fitted mixture model we can also estimate the probability that each observation A_k belongs to the first or second component.

The `spatstat` command `nnclean` performs nearest-neighbour cleaning using the code from [75]. Calling

```
> Z <- nnclean(X, k=10, plothist=TRUE)
```

fits the mixture model, prints a report, and generates the diagnostic plot in the left panel of Figure 6.27, in which the probability density of the fitted mixture model is superimposed on the histogram of observed values of the tenth-nearest neighbour distance. The resulting point pattern Z is identical to X except for two columns of marks: `class`, shown in the middle panel of Figure 6.27, a factor which classifies the points into ‘feature’ (+) and ‘noise’ (·); and `prob`, shown in the right panel, giving the fitted probability that the point belongs to the cluster component of the mixture model. The choice of $k = 10$ was arbitrary; the optimal choice of k depends on characteristics of the pattern, so k is generally chosen by trial and error [75].

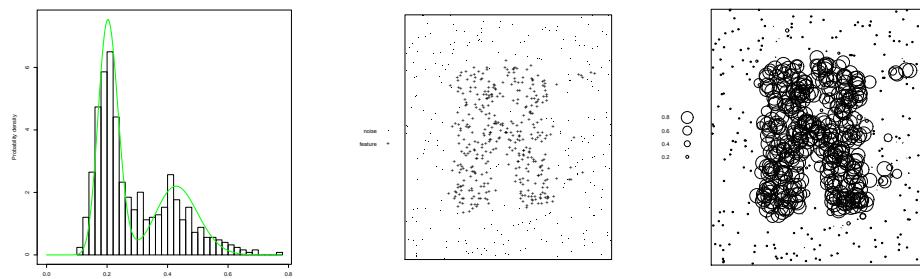


Figure 6.27: Nearest-neighbour cleaning for artificial example of high-intensity region. *Left:* histogram of 10th nearest neighbour distances, and fitted density of mixture model. *Middle:* classification of data points into ‘noise’ (·) and ‘feature’ (+) classes. *Right:* data points marked by their fitted probability of belonging to the ‘feature’ class.

An extreme example of high concentration of intensity is the South Pacific earthquakes data shown in the right panel of Figure 6.2 on page 122. This is a projection of the dataset `quakes` in the standard package `datasets`:

```
> require(datasets)
> qk <- ppp(quakes$long, quakes$lat, c(164, 190), c(-39, -10))
```

We used the `mapdata` and `maps` packages to plot the Pacific islands.

The different bandwidth selection procedures disagree widely: `bw.diggle()` gives a bandwidth of 0.108 while `bw.ppl()` gives 0.343 and `bw.scott()` gives (1.92, 1.59). This is a common feature of such highly concentrated patterns. We shall arbitrarily take a bandwidth of 0.5:

```
> dq.5 <- density(qk, 0.5)
```

To obtain comparable results from quadrat counting with hexagonal tiles, the tile size should be chosen to match the bandwidth of the kernel smoother. The mean squared length of a random point

in a hexagon of side length s centred at the origin is $(5/12)s^2$. Setting $(5/12)s^2 = 2\sigma^2$ gives $s = \sqrt{24/5}\sigma \approx 2.19\sigma$. With $\sigma = 0.5$ we get $s = 1.09$.

```
> ht.5 <- hextess(as.owin(qk), 1.09)
> hq.5 <- intensity(quadratcount(qk, tess=ht.5), image=TRUE)
```

The results are shown in Figure 6.28.

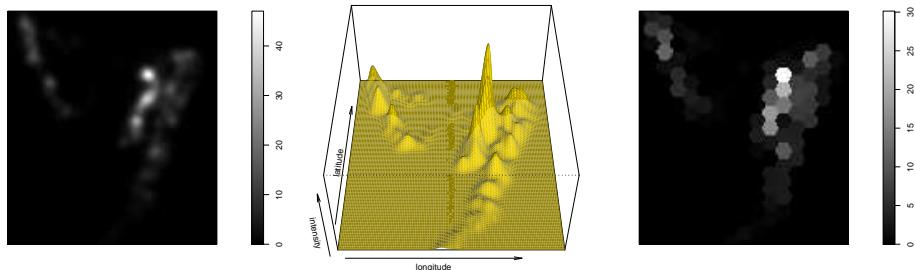


Figure 6.28: Intensity estimates for South Pacific earthquakes. *Left and Middle*: kernel smoothing, isotropic Gaussian kernel with standard deviation 0.5 degrees. *Right*: intensity estimates on hexagonal quadrats of side length 1.09 degrees.

Figure 6.29 shows the Allard-Fraley estimator computed by `clusterset(qk, what="domain")`.

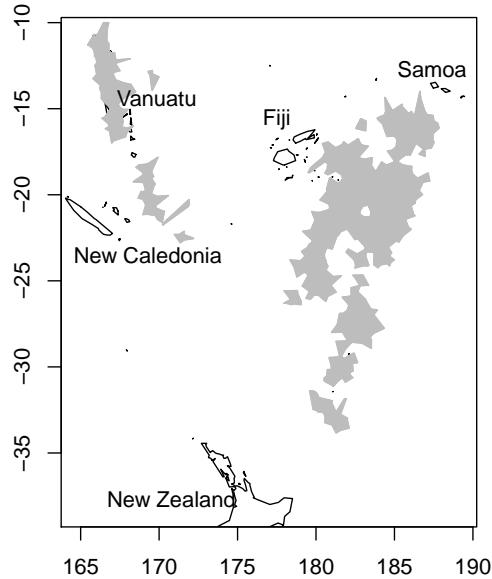


Figure 6.29: Allard-Fraley cluster set estimate for South Pacific earthquakes.

The left panel of Figure 6.30 is a diagnostic plot produced by the command `nnclean(qk, k=5, plothist=TRUE)`. It shows a histogram of the fifth-nearest-neighbour distances (with $k = 5$ chosen by trial and error) and the density of the fitted mixture model. The right panel is obtained by plotting the return value of `nnclean()`.

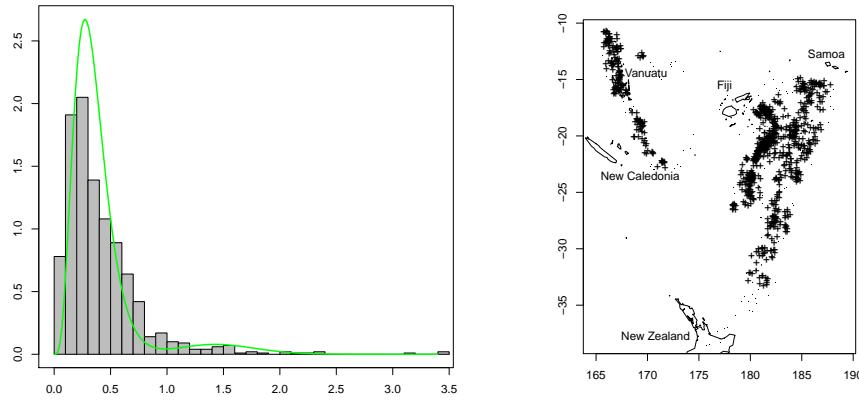


Figure 6.30: Nearest-neighbour cleaning for South Pacific earthquakes. *Left:* observed and fitted distribution of 5th nearest neighbour distances. *Right:* earthquake locations classified into cluster (+) and noise (.) .

[47]

[225]

The shapley dataset, shown in the left panel of Figure 6.31, is an example of a point pattern which is clearly not homogeneous. The data comes from a radioastronomical survey of galaxies in the Shapley Galaxy Concentration: each point is a galaxy in the distant universe. There are very dense concentrations of galaxies in some parts of the survey area.

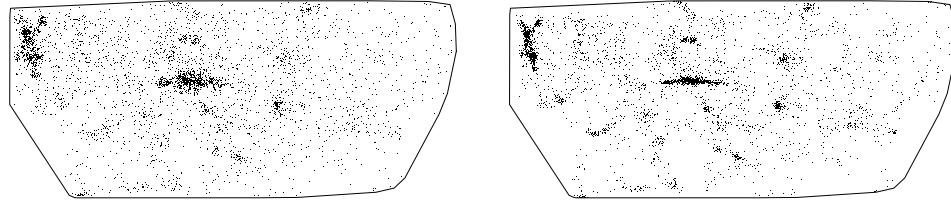


Figure 6.31: Shapley Galaxy Concentration survey dataset. *Left:* original data. *Right:* result of Choi-Hall data sharpening.

In Choi-Hall *data sharpening* [83] the points effectively exert a force of attraction on each other, and are allowed to move in the direction of the resultant force. This tends to enhance tight linear concentrations of points. The left panel of Figure 6.31 shows the result of applying the `spatstat` function `sharpen()`:

```
> Y <- sharpen(unmark(shapley), sigma=0.5, edgecorrect=TRUE)
```

Another approach to detecting local features is LISA (Local Indicators of Spatial Association) methods, in which a summary statistic is separated into contributions from each of the data points. For example the K function is expressed as a sum of the *local K functions* of each of the data points. These local functions are then compared, and classified into several groups of functions, perhaps using principal component analysis [12, 106, 105].

The `spatstat` functions `localK`, `localL`, `localpcf` compute local versions of the K -function, L -function and pair correlation function, respectively.

6.9 Kernel smoothing of marks

It is often useful to apply spatial smoothing to the *mark values* attached to the points of a point pattern.

The left panel of Figure 6.32 shows the locations of 584 Longleaf Pine (*Pinus palustris*) trees in a 200×200 metre region [271]. Circle diameters are proportional to each tree's diameter at breast height (dbh), a convenient surrogate measure of size and age. There appear to be some areas where the trees tend to be younger than in other areas. To investigate this we need to compute a spatially-varying average diameter of the trees in each neighbourhood.

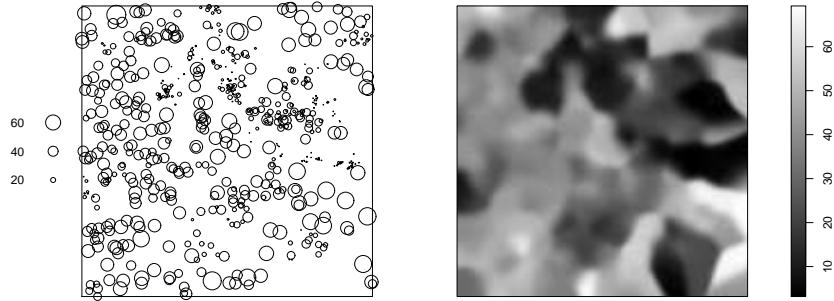


Figure 6.32: Smoothing of mark values. *Left:* Longleaf Pines data: tree locations, marked by tree diameter at breast height, in a 200 metre square plot. Diameters are not to scale. *Right:* spatially-varying average tree diameter (Nadaraya-Watson smoother of mark values) in centimetres.

Suppose the data are points x_1, \dots, x_n with corresponding marks m_1, \dots, m_n which are real numbers. The *Nadaraya-Watson smoother* [253, 336, 254] \tilde{m} is the spatial function

$$\tilde{m}(u) = \frac{\sum_i m_i \kappa(u - x_i)}{\sum_i \kappa(u - x_i)} \quad (6.23)$$

for each spatial location u . The corresponding Diggle-corrected version is

$$\tilde{m}(u) = \frac{\sum_i m_i \kappa(u - x_i) / e(x_i)}{\sum_i \kappa(u - x_i) / e(x_i)} \quad (6.24)$$

where $e(x_i)$ is the edge correction factor defined in (6.10). The function $\tilde{m}(u)$ can be taken as an estimate of the spatially-varying average mark value.

The `spatstat` package has a generic function `Smooth()` with a method `Smooth.ppp()` for point patterns. The syntax of `Smooth.ppp()` is similar to `density.ppp()` and its return value is again a pixel image. The right panel of Figure 6.32 shows the result of `Smooth.ppp()` applied to the Longleaf Pines data, with the smoothing bandwidth selected by least-squares cross-validation using the command `bw.smoothppp()`. The result strengthens the conclusion that there is a swath of younger trees across the top right quarter of the plot.

For very large values of the smoothing bandwidth, the result of `Smooth.ppp()` will become approximately constant and equal to the average mark value in the entire dataset. As bandwidth goes to zero, the expressions (6.23) and (6.24) converge to the mark value of the nearest data point (the data point x_i that is closest to the querying location u).

Various other quantities can be calculated from the smoothed marks. The kernel-smoothed *mark variance* is

$$v(u) = \tilde{m}_2(u) - (\tilde{m}(u))^2 \quad (6.25)$$

where $\tilde{m}_2(u)$ is the Nadaraya-Watson smoother of the squared marks m_i^2 . The mark variance is a measure of variability amongst the marks in the neighbourhood of the location u . If there is a smooth gradient in the mark value, the corresponding standard deviation $\sqrt{\tilde{m}_2(u)}$ is an estimate of the gradient. The mark variance is computed in *spatstat* by *markvar()*. To obtain the standard deviation:

```
> mvar <- markvar(longleaf, bw.smoothppp)
> msd <- eval.im(sqrt(mvar))
```

The result is plotted in the left panel of Figure 6.33. The highest values of standard deviation appear mainly at the borders of areas where the trees are relatively young.

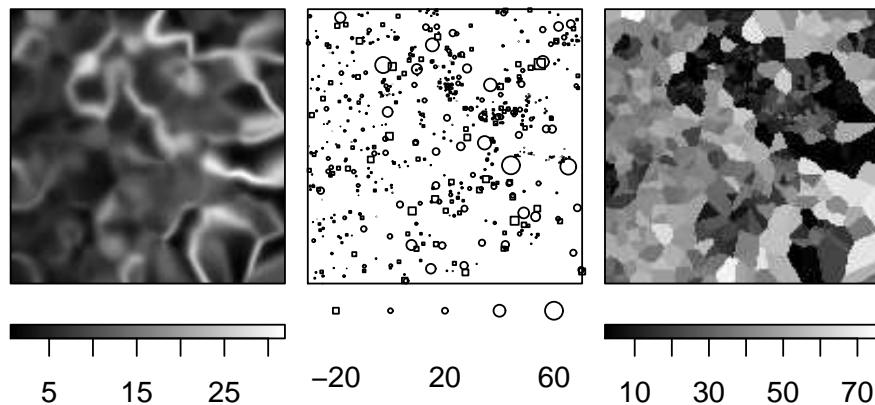


Figure 6.33: Additional smoothing products for Longleaf Pines data. *Left*: Local standard deviation of tree diameter. *Middle*: Residual diameters. Circles and squares represent positive and negative residuals, respectively. *Right*: Nearest-neighbour interpolation of diameters.

The residual $r_i = m_i - \tilde{m}_{-i}(x_i)$ is the difference between the mark value at a data point and the predicted value given by the average of the marks of neighbouring points other than x_i . It may be useful in detecting anomalies. The middle panel of Figure 6.33 shows the Longleaf Pines data marked by their residuals r_i computed by

```
> mfit <- Smooth(longleaf, bw.smoothppp, at="points")
> res <- marks(longleaf) - mfit
```

Note that *mfit[i]* is the leave-one-out estimate $\tilde{m}_{-i}(x_i)$ because of the default settings in *Smooth.ppp()*.

Nearest-neighbour interpolation, in which the value $m(u)$ is estimated by taking the mark m_i of the data point x_i that is closest to the location u , can be performed in *spatstat* by *nnmark()*. The result of *nnmark()* for the Longleaf Pines is shown in the right panel of Figure 6.33.

6.10 Multitype intensity and relative risk

If there are several different types of points, it may be the *relative* intensity of the different types that is of interest. A very important example is a spatial *case-control* study, giving the spatial locations of a set of disease cases, and of a separate set of controls (notionally a random sample from the population at risk of the disease). Then the ratio of the intensities of the two point processes is an indication of the spatially-varying risk of disease.

In the context of a case-control study, let D be the point process of disease cases and C the point process of controls. Assume that C is an independent random sample (with unknown sampling fraction f) from the population at risk. Assuming the population at risk is large, and has spatially-varying density $h(u)$ people per square kilometre, the locations of the controls C are approximately a Poisson point process with intensity $\lambda_C(u) = fh(u)$.

The hypothesis of *constant risk* states that each person in the population has the same probability p of contracting the disease. Under this hypothesis, D has intensity function $\lambda_D(u) = ph(u)$. The more general hypothesis of *spatially-varying risk* states that a person living at spatial location u has probability $p(u)$ of contracting the disease, where $p(u)$ is some function to be estimated. In this case, D has intensity function $\lambda_D(u) = p(u)h(u)$. Comparing expressions for λ_D and λ_C we find that

$$\frac{\lambda_D(u)}{\lambda_C(u)} = \frac{1}{f} p(u) \quad (6.26)$$

that is, the ratio of the intensity of cases to the intensity of controls is proportional to the spatially-varying risk of disease.

Note that we did not say the pattern of disease cases D is a Poisson process; this would only be true if different people's health outcomes are stochastically independent (in particular, not dependent on the disease status of family members, etc) and is probably not true for infectious diseases. The beauty of analysing the intensity is that it is a first-order (mean value) property of the point process, so that the equations for λ_D and λ_C above remain true even when disease cases are spatially correlated.

The customary estimator of the intensity ratio $r(u) = \lambda_D(u)/\lambda_C(u)$ is the 'plug-in' estimator obtained by substituting the kernel estimators of the numerator and denominator. Suppose the disease case locations are y_1, \dots, y_n and the control locations are x_1, \dots, x_m . If we choose Diggle's correction, and use the same bandwidth to estimate the numerator and denominator, then the estimator is

$$\tilde{r}(u) = \frac{\sum_j \kappa(u - y_j)/e(y_j)}{\sum_i \kappa(u - x_i)/e(x_i)}.$$

A related, but slightly different, approach is to combine the cases and controls into a single point process X , and to study

$$q(u) = \frac{\lambda_D(u)}{\lambda_D(u) + \lambda_C(u)}.$$

Then $q(u)$ is the probability of a case (rather than a control) given that there is a point of the combined process X at the location u . The corresponding plug-in estimator is

$$\tilde{q}(u) = \frac{\sum_j \kappa(u - y_j)/e(y_j)}{\sum_i \kappa(u - x_i)/e(x_i) + \sum_j \kappa(u - y_j)/e(y_j)}.$$

This can also be interpreted as a Nadaraya-Watson smoother of the combined point pattern with marks 1 and 0 corresponding to cases and controls, respectively.

Moving away from the case-control setting, we now suppose there are m different types of points. Let X_j be the point process consisting of the points of type j , and X_\bullet the point process of all

points regardless of type. Write $\lambda_j(u)$ for the intensity of X_j and $\lambda_\bullet(u)$ for the intensity of X_\bullet . Then we estimate the *relative risk*

$$p_j(u) = \frac{\lambda_j(u)}{\lambda_\bullet(u)}$$

by the ratio of the corresponding kernel estimates of intensity, which is equivalent to a Naradaya-Watson smoother of the indicator of type j .

For bandwidth selection, in the general case, consider the indicators z_{ij} which equal 1 when x_i belongs to type j , and equal 0 otherwise. For a particular value of smoothing bandwidth, let $\hat{p}_j(u)$ be the estimated probabilities that a point at location u will belong to type j . Then the bandwidth can be chosen to minimise either the likelihood, the squared error, or the standardised squared error, of the indicators z_{ij} relative to the fitted values $\hat{p}_j(x_i)$.

To estimate relative risk in `spatstat` the data should be represented as a multitype point pattern. The command `relrisk()` estimates the spatially-varying probability of each type of point, using kernel smoothing. The default smoothing bandwidth is selected by likelihood cross-validation.

Bandwidth selection can be performed by the function `bw.relrisk()` which supports likelihood cross-validation and least squares cross-validation. The result of `bw.relrisk()` can be printed to show the optimal bandwidth value, or plotted to show the objective function.

Figure 6.34 shows the `lansing` dataset, a survey of a 20-acre region in Lansing Woods, Michigan giving the location and botanical classification of 2251 trees. Figure 6.35 shows the same data separated by species using `split()`. The main question is whether the species are *segregated* in the sense that there are regions of the woods where one species predominates.

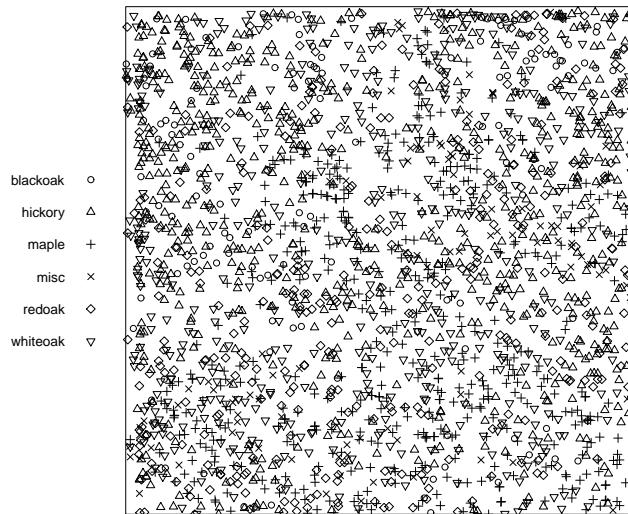


Figure 6.34: Lansing Woods data: locations and botanical classification of 2251 trees in a survey region 924 feet (282 metres) across.

After typing

```
> b <- bw.relrisk(lansing)
```

simply printing `b` yields the numerical value of the optimal bandwidth, 0.0488, while typing `plot(b)` generates the display in the left panel of Figure 6.36. The right panel is obtained by adding the argument `xlim` to the plot command to narrow the range of the `x` axis.

The smoothing estimates of the probabilities of each species are then obtained by

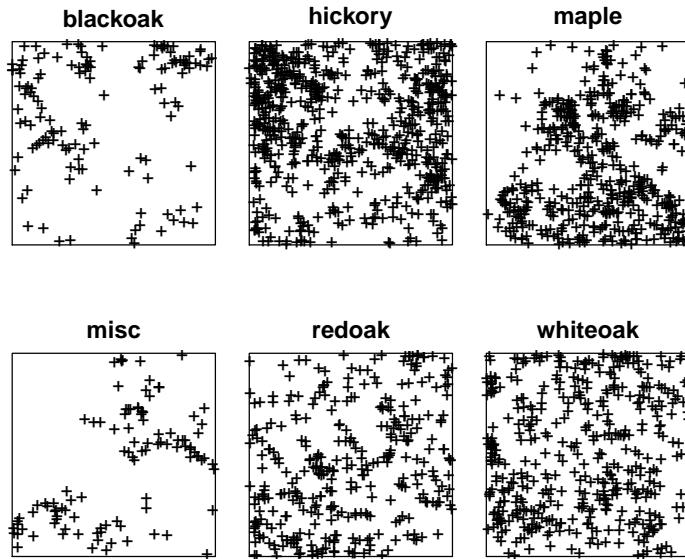


Figure 6.35: Lansing Woods data separated by species.

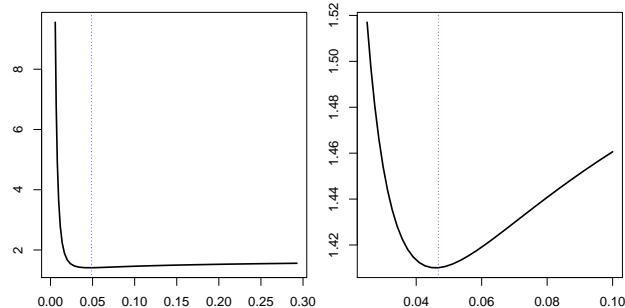


Figure 6.36: Plots of the optimization criterion for bandwidth selection in `bw.relrisk()` for the Lansing Woods data. *Left:* full range of bandwidths. *Right:* narrower range of bandwidths around the optimal value.

```
> rr <- relrisk(lansing, sigma=b)
```

which is plotted in Figure 6.37.

The following code determines, for each spatial location u , which species of tree has the highest probability at this location — effectively dividing the forest into regions where different species predominate.

```
> dominant <- im.apply(rr, which.max)
> species <- levels(marks(lansing))
> dominant <- eval.im(factor(dominant, levels=1:6,
  labels=species))
```

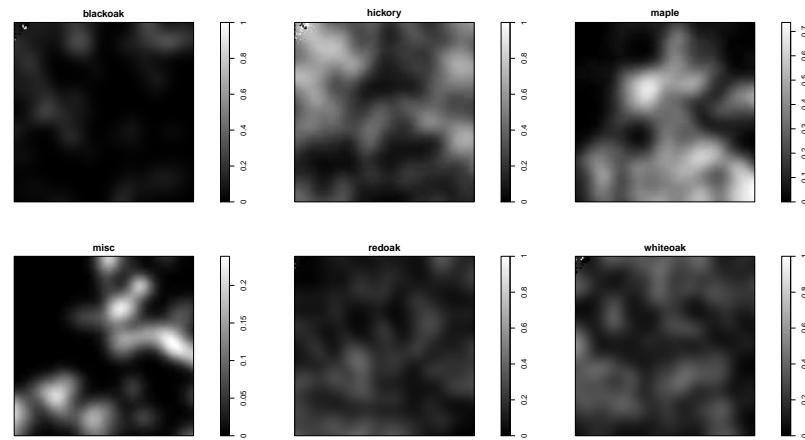


Figure 6.37: Estimates of spatially-varying proportions of each species in Lansing Woods, obtained using `relrisk()`.

The result is shown in Figure 6.38. This is a pixel image with factor values, plotted using the command `textureplot()` to show each level of the factor as a different texture.

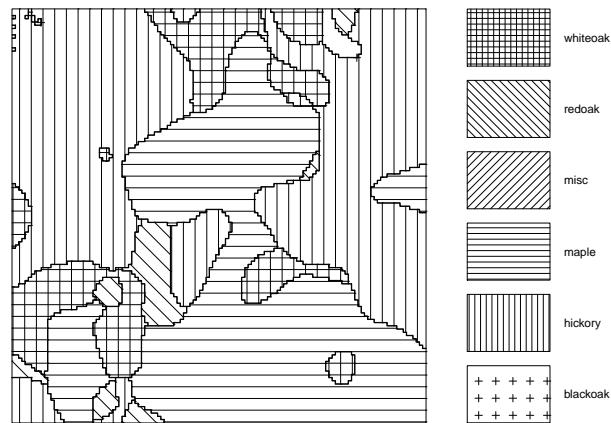


Figure 6.38: Most likely species at each location in Lansing Woods.

6.11 Intensity in 3D space and space-time

6.12 FAQ's

- Difference between intensity and density.
- Difference between intensity estimation and smoothing interpolation of marks.
- Zero or negative (!) values of estimated intensity.
-

7

Correlation

7.1 Introduction

Often the motivation for analysing point pattern data is to determine whether the points appear to have been placed independently of each other, or whether they exhibit some kind of inter-point dependence.

Figure 7.1 shows three archetypal point patterns representing ‘regularity’ (where points tend to avoid each other), ‘independence’ (complete spatial randomness) and ‘clustering’ (where points tend to be close together).

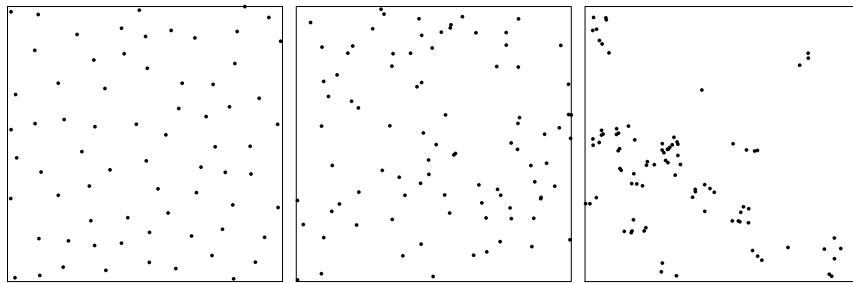


Figure 7.1: Classical trichotomy between regular (*Left*), independent (*Middle*) and clustered (*Right*) point patterns. All three patterns are in the unit square.

A standard statistical tool for measuring dependence is *correlation*, or more generally *covariance*. In this chapter we explain how to define and measure covariance in a point process, in such a way that (roughly speaking) the clustered point pattern in the right panel of Figure 7.1 has positive covariance, the completely random pattern in the middle panel has zero covariance, and the regular pattern in the left panel has negative covariance.

In statistical theory, correlation is classified as a *second moment* quantity. The “first moment” of a random variable X is its mean value; the “second moment” or “mean square” is the mean of X^2 . Together the first and second moments of random variables determine important quantities such as variance, standard deviation, covariance and correlation.

Second moment quantities for point processes are intimately related to counting *pairs* of points, or adding up contributions from each pair of points in the process. In a point process \mathbf{X} , the squared point count $n(\mathbf{X}_B)^2$ can be interpreted as the number of *pairs of points* x_i, x_j in \mathbf{X} which fall in the

nominated set B . The second moment of $n(\mathbf{X}_B)$ is the expected number of pairs of points falling in B .

This chapter covers numerous methods for analysing the second moment (correlation or covariance) structure of point processes. The most popular tools are the *pair correlation function* and Ripley's *K-function*. To give insight into these methods, we start by describing two simple, manual techniques which motivate the more advanced methods.

Correlation has the great virtue of being easy to calculate and handle, and is a powerful tool for the data analyst. There are two important caveats. First, accurate measurement of correlation requires faithful estimation of the mean (first moment) if we are to avoid problems of spurious correlation and confounding. In the point process context, this means that we must have good knowledge of the intensity before we can trust the correlation. Second, the correlation is merely a summary index of statistical association, not a characterisation of dependence or causality: "correlation is not causation". Using only correlations, we cannot discriminate between different possible causes of spatial clustering.

7.2 Manual methods

We start by describing two simple, manual techniques which motivate the more advanced methods.

7.2.1 Morisita index

If the observation window is a rectangle, a simple strategy for assessing spatial correlation is to subdivide the window into rectangular quadrats of equal size, and to count how often a pair of data points falls in the same quadrat. If there are n data points altogether, there are $n(n - 1)$ ordered pairs of distinct points. If there are m quadrats containing n_1, \dots, n_m points respectively, then there are $n_j(n_j - 1)$ ordered pairs of distinct points in the j th quadrat. The total number of ordered pairs of distinct points which fall inside the same quadrat is thus $\sum_j n_j(n_j - 1)$. The ratio

$$\frac{\sum_j n_j(n_j - 1)}{n(n - 1)}$$

is the fraction of all pairs of data points in which both points fall in the same quadrat. In a completely random (homogeneous Poisson) process, where points are independent of each other, two points fall in the same quadrat with probability $1/m$, where m is the number of quadrats, so the fraction above should equal $1/m$ on average. The ratio of the observed and expected fractions is the *Morisita index* [252]

$$M = m \frac{\sum_j n_j(n_j - 1)}{n(n - 1)}. \quad (7.1)$$

This index should be close to 1 if the points are independent, greater than 1 if they are clustered, and less than 1 if they are regular.

Repeating this calculation using different subdivisions of the window into quadrats, and plotting the Morisita index against the diameter of the quadrats, yields a *Morisita index plot*. Figure 7.2 shows that this has the ability to distinguish between the three archetypal point patterns in Figure 7.1. In `spatstat` the Morisita index plot of a point pattern X is generated by `miplot(X)`.

The Morisita index is closely related to the *index of dispersion* for quadrat counts. Suppose we calculate the sample variance of the quadrat counts,

$$s^2 = \frac{1}{m-1} \sum_{j=1}^m (n_j - \bar{n})^2$$

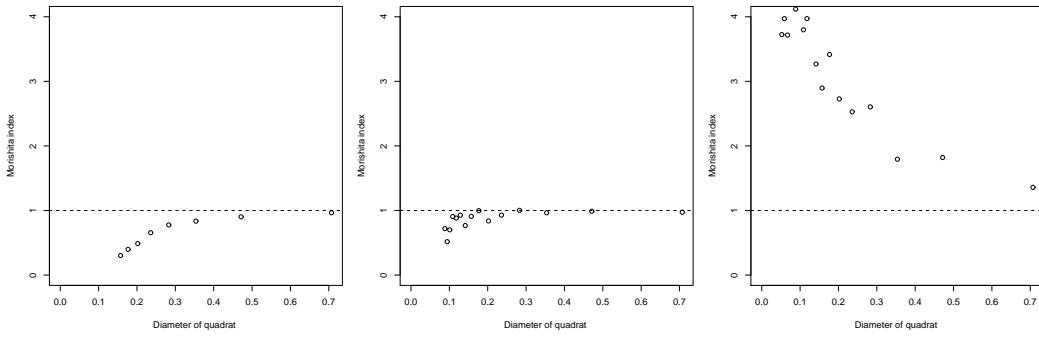


Figure 7.2: Morisita Index Plots for the three patterns in Figure 7.1.

where \bar{n} is the average quadrat count, $\bar{n} = n/m$. If the point process is Poisson, then the counts n_1, \dots, n_m are observations of independent Poisson random variables with the same mean μ . The variance of a Poisson random variable is equal to its mean. A standard index of overdispersion or underdispersion for count variables is the sample variance divided by the sample mean: for a Poisson distribution this ratio should be approximately equal to 1. Dividing the sample variance s^2 by the sample mean \bar{n} gives the *index of dispersion*

$$I = \frac{s^2}{\bar{n}} = \frac{m}{n(m-1)} \sum_{j=1}^m (n_j - \frac{n}{m})^2.$$

This is algebraically related to the Morisita index by $I = (m/(m-1))[(n-1)M - (n/m-1)]$. As discussed in Section 6.4.3 on page 131, the index of dispersion is also closely related to the χ^2 test of CSR based on quadrat counts.

Note especially that the Morisita index *assumes the intensity is homogeneous*. If this is not the case, large values of M could arise simply from spatial inhomogeneity, rather than from some form of correlation between the points.

The Morisita index is not sensitive to subtle differences in spatial scale, because it is based on subdividing the observation window coarsely into quadrats. This suggests that we look for better ways to summarise the information about pairs of points.

There is also something unsatisfactory about the theoretical derivation of the Morisita index given above. That derivation referred only to the homogeneous Poisson point process, and calculated that the Morisita index should be about 1 for that process. But if the data were generated by another kind of point process, it is unclear (at least from the previous discussion) how to interpret the value of the Morisita index. Indeed the Morisita index might not even be a well-defined property of the point process: for example, it might depend on the size of the observation window. Although these questions can be resolved, the answers are not very simple.

A good statistical index should not only be accessible by simple direct calculation from the data, but it should have a simple, direct interpretation for the point process which generated the data. In the rest of the chapter we look for such indices.

7.2.2 Fry plot

More information about spacings in the point pattern can be obtained using a *Fry plot* or Patterson plot. Originally developed for crystallography by Patterson [266, 267] this technique was independently reinvented in geophysics by Fry [150, 174].

A Fry plot can be drawn by hand, as follows (see Figure 7.3). First print the point pattern on a sheet of paper. Take a transparency or sheet of tracing paper, and mark a cross in the middle.

Place the transparency over the printout, so that the cross on the transparency lies on one of the data points. Copy the positions of all the other data points onto the transparency. Now move the cross to another data point, and repeat the copying process. After every data point has been visited, the pattern on the transparency is the Fry plot.

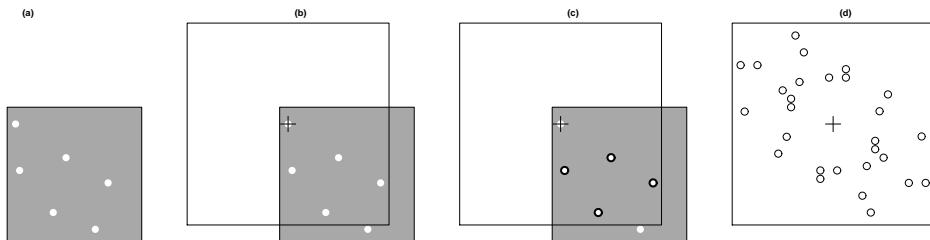


Figure 7.3: Stages in forming the Fry plot. (a): data point pattern, printed on paper. (b): transparency superimposed on point pattern so that centre of transparency (+) lies above the first data point. (c): other data points are copied (○) onto the transparency. (d): final result.

In mathematical terms the Fry plot is a scatterplot of the vector differences $x_j - x_i$ between all pairs of distinct points in the pattern.

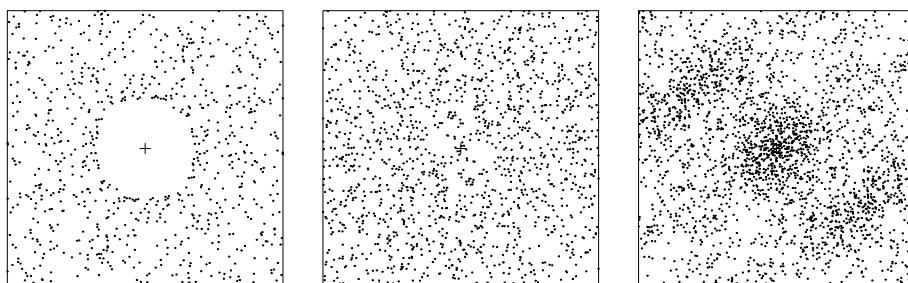


Figure 7.4: Fry plots for the three patterns in Figure 7.1.

In `spatstat` the Fry plot of a point pattern X is generated by the command `fryplot(X)` or `plot(frypoints(X))`. Additional arguments allow the plot to be restricted to a smaller area around the origin $(0,0)$, where the interesting detail is found.

Figure 7.4 shows the Fry plots for the three archetypal point patterns in Figure 7.1, restricted to squares of width 0.6 units. The origin is in the middle of each panel of Figure 7.4 and is indicated by the “+” symbol, visible only in the left panel.

The origin in the Fry plot represents a typical point of the point pattern, and the dots in the Fry plot represent the positions of other nearby points, relative to the typical point. In the left panel of Figure 7.4 there is a clear absence of dots in the middle of the panel, indicating that data points never come closer to each other than a certain minimum distance. The middle panel of Figure 7.4 shows no obvious pattern, while the right panel shows a higher concentration of dots near the origin, indicating a clustered pattern. Thus, the Fry plot is easily able to distinguish the three basic kinds of dependence between points.

Fry plots are implicitly based on the assumption that the underlying process is stationary. Under this assumption, the Fry plot contains essentially “all” information about correlations in the point process.

Fry plots can be very useful for spotting features of the point pattern which might not otherwise

be obvious. In geophysics Fry plots have proven to be very useful for inferring mechanical strain in rocks, which is reflected in the shape of an elliptical ‘hole’ in the Fry plot. However, in many other applications, the interpretation of the Fry plot is too subjective. Fry plots often need to be simplified, reduced or summarized in order to extract usable information.

7.3 The K function

A very popular technique for analysing spatial correlation in point patterns is the K -function proposed¹ by Ripley [281].

7.3.1 The empirical K -function

Suppose that the primary research question concerns the distance or spacing between points in the point pattern. It would then be natural to measure the distances $d_{ij} = \|x_i - x_j\|$ between all ordered pairs of distinct points x_i and x_j in the point pattern \mathbf{x} under consideration. These distances clearly capture a great deal of information about the spatial pattern. If the pattern is clustered, many of the pairwise distances will be small; if the pattern is regular, few of the distances will be small. This suggests that we might look at a statistical summary of the distances d_{ij} , such as the histogram.

We have argued that a good statistical summary of a point pattern should have a simple, direct interpretation in terms of the *point process* which generated the data. The histogram of observed pairwise distances d_{ij} is difficult to interpret in this way, because it depends on the shape and size of the observation window: the same point process, viewed through different windows, yields different histograms of pairwise distances.

Consider instead the empirical cumulative distribution function of the pairwise distances,

$$\begin{aligned} H(r) &= \text{fraction of values } d_{ij} \text{ less than } r \\ &= \frac{1}{n(n-1)} \sum_{i=1}^n \sum_{\substack{j=1 \\ j \neq i}}^n \mathbf{1}\{d_{ij} \leq r\} \end{aligned} \quad (7.2)$$

defined for each distance value $r \geq 0$. Here we use the “indicator” notation: $\mathbf{1}\{\dots\}$ equals 1 if the statement “...” is true, and 0 if the statement is false. The sum of these indicators is simply the number of times that the statement is true, that is, the number of values d_{ij} which are less than or equal to r . The denominator $n(n-1)$ is the total number of pairs of distinct points, so $H(r)$ is the fraction of pairs for which the distance is less than or equal to r .

Notice that $H(r)$ is analogous to the Morisita index: both quantities report the fraction of pairs of points which lie close together. The distance argument r in $H(r)$ defines “closeness”, and is analogous to the size of quadrats in the Morisita index.

We can also visualise the calculation of $H(r)$ using the Fry plot. The dots in the Fry plot are the vector differences $x_i - x_j$ between all pairs of distinct points in the point pattern dataset. The lengths of these vectors are the distances d_{ij} . To count the number of distances d_{ij} that are less than or equal to r , we simply draw a circle of radius r , centred at the origin of the Fry plot, and count the number of dots in the Fry plot which fall inside this circle. That is, $H(r)$ is the fraction of dots in the Fry plot which fall inside the circle of radius r .

The contribution from each data point x_i to the sum in (7.2) is

$$t_i(r) = \sum_{j \neq i} \mathbf{1}\{d_{ij} \leq r\},$$

¹There are similar concepts in statistical physics [264] and astronomy (cf. [239]).

the number of *other* data points x_j which lie closer than a distance r . We might call this the number of *r-neighbours* for the point x_i . Equivalently $t_i(r)$ is the number of data points which fall inside a circle of radius r centred at x_i , not counting x_i itself. Then

$$H(r) = \frac{1}{n(n-1)} \sum_{i=1}^n t_i(r) = \frac{1}{n-1} \bar{t}(r)$$

where $\bar{t}(r) = (1/n) \sum_i t_i(r)$ is the average number of *r-neighbours* per data point. This is an important clue that the quantity we really want to estimate is *the average number of r-neighbours of a typical random point*.

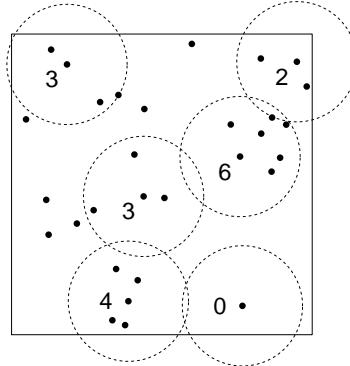


Figure 7.5: Intermediate stage in calculating the average number of *r-neighbours* of a data point. In each circle, the numeral shows the value of $t_i(r)$ for the data point x_i at the centre of the circle.

Figure 7.5 shows an intermediate stage in the calculation of $\bar{t}(r)$. The number of *r-neighbours* is shown for each of several data points. The average of these numbers for all data points is $\bar{t}(r)$.

The average number of *r-neighbours* of a data point will depend on the overall average density of points in the dataset. In order to be able to compare datasets with different numbers of points, it makes sense to standardise $\bar{t}(r)$, dividing it by the average intensity of the point pattern. It may be more appropriate to divide by $\hat{\lambda} = (n-1)/|W|$, where n is the number of points and $|W|$ is the area of the observation window. The result of this standardisation is $\bar{t}(r)/((n-1)/|W|) = |W|H(r)$.

In order to be able to compare datasets observed in different windows, we also need to take account of *edge effects* as explained in Section 7.4. This leads to a slight modification of the function $|W|H(r)$, called the **empirical K-function**,

$$\hat{K}(r) = \frac{|W|}{n(n-1)} \sum_{i=1}^n \sum_{\substack{j=1 \\ j \neq i}}^n \mathbf{1}\{d_{ij} \leq r\} e_{ij}(r) \quad (7.3)$$

where $e_{ij}(r)$ is an *edge correction weight* described in Section 7.4.

In summary, the empirical *K*-function $\hat{K}(r)$ is the cumulative average number of data points lying within a distance r of a typical data point, corrected for edge effects, and standardised by dividing by the intensity. The standardisation and edge correction make it possible to compare point patterns with different numbers of points, observed in different windows.

Warning: using the *K*-function implicitly assumes that the point process has homogeneous intensity. See Section 7.3.5.

Figure 7.6 displays the empirical K -functions for the three archetypal point patterns of Figure 7.1, showing that the K -function clearly has the ability to discriminate between the three basic kinds of inter-point dependence. The K -function for the clustered pattern lies above the K -function for a completely random pattern, which in turn lies above the K -function for a regular pattern. This is equivalent to saying that, after adjusting for intensity, a typical point in the clustered pattern has more close neighbours than a typical point in the completely random pattern, which in turn has more close neighbours than a typical point in the regular pattern.

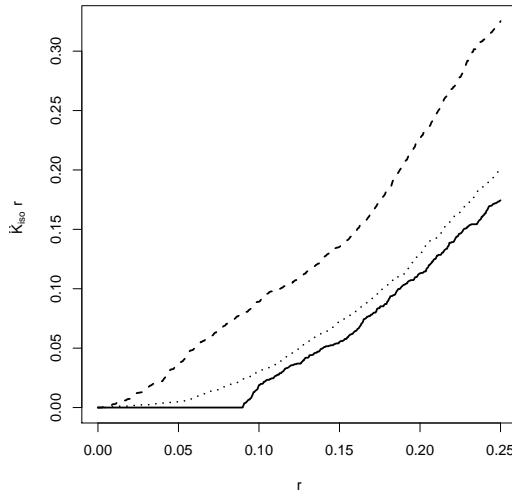


Figure 7.6: Empirical K -functions for the three patterns in Figure 7.1. Solid lines: regular pattern. Dotted lines: independent pattern. Dashed lines: clustered pattern.

When interpreting graphs of the K -function, it helps to remember that $K(r)$ is a standardised or relative quantity, rather than a direct physical quantity. The K -function of a completely-mapped forest gives us, for each r , the average number of neighbour trees lying within distance r of a typical tree, divided by the density of the forest in trees per unit area. The values of $K(r)$ are expressed in units of (number)/(number/area) = area. Standardisation makes it possible to compare the degree of regularity or clustering in forests with different average densities of trees. To recover the physically-meaningful average number of neighbours $\bar{t}(r)$, we would need to know the forest density $\bar{\lambda}$, and multiply $K(r)$ by $\bar{\lambda}$ to obtain $\bar{t}(r)$.

7.3.2 The K -function of a point process

The empirical function $\hat{K}(r)$ is a summary of the pairwise distances in the point pattern dataset, normalised to enable us to compare different datasets. But the key question about any summary statistic for a point pattern is what it means for the *point process* which generated the pattern.

The K -function of a point process \mathbf{X} will be defined as the expected number of r -neighbours of a typical point of \mathbf{X} , divided by the intensity λ . For this we need to assume ² that \mathbf{X} is **stationary** (the distribution of \mathbf{X} is the same as the distribution of the shifted process $\mathbf{X} + v$, for any vector v). This implies that \mathbf{X} has homogeneous intensity λ . We may then define

$$K(r) = \frac{1}{\lambda} \mathbb{E} [\text{number of } r\text{-neighbours of } u \mid \mathbf{X} \text{ has a point at location } u] \quad (7.4)$$

²slightly weaker assumptions are enough, and these will be stated below.

for any $r \geq 0$ and any location u . Since the process is stationary, this definition does not depend on the location u . On the right hand side of (7.4), the symbol $|$ indicates that this is a conditional expectation. Intuitively, we assume there is a random point of \mathbf{X} at the location u ; given this, we find the expected number of other points of \mathbf{X} lying within a distance r ; and finally we divide by the intensity λ , to obtain $K(r)$.

Extending the notation $t_i(r)$, let us define for any spatial location u

$$t(u, r, \mathbf{x}) = \sum_{j=1}^{n(\mathbf{x})} \mathbf{1}\{0 < \|u - x_j\| \leq r\},$$

the number of points in the point pattern \mathbf{x} that lie within a distance r of the location u , but not at u itself.

Definition 7.1. *If \mathbf{X} is a stationary point process, with intensity $\lambda > 0$, then for any $r \geq 0$*

$$K(r) = \frac{1}{\lambda} \mathbb{E}[t(u, r, \mathbf{X}) \mid u \in \mathbf{X}] \quad (7.5)$$

does not depend on the location u , and is called the K-function of \mathbf{X} .

Explicit formulas for the K -function have been found for a few point process models. For the homogeneous Poisson point process (CSR), since the points are independent, intuitively speaking, the presence of a random point at the location u will have no bearing on the presence of points at other locations, so

$$\mathbb{E}[t(u, r, \mathbf{X}) \mid u \in \mathbf{X}] = \mathbb{E}[t(u, r, \mathbf{X})].$$

But $t(u, r, \mathbf{X})$ is the number of points of \mathbf{X} falling in the disc $b(u, r)$ of radius r centred at u , and this has expected value $\lambda \text{area}(b(u, r)) = \lambda \pi r^2$. Dividing by λ shows that, for a homogeneous Poisson process,

$$K_{\text{pois}}(r) = \pi r^2 \quad (7.6)$$

regardless of the intensity.

7.3.3 Use of the K -function

7.3.3.1 Visual inspection of K -function

To study correlation in a point pattern dataset, assuming the intensity is homogeneous, we can plot the empirical K -function $\hat{K}(r)$ calculated from the data, together with the theoretical K -function of the homogeneous Poisson process $K_{\text{pois}}(r) = \pi r^2$, which serves as the benchmark of ‘no correlation’. Figure 7.7 shows this graphic for each of the three archetypal patterns in Figure 7.1.

Figure 7.7 shows that this graphic easily detects the presence and type of correlation between points in a point pattern. In the left panel, the curve for the empirical K -function (solid lines) is lower than the theoretical curve for a completely random pattern (dashed lines), $\hat{K}(r) < K_{\text{pois}}(r)$, indicating that a typical point in this pattern has fewer neighbours than would be expected if the pattern were completely random. This is consistent with a regular point process. Similarly in the right panel, the empirical curve is higher than the theoretical curve, $\hat{K}(r) > K_{\text{pois}}(r)$, indicating that a typical point has more neighbours than would be expected if the pattern were completely random; this is consistent with clustering.

The K -function is a powerful tool for investigating point patterns, but we need to remember that “correlation is not causation”. If analysis shows that $\hat{K}(r) > K_{\text{pois}}(r)$, the careful scientist will not say that this “indicates” clustering, but that it is “consistent with” clustering, or that it indicates “positive association” between points. See Section 7.3.5 for more discussion.

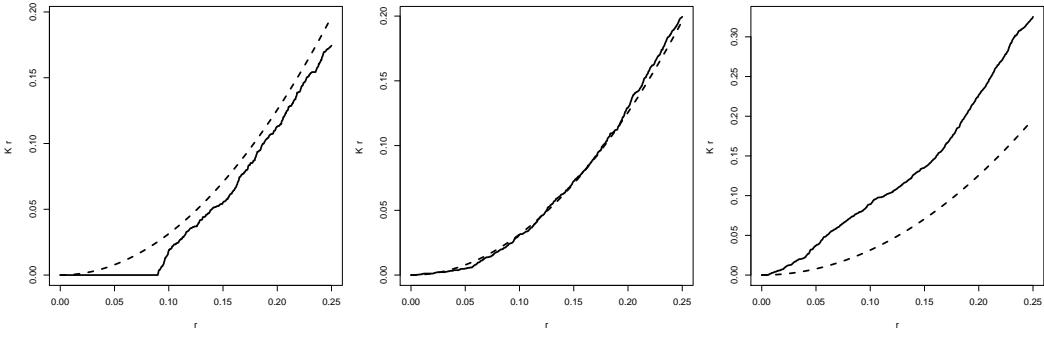


Figure 7.7: Empirical K -function (solid lines) for each of the three patterns in Figure 7.1, and the theoretical K -function for a Poisson process (dashed lines).

Transformation of K

A commonly-used transformation of K proposed by Besag [61] is the L -function

$$L(r) = \sqrt{\frac{K(r)}{\pi}} \quad (7.7)$$

which transforms the theoretical Poisson K function $K_{pois}(r) = \pi r^2$ to the straight line $L_{pois}(r) = r$, making visual assessment of the graph much easier. Figure 7.8 shows the L -functions for the three archetypal point patterns. The square root transformation also approximately stabilises the variance of the estimator (that is, the variance of the empirical function $\hat{L}(r)$ is roughly constant as a function of r), making it easier to assess deviations.

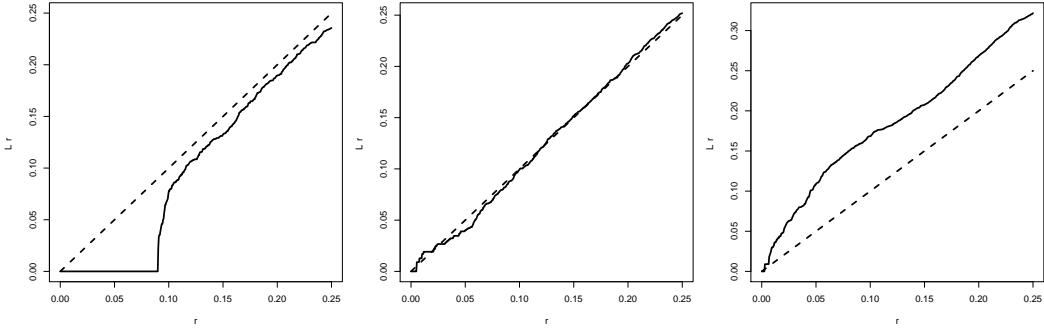


Figure 7.8: Empirical L -function (solid lines) for each of the three patterns in Figure 7.1, and the theoretical L -function for a Poisson process (dashed lines).

Besag's L -function has become such a popular transformation of Ripley's K -function that many writers in applied fields, and some software packages, apply the transformation without mentioning it. They plot $L(r)$ against r , or $L(r) - r$ against r , but still call it Ripley's K -function. This is not advisable because K and L are not equivalent in some contexts. When we say $K(r)$ we shall always mean $K(r)$.

Statistical inference

Options for formal statistical inference about the K -function include *confidence intervals* and *hypothesis tests*, illustrated in Figure 7.9.

A **confidence interval** is designed to contain the *true* value of the target quantity with a specified degree of confidence. A confidence interval is centred around an *estimated* value of the target quantity. The width of the confidence interval is an indication of the accuracy of our estimation. The left panel of Figure 7.9 shows the empirical K -function (solid lines) surrounded by a 95% confidence interval for the true value of $K(r)$ (grey shading) together with the benchmark value for CSR. More explanation about confidence intervals is given in

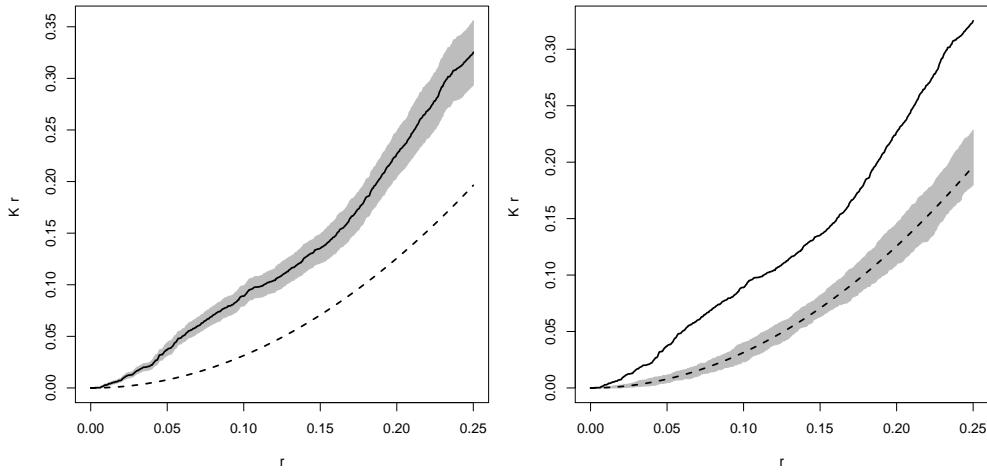


Figure 7.9: Tools for formal inference, demonstrated on the clustered pattern in the right panel of Figure 7.1. *Left:* 95% confidence intervals (shaded) for the true value of the K -function, obtained using Loh's bootstrap (function `lohboot()`). *Right:* acceptance region (shaded) for a hypothesis test of complete spatial randomness, with significance level 5%, using the envelopes of the K -functions of 39 simulated realisations of CSR.

If we have a preconceived hypothesis which should be formally tested — for example, if we want to formally test whether the point process is completely random — then a more appropriate tool is a **hypothesis test**. The test can be represented by its *acceptance interval*,³ the range of values that are deemed to be not significantly different from the hypothesised value. An acceptance interval is centred around the *hypothesised* value of the target quantity. The width of the acceptance interval reflects the inherent variability of the observations. In the right panel of Figure 7.9, the grey shaded region shows the acceptance interval for a formal hypothesis test of the null hypothesis that the point process is CSR, with significance level 0.05. For a fixed value of r , the test rejects the null hypothesis of CSR if the value of $\hat{K}(r)$ lies outside this acceptance interval. More explanation and important caveats about hypothesis tests are given in Chapter 10.

Implications for modelling

The K -function can serve as a guide for building a point process model of the phenomenon being studied. Chapter 12 presents one class of models which can be fitted directly to point pattern data using the K -function. More commonly the K -function is only a rough guide to the required behaviour of the model.

A point process model of a natural phenomenon could include several stages in which points are randomly added, combined, displaced or removed. In this regard there are two important properties of the K -function.

³strictly the “non-rejection” interval.

Firstly **the K -function is invariant under thinning**. Suppose \mathbf{X} is a stationary point process, and \mathbf{Y} is the point process obtained by independent random thinning — randomly deleting or retaining each point of \mathbf{X} , with a constant probability p of retaining each point. Then the K -function of \mathbf{Y} is identical to the K -function of \mathbf{X} .

Secondly there is the effect of **superposition**. Suppose \mathbf{X} and \mathbf{Y} are two independent, stationary point processes, with intensities $\lambda_{\mathbf{X}}, \lambda_{\mathbf{Y}}$ and K -functions $K_{\mathbf{X}}(r), K_{\mathbf{Y}}(r)$. Superimposing these two processes gives a stationary point process with intensity $\lambda = \lambda_{\mathbf{X}} + \lambda_{\mathbf{Y}}$ and K -function

$$K(r) = p_{\mathbf{X}}^2 K_{\mathbf{X}}(r) + p_{\mathbf{Y}}^2 K_{\mathbf{Y}}(r) + 2p_{\mathbf{X}}p_{\mathbf{Y}}\pi r^2 \quad (7.8)$$

where $p_{\mathbf{X}} = \lambda_{\mathbf{X}}/\lambda$ and $p_{\mathbf{Y}} = \lambda_{\mathbf{Y}}/\lambda$ are the relative proportions of points coming from \mathbf{X} and \mathbf{Y} .

7.3.4 Estimating the K and L functions in spatstat

The `spatstat` function `Kest()` computes several estimates of $K(r)$ using different edge corrections. It also returns the “benchmark” value πr^2 which is the theoretical value for $K(r)$ for a homogeneous Poisson process.

```
> K <- Kest(cells)
> Ki <- Kest(cells, correction="isotropic")
```

The argument `correction` specifies which estimate or estimates will be computed by `Kest()`. Options include "isotropic" for Ripley's isotropic correction, "translation" for the translation correction, "rigid" for the rigid motion correction, "border" for the border correction, and "none" for the uncorrected estimate. These are explained in Section 7.4. Any number of edge corrections may be selected. Specifying a single edge correction will reduce computation time, especially in simulation experiments.

Additionally `Kest()`, like all standard summary functions in `spatstat`, recognises the options "best" (representing the estimate with the best statistical performance regardless of computational cost) and "good" (the estimate with the best statistical performance for reasonable computational cost).

If no `correction` argument is given, the default is to compute the isotropic, translation, and border corrections, unless the point pattern contains more than `nlarge` points, when only the border correction will be computed. The default threshold is `nlarge = 3000` points. Setting `nlarge=Inf` will suppress this behaviour.

The `spatstat` function `Lest()` computes estimates of $L(r)$ directly from point pattern data.

```
> Lc <- Lest(cells)
```

The arguments of `Lest()` are identical to those of `Kest()`.

A previously-computed K -function can be converted to the L -function using the syntax `L <- with(K, sqrt(. / pi))`, explained on page 189, or plotted as an L -function using the syntax `plot(K, sqrt(. / pi) ~ r)` explained on page 185.

7.3.5 Caveats about the K -function

The use of the K function for analysing point patterns has become established across wide areas of applied science, following Ripley's influential paper [281] and many subsequent textbooks [108, 123, 127, 320, 283, 284, 314]. Useful and powerful as this methodology may be, there is an unfortunate tendency to apply it uncritically and to neglect other methods of analysis.

7.3.5.1 K function assumes homogeneity

The K function is defined and estimated under the *assumption that the point process is stationary (homogeneous)*. If the process is not stationary, deviations between the empirical and theoretical functions (e.g. \hat{K} and K_{pois}) are not necessarily evidence of interpoint interaction, since they may also be attributable to variations in intensity.

Figure 7.10 shows a realisation of an inhomogeneous Poisson process, and its empirical K -function. The plot of the K -function shows that, on average, a point in this pattern has more r -neighbours than would be expected for CSR. However, this happens because there is a higher density of points in one corner of the window. The points are positively associated, but not because they are clustered in the usual sense.

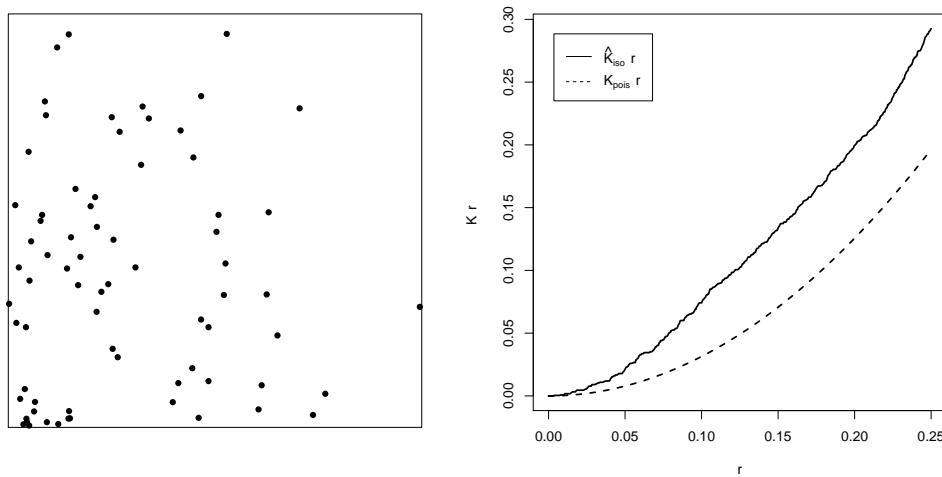


Figure 7.10: The K -function is fooled by spatial inhomogeneity. *Left:* Inhomogeneous Poisson point pattern. *Right:* Empirical K -function.

7.3.5.2 Correlation is not causation

There are many possible causes and mechanisms of correlation between points. True spatial regularity could be the result of ecological processes (territorial behaviour, competition for resources), physical forces (repulsion between objects), or human intervention. Apparent spatial regularity can occur as an artefact of the treatment of data, for example, if biological cells of appreciable size are treated as ideal points, or if spatial coordinates are discretised. Spatial clustering does not imply that the points are organised into identifiable ‘clusters’; merely that they are closer together than would be expected for a completely random pattern. True spatial clustering could be the result of biological processes (reproduction, contagion), physical forces (electrostatic or magnetic attraction) or space-time history (clustering of meteorite impacts). Apparent clustering can occur if parts of a spatial pattern are obliterated (e.g. native forest partially destroyed by fire) or if objects of one type are physically *repelled* by another type of object. Spatial inhomogeneity is often mistaken for spatial clustering as we mentioned above.

7.3.5.3 Lack of correlation does not prove independence

Correlation is a summary measure of stochastic dependence, but absence of correlation does not necessarily indicate independence. Examples are well-known in elementary statistics.

Similarly, there exist point processes whose K functions are equal to πr^2 and yet the processes

are *not* Poisson processes (so that there is dependence amongst the points). Therefore the K function does not completely characterise the point process.

An example is the *cell process* of Baddeley and Silverman [44]. Space is divided into equal cells; in each cell we place a random number N of points, according to a probability distribution that has the property $\mathbb{E}N = \text{var}N$; the points are positioned independently and uniformly. The cell process has exactly the same theoretical K function as the homogeneous Poisson process, but is manifestly different from a Poisson process. The left panel of Figure 7.11 shows a realisation of the cell process, generated by the *spatstat* function `rcell()`. The right panel shows the empirical K function, which falsely suggests that the process is Poisson. In fact the cell process would defeat any technique based on second moments, because all second-moment quantities for the cell process are identical to those for the Poisson process.

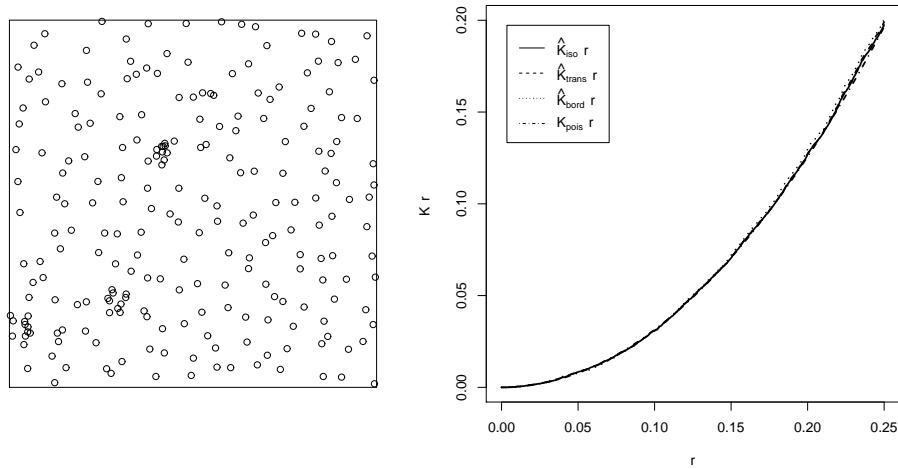


Figure 7.11: Realisation of Baddeley-Silverman cell process (Left) and its empirical K -function (Right).

7.3.5.4 Spatial scale of interaction

Summary functions like Ripley's K -function convey information across a range of spatial scales. This is an important motivation for using empirical functions, rather than simple summary statistics, and for displaying them graphically.

Many researchers use a graph of the K -function to infer "the" scale of spatial interaction in a point pattern. This scale is often estimated by reading off the position where the empirical function lies furthest away from the theoretical Poisson value, or furthest outside the simulation envelope.

This interpretation is not correct for several reasons. Firstly, "the scale of interaction" is not a well-defined concept for point processes in general. It is only meaningful for certain point process models, such as Markov point processes (Chapter 13) and some Neyman-Scott cluster processes (Chapter 12).

Secondly, even for point processes which have a well-defined scale of interaction, it is not always true that the greatest deviation in the K -function occurs when r is equal to the scale of interaction. The K -function reflects *correlation* between pairs of points, not direct dependence. Dependence between points at one scale can give rise to correlation between points at another scale.

Thirdly, the K -function is *cumulative*: $K(r)$ accumulates contributions from all distances *less than or equal to* r . If a pattern of emergent seedlings yields $\hat{K}(r) > \pi r^2$ for the distance $r = 10$ metres, this does not necessarily indicate that seedlings are clustered **at** distances of 10 metres. A

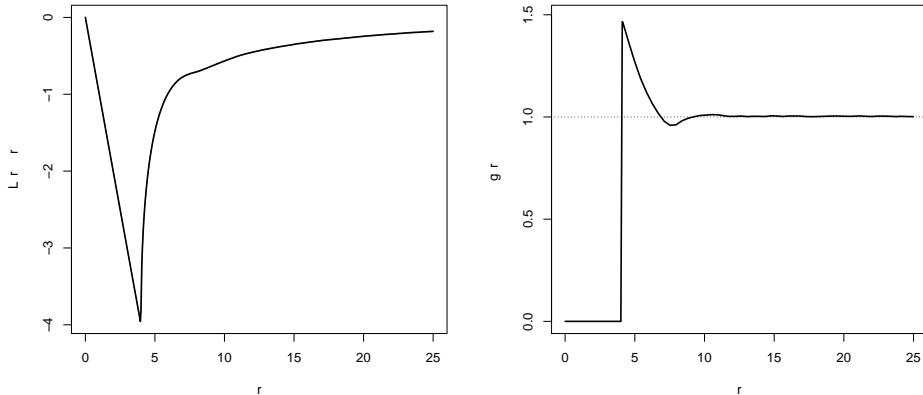


Figure 7.12: Centred L -function (*Left*) and pair correlation function (*Right*) of the Hard Core process with range $s = 4$ metres and base intensity $\beta = 0.06$ in a 100 metre square, estimated by simulating 10,000 realisations. Grey shading in left panel (barely visible) indicates 95% confidence interval.

plausible explanation is that seedlings are organised in clusters at a much smaller spatial scale, and the cumulative effect is still evident at 10 metres. An alternative, non-cumulative summary statistic is the pair correlation function (Section 7.6). The K -function is optimal for detecting interpoint interaction that occurs equally at all distances up to a certain maximum distance r .

This is from Ecological Monographs paper

Consider, for example, the ‘Hard Core’ process in which points are forbidden to occur closer than a specified *range*, s , whose value unambiguously defines the scale of interaction of the process. In this case, both the K -function and the L -function show their greatest deviation from CSR at distance s . The left-hand panel of Figure 7.12 illustrates this by plotting $L(r) - r$ against r , with $L(r)$ estimated by simulating 10,000 realisations of a Hard Core process with range $s = 4$ metres.

We might be tempted to identify the scale of interaction from the behaviour of the pair correlation function. The right-hand panel of Figure 7.12 plots $g(r)$ against r , again estimated from 10,000 simulated realisations. The shark fin shape of the pair correlation at distances between 4 and 7 metres, the trough at 7.5 metres, and the subsequent small peak at 10 metres, might naively suggest the existence of multiple scales of interaction in the underlying process; in fact, the peaks and troughs in the plot beyond 4 metres are simply echoes of a single scale of interaction. The parameter values in this example ensure that the points are very tightly packed, subject to the hard core restriction. Each point is very likely to have a neighbour lying just beyond the critical distance $s = 4$ metres, and this gives rise to the shark fin shape. Close neighbours inhibit the presence of other neighbours, giving rise to the trough at 7.5 metres. The essential problem here is that correlation is not causation.

A second example is provided by a Thomas cluster process (REF) [318, 120]. Its K -function and pair correlation function are respectively [127, 193]

$$\begin{aligned} K(r) &= \pi r^2 + \kappa^{-1} \{1 - \exp(-0.25r^2/\sigma^2)\} \\ g(r) &= 1 + (4\sigma^2\pi\kappa)^{-1} \exp(-0.25r^2/\sigma^2). \end{aligned}$$

In common with the Hard Core process described above, this cluster process embodies a single scale of interaction, as a consequence of the property that only offspring of the same parent “interact”. Unlike the Hard Core process however, it is not obvious how we might assign a numerical value to the scale. One reasonable suggestion would be to use some property of the distribution of the distance between two offspring of the same parent. This depends only on σ . Another might be some

κ	σ		
	0.01	0.02	0.05
2	0.039	0.070	0.149
5	0.036	0.065	0.137
10	0.034	0.061	0.130

Table 7.1: The distance, r , at which $L(r)$ deviates maximally from its value under CSR, in a cluster process with parent intensity κ and cluster standard deviation σ .

property of the distribution of the maximum distance between any two offspring of the same parent. This depends on both σ and μ . We cannot think of any justification for involving the parameter κ , yet this clearly features in the expressions for $K(r)$ and $g(r)$, whereas μ does not. Also, $K(r) - \pi r^2$ and $g(r) - 1$ are, respectively, increasing and decreasing functions of r , so in neither case would it be sensible to identify the scale of the process as the distance at which the function in question deviates maximally from its theoretical value under CSR. The maximum deviation of $L(r)$ from its value under CSR does occur at a finite, non-zero value of r but, as Table 7.1 illustrates, this value depends on both σ and κ .

In both of our examples, we have considered possible definitions of a scale of interaction only in terms of properties of the underlying stochastic *process*, not of any observed point *pattern*. We would argue that this is the correct way to think about the issue, since the objective of any data analysis is to understand what natural processes may or may not have generated the data. In summary, the notion of a scale of interaction is useful for heuristics, but can only be quantified precisely within the confines of a declared family of parametric models, in which case it must be expressible as some function of the model parameters, and can only be estimated by first estimating those parameters.

7.4 Edge corrections for the K -function*

This section gives more detail about the edge correction techniques used in estimating the K -function of a point process.

Most users of `spatstat` will not need to know the theory behind edge correction, the details of the techniques, or their relative merits. So long as some kind of edge correction is performed (which happens automatically in `spatstat`), the particular choice of edge correction technique is usually not critical.

However, there is a danger that the *implementations* of edge corrections in some software packages may be incorrect, which could cause substantial bias. There are certainly some misunderstandings about edge corrections that have crept into the applied literature. This section attempts to set the record straight.

For advanced users, understanding the assumptions behind edge correction is helpful in appreciating potential weaknesses of the analysis, and in resolving discrepancies between results.

* Starred sections contain advanced material, and can be skipped by most readers.

7.4.1 Estimation without edge effects

An alternative expression for the K function, that does not involve conditional expectation, is

$$K(r) = \frac{\mathbb{E} \sum_{x \in \mathbf{X} \cap B} t(x, r, \mathbf{X})}{\lambda \mathbb{E} n(\mathbf{X}_B)}, \quad (7.9)$$

holding for a stationary point process \mathbf{X} , and for any bounded region B with area $|B| > 0$. Thus $\lambda K(r)$ is the expected number of r -close pairs of points in which the first point falls in B , divided by the expected number of points falling in B .

A straightforward way to estimate the K -function is suggested the representation in equation (7.9). We simply replace the numerator and denominator of (7.9) by data-based estimates:

$$\hat{K}(r) = \frac{\sum_{x \in \mathbf{X} \cap W} t(x, r, \mathbf{X})}{\bar{\lambda} n(\mathbf{X}_B)}. \quad (7.10)$$

The numerator is the total number of r -neighbours of all random points falling in B (an arbitrary set with nonzero area), and in the denominator, $n(\mathbf{X}_B)$ is the number of points falling in B , while $\bar{\lambda} = n(\mathbf{X}_B)/|B|$ is an estimator of λ .

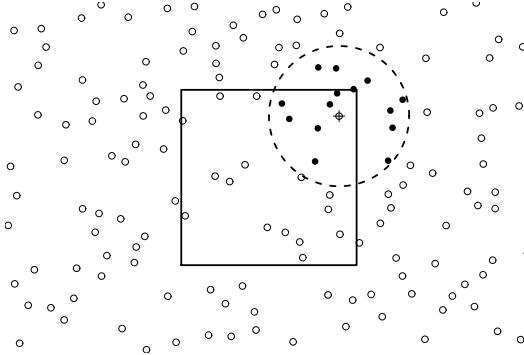


Figure 7.13: Counting, without edge effects, the number of neighbours of each wildflower within a sampling frame (black rectangle) in a field of wildflowers.

Figure 7.13 sketches how this estimator could be applied in practice. In a homogeneous field of wildflowers, we have laid out a sampling frame B (black square). Visiting one of the flowers inside the sampling frame, we push a thin stake into the soil near the flower (crosshairs), tied to a 1-metre length of string. Pulling the string taut, we walk all around the flower (dashed lines), and count how many other flowers are swept over (black dots) *regardless of whether they lie inside or outside the frame*. For the example in Figure 7.13 the count is 14. We have just counted the number of r -neighbours $t(x, r, \mathbf{X})$ for one flower, for the distance $r = 1$ metre.

Repeating this process for each flower inside the study area would give us the neighbour counts $t(x_i, r, \mathbf{X})$ for flowers x_1, \dots, x_n , say, where n is the number of flowers in the study area. The average observed neighbour count is $\bar{t}(r) = (1/n) \sum_i t(x_i, r, \mathbf{X})$. Dividing by the estimated intensity $\bar{\lambda} = n/A$, where A is the area of the study frame, gives us an estimate of $K(r)$ as in (7.10).

7.4.2 Edge effects

The situation sketched in Figure 7.13 is unusual, because we were able to look outside the sampling frame. In most research studies, this information is lost; points are recorded only if they fall inside the study region; the situation is more like Figure 7.14.

If points are observed only inside a window W , then the estimator (7.10) with $B = W$ is not

feasible. The number of points inside a circle of radius r , centred on a point of the process inside W , is *not observable* if the circle extends outside W . This is an *edge effect* problem.

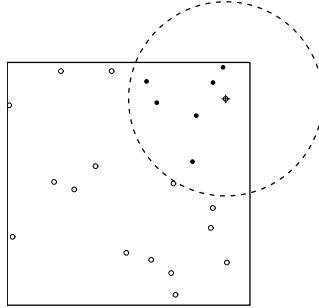


Figure 7.14: Edge effect problem for estimation of the K function. If we can only observe the points inside a window W (solid lines), then the number of points inside a circle of radius r , centred on a point of the process inside W , is not known if the circle extends outside W . The number of points *observed* inside the circle (i.e. counting only points within the window) is typically less than the true number.

It might be tempting to ignore this problem and use the “uncorrected” empirical function

$$\tilde{K}(r) = \frac{|W|}{n(n-1)} \sum_{i=1}^n \sum_{\substack{j=1 \\ j \neq i}}^n \mathbf{1}\{d_{ij} \leq r\} = |W|H(r), \quad (7.11)$$

the counterpart of (7.3) without the weighting terms $e_{ij}(r)$. However a simple experiment shows that (7.3) is severely biased as an estimator of $K(r)$. We generate a completely random point pattern, according to a uniform Poisson process with intensity $\lambda = 100$ in the unit square. The uncorrected function $\tilde{K}(r)$ is plotted in Figure 7.15 together with the correct K -function $K(r) = \pi r^2$. The uncorrected function is clearly an under-estimate of the correct K -function.

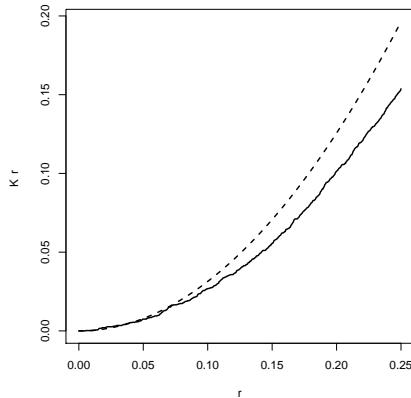


Figure 7.15: Bias in estimating $K(r)$ due to edge effects. Solid lines: uncorrected estimate $\tilde{K}(r)$ for a completely random point pattern. Dashed lines: correct K -function $K(r) = \pi r^2$.

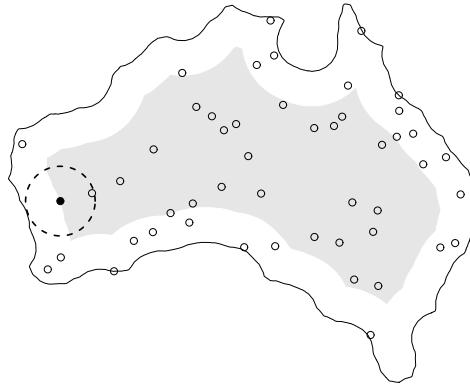


Figure 7.16: Border method of edge correction for the K -function.

7.4.3 Border correction

One simple strategy for eliminating the edge effect bias is the *border method*. When estimating $K(r)$ for a particular distance r , we restrict attention to cases where the circle of radius r lies entirely inside the window, so that the edge effect does not occur. See Figure 7.16.

In the numerator and denominator of (7.10) we restrict the summation to data points x_i for which $b(x_i, r)$ lies entirely inside W . For such points, $t(x_i, r, \mathbf{X} \cap W) = t(x_i, r, \mathbf{X})$, so the true number of r -neighbours is observable. If $d(u, \partial W)$ denotes the shortest distance from a location u to the window boundary ∂W , then we are restricting attention to data points x_i which satisfy $d(x_i, \partial W) \geq r$. These are the data points which fall in the *eroded set*

$$W_{\ominus r} = W \ominus b(0, r) = \{u \in W : d(u, \partial W) \geq r\} \quad (7.12)$$

consisting of locations in W that are at least r units away from the boundary ∂W . The eroded set is shaded in Figure 7.16.

Thus we estimate $K(r)$ by the *border correction estimate*

$$\hat{K}_{bord}(r) = \frac{\sum_{x_i \in \mathbf{x} \cap W_{\ominus r}} t(x_i, r, \mathbf{X})}{\bar{\lambda} n(\mathbf{x}_{W_{\ominus r}})} = \frac{\sum_{i=1}^n \mathbf{1}\{b_i \geq r\} t(x_i, r, \mathbf{x})}{\bar{\lambda} \sum_{i=1}^n \mathbf{1}\{b_i \geq r\}} \quad (7.13)$$

where $b_i = d(x_i, \partial W)$ is the distance to the window boundary, and $\bar{\lambda}$ is usually $n(\mathbf{x}_W)/\text{area}(W)$.

The estimate (7.13) is well-defined so long as the denominator is non-zero, that is, so long as $r < \max_i b_i$. The maximum possible value of r for which (7.13) can ever be computed is⁴ the inradius of W , the radius of the largest disc contained in W , since this is the maximum possible value of $d(\cdot, \partial W)$.

The border correction estimate (7.13) can be justified as a data-based estimate of the right hand side of (7.9) taking $B = W_{\ominus r}$. This suggests that it will have reasonable statistical properties in sufficiently large datasets. The numerator of (7.13) is an unbiased estimator of the numerator of (7.9), while the denominator of (7.13) is the product of two terms which are unbiased estimators of the corresponding terms in the denominator of (7.9). In large datasets, the border-correction estimate is consistent and approximately unbiased, under reasonable assumptions.

The border correction estimate of the K -function is relatively simple to code for any shape of window, and is fast to calculate. However, in small datasets, it can be inaccurate when compared to other methods, because it discards substantial amounts of data. For example, if W is the unit

⁴Here and throughout the discussion we assume W is a topologically regular set, meaning that it is the closure of its interior.

square and $r = 0.1$, the eroded window $W_{\ominus r}$ is a square of side 0.8 and area 0.64 so that 40% of the data is being discarded in order to estimate $K(0.1)$. In small datasets or in unusual situations, the graph of the estimated function $\hat{K}_{\text{bord}}(r)$ can have an erratic trajectory, as shown in Figure 7.17. For large values of r , the denominator of (7.13) is small, magnifying the effect of discrete jumps in the numerator and denominator. Although the true K -function must be an increasing function of r , the border correction estimate need not be so.

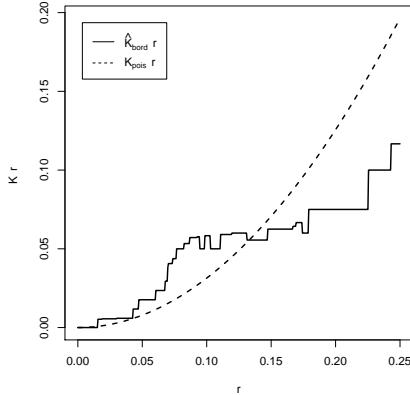


Figure 7.17: Border correction estimate of the K -function for 20 uniformly random points in the unit square.

When the number of data points is large, the border method is usually preferable to more computationally-intensive methods, because the interesting distances r are small, and the loss of statistical efficiency in the border method is negligible.

The border method is a useful remedy for edge effects in many spatial problems. It is related to the “local knowledge” principle of mathematical morphology [305, 42].

7.4.4 Isotropic correction

An alternative approach to edge effects is to regard them as a form of *sampling bias*.

Visualise the entire point process \mathbf{X} extending throughout two-dimensional space. Consider a pair of points x, x' from \mathbf{X} , and assume that x falls in the observation window. Given the location of the first point x and the distance $d = \|x - x'\|$, the second point x' must lie somewhere on the circle $b(x, d)$ of radius d centred at x . In general, only part of this circle lies inside the window: see Figure 7.18.

If the point process is *isotropic* (statistically invariant under rotation), then roughly speaking, the second point x' is equally likely to lie anywhere on this circle. The probability that x' falls inside W is the fraction of length of the circle lying within W ,

$$p(x, d) = \frac{\text{length}(W \cap \partial b(x, d))}{2\pi d}.$$

As d increases, the probability $p(x, d)$ typically decreases. This gives rise to a sampling bias: larger distances are less likely to be observed.

A standard strategy for correcting sampling bias is the Horvitz-Thompson estimator [186] in which each item in the sample is weighted by the reciprocal of its sampling probability. The analogous strategy can be applied here [36, 97]. If each pair of points x_i, x_j is weighted by the reciprocal



Figure 7.18: Calculation of sampling bias for the isotropic correction.

of the probability $p(x_i, d_{ij})$, we obtain Ripley's *isotropic correction* estimator

$$\hat{K}_{iso}(r) = \frac{1}{\bar{\lambda}n} \sum_{i=1}^n \sum_{\substack{j=1 \\ j \neq i}}^n \mathbf{1}\{d_{ij} \leq r\} \frac{2\pi d_{ij}}{\text{length}(W \cap \partial b(x_i, d_{ij}))} \quad (7.14)$$

Note that some authors state, incorrectly, that Ripley's isotropic correction uses the fraction of *area* rather than the fraction of *length* of the circle.

The isotropic-correction estimate is a non-decreasing function of r , unlike the border-correction estimate.

The Horvitz-Thompson principle is valid provided each item in the population has a non-zero probability of being sampled. In the isotropic correction, it is possible to have $p(x, d) = 0$ when d is large enough. Let \hat{R}_{iso} be the smallest distance for which there is some location x in W where $p(x, d) = 0$. Then $\bar{\lambda}^2 \hat{K}_{iso}(r)$ is an unbiased estimator of $\lambda^2 K(r)$ for all $r < \hat{R}_{iso}$. The distance \hat{R}_{iso} is the circumradius of W , the radius of the smallest circle containing W (or the minimum of the circumradii of the connected components of W , if W consists of several pieces). If W is a rectangle, \hat{R}_{iso} is equal to half the diagonal of W .

The isotropic estimator can be modified to work for longer distances. For a given distance r , let $W^*(r)$ be the subset of W consisting of locations x where $p(x, r) > 0$. Then the modified estimator due to Ohser and Stoyan [261] is

$$\hat{K}_{iso*}(r) = \frac{1}{\bar{\lambda}n} \sum_{i=1}^n \sum_{\substack{j=1 \\ j \neq i}}^n \mathbf{1}\{d_{ij} \leq r\} \frac{2\pi d_{ij}}{\text{length}(W \cap \partial b(x_i, d_{ij}))} \frac{|W|}{|W^*(d_{ij})|} \quad (7.15)$$

and $\bar{\lambda}^2 \hat{K}_{iso*}(r)$ is an unbiased estimator of $\lambda^2 K(r)$ for all $r < \hat{R}_{iso*}$. Here \hat{R}_{iso*} is the smallest distance R such that $|W^*(R)| = 0$, which is the diameter of W . Of course (7.15) agrees with (7.14) for $r < \hat{R}_{iso}$ so this calculation is only required when $\hat{R}_{iso} \leq r < \hat{R}_{iso*}$.

For any $r < \hat{R}_{iso}$, the double sum in (7.14), namely $\bar{\lambda}n \hat{K}_{iso}(r)$, is an unbiased estimator of $\lambda^2 \text{area}(W) K(r)$.

The modified isotropic correction provides estimates of $K(r)$ for much larger distances r than are possible with the border correction. However, at large distances, the variance of the estimator increases rapidly. This is a well-known problem with the Horvitz-Thompson estimator: if the probability of sampling an item is very small, it will be given a very large weight if it is sampled, so its contribution has large variance.

A pragmatic solution is to restrict the range of r values, and to truncate the edge correction

weight in (7.15) so that it never exceeds a specified upper limit. This is the procedure used in `spatstat`.

Stein [307] proposed a more satisfactory solution, in which contributions to the estimator are downweighted if they have large variance. This would be more computationally-intensive and does not appear to have been implemented in publicly-accessible code.

7.4.5 Translation correction

Another edge correction argument can be used if we are not willing to assume the point process is isotropic.

Draw arrows from every random point x to every other random point x' . An arrow will be observed only when both its endpoints x, x' fall in the observation window W . See Figure 7.19. Intuitively it is clear that a shorter arrow is more likely to fall inside W than a longer arrow, and this gives rise to a sampling bias: the observed arrows are a biased sample of all arrows.

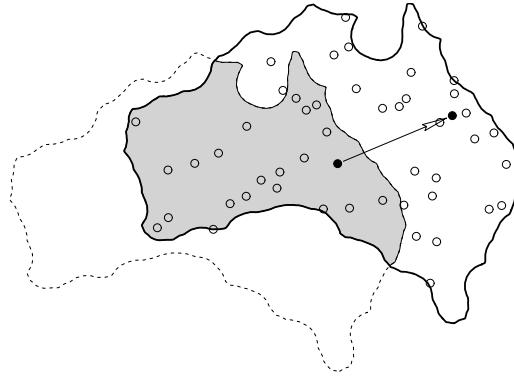


Figure 7.19: Sampling bias calculation for the translation edge correction. An arrow joining a pair of random points will be observed only if both endpoints fall in the window W . The shaded region shows the possible locations for the start of the arrow which guarantee that the arrow will be observed.

For an arrow of fixed size and direction $v = x' - x$, consider the possible positions of the starting point x for which both endpoints x and $x + v$ fall inside W . Notice that $x + v \in W$ if and only if x belongs to $W - v$, a copy of the window W shifted by the vector $-v$. Both endpoints of the arrow fall inside W if and only if $x \in W \cap (W - v)$. The region $W \cap (W - v)$ is shaded in Figure 7.19. It is the intersection of the window W with a shifted copy of itself. The probability that the starting point x falls in the shaded region $W \cap (W - v)$ is related to the area of this region.

Applying the Horvitz-Thompson principle leads to the *translation correction* [260, 261] estimator of the K -function,

$$\hat{K}_{\text{trans}}(r) = \frac{1}{\lambda n} \sum_{i=1}^n \sum_{\substack{j=1 \\ j \neq i}}^n \mathbf{1}\{d_{ij} \leq r\} \frac{|W|}{|W \cap (W - (x_j - x_i))|}. \quad (7.16)$$

In this estimator, each pair of data points (x_i, x_j) is weighted by the reciprocal of the fraction of window area in which the first data point x_i could be placed so that both points x_i, x_j would be observable (assuming their relative positions were held fixed). This effectively compensates for the sampling bias in observing such pairs.

The edge correction weight in (7.16) can be calculated using simple geometry if W is a rectangle. Otherwise, the most efficient algorithm computes the *set covariance function* $C_W(v) = |W \cap (W - v)|$

for all vectors v on a fine grid using the Fast Fourier Transform (because C is the convolution of two indicator functions) and then extracts the values $C_W(x_j - x_i)$ for each pair of points.

The Horvitz-Thompson principle gives an unbiased estimator provided that each item in the population has a non-zero probability of being sampled. Consequently $\bar{\lambda}^2 \hat{K}_{trans}(r)$ is an unbiased estimator of $\lambda^2 K(r)$ for all $r \leq R$, where R is the length of the shortest vector v such that $C_W(v) = 0$. If W is a rectangle, then R is the length of the shortest side.

The translation correction provides estimates of $K(r)$ for larger distances r than are possible with the isotropic correction, but again suffers from inflated variance at these large distances. Again the pragmatic solution used in `spatstat` is to restrict the range of r values, and to truncate the edge correction weight in (7.16) so that it never exceeds a specified upper limit.

If the point process is assumed to be isotropic as well as stationary, the translation correction can be modified to give the *rigid motion correction*

$$\hat{K}_{rigid}(r) = \frac{1}{\bar{\lambda} n} \sum_{i=1}^n \sum_{\substack{j=1 \\ j \neq i}}^n \mathbf{1}\{d_{ij} \leq r\} \frac{|W|}{\bar{C}_W(d_{ij})} \quad (7.17)$$

where \bar{C}_W is the rotational average of C_W , that is, $\bar{C}_W(r)$ is the average value of $C_W(v)$ over all vectors v of length $\|v\| = r$.

7.4.6 Discussion

Edge corrections are discussed in [312, 37].

The choice of edge correction does not, in most instances, seem to be very important, as long as *some* edge correction is applied. Must be applied correctly! The border method tends to be imprecise for values of r that are large relative to the scale of the observation window, but work satisfactorily in moderately large datasets. In huge datasets no edge correction is necessary.

Algorithm `Kest(correction="none")` works with millions of points
Discrepancies often indicate unusual features

7.5 The class "fv"

7.5.1 Objects of class "fv"

An object of class "fv" ("function value table") is a convenient way of storing and plotting several different estimates of the same function.

The value returned by `Kest()` is an object of class "fv":

```
> K <- Kest(cells)
> K
Function value object (class 'fv')
for the function r -> K(r)
.....
Description
r      distance argument r
theo   theoretical Poisson K(r)
border border-corrected estimate of K(r)
trans   translation-corrected estimate of K(r)
iso    isotropic-corrected estimate of K(r)
```

```
.....  

Default plot formula: .~r  

where "." stands for 'iso', 'trans', 'border', 'theo'  

Recommended range of argument r: [0, 0.25]  

Available range of argument r: [0, 0.25]
```

An "fv" object is a data frame (that is, it also belongs to the class "data.frame") with attributes giving extra information such as the recommended way of plotting the function. One column of the data frame contains evenly-spaced values of the distance argument r , while the other columns contain estimates of the value of the function, or the theoretical value of the function under CSR, corresponding to these distance values. The printout for K_c above indicates that the columns in the data frame are named `r`, `theo`, `border`, `trans`, and `iso`. The first column `r` contains a sequence of values of the function argument r . The next column `theo` contains the corresponding values of $K(r)$ for a homogeneous Poisson process. The columns `border`, `trans` and `iso` contain estimates of the K function using the border, translation, and isotropic edge corrections, respectively.

The function argument in an "fv" object is usually called `r`. The printout above indicates the range of values of `r` in the table as the "available range". It also gives a "recommended range" which is generally shorter than the available range. *The default plot of the object will only show the function values over the recommended range* and not over the full range of values available. This is done so that the interesting detail is clearly visible in the default plot. Values outside the recommended range may be unreliable due to increased variance or bias, depending on the edge correction.

7.5.2 Plotting "fv" objects

If `f` is an object of class "fv", then `plot(f)` is dispatched to the method `plot.fv()`. The default behaviour of `plot(f)` is to generate a plot containing several curves, each representing a different edge-corrected estimate of the same target function, plotted against the distance argument r . For example, the command `plot(Kest(redwood))` generates Figure 7.20, which shows three different estimates of the K -function for the redwood dataset, together with the theoretical curve $K(r) = \pi r^2$ for a homogeneous Poisson process, all plotted against the distance argument r . The legend indicates the meaning of each curve. The main title identifies the function object that was plotted.

The return value from `plot.fv()` contains more detailed information about the meaning of the curves. For Figure 7.20, the return value is

	lty	col	label	meaning
iso	1	1	<code>hat(K)[iso](r)</code>	isotropic-corrected estimate of $K(r)$
trans	2	1	<code>hat(K)[trans](r)</code>	translation-corrected estimate of $K(r)$
border	3	1	<code>hat(K)[bord](r)</code>	border-corrected estimate of $K(r)$
theo	4	1	<code>K[pois](r)</code>	theoretical Poisson $K(r)$

Here `lty` and `col` are the graphics parameters controlling the line type and line colour, and `label` is the mathematical notation for each edge-corrected estimate, in the syntax recognised by R graphics functions.

The default plot in Figure 7.20 can easily be modified. Set `legend=FALSE` to suppress the legend, and `main=""` to suppress the main title. The legend position is automatically computed to avoid overlap with the plotted curves, but this can be overridden by `legendpos`. For further options see `help(plot.fv)`.

The variables plotted on the x and y axes are determined by the second argument to `plot.fv()`, which should be a `formula`, involving the names of columns of the object. As usual, the left side of the formula represents what variables will be plotted on the y axis, and the right side determines the

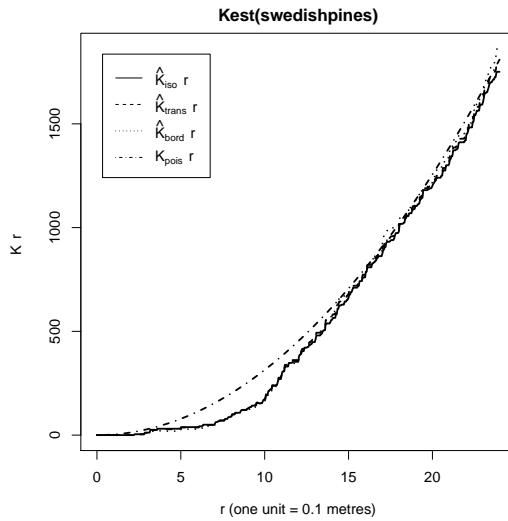


Figure 7.20: Default plot produced by `plot.fv()` in response to the command `plot(Kest(swedishpines))`.

x variable for the plot. For example, in the object `Kest(swedishpines)`, the column named `iso` contains the values of the isotropic correction estimate. The command

```
> K <- Kest(swedishpines)
> plot(K, iso ~ r)
```

plots the isotropic correction estimate against r as shown in Figure 7.21.

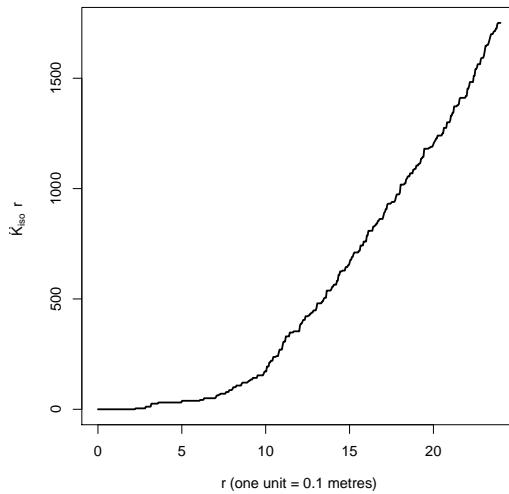


Figure 7.21: Plot of isotropic-corrected estimate against r .

Both sides of the plot formula are interpreted as *mathematical expressions*, so that operators like “+”, “-”, “*”, “/” have their usual meaning in arithmetic. The right-hand side of the formula can be any expression that evaluates to a numeric vector, and the left hand side is any expression that evaluates to a vector or matrix, of compatible dimensions.

If the left hand side evaluates to a matrix then each column of that matrix is plotted against the specified x variable as a separate curve. In particular the left hand side of the formula may invoke the function `cbind()` to indicate that several different curves should be plotted. For example, to plot only the translation and isotropic correction estimators and the theoretical curve (i.e. to omit the border method estimator) an appropriate syntax is: `plot(Kc, cbind(iso, trans, theo) ~ r)` (results not shown).

The plot formula may also involve the names of constants like `pi`, standard functions like `sqrt`, and some special abbreviations listed in Table 7.2. The symbol `.x` represents the function argument, usually named `r`. One of the columns of function values will be designated as the “best” estimate, for use by some other commands in `spatstat`. This column is identified by the symbol `.y`. Some or all of the columns of function values are designated as “acceptable” estimates for the default plot, and these are identified by the symbol “`.`”. The default plotting formula is `. ~ .x` indicating that the acceptable estimates will be plotted against the function argument.

<code>.x</code>	argument of function
<code>.y</code>	best estimate of function
<code>.</code>	all estimates of function for default plot
<code>.a</code>	all columns of function values

Table 7.2: Recognised abbreviations for columns of an “fv” object. To expand these abbreviations, use `fvnames(f, ".x")` and so on.

Transformations can be applied to the function values when executing the `plot` command. For example, to divide each of the function estimates by the theoretical Poisson value, one could use `plot(K, . / theo ~ r)`. The resulting plot is shown in the left panel of Figure 7.22. Alternatively one could plot the function estimates *against* the Poisson value, using `plot(K, . ~ theo)`. This is shown in the right panel of Figure 7.22.

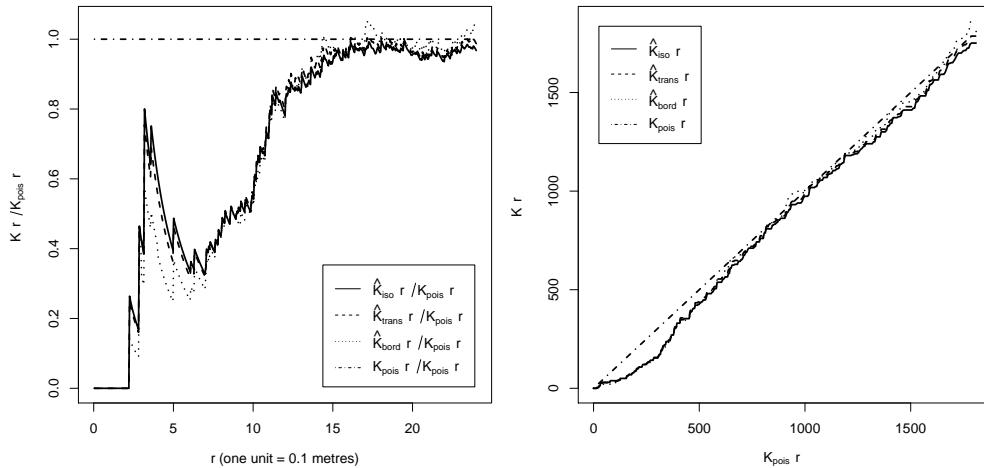


Figure 7.22: *Left:* Estimates of $K(r)$ divided by πr^2 , plotted against r . *Right:* Estimates of $K(r)$ plotted against πr^2 .

The mathematical labels for the plot axes, and for the individual curves, are constructed automatically by `spatstat` from the plot formula. If the plot formula involves the names of external variables, these will be rendered in Greek where possible. For example, to plot the average number of trees surrounding a typical tree in the Swedish Pines data,

```
> K <- Kest(swedishpines)
> lambda <- intensity(swedishpines)
> plot(K, lambda * . ~ r)
```

It is important that the intensity be named `lambda` so that it will be rendered as λ and the y axis will be labelled $\lambda K(r)$.

Mention Cox proposal (in discussion of Ripley 1977) $K(r)$ vs r^2

Refer again to “recommended range” of r

Explain how to change scale using `xlim`, `ylim`.

7.5.3 Manipulating "fv" objects

An "fv" object can be manipulated using the operations listed in Table 7.3.

<code>f</code>	print a description
<code>print(f)</code>	print a description
<code>plot(f)</code>	plot the function estimates
<code>as.data.frame(f)</code>	strip extra information (returns a data frame)
<code>f\$iso</code>	extract column named <code>iso</code> (returns a numeric vector)
<code>f[i,j]</code>	extract subset (returns an "fv" object)
<code>subset(f, ...)</code>	extract subset (returns an "fv" object)
<code>with(f, expr)</code>	perform calculations with columns of data frame
<code>eval.fv(expr)</code>	perform calculations with several "fv" objects
<code>cbind(f1, f2, ...)</code>	combine "fv" objects f_1, f_2, \dots
<code>bind.fv(f, d)</code>	combine an "fv" object f and data frame d
<code>collapse.fv(f1, f2, ...)</code>	combine several redundant "fv" objects
<code>compatible(f1, f2, ...)</code>	check whether "fv" objects are compatible
<code>harmonise(f1, f2, ...)</code>	make "fv" objects compatible
<code>min(f), max(f), range(f)</code>	range of function values
<code>Smooth(f)</code>	apply smoothing to function values
<code>deriv(f)</code>	derivative of function
<code>stieltjes(g,f)</code>	compute Stieltjes integral with respect to f
<code>as.function(f)</code>	convert to a function.

Table 7.3: Operations for manipulating an "fv" object f .

Values of the function

An "fv" object is essentially a table of values of a function. These values can be extracted directly, since the object is also a data frame. A single column of values can be extracted using the `$` operator in the usual way.

The subset extraction operator “[” has a method for "fv" objects. This always returns another "fv" object, so it will refuse to remove the column of values of the function argument r , for example. To extract entries of the table in any desired fashion, it is typically best to strip off the extra information by converting the object to a data frame using `as.data.frame()` and then using “[”.

The table of function values can also be converted to a true function in the R language using `as.function()`. This makes it easy to evaluate the function at any desired distance r .

```
> KS <- Kest(swedishpines)
> K <- as.function(KS)
> K(9)
[1] 129.096
```

By default, the result K is a function in R, with a single argument r (or whatever the original function argument was called). The new function accepts numeric values or numeric vectors of distance values, and returns the values of the “best” estimate of the function, interpolated linearly between entries in the table. If one of the other function estimates is required, use the argument `value` to select it. Several estimates can be chosen:

```
> K <- as.function(KS, value=".")  
> K(9)  
[1] 129.096  
> K(9, "trans")  
[1] 130.4998
```

Calculating with columns

To manipulate or combine one or more columns of data in an “fv” object, it is typically easiest to use `with.fv()`, a method for the generic `with()`. For example:

```
> Kr <- Kest(redwood)  
> y <- with(Kr, iso - theo)  
> x <- with(Kr, r)
```

The results x and y are numeric vectors, where x contains the values of the distance argument r , and y contains the difference between the columns `iso` (isotropic correction estimate) and `theo` (theoretical value for CSR) for the K function estimate of the redwood seedlings data. For this to work, we have to know that Kr contains columns named `r`, `iso` and `theo`. Printing the object will reveal this information.

The general syntax is `with(X, expr)` where X is an “fv” object and `expr` can be any expression involving the names of columns of X . The expression can include functions, so long as they are capable of operating on numeric vectors. The expression can also involve the abbreviations listed in Table 7.2. Thus: `Kr.tran <- with(Kr, . - pi*r^2)` subtracts the “theoretical” value from all the available edge correction estimates. In this case the result `Kr.tran` is an “fv” object. You can also get a single numeric result:

```
> with(Kr, max(abs(iso-theo)))  
[1] 0.04945199
```

Calculating with several “fv” objects

To manipulate or combine several “fv” objects, `spatstat` provides the special function `eval.fv()`. Its argument should be an expression involving *the names* of the “fv” objects which are to be combined. For example, to find the difference between the K -functions of the `redwood` and `cells` datasets,

```
> K1 <- Kest(redwood)  
> K2 <- Kest(cells)  
> DK <- eval.fv(K1-K2)
```

Note that something like `eval.fv(Kest(redwood) - Kest(cells))` would not work, because `redwood` and `cells` are not “fv” objects.

The objects in the expression should be “compatible” in the sense that they have the same column names, and the same vector of r values. However `eval.fv` will attempt to reconcile incompatible objects. (The `spatstat` generic function `compatible()` determines whether two or more objects are compatible, and the generic function `harmonise()` makes them compatible, if possible.)

7.6 The pair correlation function

7.6.1 The pair correlation function of a point process

A plot of the K -function can be difficult to interpret correctly in terms of the behaviour of the point process, because of its “cumulative” nature. The value of $K(r)$ contains contributions for all interpoint distances *less than or equal to* r .

An alternative tool is the **pair correlation function** $g(r)$ which contains contributions only from interpoint distances *equal to* r . It is defined by

$$g(r) = \frac{K'(r)}{2\pi r} \quad (7.18)$$

where $K'(r)$ is the derivative of the K -function with respect to r .

References [314, pp. 249, 284–291].

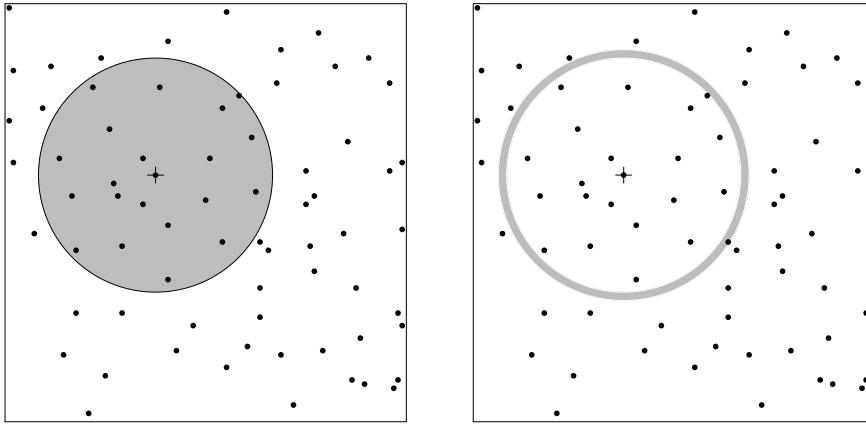


Figure 7.23: Geometry of the K -function (*Left*) and the pair correlation function (*Right*).

In geometric terms, $K(r)$ is defined by drawing a circle of radius r centred on a point of the point process, and counting the number of other points falling in the circle (see left panel of Figure 7.23). To define $g(r)$, draw two concentric circles of radius r and $r + h$, where h is a small increment of distance, and count only those points falling in the ring between the two circles (see right panel of Figure 7.23). This captures the interpoint distances d_{ij} that lie in the narrow range between r and $r + h$. The expected count is $\lambda K(r + h) - \lambda K(r)$. We standardise the expected count by dividing it by the expected value for complete spatial randomness, which is $\lambda\pi(r + h)^2 - \lambda\pi r^2$, yielding

$$g_h(r) = \frac{\lambda K(r + h) - \lambda K(r)}{\lambda\pi(r + h)^2 - \lambda\pi r^2} = \frac{K(r + h) - K(r)}{2\pi rh + \pi h^2}.$$

When h is very small, πh^2 becomes negligible, and $(K(r + h) - K(r))/h$ becomes the derivative of K , so we obtain the pair correlation function (7.18).

Imagine the observation window is divided into a fine grid of pixels. The probability that a given pixel contains a random point is $p \sim \lambda a$ where a is the pixel area. Now consider two pixels with centres u and v , separated by a distance r . See Figure 7.24. Let $p_2(u, v)$ be the probability that *both* pixels contain random points. If the process is Poisson, then events in these pixels are independent,

and $p_2(u, v) = p^2$. For any stationary and isotropic point process, it turns out that

$$g(r) \sim \frac{p_2(u, v)}{p^2}. \quad (7.19)$$

That is, $g(r)$ is the probability of observing a pair of random points separated by a distance r , divided by the corresponding probability for a Poisson process. The value $g(r) = 1$ is consistent with complete spatial randomness, because of the way the function has been standardised. A value $g(r) < 1$ indicates that interpoint distances equal to r are less frequent than would be expected for a completely random process, so this suggests regularity. A value $g(r) > 1$ indicates that this interpoint distance is more frequent than expected for a completely random pattern, which suggests clustering. These interpretations are less ambiguous than the corresponding statements for the K -function, because they refer only to interpoint distances equal to r .

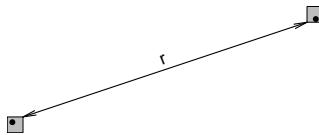


Figure 7.24: Interpretation of pair correlation as giving the probability that two small regions, separated by distance r , both contain a random point.

Notice that the value $g(r)$ is not a correlation in the sense used by statisticians. The correlation between two random variables is a number standardised to lie in the range between -1 and +1, with the value 0 indicating a lack of correlation. It turns out to be impractical to standardise the correlation structure of point processes in this way. Instead, $g(r)$ is the kind of correlation often used in physics: the possible values range from 0 to infinity, and the value 1 is associated with a lack of correlation.

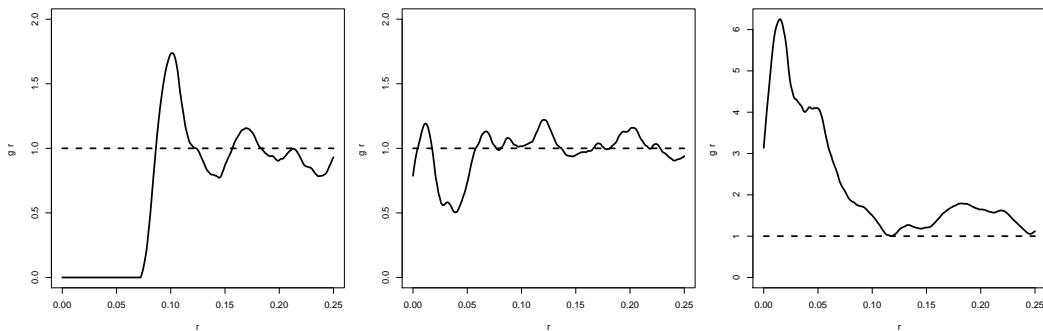


Figure 7.25: Empirical pair correlation function for each of the three patterns in Figure 7.1.

Figure 7.25 shows estimates of the pair correlation function for the three archetypal point patterns. Compare these with the empirical K -functions shown in Figure 7.7 on page 171.

The left panel of Figure 7.7 shows that the K -function for the regular pattern is below the Poisson curve for all distances r , suggesting (if we are not careful) that there is inhibition at *all* distances. The plot of the pair correlation has a different message: $g(r)$ is below the Poisson value, $g(r) < 1$, for distances r up to 0.07 units, then it climbs steeply to a value *greater than* the Poisson value, $g(r) > 1$ at about $r = 0.10$ units, before declining back to 1. In fact $g(r)$ is zero for $r \leq 0.07$ because this extremely regular pattern has no interpoint distances shorter than 0.07 units. The reason for the apparent discrepancy between g and K is clear: short interpoint distances are completely absent in

this pattern, and this deficit affects the cumulative count of pairwise distances in the K -function, even at much larger distances r .

The K -function and the pair correlation function g are mathematically interrelated: g can be obtained from K through (7.18), and K can be obtained from g by the reverse relationship

$$K(r) = \int_0^r sg(s) ds. \quad (7.20)$$

One may ask why anyone would use the K -function, if the pair correlation function g is easier to interpret. The main reason is that K seems to perform better as a basis for statistical inference (such as hypothesis tests).

Also, in theory g does not necessarily exist: it is essentially the derivative of the K -function, and this does not exist if the K -function has jumps. For example, for a regular grid of points (randomly translated to make it a stationary point process), with points spaced at multiples of s units, the K -function has a jump at $r = s$ and at $r = 2d$, $r = s\sqrt{2}$ and so on. This process has no pair correlation function. This example could be realistic for some crystalline structures.

7.6.2 Estimating the pair correlation function

A popular way to estimate a probability density from data is to smooth the data using a kernel [300, 296]. Similarly a good way to estimate the pair correlation function is by kernel smoothing.

Taking any edge-correction estimator of the K -function of the general form (7.3), we replace the indicator $\mathbf{1}\{d_{ij} \leq r\}$ by a kernel function to obtain an estimate of $K'(r)$, then plug into the formula (7.18) for the pair correlation function, to obtain the “fixed-bandwidth” kernel estimator [314, eq. (15.15), p. 284]

$$\hat{g}(r) = \frac{1}{2\pi r} \sum_{i=1}^n \sum_{\substack{j=1 \\ j \neq i}}^n \kappa_h(r - d_{ij}) e_{ij}(r) \quad (7.21)$$

where $d_{ij} = \|x_i - x_j\|$ is the interpoint distance between the i th and j th data points, $e_{ij}(r)$ is an edge correction weight, and κ_h is the smoothing kernel, with smoothing bandwidth $h > 0$.

The kernel κ_h is a rescaled version of a template kernel κ ,

$$\kappa_h(x) = \frac{1}{h} \kappa\left(\frac{x}{h}\right) \quad (7.22)$$

where κ is any chosen function that is a probability density over the real line with mean 0. For example, κ could be the standard Normal (Gaussian) density, so that κ_h would be the normal density with mean 0 and standard deviation h . The convention followed in R and in `spatstat` is that κ has standard deviation 1, so that κ_h has standard deviation h .

The usual choice of smoothing kernel for pair correlation functions is the *Epanechnikov kernel* with half-width w ,

$$\varepsilon_w(x) = \frac{3}{4w} \left(1 - \frac{x^2}{w^2}\right)_+, \quad (7.23)$$

a quadratic function truncated to the interval $[-w, w]$. The standard deviation of $\varepsilon_w(x)$ is $h = w/\sqrt{5}$, so the kernel of bandwidth h is $\kappa_h(x) = \varepsilon_{h\sqrt{5}}(x)$.

To compute (7.21) in practice we must choose a bandwidth value h , and as usual, this involves a compromise between bias and variability. For excessively large bandwidth h , variability will be well-controlled but the shape of the function \hat{g} will be over-smoothed, so that important detail in the pair correlation function may be missed. For excessively small bandwidth, over-smoothing will not occur, but the estimates $\hat{g}(r)$ will have high variance, and the shape of \hat{g} will be erratic.

The standard rule of thumb, recommended by Stoyan is to take the half-width w of the Epanechnikov kernel (7.23) to be $w = c/\sqrt{\hat{\lambda}}$ where c is a constant between 0.1 and 0.2. This corresponds to taking the bandwidth

$$h = \frac{c'}{\sqrt{\hat{\lambda}}} \quad (7.24)$$

where $c' = c/\sqrt{5}$ is between 0.045 and 0.090.

To compute the estimated pair correlation function in **spatstat** use `pcf()`.

```
> g <- pcf(cells)
> plot(g)
```

The function `pcf()` is generic, with a method for point patterns. By default, `pcf.ppp()` uses the estimator (7.21), with the Epanechnikov kernel, with bandwidth h selected by Stoyan's rule of thumb (7.24) with $c = 0.15$.

In analysing a point pattern dataset, attention is often focused on the behaviour of $g(r)$ for small r , which reflects the correlation between close pairs of points. Unfortunately, estimation of $g(r)$ at small distances is intrinsically difficult, because geometry dictates that there will typically be few observations at small distances. The estimator (7.21) has poor performance at small distances. Theoretically the variance of $\hat{g}(r)$ becomes infinite as r goes to zero, for many point processes including CSR. The left panel of Figure 7.26 shows a real example where $\hat{g}(r)$ has an infinite asymptote at $r = 0$. These problems are due to the factor $1/r$ in (7.21). An alternative estimator with usually better performance is

$$\hat{g}(r) = \frac{1}{2\pi} \sum_{i=1}^n \sum_{\substack{j=1 \\ j \neq i}}^n \frac{\kappa_h(r - d_{ij})}{d_{ij}} e_{ij}(r) \quad (7.25)$$

in which the contribution to $\hat{g}(r)$ from an observed interpoint distance d_{ij} involves a factor $1/d_{ij}$ rather than $1/r$. This estimator can be invoked by specifying the argument `divisor="d"` in `pcf.ppp()`.

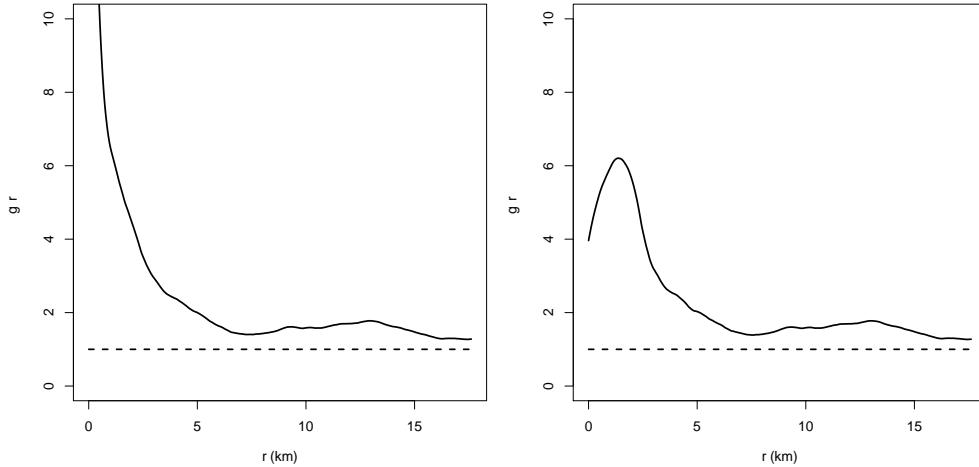


Figure 7.26: Estimates of the pair correlation function of the Queensland copper deposits of Figure 1.11, using (Left) the divide-by- r estimator (7.21) and (Right) the divide-by- d estimator (7.25).

Another useful trick applies when an estimate of $K(r)$ is already available; then we can calculate

$g(r)$ from $K(r)$ through equation (7.18). This is particularly useful in large datasets, where direct estimation of $g(r)$ can be time-consuming. The conversion of $K(r)$ to $g(r)$ is done by the method `pcf.fv()`, which computes the derivative by numerical differentiation using spline smoothing. Effectively the function $K(r)$ is first smoothed by fitting polynomial approximations to $K(r)$ over successive ranges of r values. The derivatives of the polynomials are then computed, using elementary calculus.

```
> K <- Kest(shapley)
> g <- pcf(K, spar=0.5)
```

Since `K` is an object of class "fv", the last command is dispatched to `pcf.fv()`. Additional arguments to `pcf.fv()` such as `spar` are passed to the function `smooth.spline()` in the `stats` package, which performs the spline smoothing. Using a little algebra, we may apply smoothing either to $K(r)$ or to the transformations $K(r)/(2\pi r)$, $K(r)/(\pi r^2)$ or $L(r) = \sqrt{K(r)/\pi}$, by specifying `method="a"`, `"b"`, `"c"` or `"d"` respectively. Methods `"b"` to `"d"` offer improved performance at small distances r . The default is `method="c"` which seems to perform best overall.

7.7 Standard errors and confidence intervals

Statistical estimates should be accompanied by a measure of accuracy (such as a standard error) or uncertainty (such as a confidence interval). This is not straightforward for the K -function: there is no simple expression for the standard error or variance of $\widehat{K}(r)$, even in a Poisson process, because the contributions to $\widehat{K}(r)$ from different data points x_i are not independent.

7.7.1 Variance under CSR

Approximations to the standard error of $\widehat{K}(r)$ assuming complete spatial randomness are available, using the techniques of U -statistics [284, 260, 234, 127]. Ripley [284] gave an approximate variance estimate for the isotropic corrected K function estimator,

$$\text{var } \widehat{K}_{iso}(r) \approx 2 \left(\frac{A}{n-1} \right)^2 \left(\frac{\pi r^2}{A} + \frac{0.96Pr^3}{A^2} + 0.13 \frac{nPr^5}{A^3} \right) \quad (7.26)$$

where n is the number of data points and A and P are the area and perimeter length of the window W . Lotwick and Silverman [234] gave a more intricate and accurate approximation for the case of a rectangular window. These approximations are chiefly useful for predicting the accuracy that can be expected from a survey region of given size and shape. They are computed by `Kest()` if the argument `var.approx=TRUE` is given. For the L -function, the corresponding approximations for $\text{var } \widehat{L}_{iso}(r)$ are calculated by the delta method.

7.7.2 Block bootstrap

Bootstrap methods [138, 185] make it possible to estimate variance from the data, without assuming the Poisson model. A very simple version of this approach is *block variance estimation*. Assuming the window W is a rectangle, divide it into equal quadrats B_1, \dots, B_m . Suppose our estimator of $K(r)$ is (7.3). For a particular quadrat B , let

$$\widehat{K}(r, B) = \frac{m|W|}{n(n-1)} \sum_{x_i \in B} \sum_{j \neq i} \mathbf{1}\{d_{ij} \leq r\} e_{ij}(r) \quad (7.27)$$

be an estimate of $K(r)$ based only on the pairs (x_i, x_j) for which x_i falls in the quadrat B , while x_j may lie anywhere in W . The usual estimate $\hat{K}(r)$ is the average of the estimates $\hat{K}(r, B_k)$ over all $k = 1, \dots, m$. We compute the pointwise sample mean, sample variance, and sample standard deviation of these estimates, yielding an elementary bootstrap estimate [127, eq. (4.21), p. 52] of the standard error for $\hat{K}(r)$. Assuming $\hat{K}(r)$ is approximately normally distributed, we can then calculate pointwise 95% confidence intervals for the true value of $K(r)$.

The block variance estimate is computed by the `spatstat` function `varblock()`:

```
> X <- copper$SouthPoints
> Kvb <- varblock(X, Kest, nx=3, ny=3)
> plot(Kvb, iso ~ r, shade=c("loiso", "hiiso"))
```

The result is shown in the left panel of Figure 7.27. For a fixed value of r , the grey shading gives a 95% confidence interval for the true value of $K(r)$. Formally this means that, in 95% of outcomes of this procedure, the random interval computed here would embrace the true value of $K(r)$.

Notice that the confidence interval becomes wider as r increases, because the variance of $\hat{K}(r)$ is roughly proportional to r^2 . If we use $L(r)$ instead of $K(r)$, the variance will be stabilised: $\text{var}\hat{L}(r)$ is approximately constant as a function of r . The confidence intervals will then have roughly constant width.

An important caveat is that these are *pointwise* calculations (i.e. performed separately for each value of distance r) yielding *pointwise* confidence intervals (i.e. valid only for a single value of r at a time). Our confidence that the true K -function lies *entirely inside* the grey shaded region over all values of r , is vastly less than 95%. A solution to this problem is explained below.

7.7.3 Loh's bootstrap

A more flexible bootstrap approach was developed by Loh [232]. First we decompose the empirical K -function (7.3) into the contributions from each individual data point x_i ,

$$\hat{K}(r, x_i) = \frac{|W|}{n-1} \sum_{j \neq i} \mathbf{1}\{d_{ij} \leq r\} e_{ij}(r) \quad (7.28)$$

called the **local K** -functions. The usual estimate $\hat{K}(r)$ is the average of the estimates $\hat{K}(r, x_i)$ over all $i = 1, \dots, n$. We could again calculate the pointwise sample mean and sample variance of these local K -functions. Instead, Loh's approach is to resample from the suite of local K -functions. Choose an independent random sample of size n , with replacement, from the numbers 1 to n . If the sample is i_1, \dots, i_n , we calculate the average of the corresponding local K -functions,

$$K^*(r) = \frac{1}{n} \sum_{k=1}^m \hat{K}(r, x_{i_k}). \quad (7.29)$$

This gives a resampled version of $\hat{K}(r)$. Repeating this procedure a large number N of times, we obtain a bootstrap sample of the distribution of $\hat{K}(r)$. Pointwise confidence intervals can then be computed directly from the observed quantiles of the bootstrap sample. This calculation is performed by the `spatstat` function `lohboot()`:

```
> Kloh <- lohboot(X, "Kest")
> plot(Kloh, cbind(iso, theo) ~ r, shade=c("lo", "hi"))
```

The result is shown in the right panel of Figure 7.27. The default behaviour is to compute a pointwise 95% confidence interval.

The grey-shaded intervals in Figure 7.27 are pointwise confidence intervals. Ideally we would prefer a *global* confidence interval or confidence band, having a 95% chance of containing the

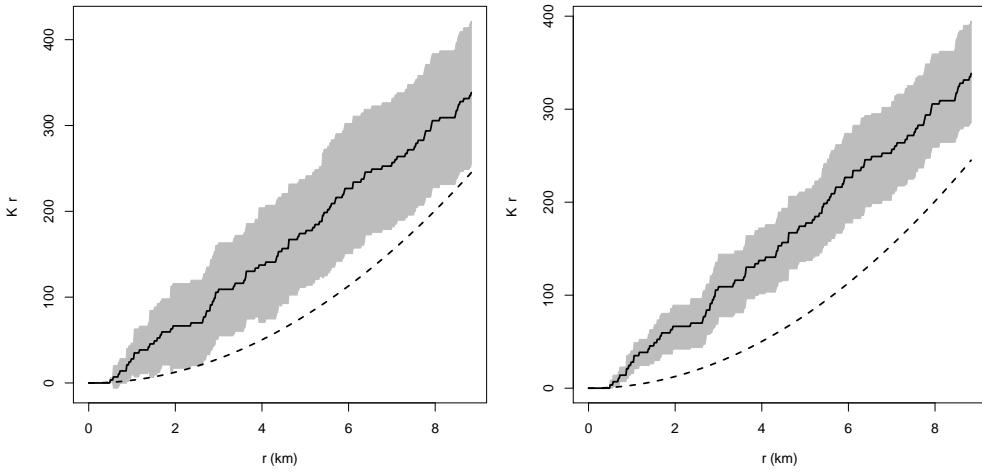


Figure 7.27: Pointwise 95% confidence intervals for the true K -function of the Queensland copper deposits, southern half, using (Left) the spatial block bootstrap `varblock`, and (Right) Loh's bootstrap `lohboot`.

entire graph of the true K -function. This is possible in Loh's approach. First we replace $K(r)$ by $L(r) = \sqrt{K(r)/\pi}$ to stabilise the variance. For each resampled L -function $L^*(r)$, we calculate the deviation

$$D^* = \max_{0 \leq r \leq r_{max}} |L^*(r) - \hat{L}(r)|,$$

the maximum vertical separation between the graphs of $L^*(r)$ and $\hat{L}(r)$ over the interval of r values up to a nominated maximum distance r_{max} . If there are N resampled L -functions, this gives N deviation values; we take the 95th percentile of these deviations, say $D_{0.95}$, and construct the **global confidence band** with boundaries

$$\begin{aligned} L_-(r) &= \hat{L}(r) - D_{0.95} \\ L_+(r) &= \hat{L}(r) + D_{0.95}. \end{aligned} \tag{7.30}$$

This has the desired interpretation that, in 95% of outcomes of this procedure, the true L -function will lie entirely between the two limits $L_-(r), L_+(r)$. This procedure is performed by `lohboot()` with the argument `global=TRUE`:

```
> Lg <- lohboot(X, "Lest", global=TRUE)
```

A global confidence band for $K(r)$ can then be obtained by back-calculation:

```
> Kg <- eval.fv(pi * Lg^2)
```

The results are plotted in Figure 7.28.

7.8 Testing statistical significance

The K -function estimated from a point pattern dataset, $\hat{K}(r)$, is often compared graphically with the theoretical K -function for the homogeneous Poisson process, $K_{pois}(r) = \pi r^2$, serving as the

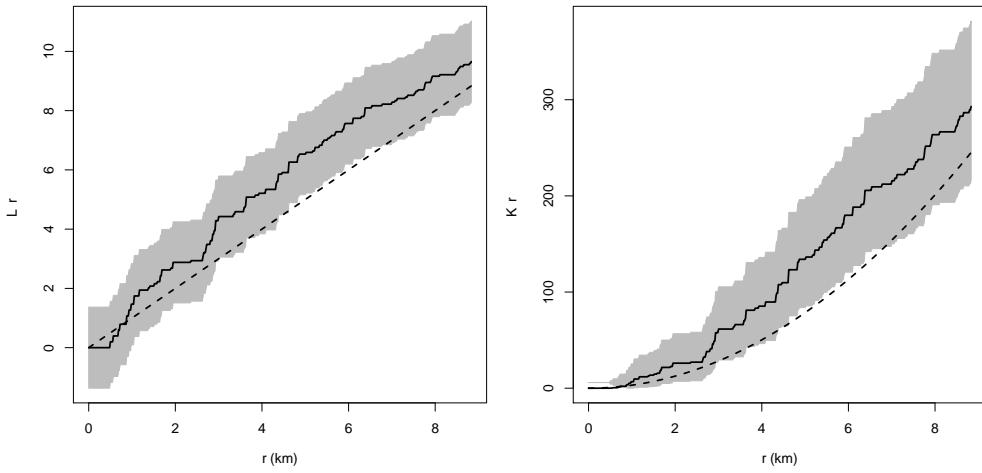


Figure 7.28: Global 95% confidence interval for the true L -function (Left) and K -function (Right) of the Queensland copper deposits, southern half, using Loh's bootstrap.

benchmark of a completely random pattern. In the examples above, large discrepancies between $\hat{K}(r)$ and $K_{pois}(r)$ suggested that the example datasets are not completely random. However, even with a completely random pattern, we will never obtain perfect agreement between \hat{K} and K_{pois} , because of random variability. We need a rule for deciding whether the difference between \hat{K} and K_{pois} is large enough (relative to the scale of variability) to convince us that the point process is not completely random: that is, whether the discrepancy between \hat{K} and K_{pois} is *statistically significant*. This rule is a *significance test* or *hypothesis test*.

Comparing our data with a completely random process is not the only comparison of interest. There might be a more appropriate “null hypothesis” for our study, a point process model with theoretical K -function $K_0(r)$. We would then be assessing statistical significance of deviations between $\hat{K}(r)$ and $K_0(r)$.

There are also *two-sample* comparisons in which we test for a significant difference between the K -functions of two point patterns, for example, surveys of the same species of tree in two different forests.

The summary statistic used in the significance test could be the K -function, the pair correlation function g , or some other statistic.

7.8.1 Pointwise envelopes

Suppose we are testing whether the point pattern data are consistent with complete spatial randomness. In order to assess the statistical significance of deviations between the observed K -function and the theoretical K -function for CSR, essentially we need to know how much variability should be expected in the estimate $\hat{K}(r)$ if the pattern is completely random.

The left panel of Figure 7.29 shows the K -function estimated from the `cells` dataset (thick black line), and also the K -functions of 39 simulated realisations of CSR with the same number of points (grey lines). Each grey line could have been generated by typing

```
plot(Kest(runifpoint(npointr(cells)), Window(cells))))
```

The spread of the grey lines typifies the variability in $\hat{K}(r)$ that we should expect if the pattern is completely random with the same number of points as the data. Clearly, the estimate of $K(r)$ for the `cells` data lies outside this range, for some values of r .

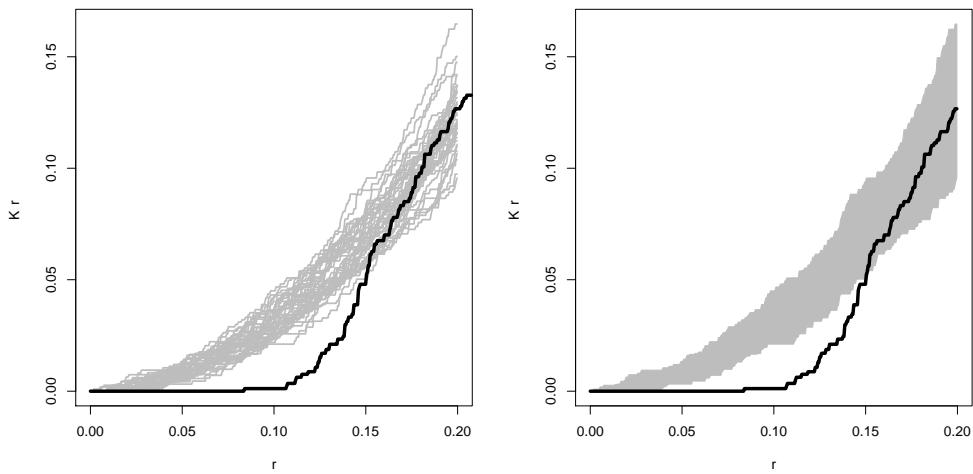


Figure 7.29: Construction of envelopes of the K function. *Left:* the K function estimate of the `cells` dataset (black line), and of 39 simulated realisations of CSR with the same number of points (grey lines). *Right:* grey shading between the upper and lower *envelopes* (maximum and minimum values) of the K -functions of the simulated patterns.

In the right panel of Figure 7.29 we have shaded the region between the highest and lowest grey lines, that is, the minimum and maximum values of the K -functions of the simulated patterns, called the upper and lower *envelopes* of the simulated functions.

The envelope plot can be interpreted as a significance test, in the following way. Choose a particular distance r , say $r^* = 0.1$. If the `cells` data were completely random, then the data and simulated patterns would be statistically equivalent, since they would all be completely random patterns with the same number of points. Consequently the values of $\hat{K}(r^*)$ for the data and simulated patterns would be statistically equivalent numbers. By symmetry, since there are 40 values altogether (one data value and 39 simulated values), there would be a chance of 1 in 40 that the value for the data is the smallest, i.e. that the data value is smaller than all of the simulated values. This has in fact occurred in Figure 7.29, and the probability that this happens completely by chance is $1/40 = 0.025$, so we could declare the result to be statistically significant with a p -value of 0.025.

This is an example of a *Monte Carlo test*, a test based on simulations from the null hypothesis, using a symmetry principle. A full explanation of Monte Carlo tests is given in Chapter 10.

We have just described a “one-sided” test, in which the result is declared statistically significant if the data value lies below the minimum of the simulated values. In a “two-sided” test, the result is declared statistically significant if the data value lies either below the minimum or above the maximum of the simulated values. In the example above, the probability that this happens by chance would be $2/40 = 0.05$.

The `spatstat` function `envelope()` computes simulation envelopes. The right panel of Figure 7.29 was generated by the commands

```
> E <- envelope(cells, Kest, nsim=20, fix.n=TRUE)
> plot(E)
```

The result of `envelope()` is an object of class "envelope" which also belongs to the class "fv". It can be plotted, printed, and manipulated using the tools for "fv" objects described in Section 7.5. See Table 7.3 on page 188. The print method gives a lot of detail:

```
> E
```

```

Pointwise critical envelopes for K(r)
and observed value for 'cells'
Edge correction: "iso"
Obtained from 20 simulations of CSR
Alternative: two.sided
Significance level of pointwise Monte Carlo test: 2/21 = 0.0952
.....
Description
r    distance argument r
obs  observed value of K(r) for data pattern
theo theoretical value of K(r) for CSR
lo   lower pointwise envelope of K(r) from simulations
hi   upper pointwise envelope of K(r) from simulations
.....
Default plot formula: .~r
where "." stands for 'obs', 'theo', 'hi', 'lo'
Columns 'lo' and 'hi' will be plotted as shading (by default)
Recommended range of argument r: [0, 0.25]
Available range of argument r: [0, 0.25]

```

Arguments to `envelope()` allow us to specify the data pattern, the summary function to be used, the number of simulations, and so on. Table 10.1 on page 355 lists the most useful arguments to `envelope()`. Numerous modifications to the test procedure are possible. By default, `envelope()` assumes a two-sided test using the maximum and minimum of the simulated functions. One could use the k th largest and k th smallest simulated values to determine the envelopes, by specifying the argument `nrank` equal to k .

The right panel of Figure 7.29 indicates that we would have obtained a statistically significant result if we had chosen the distance value r^* to be anywhere between 0.05 and 0.15 distance units. This can be very useful for understanding the spatial scale of correlation in the point pattern. However, there is a very important caveat here. The rationale for declaring statistical significance, explained above, assumes that the distance r^* is fixed, and that it was chosen in advance, without looking at the Figure. It does not allow us to first generate the Figure and then choose a favorable value of r^* . Effectively, each distance value r represents a different hypothesis test. If we were to scan the entire graph, looking at the test outcomes for each value of r , and pick a value of r which gave us the answer we wanted, this would clearly be unfair. The probability that the graph of $\widehat{K}(r)$ does outside the envelope of the simulated functions for some value of r , is much greater than 0.05, so this is not statistically significant. This is the problem of *multiple testing* or the *look elsewhere effect*.

BEGIN: Alternative forms of words

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 (making the conclusion much less ‘significant’).

Asking whether there is a statistically significant difference between $\widehat{K}(r)$ and $K_{pois}(r)$ at a pre-specified distance r , is different from asking whether there is a significant difference at *any* distance r .

END: alternative forms of words

Envelopes computed in the manner of Figure 7.29 are called *pointwise* envelopes because the maximum and minimum are calculated separately for each distance r (i.e. for each ‘point’ on the horizontal axis). The printed output above for the “envelope” object `E` reminds us that it contains pointwise envelopes, so their correct interpretation depends on fixing a value of r in advance. In the wrong hands these can be misinterpreted.

7.8.2 Global envelopes

The problem of multiple testing can be avoided using “global” envelopes, shown in Figure 7.30. As distinct from the “pointwise envelopes” in Figure 7.29, the “global envelopes” in Figure 7.30 delimit a zone of *constant width*. The width is determined by finding the most extreme deviation from the theoretical K -function that is achieved by any of the simulated K -functions, at any distance r along the horizontal axis.

That is, we compute the maximum vertical deviation between the graphs of \hat{K} and K_{pois} ,

$$D = \max_r |\hat{K}(r) - K_{pois}(r)|. \quad (7.31)$$

This is computed for each of the m simulated datasets, giving values D_1, \dots, D_m . The maximum deviation $D_{max} = \max_j D_j$ is taken. The global envelopes are

$$\begin{aligned} A(r) &= K_{pois}(r) - D_{max} \\ B(r) &= K_{pois}(r) + D_{max} \end{aligned} \quad (7.32)$$

The graph of the estimated K function for the data transgresses these limits with probability $1/(m+1)$ if the null hypothesis is true. Taking $m = 19$ gives a test with significance level 0.05.

Global envelopes were also proposed by Ripley [281, 282, 283], but do not appear to have been used in the ecological literature; in particular, they are not discussed by Kenkel [197] or Loosmore and Ford [233].

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

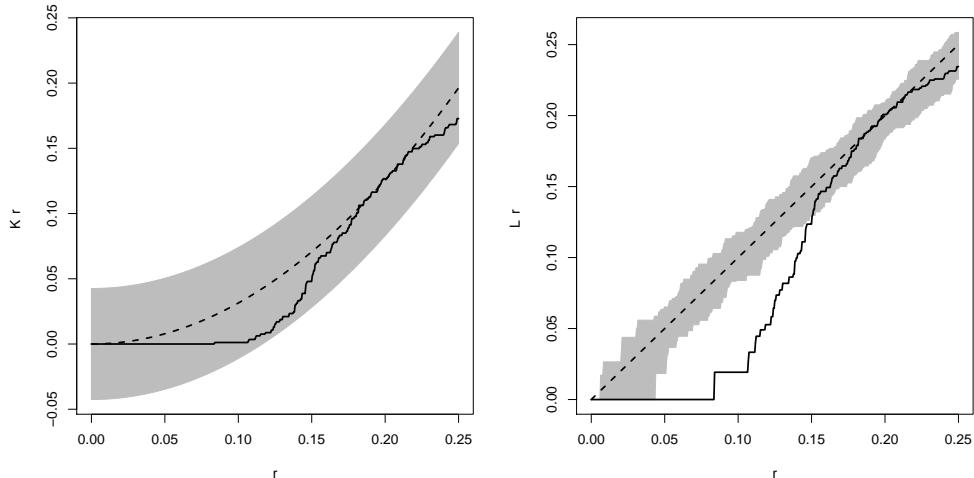


Figure 7.30: Global envelopes relating to the `cells` data. *Left:* global envelope for the K function. *Right:* global envelope for the L function.

A more powerful test is obtained if we (approximately) stabilise the variance, by using the L function in place of K , as shown in the right panel of Figure 7.30. This clearly rejects the null hypothesis of CSR at the 0.05 significance level.

7.8.3 Envelopes for any null hypothesis

Instead of testing for significant deviations from CSR, we can test for significant deviations from any null hypothesis, using the same Monte Carlo test procedure. The only requirement is that we must be able to simulate realizations of the point process that is specified by the null hypothesis.

The `envelope()` function can generate envelopes based on *any* null hypothesis. The user must specify the null hypothesis, either by

- a fitted point process model (object of class "ppm" or "kppm") representing the null hypothesis; or
- a recipe for generating simulated realizations of the null hypothesis; or
- a list of point patterns, taken to be realizations of the null hypothesis.

We explain these in reverse order, below.

List of simulated patterns

Diggle suggested that an appropriate model for the `hamster` dataset was a Simple Sequential Inhibition model with hard core distance . A simulated realisation `X` of this model can be computed by

```
> ham <- rescale(unmark(hamster))
> n <- npoints(ham)
> W <- Window(ham)
> r <- min(nndist(ham)) * n/(n+1)
> X <- rSSI(r, n, W)
```

We could simply generate a list of 39 such patterns:

```
> SimList <- list()
> for(i in 1:39) SimList[[i]] <- rSSI(r, n, W)
```

then use this list as the argument `simulate` for the `envelope()` command:

```
> ES <- envelope(ham, Lest, nsim=39, simulate=SimList)
```

If global envelopes (Section 7.8.2) are desired, we also need to compute the expected value of the summary function under the null hypothesis, in order to calculate the deviation D analogous to (7.31). By default, `envelope()` estimates the theoretical value using a separate set of `nsim` simulations from the null hypothesis, so that altogether $2 * \text{nsim}$ simulations are required. To obtain global envelopes we would type

```
> for(i in 1:78) SimList[[i]] <- rSSI(r, n, W)
> EG <- envelope(ham, Lest, nsim=39, simulate=SimList, global=TRUE)
```

The result is plotted in the left panel of Figure 7.31 using the plot formula `. - r ~ .` to bring out more detail.

Simulation recipe

An alternative way to do this is to specify the “recipe” for generating simulated realisations:

```
> recipe <- expression(rSSI(r, n, W))
```

An expression in the R language represents a command that has not yet been executed. It can be executed or “evaluated” using `eval`.

```
> recipe
expression(rSSI(r, n, W))
> eval(recipe)
```

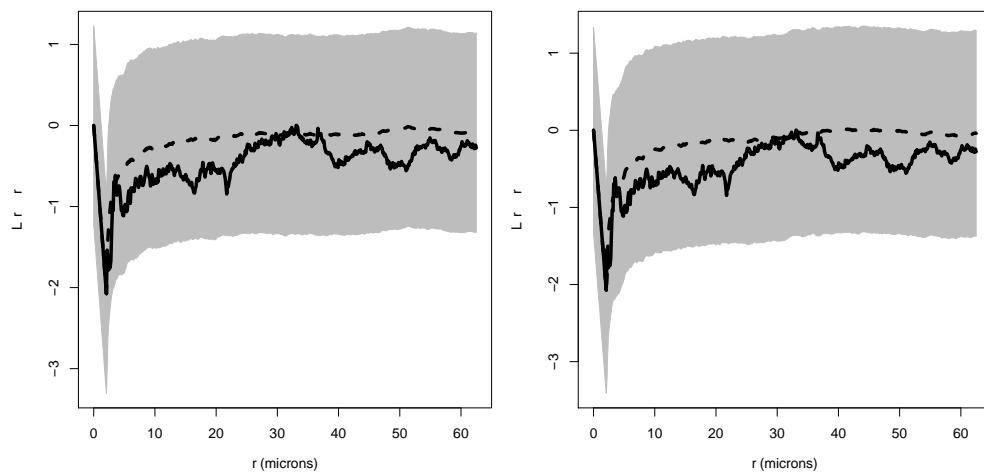


Figure 7.31: Global envelopes for testing the null hypothesis of Simple Sequential Inhibition (*Left*) and a Gibbs Hard Core process (*Right*) for the hamster data.

```
planar point pattern: 303 points
window: rectangle = [0, 250] x [0, 250] microns
```

If we repeatedly type `eval(recipe)`, a different simulated realisation of SSI will be generated each time. The recipe can be passed to `envelope()`:

```
> ES2 <- envelope(ham, Lest, nsim=39, global=TRUE,
                     simulate=recipe)
```

The recipe is then evaluated 39 times to estimate the theoretical L -function for the SSI model, and 39 more times to calculate the envelopes.

Fitted point process model

Envelopes can be based on a point process model which has been fitted to the point pattern dataset. The model-fitting functions `ppm()` and `kppm()` are described in Chapters 9 and 12 respectively. For a brief example, suppose we fit a “hard core” point process model to the unmarked hamster data:

```
> fit <- ppm(ham ~ 1, Hardcore())
```

then simulation envelopes based on this fitted model can be computed simply by

```
> EH <- envelope(fit, Lest, nsim=39, global=TRUE)
```

The result is shown in the right panel of Figure 7.31.

7.8.4 Non-graphical tests

7.9 Detecting anisotropy

A point process is ‘isotropic’ if all its statistical properties are unchanged when it is rotated. If there is any statistical property of the process that does change when the process is rotated, then the

process is ‘anisotropic’. A point process could be stationary, but not isotropic. The microstructure of a mineral could be homogeneous, but with a preferential direction, perhaps reflecting the history of deformation.

The following code simulates a homogeneous, anisotropic point process. First we generate a Simple Sequential Inhibition process, and then deform space by squashing the y axis.

```
> X <- rSSI(0.05, win=owin(c(0,1), c(0, 3)))
> Y <- affine(X, mat=diag(c(1, 1/3)))
```

Figure 7.32 shows the simulated pattern Y and its Fry plot, clearly indicating that the pattern is anisotropic.

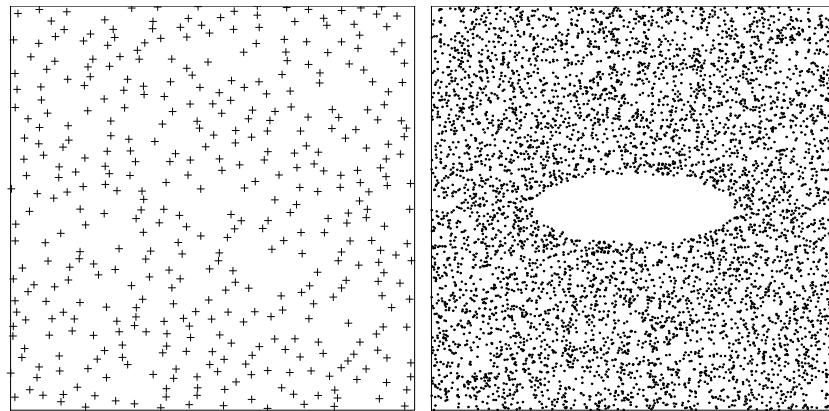


Figure 7.32: Anisotropy. *Left*: synthetic point pattern exhibiting anisotropy. *Right*: Fry plot.

7.9.1 Sector K -function

Various modifications of the K -function have been suggested in order to measure anisotropy. Instead of counting all data points falling inside a circle of radius r centred at a data point, we could replace the circle by another geometrical shape.

The *sector* shown in Figure 7.33 is the intersection between a disc of radius r centred at a data point (where r is the function argument) and two lines at orientations of α and β (which are fixed parameters) measured anticlockwise from the x -axis. The *sector K -function* of a stationary point process is $1/\lambda$ times the expected number of points lying within this sector, given that the vertex of the sector is a point of the process.

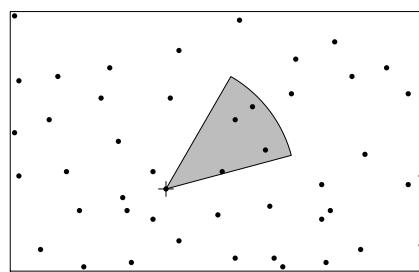


Figure 7.33: Geometry of the sector K -function.

The `spatstat` function `Ksector` computes the sector K -function using the translation edge

correction, together with the theoretical value for a Poisson process, $K_{sector, pois}(r) = (\beta - \alpha)r^2/2$ if α, β are expressed in radians.

For example, to compute the sector K -functions for two 30-degree angle sectors centred on the x and y axes respectively:

```
> Khoriz <- Ksector(Y, begin = -15, end = 15, units="degrees")
> Kvert <- Ksector(Y, begin = 90-15, end = 90+15, units="degrees")
```

Anisotropy would be suggested if these two functions appeared to be unequal. We may then compare them by superimposing the plots, or by using `eval.fv` to compute the difference between the functions. The left panel of Figure 7.34 shows the result of

```
> plot(Khoriz, trans/theo ~ r, lty=2)
> plot(Kvert, trans/theo ~ r, add=TRUE)
```

where we have divided each estimate by the theoretical value.

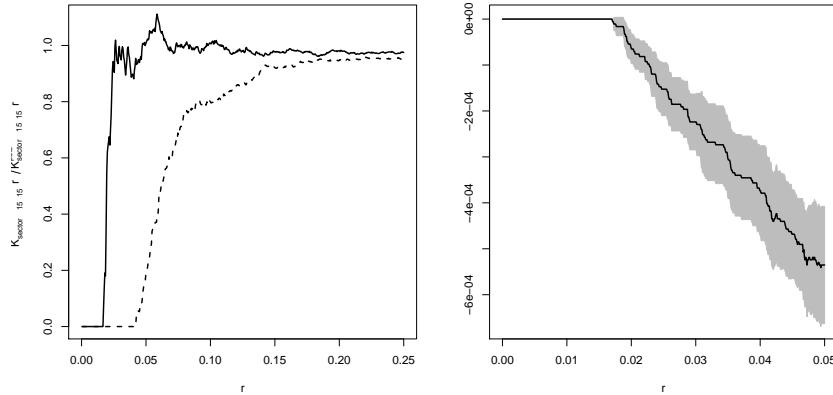


Figure 7.34: Anisotropy analysis using `Ksector`. *Left*: Superimposed plot of normalised `Ksector` functions for 30-degree sectors centred on the horizontal axis (dashed line) and vertical axis (solid line). *Right*: 95% confidence interval for the difference between horizontal and vertical sector K -functions using block bootstrap.

The right panel of Figure 7.34 shows the difference $K_{horiz} - K_{vert}$ and a 95% confidence interval computed by the following code:

```
> dK <- function(X, ...) {
  K1 <- Ksector(X, ..., begin = -15, end = 15, units="degrees")
  K2 <- Ksector(X, ..., begin = 90-15, end = 90+15, units="degrees")
  eval.fv(K1-K2)
}
> CIDK <- varblock(Y, dK, nx=5)
```

The sector K function requires a choice of the limiting angles α and β , so is best used when there is some prior knowledge about the preferential directions in the pattern.

7.9.2 Pair orientation distribution

Consider all pairs of points in a point pattern that lie more than r_1 and less than r_2 units apart, where $r_1 < r_2$ are fixed. For each such pair, measure the *direction* of the arrow joining the points,

as an angle in degrees anticlockwise from the x axis. The cumulative distribution function of these directions is the *point pair orientation distribution function*.

The geometry is shown in Figure 7.35. The point pair orientation distribution function is [?, (14.53), p. 271]

$$O_{r_1, r_2}(\phi) = \frac{\mathbb{E} [\text{number of points in } O(u, r_1, r_2, \phi) \mid \mathbf{X} \text{ has a point at location } u]}{\mathbb{E} [\text{number of points in } O(u, r_1, r_2, 360) \mid \mathbf{X} \text{ has a point at location } u]}, \quad 0 \leq \phi \leq 360 \quad (7.33)$$

where $O(u, r_1, r_2, \phi)$ is the region bounded by circles of radius r_1 and r_2 centred at u and lines at orientation 0 and ϕ through u , shown in Figure 7.35.

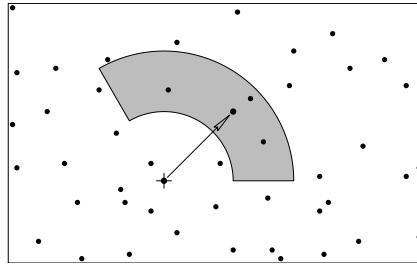


Figure 7.35: Geometry of the pair orientation distribution.

The `spatstat` function `pairorient` computes the point pair orientation distribution function. Usually it is better to estimate the derivative, using `deriv.fv` specifying that the derivative is a periodic function.

```
> f <- pairorient(Y, 0.02, 0.05, units="degrees")
> Df <- deriv(f, spar=0.6, Dperiodic=TRUE)
```

The graphs of f and Df shown in Figure 7.36 strongly indicate a preferential direction around 90 degrees.

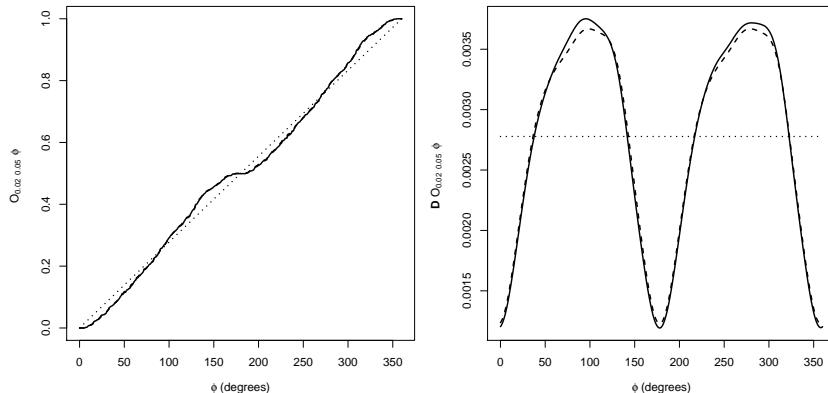


Figure 7.36: Pair orientation distribution. *Left:* Cumulative distribution function. *Right:* Density function. Dotted lines correspond to uniform distribution of angles.

The point pair orientation distribution can be an effective tool for finding a preferential direction, but seems to be quite sensitive to the choice of the distances r_1 and r_2 .

7.9.3 Anisotropic pair correlation function

These considerations lead us back to the Fry plot (Section 7.2.2). In a stationary point process, virtually all information about the correlation structure is contained in the Fry plot. For example, the K -function could be calculated from the Fry plot, by counting how many dots in the Fry plot fall in a circle of radius r centred at the origin, and standardising appropriately. Replacing the circle by another “test set”, such as a sector or a ring, gives the other summary functions described in this chapter.

By counting dots in the Fry plot we are effectively treating the Fry plot as a point process in its own right, and finding the *intensity* of this process. The intensity measure of the Fry plot, appropriately standardised, is called the *reduced second moment measure* of the original point process. It generalises Ripley’s K -function from circles to test sets of general shape.

Definition 7.2. For a stationary point process \mathbf{X} with intensity λ , the *reduced second moment measure* is the measure defined for any test set A in two-dimensional space by

$$\mathcal{K}(A) = \frac{1}{\lambda} \mathbb{E} [\text{number of points of } \mathbf{X} (\text{except } u) \text{ falling in } A + u \mid \mathbf{X} \text{ has a point at } u] \quad (7.34)$$

for an arbitrary location u .

For example, if we take the test set $A = b(0, r)$, the disc of radius r centred at the origin, then $A + u$ is the disc of radius r centred at u , and equation (7.34) reduces to (7.4), so that $\mathcal{K}(A) = K(r)$, the value of Ripley’s K -function for distance r . See starred section for details

The measure \mathcal{K} can be regarded as the generalisation of the K -function, and also as a standardised version of the intensity measure of the Fry plot. The connection between (7.34) and the Fry plot is that a point x_j in \mathbf{X} falls in $A + u$ if and only if the vector difference $x_j - u$ falls in A .

Be warned that some writers use the term *reduced second moment measure* when they mean the K -function. This has caused confusion. As originally defined, the reduced second moment measure is a measure, while the K -function is a function obtained by evaluating this measure for discs of increasing radius.

In most applications we can assume that the Fry plot has an intensity function, so that \mathcal{K} has an intensity or density function, the *anisotropic pair correlation function* $g(v)$ defined for vectors v . This can be estimated by kernel smoothing of the Fry plot with edge correction and standardisation. The left panel of Figure 7.37 shows an estimate of the anisotropic pair correlation function for the point pattern data in Figure 7.32.

The anisotropic pair correlation function $g(v)$ is a generalisation of the pair correlation function $g(r)$ to anisotropic, stationary point processes. It allows the correlation between pairs of points to be dependent on their relative position and not only on the distance between them. Again, $g(v)$ is standardised so that a Poisson point process would yield a constant intensity equal to 1.

The anisotropic pair correlation function can be interpreted as a joint probability, as in Figure 7.24 on page 191. Imagine the observation window is divided into a fine grid of pixels, and consider two pixels u and v separated by the vector $v - u$. If $p_2(u, v)$ is the probability that *both* pixels contain random points, then

$$g(v - u) \sim \frac{p_2(u, v)}{p^2} \quad (7.35)$$

where p^2 is the corresponding probability for a Poisson process.

The `spatstat` package provides a function `Kmeasure()` which computes an estimate of the anisotropic pair correlation function (the density of the reduced second moment measure \mathcal{K}). The algorithm approximates the point pattern and its window by pixel images, introduces a Gaussian smoothing kernel, and uses the Fast Fourier Transform to compute a density estimate of \mathcal{K} , effectively using the translation edge correction. The standard deviation σ of the kernel must be chosen by hand; there do not seem to be any established rules for bandwidth selection in this context.

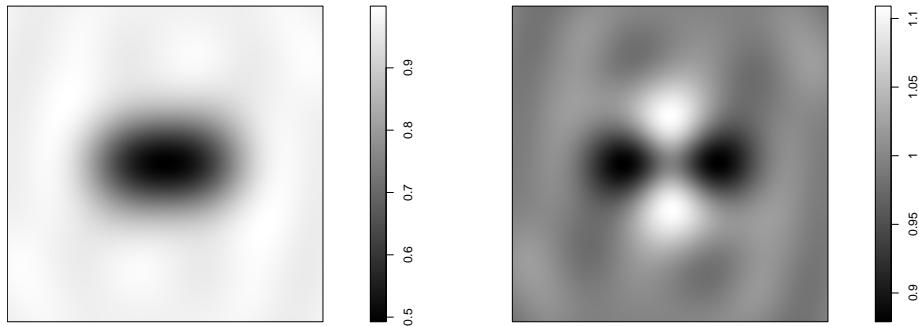


Figure 7.37: Anisotropic pair correlation function. *Left:* image of estimated anisotropic pair correlation $g(u) = g(0,u)$ as a function of vector difference u , with origin at centre of image. Detail of function inside the square of side length 0.2 centred at the origin. *Right:* anisotropy ratio $r(u) = g(u)/\bar{g}(\|u\|)$ where \bar{g} is the rotational mean of g .

The left panel of Figure 7.37 was generated by the commands

```
> ganiso <- Kmeasure(Y, sigma=0.02, eps=0.001)
> detail <- square(c(-0.1,0.1))
> plot(ganiso[detail])
```

Choosing the bandwidth `sigma=0.02` and pixel resolution `eps=0.001` ensures that we do not over-smooth interesting detail, which (according to the Fry plot) is at a scale of about 0.1 units.

Evidence for anisotropy can be judged by comparing the value of $g(v)$ for different vectors v of the same length but different orientations. A useful tool is to compute the *rotational mean*

$$\bar{g}(r) = \frac{1}{2\pi} \int_0^{2\pi} g((r \cos \theta, r \sin \theta)) d\theta \quad (7.36)$$

which gives, for each distance r , the average value of $g(v)$ over all vectors of length r . Then the *anisotropy ratio* $g(v)/\bar{g}(\|v\|)$ is a comparison of the anisotropic pair correlation with its rotational mean.

The `spatstat` function `rotmean` computes the rotational mean of any image. The right panel of Figure 7.37 shows the result of plotting the anisotropy ratio $g(v)/\bar{g}(\|v\|)$, computed by

```
> giso <- rotmean(ganiso, result="im")
> grel <- eval.im(ganiso/giso)
> plot(grel[detail])
```

The right panel of Figure 7.37 gives a clear impression of anisotropy, with anisotropy ratios above 1 in the vertical direction and below 1 in the horizontal direction, at distances of about 0.04 units. The left panel shows that this corresponds to pair correlation values closer to 1 in the vertical direction, and further away from 1 in the horizontal direction, at this distance range. This is consistent with the appearance of the Fry plot.

For a stationary isotropic point process there is a close relationship between the K -function and the isotropic pair correlation function through equations (7.18) and (7.20). Equation (7.20) can be generalised to a stationary point process:

$$\mathcal{K}(A) = \int_A g(v) dv. \quad (7.37)$$

In particular, for $A = b(0, r)$,

$$K(r) = \int_{b(0,r)} g(v) dv \quad (7.38)$$

Equation (7.37) can be used to compute an estimate of the second moment measure $\mathcal{K}(A)$ for any region A . We first use `Kmeasure` to compute an estimate of g as a pixel image, then compute the integral of the pixel image over the domain A using `integral.im`. For example, if A is the square window `detail` created above,

```
> integral.im(ganiso[detail, drop=FALSE])
[1] 0.03746245
```

The pair correlation function also has a direct relationship to the second moment of the number of points falling in a region B . If $N = n(\mathbf{X}_B)$ then $N(N - 1)$ is the number of ordered pairs of distinct points falling in B ; the expected number of such pairs is

$$\mathbb{E}[N(N - 1)] = \lambda^2 \int_B \int_B g(v - u) du dv \quad (7.39)$$

as we can see intuitively by dividing B into a fine grid of pixels, and summing over all pairs of distinct pixels the joint probability that both pixels contain random points.

There is a version of Campbell's formula for second moments. Consider the sum, over all pairs of random points x_i, x_j , of some function $f(x_i, x_j)$. Its expectation is

$$\mathbb{E} \left[\sum_{i=1}^n \sum_{\substack{j=1 \\ j \neq i}}^n f(x_i, x_j) \right] = \lambda^2 \int \int f(u, v) g(v - u) du dv \quad (7.40)$$

provided the expectation is finite.

7.10 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.

7.10.1 The general pair correlation function

First we need to extend the concept of the pair correlation function to point processes which cannot be assumed to be stationary or isotropic.

Suppose the point process \mathbf{X} has intensity function $\lambda(u)$, so that

$$\mathbb{E}n(\mathbf{X}_A) = \int_A \lambda(u) du$$

for any bounded region A . Additionally suppose that \mathbf{X} has **second moment intensity function** $\lambda_2(u, v)$ defined as the function that satisfies

$$\mathbb{E}[n(\mathbf{X}_A) n(\mathbf{X}_B)] = \int_A \int_B \lambda_2(u, v) du dv \quad (7.41)$$

for any disjoint regions A and B . For a Poisson process with intensity function $\lambda(u)$, we have $\lambda_2(u, v) = \lambda(u)\lambda(v)$.

The second moment intensity function has a simple interpretation. Again imagine that the observation window is divided into a fine grid of pixels of area a . Let $p(u)$ be the probability that the pixel with centre u contains a random point. Then $p(u) \sim \lambda(u)a$. Let $p_2(u, v)$ be the joint probability that the two pixels with centres u and v both contain random points. Then $p_2(u, v) \sim \lambda_2(u, v)a^2$.

In this general setting we can define the general *pair correlation function*

$$g_2(u, v) = \frac{\lambda_2(u, v)}{\lambda(u)\lambda(v)} \quad (7.42)$$

for any two different spatial locations u and v . Using the approximations above, $g(u, v) \sim p_2(u, v)/(p(u)p(v))$ is the probability that pixels centred at u and v both contain random points, divided by the corresponding probability for a Poisson process with the same intensity function $\lambda(u)$.

7.10.2 Inhomogeneous K and g functions

Estimation of the function $g(u, v)$ is not practically possible without some further assumption on the form of the function. A strategy proposed by Baddeley *et al.* [29] effectively assumes

$$g_2(u, v) = g(v - u), \quad (7.43)$$

that is, the pair correlation between u and v depends only on their relative position. This would be true if the process is stationary, but is also true for an inhomogeneous Poisson process, and for many other processes.

Assuming (7.43), it is possible to define a counterpart of $K(r)$ called the *inhomogeneous K -function*. The idea is that each point x_i will be weighted by $w_i = 1/\lambda(x_i)$, the reciprocal of the intensity at x_i , and each pair of points x_i, x_j will be weighted by $w_{ij} = 1/(\lambda(x_i)\lambda(x_j))$.

The *inhomogeneous K -function* is defined as

$$K_{inhom}(r) = \mathbb{E} \left[\sum_{x_j \in \mathbf{X}} \frac{1}{\lambda(x_j)} \mathbf{1} \{ 0 < \|u - x_j\| \leq r \} \mid u \in \mathbf{X} \right] \quad (7.44)$$

assuming this does not depend on location u . If (7.43) holds, then (7.44) does not depend on u .

Thus, $K_{inhom}(r)$ 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 stationary, then $\lambda(u)$ is constant and $K_{inhom}(r)$ reduces to the usual K function (7.5).

For an inhomogeneous Poisson process with intensity function $\lambda(u)$, the inhomogeneous K function is

$$K_{inhom, pois}(r) = \pi r^2 \quad (7.45)$$

exactly as for the homogeneous case.

The standard estimators of K can be extended to the inhomogeneous K function:

$$\widehat{K}_{inhom}(r) = \frac{1}{D^p \text{area}(W)} \sum_i \sum_{j \neq i} \frac{\mathbf{1} \{ \|x_i - x_j\| \leq r \}}{\widehat{\lambda}(x_i) \widehat{\lambda}(x_j)} e(x_i, x_j; r) \quad (7.46)$$

where $e(u, v, r)$ is an edge correction weight as before, and $\widehat{\lambda}(u)$ is an estimate of the intensity function $\lambda(u)$. The constant D^p in (7.46) is the p th power of

$$D = \frac{1}{\text{area}(W)} \sum_i \frac{1}{\widehat{\lambda}(x_i)}$$

which has expected value 1 if the intensity is estimated without error. Choosing the power p equal to

1 or 2 improves statistical performance. Theoretical results in [29] show that, if the intensity function is *known*, or can be estimated with very high accuracy, then $\widehat{K}_{inhom}(r)$ is an unbiased estimator of $K_{inhom}(r)$ when $p = 0$, and is approximately unbiased when $p = 1$ or 2.

In practice, the intensity function will need to be estimated from data, so the estimator (7.46) could be biased. The intensity function is usually estimated from the *same* point pattern data, which can cause difficult biases. The estimator (7.46) requires the estimated intensities $\widehat{\lambda}(x_i)$ at the data points x_i ; many techniques for estimating a point process intensity function tend to overestimate the intensity at the data points themselves.

Suppose we use a kernel smoothing estimate of the point process intensity,

$$\widehat{\lambda}(u) = \sum_i \kappa(u - x_i),$$

where κ is the smoothing kernel. Substituting $u = x_i$ we find that the estimated intensity $\widehat{\lambda}(x_i)$ at a data point x_i is always *at least* $\kappa(x_i - x_i) = \kappa(0)$, the value of the kernel at zero distance. The estimate $\widehat{\lambda}(x_i)$ has a strong positive bias. A common remedy is to omit the contribution from x_i when estimating the intensity at $u = x_i$, giving the *leave-one-out* intensity estimator

$$\widehat{\lambda}_{-i}(x_i) = \sum_{j \neq i} \kappa(x_i - x_j). \quad (7.47)$$

This tends to be only slightly negatively biased.

The inhomogeneous K function is computed in `spatstat` by the command `Kinhom(X, lambda)` where X is the point pattern and `lambda` is the estimated intensity function. Here `lambda` may be a pixel image, a `function(x, y)` in the R language, a fitted point process model, a numeric vector giving the values $\widehat{\lambda}(x_i)$ at the data points x_i only, or it may be omitted (and will then be estimated from X). By default, the data-dependent denominator D^1 is used: the power p is controlled by the argument `normpower`.

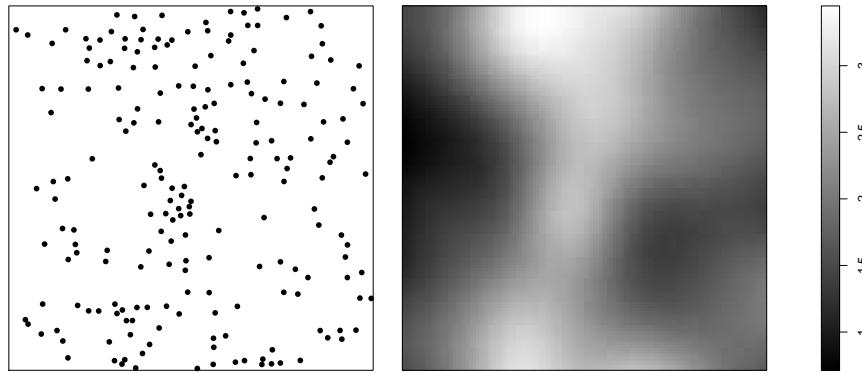


Figure 7.38: Full dataset of Japanese Black Pines (*Left*) and kernel-smoothed intensity estimate (*Right*). Smoothing bandwidth selected by likelihood cross-validation.

Figure 7.38 shows the full dataset of Japanese Black Pine seedlings and saplings in a 10 metre square study region recorded by Numata [256]. A kernel-smoothed intensity estimate is shown, with smoothing bandwidth selected by likelihood cross-validation using `bw.ppl()` (see Section 6.5.1.2 on page 133):

```
> numata <- residualspaper$Fig1
> lambda <- density(numata, bw.ppl)
```

The chosen bandwidth seems appropriate, and the intensity estimate gives a strong impression of inhomogeneity. We now compute the inhomogeneous K -function:

```
> numataK <- Kinhom(numata, lambda)
```

The result is shown in Figure 7.39. The edge-corrected estimates are close to the Poisson theoretical value, suggesting that the data are consistent with an inhomogeneous Poisson process with the intensity shown in Figure 7.38.

If the intensity function is not given in the call to `Kinhom` then it will be estimated by kernel smoothing using the leave-one-out estimator. We could have obtained the same result by

```
> numataK <- Kinhom(numata, sigma=bw.ppl)
```

The argument `sigma=bw.ppl` is passed to `density.ppp`.

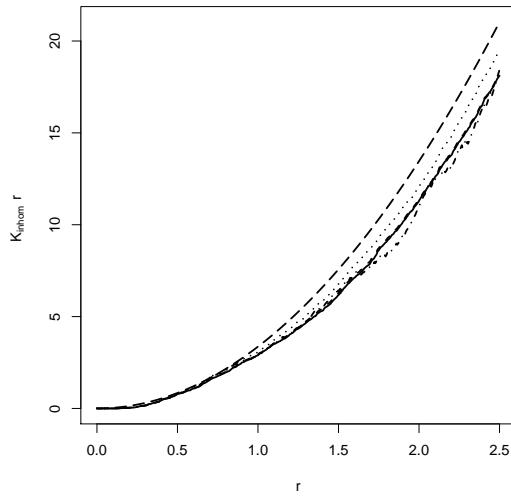


Figure 7.39: Inhomogeneous K function for Numata data using estimate of intensity function from Figure 7.38.

The inhomogeneous analogue of the L -function is

$$L_{inhom}(r) = \sqrt{K_{inhom}(r)/\pi}. \quad (7.48)$$

For an inhomogeneous Poisson process, $L_{inhom, pois}(r) \equiv r$. The corresponding empirical L -function is again justified by the fact that it approximately stabilises the variance. The inhomogeneous L -function can be computed in `spatstat` using `Linhom()`, which has the same arguments as `Kinhom()`.

The relationship (7.38) between the K -function and pair correlation function of a stationary point process, extends to the inhomogeneous K -function:

$$K_{inhom}(r) = \int_{b(0,r)} g_2(0,v) dv. \quad (7.49)$$

In general $g_2(0,v)$ depends on the orientation of v as well as the distance $\|v\|$. The rotational mean of the pair correlation

$$\bar{g}(r) = \frac{1}{2\pi} \int_0^{2\pi} g_2(0, (r \cos \theta, r \sin \theta)) d\theta$$

is sometimes called the “inhomogeneous pair correlation function” $g_{inhom}(r)$, and satisfies

$$g_{inhom}(r) = \frac{K'_{inhom}(r)}{2\pi r}.$$

The inhomogeneous pair correlation function is computed by `pcf.inhom()`:

```
> g <- pcf.inhom(bei)
```

The previously mentioned method `pcf.fv()`, which converts a K -function estimate into a pair correlation function estimate by numerical differentiation, also works for estimates of the inhomogeneous K function. Thus the following is an alternative to the foregoing estimation procedure:

```
> g <- pcf(Kinhom(bei))
```

To construct confidence intervals for the true value of $K_{inhom}(r)$ or $g_{inhom}(r)$ one can use `varblock()` or `looboot()` as described above.

To test the hypothesis that the point process is an inhomogeneous Poisson process, use `envelope()`, `dclf.test()` or `mad.test()`. Some care is required: as explained in Chapter 10, it is important to ensure that the simulated patterns are treated in exactly the same way as the original data was treated. Figure 7.40 shows the global envelopes of the centred inhomogeneous L function for the full Japanese Pines data of Figure 7.38, generated by

```
> lam <- density(numata, bw.ppl)
> E <- envelope(numata, Linhom, sigma=bw.ppl,
+                 simulate=expression(rpoispp(lam)),
+                 nsim=19, global=TRUE)
```

The plot indicates significant evidence against the inhomogeneous Poisson model, with a suggestion that there is inhibition at small distances.

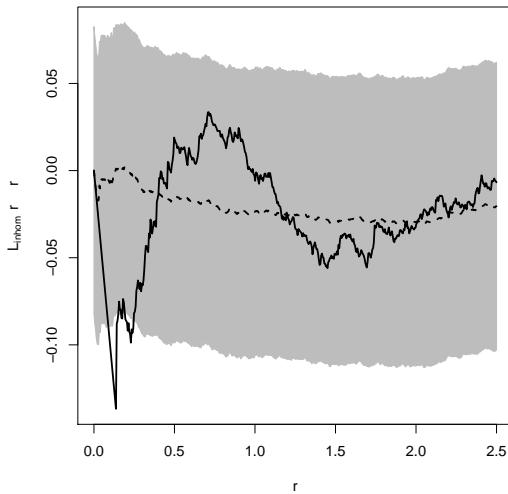


Figure 7.40: Global envelopes of $L_{inhom}(r) - r$ for 19 simulated realisations of an inhomogeneous Poisson process with intensity estimated by kernel smoothing.

7.10.3 Local scaling

The inhomogeneous K -function effectively assumes that the *spatial scale* of interaction remains constant, while the intensity is spatially-varying. This is not an appropriate assumption for the Bronze Filter data in Figure 1.9 on page 8, an inhomogeneous point pattern in which the spacing between points increases gradually from left to right.

An alternative approach to inhomogeneity [168, 273, 167] is to assume that the point process is equivalent, in small regions, to a rescaled version of a ‘template’ process, where the template process is stationary and isotropic, and the rescaling factor can vary from place to place. This could be an appropriate model for Figure 1.9.

We would then be assuming that, for two locations u and v sufficiently close together,

$$g(u, v) = g_1 \left(\frac{\|u - v\|}{s} \right) \quad (7.50)$$

where g_1 is the pair correlation function of the ‘template’ process, and s is the local scaling factor applicable to both locations u and v . Rescaling the spatial coordinates by a factor $1/s$ rescales the intensity by s^2 , so the appropriate value is $s = 1/\sqrt{\lambda}$ where λ is the local intensity in the neighbourhood of u and v .

In practice, we would first estimate the intensity function of the original data by $\widehat{\lambda}(u)$, then for each pair of data points x_i, x_j define the rescaled distance

$$d_{ij}^* = \frac{\|x_i - x_j\|}{s(x_i, x_j)}$$

where the rescaling factor is

$$s(x_i, x_j) = \frac{1}{2} \left(\frac{1}{\sqrt{\widehat{\lambda}(x_i)}} + \frac{1}{\sqrt{\widehat{\lambda}(x_j)}} \right).$$

An edge-corrected estimator of Ripley’s original K -function is then applied to the distances d_{ij}^* to give the *locally-scaled K-function* [168, 273].

The `spatstat` package provides the commands `Kscaled()` and `Lscaled()` which compute the locally-scaled K and L functions. Their syntax is similar to `Kinhom()`.

To estimate a locally-scaled K -function for the bronze filter data, we first estimated the intensity, assuming it is an exponential function of the x coordinate, by fitting a point process model (see Chapter 9):

```
> X <- unmark(bronzefilter)
> fit <- ppm(X ~ x)
> lam <- predict(fit)
```

The locally-scaled K -function was then estimated by

```
> Kbro <- Kscaled(X, lam)
```

The result is plotted in Figure 7.41.

7.11 Theory*

* Starred sections contain advanced material, and can be skipped by most readers.

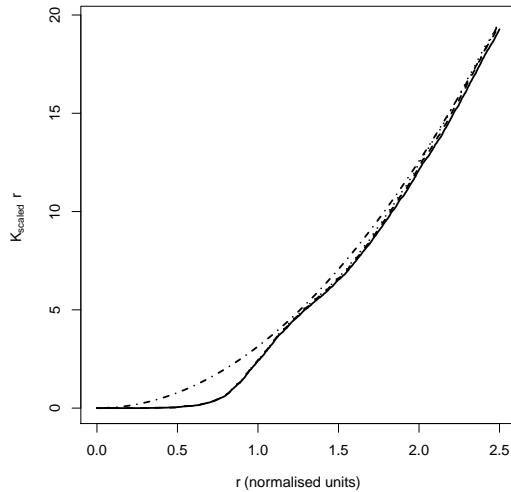


Figure 7.41: Locally-scaled K -function of the bronze filter data.

7.11.1 Second moment measures

Let \mathbf{X} be a point process. We are interested in the variance of the count $n(\mathbf{X}_B)$,

$$\text{var} n(\mathbf{X}_B) = \mathbb{E}[n(\mathbf{X}_B)^2] - [\mathbb{E}n(\mathbf{X}_B)]^2$$

and the covariance of two such counts,

$$\text{cov}[n(\mathbf{X}_{B_1}), n(\mathbf{X}_{B_2})] = \mathbb{E}[n(\mathbf{X}_{B_1})n(\mathbf{X}_{B_2})] - [\mathbb{E}n(\mathbf{X}_{B_1})][\mathbb{E}n(\mathbf{X}_{B_2})].$$

A key observation is that $n(\mathbf{X}_{B_1})n(\mathbf{X}_{B_2})$ is equal to the number of ordered pairs (x, x') of points in the process \mathbf{X} such that $x \in B_1$ and $x' \in B_2$.

Definition 7.3. Let \mathbf{X} be a point process on a space S . Then $\mathbf{X} \times \mathbf{X}$ is a point process on $S \times S$ consisting of all ordered pairs (x, x') of points $x, x' \in \mathbf{X}$. The intensity measure v_2 of $\mathbf{X} \times \mathbf{X}$ is a measure on $S \times S$ satisfying

$$v_2(A \times B) = \mathbb{E}[N_{\mathbf{X}}(A)N_{\mathbf{X}}(B)].$$

This measure v_2 is called the second moment measure of \mathbf{X} .

Clearly, the second moment measure contains all information about the variances and covariances of the variables $N_{\mathbf{X}}(A)$. Campbell's formula applied to $\mathbf{X} \times \mathbf{X}$ becomes

$$\mathbb{E}\left[\sum_{x \in \mathbf{X}} \sum_{y \in \mathbf{X}} f(x, y)\right] = \int_S \int_S f(x, y) d v_2(x, y)$$

for a measurable function $f : S \times S \rightarrow \mathbb{R}$.

Example 7.1. For the uniform Poisson point process of intensity $\lambda > 0$ in \mathbb{R}^d , the second moment measure satisfies

$$v_2(A \times B) = \lambda^2 |A| |B| + \lambda |A \cap B|.$$

To simplify the calculation of certain moments, we introduce the *second factorial moment measure*

$$v_{[2]}(A \times B) = \mathbb{E}[n(\mathbf{X}_A)n(\mathbf{X}_B)] - \mathbb{E}[n(\mathbf{X}_{A \cap B})].$$

This is the intensity measure of the process $\mathbf{X} * \mathbf{X}$ of all ordered pairs of *distinct* points of \mathbf{X} . It satisfies

$$\mathbb{E}\left[\sum_{x \in \mathbf{X}} \sum_{y \in \mathbf{X}, y \neq x} f(x, y)\right] = \int_S \int_S f(x, y) d\nu_{[2]}(x, y).$$

The name ‘factorial’ is derived from

$$\begin{aligned} \nu_{[2]}(A \times A) &= \mathbb{E}[n(\mathbf{X}_A)^2] - \mathbb{E}[n(\mathbf{X}_A)] \\ &= \mathbb{E}[n(\mathbf{X}_A)[n(\mathbf{X}_A) - 1]]. \end{aligned}$$

For example, for the uniform Poisson process of intensity λ , the second factorial moment measure is $\nu_{[2]} = \lambda^2 \lambda_d \otimes \lambda_d$.

Definition 7.4. A point process \mathbf{X} on \mathbb{R}^d is said to have second moment density λ_2 if

$$\nu_{[2]}(C) = \int_C \lambda_2(x, y) dx dy \quad (7.51)$$

for any compact $C \subset \mathbb{R}^d \times \mathbb{R}^d$.

Informally, $\lambda_2(x, y)$ gives the joint probability that there will be points of \mathbf{X} at two specified locations x and y :

$$\mathbb{P}\{N(dx) > 0, N(dy) > 0\} \sim \lambda_2(x, y) dx dy$$

For example, the uniform Poisson process has second moment density $\lambda_2(x, y) = \lambda^2$. The binomial process of n points in W has

$$\lambda_2(x, y) = \frac{n(n-1)}{|W|^2}$$

if $x, y \in W$, and zero otherwise.

Definition 7.5. Suppose \mathbf{X} is a point process on \mathbb{R}^d which has an intensity function $\lambda(x)$ and a second moment density $\lambda_2(x, y)$. Then we define the pair correlation function of \mathbf{X} by

$$g_2(x, y) = \frac{\lambda_2(x, y)}{\lambda(x)\lambda(y)}.$$

Example 7.2. For a uniform Poisson process of intensity λ , we have $\lambda(x) \equiv \lambda$ and $\lambda_2 \equiv \lambda^2$, so that

$$g_2(x, y) \equiv 1.$$

Example 7.3. For a binomial process of n points in a region W , we have

$$g_2(x, y) \equiv 1 - \frac{1}{n}.$$

Note that the pair correlation function always satisfies $g_2(x, y) \geq 0$. It should be regarded as a “non-centred” analogue of the usual correlation of random variables. The value $g = 1$ corresponds to a lack of correlation in the usual statistical sense: if $g_2 \equiv 1$ then $\text{cov}[n(\mathbf{X}_B), n(\mathbf{X}_{B'})] = 0$ for disjoint sets B, B' .

7.11.2 Second moments for stationary processes

For a stationary point process in \mathbb{R}^d , there is a “disintegration” of the second moment measure.

Theorem 7.1. *Let \mathbf{X} be a stationary point process on \mathbb{R}^d with intensity λ . Then there is a measure \mathcal{K} on \mathbb{R}^d such that, for a general integrand f ,*

$$\mathbb{E} \left[\sum_{x \in \mathbf{X}} \sum_{y \in \mathbf{X}, y \neq x} f(x, y) \right] = \lambda \int \int f(x, x+u) d\mathcal{K}(u) dx. \quad (7.52)$$

\mathcal{K} is called the reduced second moment measure of \mathbf{X} .

To understand the measure \mathcal{K} , we notice that for $A, B \subset \mathbb{R}^d$

$$\begin{aligned} \lambda |A| \mathcal{K}(B) &= \mu(A \times B) \\ &= \int \int 1_A(s) 1_B(t) d\mu(s, t) \\ &= \int \int 1_A(x) 1_B(y-x) d\nu_{[2]}(x, y) \\ &= \mathbb{E} \left[\sum_{x \in \mathbf{X}} \sum_{y \in \mathbf{X}, y \neq x} 1_A(x) 1_B(y-x) \right] \end{aligned}$$

This may also be obtained directly from (7.52) by taking $f(x, y) = 1_A(x) 1_B(y-x)$. Since $\lambda |A| = \mathbb{E}n(\mathbf{X}_A)$, we have

$$\mathcal{K}(B) = \frac{\mathbb{E} \sum_{x \in \mathbf{X} \cap A} n(\mathbf{X}_{(B+x) \setminus x})}{\mathbb{E}n(\mathbf{X}_A)} \quad (7.53)$$

The right hand side of (7.53) may be interpreted as the average, over all points x of the process, of the number of other points y of the process such that $y - x \in B$.

Example 7.4. *For the uniform Poisson process,*

$$\begin{aligned} \nu_{[2]} &= \lambda^2 L_d \otimes L_d \\ \mu &= \lambda^2 L_d \otimes L_d \\ \mathcal{K} &= \lambda L_d \end{aligned}$$

Example 7.5. *Suppose \mathbf{X} is a stationary process on \mathbb{R}^d which has a second moment density function λ_2 . Then by comparing (7.51) with (7.52) we can see that $\lambda_2(x, y)$ depends only on $y - x$, say*

$$\lambda_2(x, y) = h(y - x),$$

for some function h , and we can write

$$\mathcal{K}(B) = \frac{1}{\lambda} \int_B h(u) du.$$

Example 7.6. *The randomly translated square grid was introduced in Example ???. This is a stationary process. Following through the derivation above, we find that the reduced second moment measure \mathcal{K} puts mass 1 at each integer point (ns, ms) for all integers n, m , except that there is no atom at $(0, 0)$.*

Intuitively this reflects the fact that, if we know there is a point of \mathbf{X} at the origin, then this determines the position of the entire grid of points, and we know there will be a point of \mathbf{X} at each location (ns, ms) .

This point process does not have a second moment density λ_2 .

Lemma 7.1 (Invariance of K under thinning). *Suppose \mathbf{X} is a stationary point process, and \mathbf{Y} is obtained from \mathbf{X} by random thinning (each point of \mathbf{X} is deleted or retained, independently of other points, with retention probability p). Then the K -functions of \mathbf{X} and \mathbf{Y} are identical.*

7.12 FAQ's

- When I plot the estimated K -function of my data using the command `plot(Kest(X))`, the horizontal axis is the distance in metres, but what is the quantity on the vertical axis, and what units is it expressed in?

The quantity on the vertical axis is the average number of neighbours of a typical point, divided by the intensity (average number of points per unit area) so that different patterns can be compared. It is measured in units of area (number of points divided by points-per-unit-area). The reference benchmark is that for a completely random process the value of $K(r)$ is the area of the disc of radius r .

- When I plot the estimated K -function of my data using the command `plot(Kest(X))`, the scale marks on the y-axis are huge numbers like $1e9$. Is this wrong? I don't see anything like this in your book.

The values of $K(r)$ are areas, and should be of the same order as the area of the observation window, expressed in the units you are using for the spatial coordinates. If your window is 30 kilometres across and the coordinates are recorded in metres, the window area is about $30000^2 \approx 10^9$ square metres. To avoid numerical overflow, it would be wise to rescale the spatial coordinates, for example converting metres to kilometres.

- When I plot the estimated K -function of my data using the command `plot(Kest(X))`, I don't understand the meaning of the different coloured curves.

These curves are different estimates of the K function, using different 'edge correction' techniques. Usually one of the curves is the theoretical value of the K function, $K(t) = \pi t^2$, corresponding to a completely random pattern.

The accompanying legend (plotted unless `legend=FALSE`) gives a mathematical expression for each of the edge corrections, and shows the corresponding line colour and line type used to plot the estimate.

The plot also generates printed output. The return value of `plot.fv` is a table giving the line colour, line type, keyword, mathematical expression, and long text description of each curve that was plotted. For further information about the different edge corrections, see `help(Kest)`.

The estimates of $K(r)$ by different edge correction techniques should be roughly equal. If the curves for the isotropic correction (`iso`), translation correction (`trans`) and border correction (`border`) estimates are wildly different, this suggests that estimation of K is difficult for these data (e.g. because there are too few data points, or the window is very narrow, or there are data points very close to the edge of the window.)

For information on how to modify the plot of the K functions, see `help(plot.fv)` or the examples in `help(Kest)`.

- I can't seem to control the range of r values in `plot(Kest(X))`. How can I control it? How is the default plotting range determined?

Use the argument `xlim` to control the range of the x axis. For example, `plot(Kest(X), xlim=c(0, 7))`. See `help(plot.fv)`.

An object of class "fv" contains function values for a certain range of r values (the "available range", which is usually the maximum possible range). However the default behaviour of `plot.fv()` is to plot the function values for a narrower range of r values (the "recommended range") which usually contains the important detail in the function. Both the available range and recommended range are printed when the "fv" object is printed.

Using the argument `xlim` when plotting the K -function will override the recommended range. However, obviously we cannot choose `xlim` to be wider than the *available* range of r values. To extend the available range, you would need to re-compute the K function, specifying the argument `r`. This should be a vector of closely-spaced values of r ranging from 0 to the desired maximum.

— SUBSEQUENT SOFTWARE UPDATES —

This is a list of relevant changes to `spatstat` since the last version of the workshop notes in 2010. These changes may not yet have been covered in the text of this chapter, if the text was cut-and-pasted from existing sources. Chapter authors should try to merge this information into the chapter text. **DELETE ITEMS FROM THIS LIST IF THEY ARE COVERED IN THE CHAPTER TEXT.**

- `Kcross`, `Kdot`, `Kmulti`: New argument `ratio` determines whether the numerator and denominator of the estimate of the multitype K-function will be stored. This enables analysis of replicated point patterns, using `pool.rat` to pool the K function estimates.
- `Kmulti`: The arguments `I`, `J` can now be any kind of subset index or can be functions that yield a subset index.
- `localKinhom`, `localLinhom`, `localpcfinhom`: Inhomogeneous versions of `localK`, `localL`, `localpcf`
- `Kmulti.inhom`: Counterpart of `Kmulti` for spatially-varying intensity.
- `rat`: New class of 'ratio objects'
- `pool.rat`: New method for `pool`. Combines K function estimates for replicated point patterns (etc) by computing ratio-of-sums
- `crossdist.ppp`, `crossdist.pp3`, `crossdist.default`: New argument `squared` allows the squared distances to be computed (saving computation time in some applications)
- `fryplot`: Now has arguments `to` and `from`, allowing selection of a subset of points.
- `connected.ppp`: Find clumps in a point pattern.
- `Kinhom`: New argument `update`. If `lambda` is a fitted model (class `ppm` or `kppm`) and `update=TRUE`, the model is re-fitted to the data pattern, before the intensities are computed.
- `envelope`: Now issues a warning if the usage of `envelope()` appears to be 'invalid' in the sense that the simulated patterns and the data pattern have not been treated equally.



8

Shortest distances and empty spaces

Statistics such as Ripley's K -function, which measure the spatial correlation in a point process (Chapter 7), are the most direct and most popular tools for assessing dependence. However, correlation is only a simple summary of dependence, and is blind to some aspects of dependence, as we saw in Chapter 7.

Additional information about a point pattern can often be revealed by measuring *shortest distances* in the pattern, such as the “nearest-neighbour” distance from each data point to the nearest other data point. In a forestry survey, it would make good practical sense to measure the distance from each sampled tree to the nearest other tree. Obviously this carries important information about the spatial arrangement of trees. This Chapter introduces analytical tools based on shortest distances.

There is a kind of duality between measuring shortest distances between points and counting points in a given area. At a bus stop we can either say that (on average) there are 4 buses per hour, or that there is 1/4 hour between buses. This duality runs deep in the mathematical theory, telling us that the shortest distances in a point pattern provide information which is complementary to the correlation structure. A good statistical analysis of spatial point patterns should therefore study both the correlation and the shortest distances.

Section 8.1 describes simple techniques based on shortest distances. Section 8.2 introduces the most important summary functions, the empty space function $F(r)$ and the nearest-neighbour function $G(r)$. Plotting and interpretation of these functions is covered in Section 8.3. Section 8.4 discusses the hazard rate of the empty space function, and Section 8.5 introduces the J -function, a combination of F and G . Inhomogeneous versions of F , G and J are described in Section 8.6. Distances from a point process to another spatial pattern are considered in Section 8.7. Technical details of edge correction for estimating F and G are presented in the starred Section 8.8.

8.1 Manual methods

8.1.1 Measuring distances in point patterns

There are several different kinds of distances that could be measured in a point pattern:

- the **pairwise distances** $d_{ij} = ||x_i - x_j||$ between all distinct pairs of points x_i and x_j ($i \neq j$) in the pattern;
- the **nearest neighbour distances** $c_i = \min_{j \neq i} d_{ij}$, the distance from each point x_i to its nearest neighbour (left panel of Figure 8.1);
- the **empty space distance** $d(u) = \min_j \|u - x_j\|$, the distance from a fixed reference location u in the window to the nearest data point (right panel of Figure 8.1).

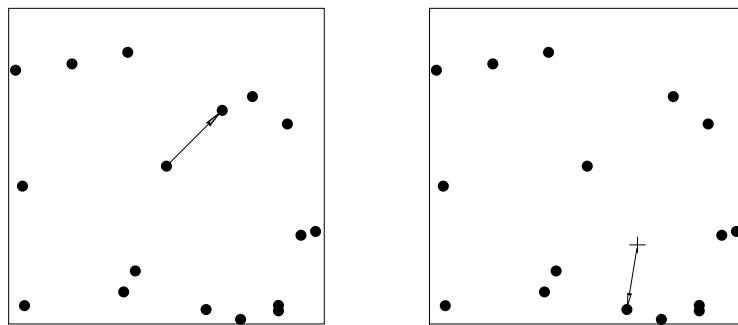


Figure 8.1: *Left*: the nearest neighbour distance is the distance from a data point (\bullet) to the nearest other data point. *Right*: the empty space distance is the distance from a fixed location (+) to the nearest data point.

Table 8.1 shows the `spatstat` functions for computing these distances in a point pattern dataset. If X is a point pattern object, `pairdist(X)` returns the matrix of pairwise distances:

```
> M <- pairdist(redwood)
> M[1:3, 1:5]
     [,1]   [,2]   [,3]   [,4]   [,5]
[1,] 0.000 0.082 0.120 0.134 0.141
[2,] 0.082 0.000 0.045 0.057 0.060
[3,] 0.120 0.045 0.000 0.060 0.028
```

while `nndist(X)` returns the vector of nearest neighbour distances:

```
> v <- nndist(redwood)
> v[1:3]
[1] 0.082 0.045 0.028
```

The command `distmap(X)` returns a pixel image whose pixel values are the empty space distances to the pattern X measured from every pixel:

```
> Z <- distmap(redwood)
> plot(Z)
```

The result is shown in the left panel of Figure 8.2.

The function `distmap` uses a very fast image processing algorithm for calculating approximate distances [289, 290, 66, 67], which we have modified¹ to compute “almost exact” distances $d(u)$ from the centre of each pixel. This will be sufficient for most purposes.

However, if exact values of empty space distance $d(u)$ are required, or if only a few values of $d(u)$ need to be computed, use `nncross()` or `distfun()`. If U and X are point patterns, the command `nncross(U, X, what="dist")` computes the exact distance from each point in U to the nearest point in X , and returns a numeric vector with one entry for each point of U . The command `f <- distfun(X)` returns a function f in the R language that can be evaluated at any spatial location to give the exact value of empty space distance.

¹The modified algorithm runs the classical distance transform algorithm to determine, for each pixel centre u , which data point x_i is the closest. It then computes the exact distance from u to x_i . In marginal cases, the wrong index i may be selected, and the result will be slightly larger than it should be.

DISTANCES	
<code>pairdist(X)</code>	matrix of pairwise distances in X
<code>crossdist(X, Y)</code>	matrix of pairwise distances from X to Y
<code>nndist(X)</code>	vector of nearest-neighbour distances in X
<code>distmap(X)</code>	pixel image of empty space distances to X
<code>distfun(X)</code>	function that computes empty space distance to X
<code>nncross(X, Y, what="dist")</code>	vector of distances from X to nearest point in Y
INDICES	
<code>nnwhich(X)</code>	vector of indices of nearest neighbours in X
<code>nncross(X, Y, what="which")</code>	vector of indices from X to nearest point in Y
<code>nnmap(X)</code>	pixel image of index of nearest point of X
<code>nncfun(X)</code>	function that computes index of nearest point of X

Table 8.1: Functions for measuring distances in a point pattern X and for identifying the nearest point.

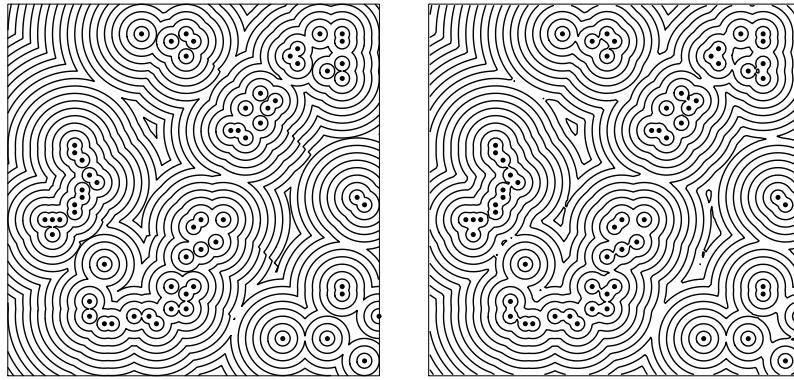


Figure 8.2: Contours of the distance function $d(u)$ for the `redwood` dataset. *Left:* computed approximately by `distmap`. *Right:* computed exactly by `distfun`. Note slight artefacts in the left panel. You are feeling sleepy.

```
> U <- runifpoint(3, Window(redwood))
> Z <- distmap(redwood)
> Z[U]
[1] 0.052 0.073 0.043
> nncross(U, redwood, what="dist")
[1] 0.052 0.073 0.040
> f <- distfun(redwood)
> f(U)
[1] 0.052 0.073 0.040
```

When constructing a spatial covariate for use in fitting a point process model to data (see Part III of the book), it is best to use `distfun()`. If it is required to compute a pixel image containing the *exact* distance values at the pixel centres, use `as.im(distfun(X), ...)` where the additional arguments control the pixel resolution. This idiom was used to make the right panel of Figure 8.2.

Instead of the nearest neighbour we may also consider the second-nearest, third-nearest and so on. The functions `nndist()`, `nncross()` and `distfun()` have an argument `k` specifying the order of neighbour: `k=2` would specify the second-nearest neighbour.

```
> nncross(U, redwood, k=2, what="dist")
[1] 0.084 0.105 0.053
> f2 <- distfun(redwood, k=2)
> f2(U)
[1] 0.084 0.105 0.053
```

To determine *which* point is the nearest neighbour of a given point, use one of the functions `nnwhich()`, `nncross()`, `nnmap()` or `nnfun()`. For a point pattern X , the result of `nnwhich(X)` is a vector v of integers such that $v[i]=j$ if the nearest neighbour of $X[i]$ is $X[j]$. For two point patterns U and X , the result of `nncross(U, X, what="which")` is a vector v such that $v[i]=j$ if, starting from the point $U[i]$, the nearest point in X is $X[j]$. The result of `nnmap(X)` is a pixel image with integer values giving for each pixel the index of the nearest point of X . The result of `nnfun(X)` is a function in the R language which can be evaluated at any spatial location u to give the index of the nearest point of X . These functions all accept the argument k specifying the order of neighbour.

8.1.2 Tests of CSR based on shortest distances

Early literature in statistical ecology includes several methods for deciding whether a point pattern is completely random, using nearest-neighbour and empty-space distances that could be measured in a field study. We present two of them here.

The nearest-neighbour distances c_i, c_j of two nearby plants x_i, x_j are statistically dependent. To avoid problems arising from this dependence, the classical techniques measure the nearest-neighbour distances only from a *random sample* of data points. Similarly the empty-space distances for two nearby locations are statistically dependent, so a *random sample* of spatial locations will be used to measure empty space distances.

Clark and Evans [86] proposed taking the average of the nearest neighbour distances c_i for m randomly-sampled points in a point pattern (or for all data points), and dividing this by the expected value $\mathbb{E}[C]$ for a completely random process with the same intensity, to obtain an index of spatial regularity. In a Poisson process of intensity λ the expected distance to the nearest neighbour is $\mathbb{E}[C] = 1/(2\sqrt{\lambda})$. The *Clark-Evans index* is

$$R = \frac{\bar{c}}{\mathbb{E}[C]} = \frac{2\sqrt{\bar{\lambda}}}{m} \sum_{i=1}^m c_i. \quad (8.1)$$

where m is the number of sampled points, and $\bar{\lambda} = n/|W|$ is the estimated intensity for the entire point pattern consisting of n data points in a window W . The index R is dimensionless; the value $R = 1$ is consistent with a completely random pattern; $R > 1$ suggests ordering, while $R < 1$ suggests clustering.

The *Clark-Evans test* of CSR is performed by approximating the distribution of R under CSR by a normal distribution with mean 1 and variance $s^2 = (1/\pi - 1/4)/(m\lambda)$. For a two-sided test, we reject the null hypothesis of complete spatial randomness at the 5% significance level if $|(R - 1)/s| > 1.96$.

Without correction for edge effects, the value of R will be positively biased [134]. Edge effects arise because, for a data point close to the edge of the window, the true nearest neighbour may actually lie outside the window. Hence observed nearest neighbour distances tend to be larger than the true nearest neighbour distances.

The `spatstat` functions `clarkevans()` and `clarkevans.test()` perform these calculations. The argument `correction` specifies an edge correction to be applied. In some cases the test will be performed using Monte Carlo simulation.

```
> clarkevans(redwood)
```

```

naive Donnelly      cdf
0.6187   0.5850   0.5836
> clarkevans.test(redwood, correction="donnelly",
                    alternative="clustered")
Clark-Evans test
Donnelly correction
Monte Carlo test based on 999 simulations of CSR with fixed n

data: redwood
R = 0.585, p-value = 0.001
alternative hypothesis: clustered (R < 1)

```

An important weakness of the Clark-Evans test is that it *assumes that the point process is stationary*. An inhomogeneous point pattern will typically give $R < 1$, and can produce spurious significance.

Hopkins and Skellam [184, 301] proposed taking the nearest-neighbour distances c_i for m randomly-sampled data points, and the empty-space distances $e_j = d(u_j)$ for an equal number m of randomly-sampled spatial locations. If the point pattern is completely random, points are independent of each other, so the distance from a data point to the nearest other point should have the same probability distribution as the distance from a fixed spatial location to the nearest data point. That is, the values c_i and e_j should have the same distribution. The *Hopkins-Skellam index* is

$$A = \frac{\sum_{i=1}^m c_i^2}{\sum_{j=1}^m e_j^2}. \quad (8.2)$$

Again this is a dimensionless index; $A = 1$ is consistent with a completely random pattern; $A < 1$ is consistent with clustering, and $A > 1$ with regularity. The *Hopkins-Skellam test* compares the value of A to the F distribution with parameters $(2m, 2m)$. In `spatstat` the functions `hopskel()` and `hopskel.test()` perform these calculations.

```

> hopskel(redwood)
[1] 0.142
> hopskel.test(redwood, alternative="clustered")
      Hopkins-Skellam test of CSR
      using F distribution

data: redwood
A = 0.1681, p-value < 2.2e-16
alternative hypothesis: clustered (A < 1)

```

Interestingly the Hopkins-Skellam index is much less sensitive than the Clark-Evans index, to problems such as edge effect bias and spatial inhomogeneity. These phenomena have roughly equal effect on the nearest-neighbour distances and the empty-space distances, so the ratio (8.2) is insensitive to them.

Summary statistics like the average nearest-neighbour distance, Clark-Evans index and Hopkins-Skellam index are simple and practical to apply in the field. They may be useful when a simple numerical index of spatial pattern is needed, for example, when we need to compare a large number of different point patterns, or to monitor changes in spatial clustering over time. Their main weakness is that they compress all the spatial information into a single number, conflating information from different spatial scales and different spatial locations.

8.1.3 Exploratory graphics

Nearest-neighbour and empty-space distances can also be represented spatially and graphically. This can be useful for exploratory investigation of spatial point pattern data.

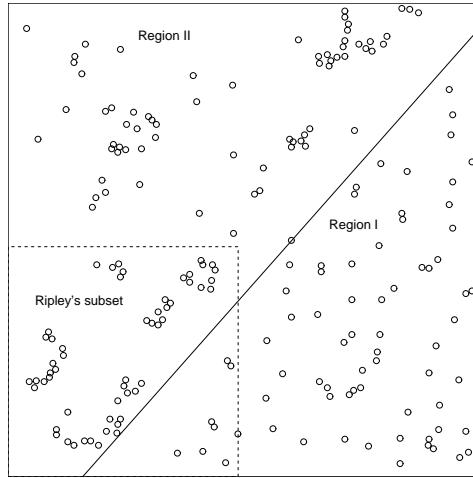


Figure 8.3: California redwood seedlings and saplings, full dataset of Strauss [316].

Figure 8.3 shows the locations of 195 seedlings and saplings of California giant redwood trees in a square sampling region, described and analysed by Strauss [316], and available in `spatstat` as the dataset `redwoodfull`. Most writers have analysed a homogeneous subset of the data, extracted by Ripley [281], which is available as `redwood`. Strauss [316] divided the sampling region into two subregions I and II demarcated by a diagonal line shown in the Figure. The spatial pattern appears to be slightly regular in region I and strongly clustered in region II.

The *Stienen diagram* of a point pattern is obtained by drawing a circle around each data point, of diameter equal to its nearest neighbour distance. Imagine circular balloons, centred at each point of the pattern, growing at a constant rate. Each balloon stops growing when it touches another balloon. The result is the Stienen diagram, generated in `spatstat` by the function `stienen()`, or by

```
> plot(X %mark% nndist(X), markscale=1)
```

where `X` is the point pattern. The left panel of Figure 8.4 shows the Stienen diagram for the full California Redwoods data, generated by

```
> stienen(redwoodfull)
```

Circles are shaded in grey if they are observed without edge effects — that is, if the observed nearest neighbour distance is shorter than the distance to the window boundary.

The Stienen diagram can be useful for revealing aspects of spatial pattern, such as a trend in spatial scale. A pair of circles touching each other represents a pair of points which are mutual nearest neighbours. The fraction of area covered by the Stienen circles is related to the mean square nearest-neighbour distance.

The *Dirichlet tile* associated with a particular data point x_i is the region of space that is closer to x_i than to any other point in the pattern \mathbf{x} :

$$C(x_i | \mathbf{x}) = \{u \in \mathbb{R}^2 : \|u - x_i\| = \min_j \|u - x_j\|\}. \quad (8.3)$$

The Dirichlet tiles are convex polygons (also known as Thiessen polygons, fundamental poly-

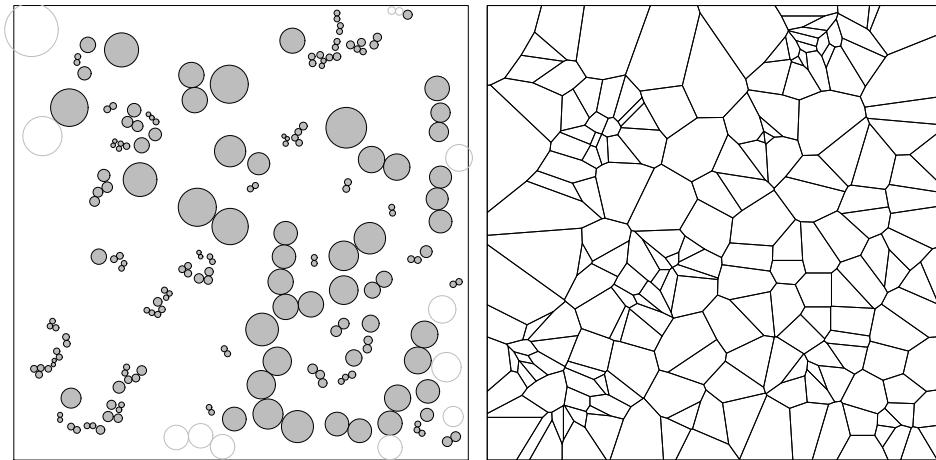


Figure 8.4: Stienen diagram (*Left*) and Dirichlet tessellation (*Right*) for the full California Redwoods data of Figure 8.3.

gons, or domains of influence) which divide two-dimensional space into disjoint regions, forming the *Dirichlet tessellation* (also known as the Voronoi diagram). The right panel of Figure 8.4 shows the Dirichlet tessellation of the full California Redwoods data, generated by `plot(dirichlet(redwoodfull))`.

The function `dirichlet()` computes the Dirichlet tiles themselves, as polygonal windows, returning a tessellation object. The functions `nncross()`, `nnmap()`, `nnfun()` or `tile.index()` can be used to identify which Dirichlet tile contains a given spatial location u . The tile containing u is $C(x_i | \mathbf{x})$ where x_i is the point of \mathbf{x} nearest to u . An image plot of `nnmap(X)` or `nnfun(X)` is essentially a discretised image of `dirichlet(X)`.

Pioneering epidemiologist John Snow [304] used a Dirichlet tessellation to show that the majority of people who died in the Soho cholera epidemic of 1854 lived closer to the infected Broad Street pump than to any other water pump.

[2, 7, 81, 119, 262]

8.2 Nearest-neighbour function G and empty-space function F

Valuable information about the spatial arrangement of points is conveyed by the nearest-neighbour distances. Much of this information is lost if we simply take the average of the nearest-neighbour distances. A better summary of the information is the cumulative distribution function of the nearest neighbour distances, $G(r)$, called the *nearest neighbour distance distribution function* or just the *nearest neighbour function*.

Correspondingly, instead of taking the mean or mean square of the empty space distances, a better summary is their cumulative distribution function $F(r)$, called the *empty space function*.

The nearest neighbour function G and empty space function F are important properties of the point process, analogous to the K -function in some ways, but based on a completely different construction.

8.2.1 Definitions of F and G for a stationary point process

Empty space function F of a point process

If \mathbf{X} is a spatial point process, the distance

$$d(u, \mathbf{X}) = \min\{|u - x_i| : x_i \in \mathbf{X}\} \quad (8.4)$$

from a fixed location $u \in \mathbb{R}^2$ to the nearest point of the process is called the ‘empty space distance’ (or ‘spherical contact distance’ or ‘void distance’).

For a stationary point process, the cumulative distribution function of the empty space distance is the *empty space function*

$$F(r) = \mathbb{P}\{d(u, \mathbf{X}) \leq r\}, \quad (8.5)$$

defined for all distances $r \geq 0$, where u is an arbitrary reference location. Since the process is stationary, this definition does not depend on u . Alternative names for F are the “spherical first contact distribution” and the “point-to-nearest-event distribution”.

An estimate of F can be interpreted as a summary of spatial pattern, as discussed in [108, chaps. 8–9], [297, pp. 488,491], [311, pp. 43,75,80–92,126–127], [107, §4], [245, p. 153 ff.], [123, 283, 284]. The values of $F(r)$ are probabilities (between 0 and 1) giving, for any fixed reference location u , the chance that there will be a point of \mathbf{X} lying within distance r of this location. The value of $F(r)$ increases as a function of r , starting from $F(0) = 0$.

For a stationary process, F is always differentiable, so that it has a probability density function [41, 175].

Nearest neighbour function G of a point process

If x_i is one of the points in a point pattern \mathbf{x} , the nearest neighbour distance $c_i = \min_{j \neq i} \|x_j - x_i\|$ can also be written as

$$c_i = d(x_i, \mathbf{x} \setminus x_i),$$

the shortest distance from x_i to the pattern $\mathbf{x} \setminus x_i$ consisting of all points of \mathbf{x} except x_i . We use this notation frequently below.

For a stationary point process \mathbf{X} the *nearest neighbour function* $G(r)$ is defined by

$$G(r) = \mathbb{P}\{d(u, \mathbf{X} \setminus u) \leq r \mid \mathbf{X} \text{ has a point at } u\} \quad (8.6)$$

for any $r \geq 0$ and any location u . That is, $G(r)$ is the cumulative distribution function of the nearest neighbour distance $d(u, \mathbf{X} \setminus u)$ at a typical point of \mathbf{X} . Since the process is stationary, this definition does not depend on the location u .

The values of $G(r)$ are probabilities and are a non-decreasing function of distance r , starting from $G(0) = 0$.

In general, the nearest-neighbour function G may not be differentiable, so that G may not have a probability density. For example, a randomly-translated regular grid of points is a stationary point process; the distance from each point to its nearest neighbour is a fixed value, so $G(r)$ is a step function.

8.2.2 Values of F and G for complete randomness

Empty space function F for Poisson process

In order to calculate $F(r)$ for a point process, note the key fact that

$$d(u, \mathbf{X}) > r \text{ if and only if } n(\mathbf{X}_{b(u,r)}) = 0 \quad (8.7)$$

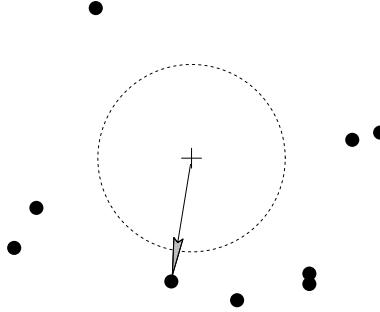


Figure 8.5: Duality between distances and counts. The distance $d(u, \mathbf{X})$ from a fixed location u (+) to the nearest random point (\bullet) satisfies $d(u, \mathbf{X}) > r$ if and only if there are no random points in the disc of radius r centred at u .

where $b(u, r)$ is the disc of radius r centred at u . That is, the distance to the nearest point of \mathbf{X} is greater than r , if and only if the disc of radius r contains no points of \mathbf{X} . See Figure 8.5.

This makes it possible to calculate $F(r)$ for a completely random pattern. If \mathbf{X} is a uniform Poisson process in \mathbb{R}^2 of intensity λ , the number of points falling in a set B has a Poisson distribution with mean $\lambda|B|$. The probability that no points fall in B is the probability of zero counts in the Poisson distribution, $\exp(-\lambda|B|)$. If B is a disc of radius r , then $|B| = \pi r^2$, so we get $\mathbb{P}\{d(u, X) > r\} = \mathbb{P}\{n(\mathbf{X}_{b(u,r)}) = 0\} = \exp(-\lambda\pi r^2)$, which gives

$$F_{pois}(r) = 1 - \exp(-\lambda\pi r^2). \quad (8.8)$$

This is the theoretical empty space function for complete spatial randomness. Notice that (8.8) depends on the intensity λ . The function is plotted in Figure 8.6 for the case $\lambda = 1$; for a general λ the plot is identical except that the horizontal axis is rescaled by the factor $1/\sqrt{\lambda}$.

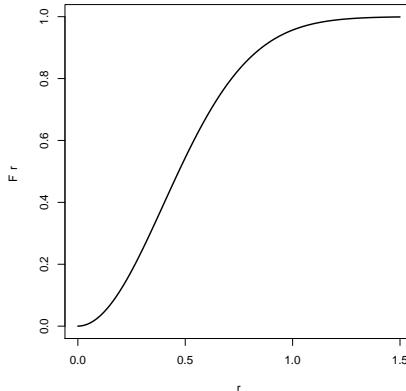


Figure 8.6: Theoretical empty space function for a homogeneous Poisson process with intensity $\lambda = 1$. The mean is 0.5 and the median is about 0.47.

Under complete spatial randomness, the empty space distance has mean value $\mathbb{E}[d(u, \mathbf{X})] = 1/(2\sqrt{\lambda})$ and variance $\text{var}[d(u, \mathbf{X})] = (1/\pi - 1/4)/\lambda$. The mean squared empty space distance is $\mathbb{E}[d(u, \mathbf{X})^2] = 1/(\pi\lambda)$.

It is also interesting to study the “contact disc”, the disc of radius $d(u, \mathbf{X})$ centred at u . The area of the contact disc, $A = \pi d(u, X)^2$ has a negative exponential distribution with rate λ :

$$\mathbb{P}\{A \leq a\} = 1 - e^{-\lambda a}, \quad a \geq 0.$$

Nearest neighbour function G for Poisson process

The nearest neighbour function for complete spatial randomness is now easy to calculate. Suppose \mathbf{X} is a uniform Poisson process in \mathbb{R}^2 of intensity λ . Since the points of \mathbf{X} are independent of each other, the information that there is a point of \mathbf{X} at the location u does not affect the probability distribution of the rest of the process. Therefore

$$\begin{aligned} G(r) &= \mathbb{P}\{d(u, \mathbf{X} \setminus u) \leq r \mid \mathbf{X} \text{ has a point at } u\} \\ &= \mathbb{P}\{d(u, \mathbf{X} \setminus u) \leq r\}. \end{aligned}$$

But this is the empty space function of the Poisson process, which has already been calculated in (8.8). That is, *for a homogeneous Poisson process, the nearest neighbour function is*

$$G_{pois}(r) = 1 - \exp(-\lambda \pi r^2) \tag{8.9}$$

identical to the empty space function for the same process, $G_{pois} \equiv F_{pois}$. For a completely random pattern, the distribution of the nearest neighbour distance is the same as that of the empty space distance.

For a general point process, of course, F and G will be different functions.

8.2.3 Estimation from data

Estimators of the empty space function $F(r)$ and the nearest-neighbour function $G(r)$ from a point pattern dataset are described in Section 8.8. These estimators *assume that the point process is stationary*. They are edge-corrected versions of the empirical cumulative distribution functions of the nearest-neighbour distances at all data points (for G) and the empty-space distances at a grid of test locations (for F).

Experience suggests that the choice of edge correction is not very important so long as some edge correction is performed. However, some software implementations of these edge corrections may be incorrect, which could introduce substantial bias. Trustworthy R packages include `spatial`, `splancs` and `spatstat`.

In `spatstat`, estimates of $F(r)$ and $G(r)$ are computed from a point pattern X by the commands `Fest(X)` and `Gest(X)`, respectively. The syntax for these commands is very similar to `Kest()`.

```
> Fs <- Fest(swedishpines)
> Gs <- Gest(swedishpines)
```

Tip: Don't use the single letter `F` as the name of an object. `F` is a recognised variable in the R base environment, initially set to the logical value `FALSE`.

The result of `Fest()` or `Gest()` is again an object of class "fv", containing several estimates of the function using different edge corrections, together with the theoretical value for a homogeneous Poisson process with the same average intensity.

```
> Fs
Function value object (class 'fv')
for the function r -> F(r)
```

```
.....
Math.label      Description
r              r          distance argument r
theo    F[pois](r)   theoretical Poisson F(r)
cs     hat(F)[cs](r) Chiu-Stoyan estimate of F(r)
rs     hat(F)[bord](r) border corrected estimate of F(r)
km     hat(F)[km](r) Kaplan-Meier estimate of F(r)
hazard  hat(h)[km](r) Kaplan-Meier estimate of hazard function h(r)
theohaz h[pois](r)   theoretical Poisson hazard h(r)
.....
Default plot formula: .~r
where "." stands for 'km', 'rs', 'cs', 'theo'
Recommended range of argument r: [0, 7.6875]
Available range of argument r: [0, 22.312]
Unit of length: 0.1 metres
```

Like the K -function, the functions F and G are typically inspected by plotting the empirical function calculated from the data, together with the theoretical empty space function of the homogeneous Poisson process with the same average intensity, plotted against distance r . This is the default behaviour in `spatstat`. Figure 8.7 shows the plots generated by the commands

```
> Swedish <- rescale(swedishpines)
> plot(Fest(Swedish))
> plot(Gest(Swedish))
```

Each plot has four curves, as indicated in the legend: three very similar estimates of the function using different edge corrections, and the theoretical function for complete spatial randomness.

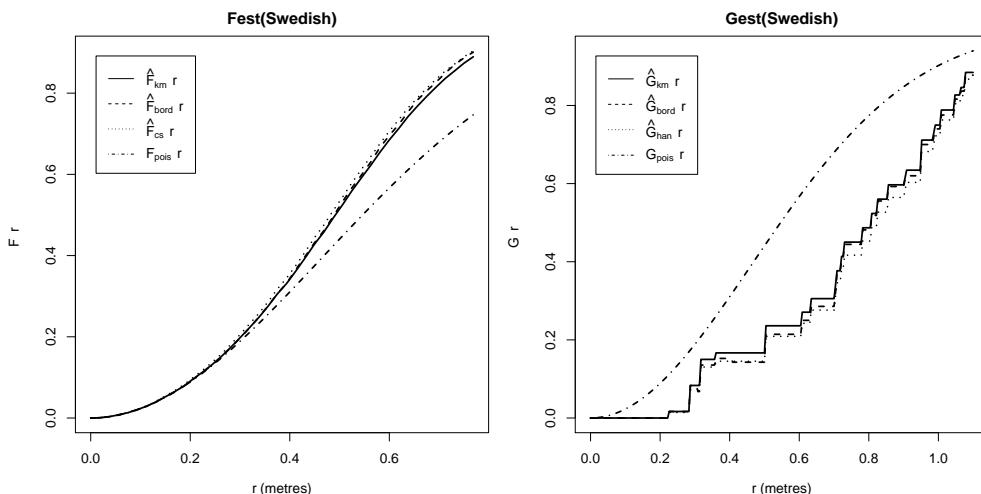


Figure 8.7: Distance analysis of Swedish Pines data: empty space function (*Left*) and nearest neighbour function (*Right*).

Reading off the right panel in Figure 8.7, the value $G(r) = 0.5$ is achieved at about $r = 0.8$ metres, so the median of the distribution of nearest-neighbour distance in the Swedish Pines is about 80 centimetres. For a completely random process of the same intensity, the median nearest neighbour distance would be about 55 centimetres.

8.2.4 Interpretation of the empty space function $\hat{F}(r)$

Figure 8.8 shows the estimated empty space function $\hat{F}(r)$ for each of the three archetypal patterns in Figure 7.1, using only one edge correction for simplicity.

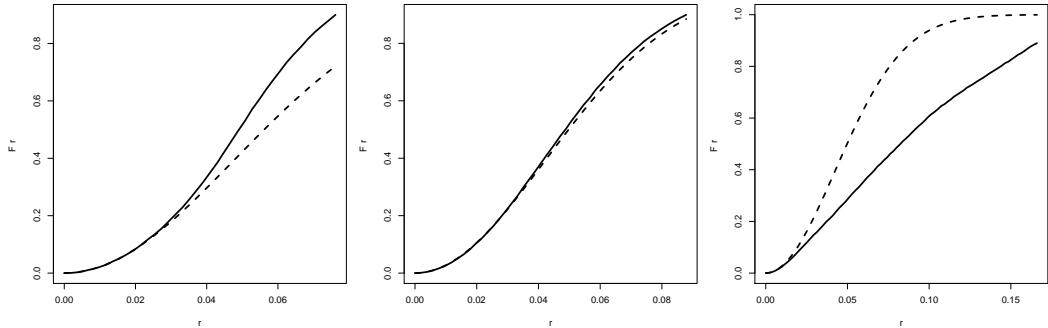


Figure 8.8: Empirical empty space function \hat{F} (solid lines) for each of the three patterns in Figure 7.1, and the theoretical function for a Poisson process (dashed lines). Generated using the plot formula `cbind(km, theo) ~ r`.

The interpretation of deviations in F is the opposite of that for the K -function. In the left panel of Figure 8.8, the curve for the empirical empty space function (solid lines) is above the theoretical curve for a completely random pattern (dashed lines), $\hat{F}(r) > F_{pois}(r)$. That is, for a given distance r , the probability that $d(u, \mathbf{X}) \leq r$ is greater in this pattern than it would be in a completely random pattern. Consequently the empty space distances in this pattern are *shorter* than expected if the pattern were completely random. This is consistent with a regular point process.

Similarly in the right panel, the empirical curve is below the theoretical curve, $\hat{F}(r) < F_{pois}(r)$, indicating that empty space distances are larger than would be expected if the pattern were completely random; this is consistent with clustering.

It may be helpful to interpret the value of $F(r)$ as the average *fraction of area* occupied by the level set of the distance function

$$\mathbf{X}_{\oplus r} = \{u \in \mathbb{R}^2 : d(u, \mathbf{X}) \leq r\}, \quad (8.10)$$

the set of all spatial locations which lie at most r units away from the point process \mathbf{X} . This region is sketched in Figure 8.9: it is the union of discs of radius r centred at each point of \mathbf{X} . It is variously called a “buffer region”, “Steiner set” or “morphological dilation” of \mathbf{X} by distance r . As r increases, the shaded region in Figure 8.9 grows and eventually fills the space. The space is filled faster if the data points are regularly spaced, and is filled more slowly if the data points are clustered together.

8.2.5 Interpretation of the nearest neighbour function G

Figure 8.10 shows the estimated empty space function $\hat{F}(r)$ for each of the three archetypal patterns in Figure 7.1, using only one edge correction.

The interpretation of deviations in $G(r)$ is similar to that for the K -function, and the opposite of that for $F(r)$. In the left panel of Figure 8.10, the curve for the empirical nearest neighbour function (solid lines) is below the theoretical curve for a completely random pattern (dashed lines), $\hat{G}(r) < G_{pois}(r)$. The nearest-neighbour distances in the data are *longer* than would be expected for a completely random pattern with the same average intensity. This is consistent with a regular pattern. Indeed, $\hat{G}(r)$ is zero for distances less than about 0.09, which indicates that there are no nearest-neighbour distances shorter than 0.09. In the right panel of Figure 8.10, the empirical curve lies

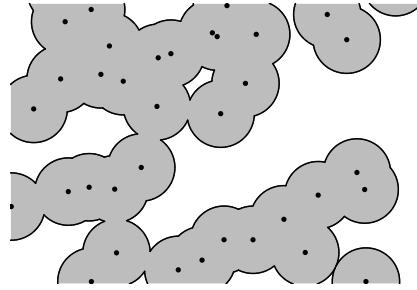


Figure 8.9: Dilation set $\mathbf{X}_{\oplus r}$ (grey shading) of a point process \mathbf{X} (\bullet). The empty space probability $F(r)$ is the average fraction of area occupied by $\mathbf{X}_{\oplus r}$.

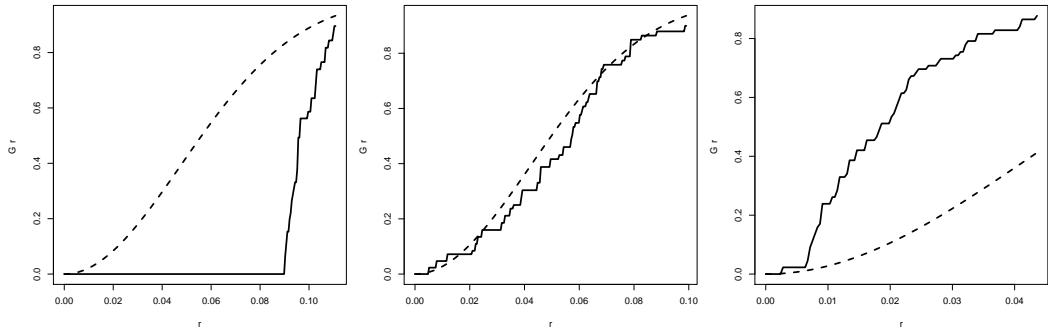


Figure 8.10: Empirical nearest neighbour function \hat{G} (solid lines) for each of the three patterns in Figure 7.1, and the theoretical function for a Poisson process (dashed lines). Generated using the plot formula `cbind(km, theo) ~r`.

above the theoretical curve for a completely random pattern, indicating that the nearest-neighbour distances in the data are *shorter* than expected for a completely random pattern. This is consistent with clustering.

8.2.6 Implications for modelling

If the purpose of estimating the functions F and G is to guide the selection of appropriate point process models, then the functions $1 - F(r)$ and $1 - G(r)$ may be more useful.

In a modelling scenario where random points come from several different sources, $1 - F(r)$ is obtained by multiplying contributions from each of the sources. If \mathbf{X} and \mathbf{Y} are two *independent* stationary point processes with empty space functions $F_{\mathbf{X}}(r)$ and $F_{\mathbf{Y}}(r)$, then their superposition $\mathbf{X} \cup \mathbf{Y}$ has empty space function $F_{\mathbf{X} \cup \mathbf{Y}}(r)$ where

$$1 - F_{\mathbf{X} \cup \mathbf{Y}}(r) = (1 - F_{\mathbf{X}}(r)) (1 - F_{\mathbf{Y}}(r)) \quad (8.11)$$

as we can see easily by remembering that $1 - F(r)$ is the probability that a disc of radius r contains no points.

The corresponding relation for G is more complicated:

$$1 - G_{\mathbf{X} \cup \mathbf{Y}}(r) = \frac{\lambda_{\mathbf{X}}}{\lambda_{\mathbf{X}} + \lambda_{\mathbf{Y}}} (1 - G_{\mathbf{X}}(r)) (1 - F_{\mathbf{Y}}(r)) + \frac{\lambda_{\mathbf{Y}}}{\lambda_{\mathbf{X}} + \lambda_{\mathbf{Y}}} (1 - F_{\mathbf{X}}(r)) (1 - G_{\mathbf{Y}}(r)) \quad (8.12)$$

where $\lambda_{\mathbf{X}}, \lambda_{\mathbf{Y}}$ are the intensities of \mathbf{X} and \mathbf{Y} respectively. This arises because a typical point $\mathbf{X} \cup \mathbf{Y}$

originally came from \mathbf{X} with probability $p_{\mathbf{X}}$ and from \mathbf{Y} with probability $1 - p_{\mathbf{X}}$, where $p_{\mathbf{X}} = \lambda_{\mathbf{X}} / (\lambda_{\mathbf{X}} + \lambda_{\mathbf{Y}})$. For a typical point of \mathbf{X} , say, the probability that there are no other points of \mathbf{X} within a distance r is of course $1 - G_{\mathbf{X}}(r)$, and because of independence, the probability that there are no points of \mathbf{Y} within distance r is $1 - F_{\mathbf{Y}}(r)$. The complexity of (8.12) is best handled using the J -function, discussed in Section 8.5.

8.3 Formal inference and diagnostic plots

Confidence intervals and simulation envelopes are available for the empty space function and nearest-neighbour function, using the procedures described in Section 7.7. Since $F(r)$ and $G(r)$ are cumulative distribution functions, some additional tricks are available, and are sketched below.

Figure 8.11 shows pointwise 95% confidence intervals for the true values of $F(r)$ and $G(r)$ for the Swedish Pines data, obtained by the spatial block bootstrap with a 5×5 grid of blocks, using

```
> Fci <- varblock(Swedish, Fest, nx=5, correction="best")
> Gci <- varblock(Swedish, Gest, nx=5, correction="best")
```

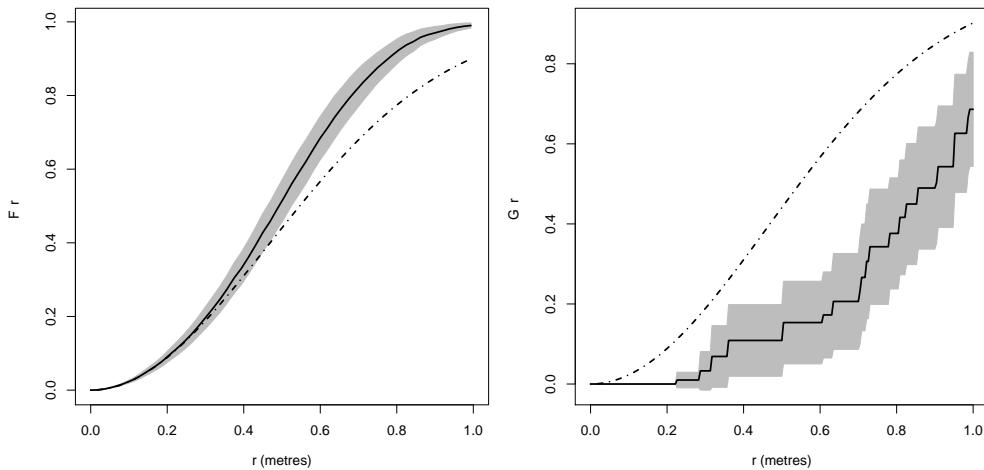


Figure 8.11: Pointwise 95% confidence intervals for the true value of the empty space function (*Left*) and the nearest-neighbour function (*Right*) for the rescaled Swedish Pines data.

Figure 8.12 shows pointwise 5% significance envelopes for the null hypothesis of complete spatial randomness, generated by

```
> Fenv <- envelope(Swedish, Fest, nsim=39, fix.n=TRUE)
> Genv <- envelope(Swedish, Gest, nsim=39, fix.n=TRUE)
```

The shaded regions in Figures 8.11 and 8.12 have a ‘spindle’ shape (which can be revealed more clearly by plotting the same objects using the plot formula $hi - lo \sim r$). This shape is to be expected, because $F(r)$ and $G(r)$ are probabilities, and the estimate of a population proportion is usually governed by the variance properties of the binomial distribution. This holds in our context, where limit theorems [35, 194] imply that $\text{var}[\hat{F}(r)]$ should be approximately proportional to $F(r)(1 - F(r))$, except for values of r close to 0 or 1. This can be confirmed by plotting the same objects using the formula $hi - lo \sim \sqrt{\text{theo} * (1-\text{theo})}$.

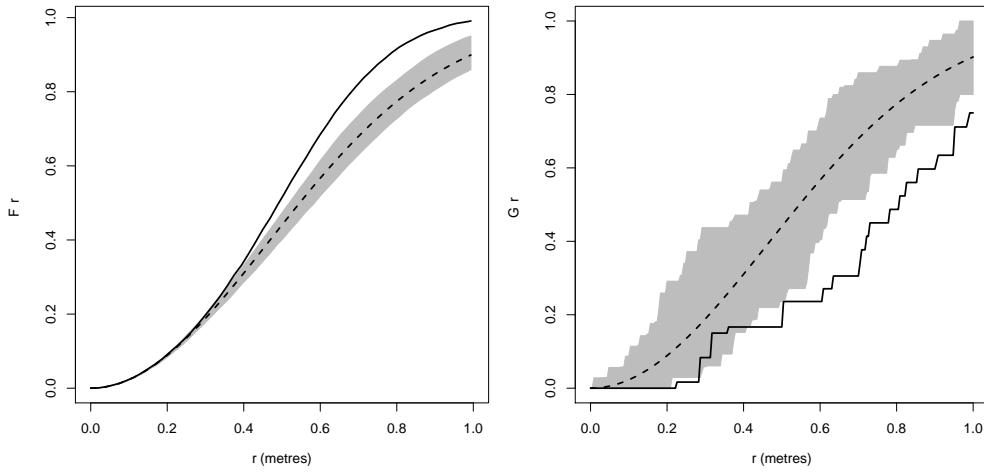


Figure 8.12: Pointwise 5% significance envelopes for the empty space function (*Left*) and the nearest-neighbour function (*Right*) for testing the hypothesis of complete spatial randomness on the rescaled Swedish Pines data.

The variance-stabilising transformation for the binomial distribution is *Fisher's arcsine transformation*

$$\Phi(p) = \arcsin \sqrt{p}. \quad (8.13)$$

Therefore the variance of $\Phi(\hat{F}(r))$ should be approximately constant as a function of r .

Variance stabilisation is particularly useful for constructing global simulation envelopes. Figure 8.13 shows a global envelope for $\Phi(F(r))$ based on simulations from complete spatial randomness, generated by

```
> Phi <- function(x) asin(sqrt(x))
> Fglob <- envelope(Swedish, Fest, nsim=19, fix.n=TRUE,
                      global=TRUE, ginterval=c(0,1),
                      transform=expression(Phi(.)))
```

and plotted in the default style. Note that, although we could have given any name to the R function which implements the arcsine transformation (8.13), by choosing the name `Phi` we ensured that it would be rendered as a Greek letter in the plot labels.

Another feature in Figure 8.12 is the ‘‘elephant’s foot’’ at the bottom left corner of the right-hand panel. For any point pattern \mathbf{x} , the graph of the empirical nearest-neighbour function $\hat{G}(r)$ has a flat segment at zero height at the bottom left, because $\hat{G}(r) = 0$ for all r less than the *minimum nearest-neighbour distance* $c_{\min} = \min_i c_i$.

At such small values of r , the asymptotic normal approximation to $\hat{G}(r)$ breaks down completely, and Φ does not stabilise the variance. Small values of r should therefore be avoided when constructing a global envelope or global confidence interval for $G(r)$. How small? For a homogeneous Poisson process observed in a window W , the *minimum* nearest neighbour distance has very roughly² the distribution (8.8) with the parameter λ replaced by $\lambda^2|W|/2$. The expected minimum nearest-neighbour distance is about $r_0 = 1/(\lambda \sqrt{2|W|})$ which we could estimate by $\hat{r}_0 = \sqrt{|W|}/(n\sqrt{2})$

²This is based on the heuristic that the n points are $n/2$ pairs of mutual nearest neighbours, with nearest-neighbour distances independent between each pair. Since πc_i^2 is exponential with rate λ , the minimum of $n/2$ such values, namely πc_{\min}^2 , is exponential with rate $n\lambda/2$.

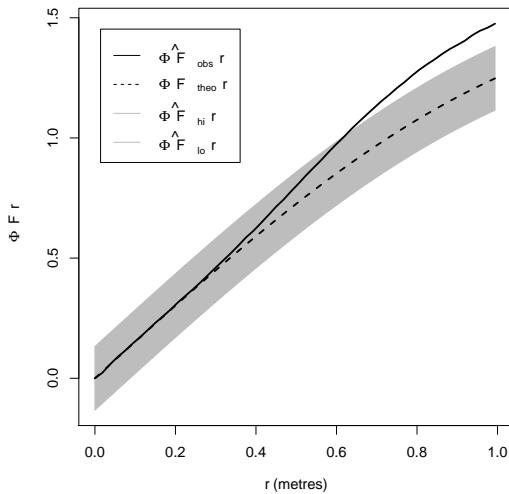


Figure 8.13: Global envelope of the variance-stabilised empty space function.

where n is the number of data points. Referring to Figure 8.6, the minimum nearest neighbour distance is very unlikely to exceed $3r_0$. Thus, we should avoid distances smaller than about $3r_0$.

P–P and Q–Q plots [339] are useful statistical graphics for comparing observed and predicted distributions. The P–P plot for the empty space function (say) is a plot of the empirical probability $\hat{F}(r)$ against the theoretical probability $F_{pois}(r)$ for every r . This can be generated using a plot formula in `plot.fv()` (Section 7.5.2, page 185):

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

Again the symbol `.` stands for ‘all recommended estimates of the function’. The result is shown in the left panel of Figure 8.14. Theoretical probabilities are plotted on the x axis, and empirical probabilities on the y axis. The diagonal line $y = x$ corresponds to perfect agreement between the observed data and a Poisson process. Reading off the plot, we find that the median empty-space distance in a Poisson process ($x = 0.5$ in the plot) is equal to the 60th percentile of empty-space distances in the data ($y = 0.6$ in the plot).

The *Q–Q plot* for the nearest neighbour function (say) is the plot of empirical quantiles $\hat{G}^{-1}(p)$ against theoretical quantiles $G_{pois}^{-1}(p)$ for every p between 0 and 1, where \hat{G}^{-1} and G_{pois}^{-1} are the inverse functions. This can be generated by

```
> plot(QQversion(Gest(Swedish)))
```

The result is shown in the right panel of Figure 8.14. Empirical quantiles are plotted on the x -axis, and the corresponding theoretical quantiles on the y -axis. For example the observed proportion of nearest neighbour distances less than 80 centimetres ($x = 0.8$) is equal to the theoretically expected proportion of nearest neighbour distances less than 55 centimetres ($y = 0.55$). The diagonal line $y = x$ corresponds to perfect agreement between the observed data and a Poisson process. Another straight line would correspond to a linear relationship between the observed and theoretical nearest-neighbour distances: a straight line through the origin would correspond to a Poisson process with a different intensity.

Aitkin and Clayton [4] suggested applying this variance-stabilising transformation to *both* axes in a P–P plot to preserve the linear relationship. For the previously-computed pointwise confidence intervals and envelopes `Fci`, `Gci`, `Fenv`, `Genv` this could be achieved by plotting the object using the formula `Phi(.) ~ Phi(theo)`.

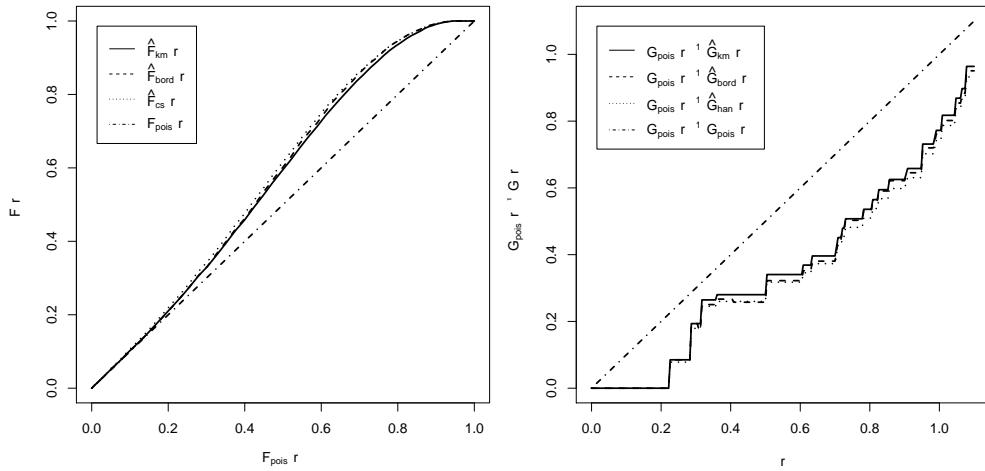


Figure 8.14: P–P and Q–Q plots. *Left:* P–P plot of empty space function; *Right:* Q–Q plot of nearest neighbour function. Rescaled Swedish Pines data.

The global envelope F_{glob} was computed using `transform = expression(Phi(.))`, so the appropriate plot formula is `. ~ theo`. The result is shown in Figure 8.15. To make a global envelope for the variance-stabilised nearest neighbour function $\Phi(G(r))$, we should avoid small distances:

```
> r0 <- sqrt(area.owin(Window(Swedish))/2)/npoints(Swedish)
> Gglob <- envelope(Swedish, Gest, nsim=19, fix.n=TRUE,
  global=TRUE, ginterval=c(3*r0,1),
  transform=expression(Phi(.)))
```

then plot this with the formula `. ~ theo`. The result is shown in the right panel of Figure 8.15.

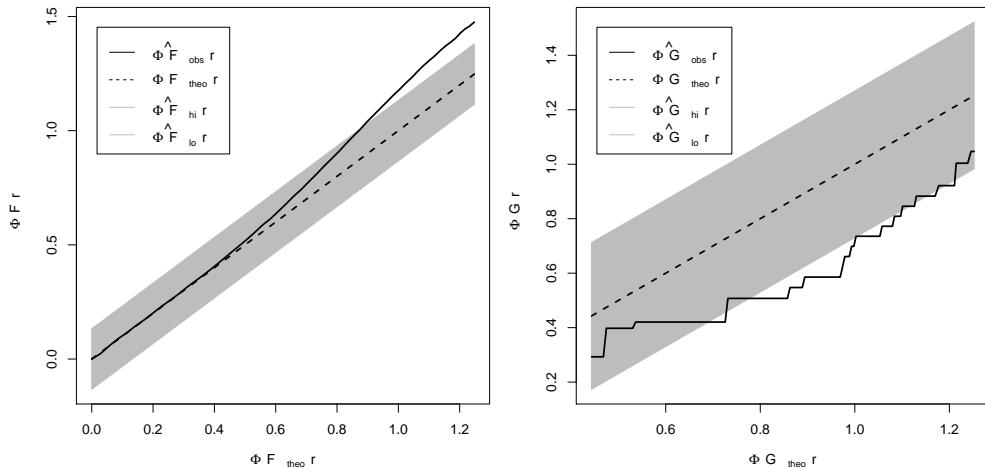


Figure 8.15: Global envelopes for variance-stabilised summary functions. *Left:* Empty space function. *Right:* Nearest neighbour function, avoiding short distances.

8.4 Empty space hazard

[40, 176, 41, 175]

In Chapter 7 we saw that the cumulative nature of the K -function has some disadvantages. Similar comments apply to the cumulative distribution functions $F(r)$ and $G(r)$. While they are very useful for assessing statistical significance, their interpretation is complicated by the fact that the value of $F(r)$ or $G(r)$ contains information from all distances s less than or equal to r .

There is a need for alternative summary functions (derived from F and G) which contain only contributions from distances *equal to* r . For technical reasons we will concentrate mostly on the empty space function F .

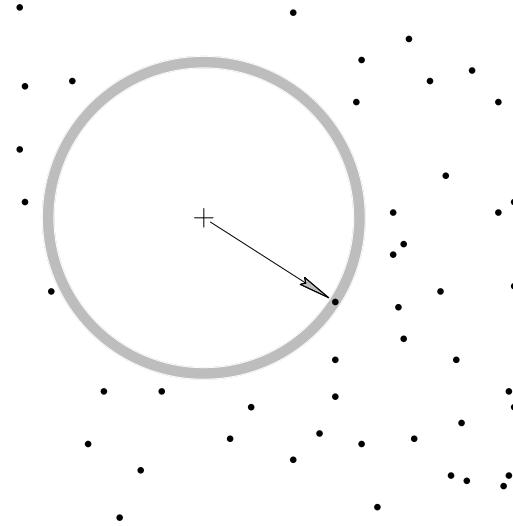


Figure 8.16: Geometry for derivative and hazard rate of empty space function.

The most obvious choice for a non-cumulative alternative to F is the *derivative* $f(r) = F'(r)$, which is the probability density function of the empty-space distance $d(u, \mathbf{X})$ from an arbitrary fixed location u to the nearest point in \mathbf{X} . Its geometrical interpretation is sketched in Figure 8.16. Consider two circles, of radius r and $r + \Delta r$, centred at the reference location u . The distance $d(u, \mathbf{X})$ lies between r and $r + \Delta r$ if and only if there are no points of \mathbf{X} in the inner circle of radius r , and there is a point of \mathbf{X} in the outer circle of radius $r + \Delta r$. This has probability $F(r + \Delta r) - F(r)$. Dividing by Δr and taking the limit as $\Delta r \rightarrow 0$ we obtain the derivative $f(r)$.

For a homogeneous Poisson process, the density $f(r)$ of the empty space function can be calculated from (8.8):

$$f_{\text{pois}}(r) = F'_{\text{pois}}(r) = 2\pi r \lambda \exp(-\lambda \pi r^2). \quad (8.14)$$

This is not very convenient as a benchmark for practical interpretation.

Refer again to the geometry of Figure 8.16. The probability of the event depicted in Figure 8.16 is approximately $f(r)\Delta r$, or

$$[\lambda 2\pi r \Delta r] \exp(-\lambda \pi r^2).$$

The term $\exp(-\lambda \pi r^2)$ is the probability of the event that there are no points of \mathbf{X} in the disc of radius r . The term $\lambda 2\pi r \Delta r$ is (for small Δr) the probability of the event that at least one point of \mathbf{X} falls in the ring between the two circles of radius r and $r + \Delta r$. For a Poisson process, these two events are independent, so the probability that they both occur is the product of their probabilities. This explains the complicated form of (8.14).

A better alternative to $f(r)$ is the *hazard rate*

$$h(r) = \frac{f(r)}{1 - F(r)}. \quad (8.15)$$

The hazard rate has the following simple interpretation: *given* that the empty space distance $d(u, \mathbf{X})$ is greater than r , the probability that it falls between r and $r + \Delta r$ is equal to $h(r)\Delta r$. In Figure 8.16, $h(r)\Delta r$ is the conditional probability that a point falls in the shaded ring, *given* that no point falls in the inner circle.

For complete spatial randomness, the hazard rate for the empty space distance is

$$h_{pois}(r) = 2\pi\lambda r, \quad (8.16)$$

a linear function of r , making it very useful as a benchmark.

The hazard rate is a useful tool in biostatistics and risk analysis for studying the distribution of lifetimes, survival times, time-to-first-failure, and similar quantities. For human lifetimes, the hazard rate answers the question: given that I have reached age n years, what is the chance that I will die before my $(n + 1)$ th birthday? Human mortality, machine reliability and so on, are much easier to interpret using the hazard rate than the probability density.

The `spatstat` command `Fest()`, in addition to calculating estimates of the empty space function $F(r)$, also computes an estimate $\hat{h}(r)$ of the hazard rate $h(r)$ of the empty space function, and the corresponding theoretical hazard $h_{pois}(r)$ for a Poisson process with the same intensity. The hazard rate estimate is not plotted in the default plot, because $h(r)$ is on a completely different scale from $F(r)$. However, the estimated and theoretical hazard functions are shown as columns of the result of `Fest()` if the result is printed (see page 8.2.3 for an example). These columns can be plotted by specifying them in the plot formula, for example,

```
> plot(Fest(cells), cbind(hazard, theohaz) ~ r)
```

Figure 8.17 shows the estimated empty space hazard $h(r)$ for the three archetypal point patterns in Figure 7.1.

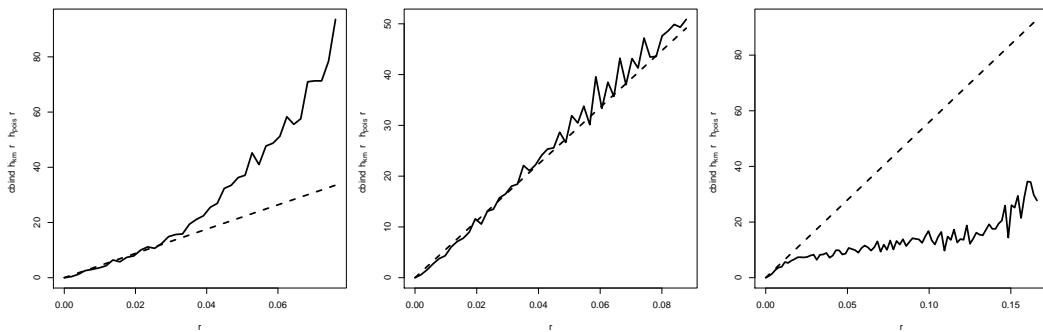


Figure 8.17: Estimated empty space hazard \hat{h} (solid lines) for each of the three patterns in Figure 7.1, and the theoretical hazard function for a Poisson process (dashed lines). Generated using the plot formula `cbind(hazard, theohaz) ~ r`.

The `spatstat` function `Fhazard()` is a wrapper for `Fest` which extracts the hazard estimates. It should be used when computing confidence intervals and simulation envelopes. Figure 8.18 shows an example computed by

```
> hazenv <- envelope(Swedish, Fhazard, nsim=39, fix.n=TRUE,
  transform=expression(./(2*pi*r)))
```

Dividing the hazard rate $h(r)$ by $2\pi r$ means that complete spatial randomness corresponds to a constant value. This transformation also seems to give approximately constant variance.

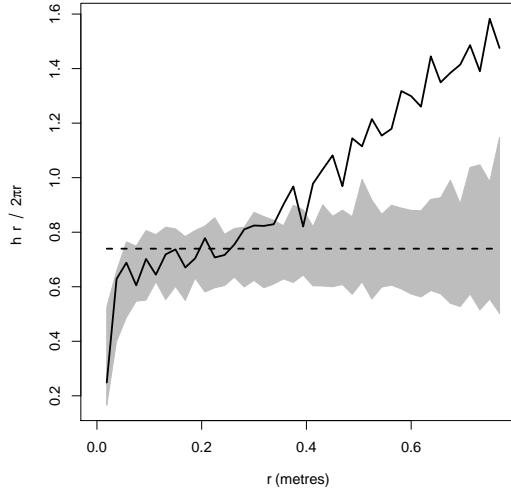


Figure 8.18: The standardised hazard function $h(r)/(2\pi r)$ for the rescaled Swedish Pines data, and the pointwise envelopes of 39 simulations of complete spatial randomness.

Estimates of the hazard rate are usually computed directly from data using specialised estimators described in Section 8.8. If an estimate of the function $F(r)$ is already available, then $h(r)$ can be computed as the derivative

$$h(r) = \frac{d}{dr} [-\log(1 - F(r))]. \quad (8.17)$$

Other material justifying the use of hazard rate:

The interpretation of distance as a waiting time or failure time was proposed in [39, 156]; earlier Miles [246, 248] had shown that many distance and size variables arising in stochastic geometry are generalisations of one-dimensional waiting times. See [251, 347].

The transformation from F to h is a convenient simplification, at least in the Poisson case. Daley and Vere-Jones [112, pp. 242–245] have discussed the (theoretical) survival function and hazard rate of F for Poisson cluster point processes.

The function $-\log(1 - F)$ has been much used in connection with parameter estimation for the Boolean model [108, §9.6], [170, pp. 291–294], [297, pp. 495–502], [311, pp. 84–92], [284, chap. 6]. Indeed $-\log(1 - F)$ is a restriction (to the special case of discs) of the functional $\psi(K) = -\log \mathbb{P}\{X \cap K = \emptyset\}$ defined for compact sets $K \subset \mathbb{R}^d$; ψ is a standard functional in stochastic geometry, connected with characterisations of infinite divisibility, convexity and other properties [242, chap. 3, thm. 5.3.1, 5.5.1].

Finally, the use of h parallels the use of kernel estimators of the density of the nearest-neighbour distribution function [131], [245, pp. 137–138] and of the pair correlation function [311, p. 126], [132, 309, 313] (but see [241, 264]). Kernel estimators of the density of F have been proposed in [133, 144].

8.5 *J*-function

8.5.1 Comparing nearest-neighbour and empty-space distances

Nearest-neighbour distances and empty-space distances have the same probability distribution, if the pattern is completely random. Under various departures from complete spatial randomness, the nearest-neighbour and empty-space distances tend to respond in opposite directions — one becoming larger while the other becomes smaller. This suggests that a comparison of these two types of distance could be useful in assessing departure from CSR.

The Hopkins-Skellam test (Section 8.1.2) is an example of such a technique. The test statistic (8.2) is the ratio of the mean squares of the nearest-neighbour and empty-space distances.

Diggle [121, eq. (5.7)] proposed the diagnostic

$$D_{max} = \max_r |\hat{G}(r) - \hat{F}(r)|$$

as a measure of deviation from the Poisson process. A Monte Carlo test based on D_{max} is equivalent to a global envelope test based on

$$D(r) = G(r) - F(r).$$

This function is identically equal to zero for a homogeneous Poisson process. Values $D(r) > 0$ are consistent with clustering, while $D(r) < 0$ is consistent with regularity. The function can be calculated in spatstat using `eval.fv()`:

```
> FS <- Fest(Swedish)
> GS <- Gest(Swedish)
> DD <- eval.fv(GS-FS)
```

In order to construct a confidence interval or simulation envelope, this should be coded as a single function:

```
> DigDif <- function(X, ..., r=NULL) {
  FX <- Fest(X, ..., r=r)
  GX <- Gest(X, ..., r=FX$r)
  eval.fv(GX-FX)
}
```

The argument `r` is not necessary here, but ensures that the two function objects have exactly the same vector of r values. Then a simulation envelope for testing CSR (for example) could be constructed by

```
> DE <- envelope(Swedish, DigDif, nsim=19, fix.n=TRUE, global=TRUE)
```

Weaknesses of the Hopkins-Skellam index and the diagnostic D_{max} include the loss of information about spatial scale, and the lack of a clear interpretation for their numerical values, other than with reference to CSR. The function $D(r)$ does have an interpretation as a score residual for testing the null hypothesis of CSR against a particular alternative hypothesis; see Section 11.8.4.

8.5.2 The *J* function

A useful combination of F and G is the *J function* [322] of a stationary point process,

$$J(r) = \frac{1 - G(r)}{1 - F(r)}, \quad (8.18)$$

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

$$J_{pois}(r) \equiv 1. \quad (8.19)$$

Values $J(r) > 1$ suggest regularity, and $J(r) < 1$ suggest clustering (at scales less than or equal to r).

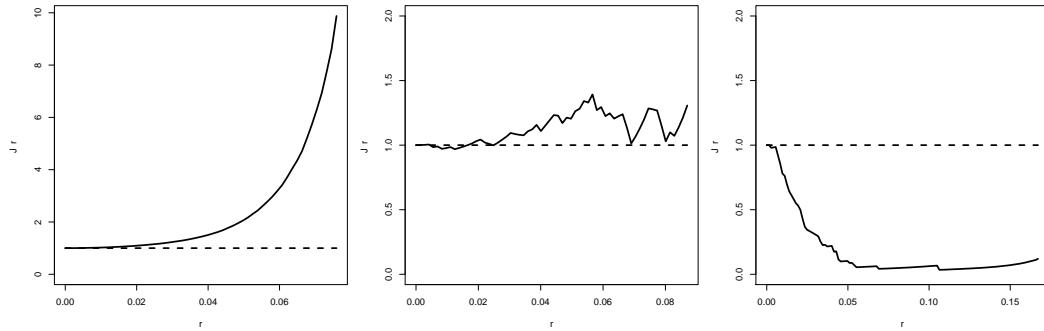


Figure 8.19: Empirical J -function (solid lines) for each of the three patterns in Figure 7.1, and the theoretical function for a Poisson process (dashed lines). Generated using the plot formula `cbind(km, theo) ~ r`.

Notice that $1 - G(r)$ is the probability that the nearest-neighbour distance will be *greater than* r , and $1 - F(r)$ is the corresponding probability for the empty-space distances. A value $J(10) = 3$, for example, would mean that the distance threshold of 10 units is exceeded 3 times more often by the nearest-neighbour distances than by the empty-space distances.

The `spatstat` function `Jest()` computes edge-corrected estimates of the J -function from a point pattern dataset, assuming the point process is stationary. Its syntax is identical to that of `Gest()` and `Fest()`.

The empirical J -function is insensitive to edge effects, because the nearest-neighbour and empty-space distances are subject to similar edge effects which “cancel out” in the ratio $(1 - G)/(1 - F)$ to a first approximation [27]. In many applications the uncorrected estimate of the J -function can be used, saving computation time and maximising the use of data.

Based on fundamental GNZ formula – canonical comparison

The J -function has good properties with respect to some operations on point processes. One particularly appealing property 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

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

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

The J -function can be evaluated for many point processes even when the F and G functions cannot [322]. For example, it can be evaluated for Neyman-Scott cluster processes. A general formula for the J -function of a Gibbs process is also available (Section 8.9.4) and can be used at least to derive qualitative properties.

Approximation

$$J(r) - 1 \approx -\lambda(K(r) - \pi r^2)$$

to second order [321].

Applications in astronomy [198, 200, 199] and soil science [306].

Like the K -function, the functions F , G and J do not completely characterise the point process. In particular, if data analysis suggests that $J(r) \equiv 1$, this does not guarantee that the point process is completely random. A counterexample in one dimensional space was found in [53].

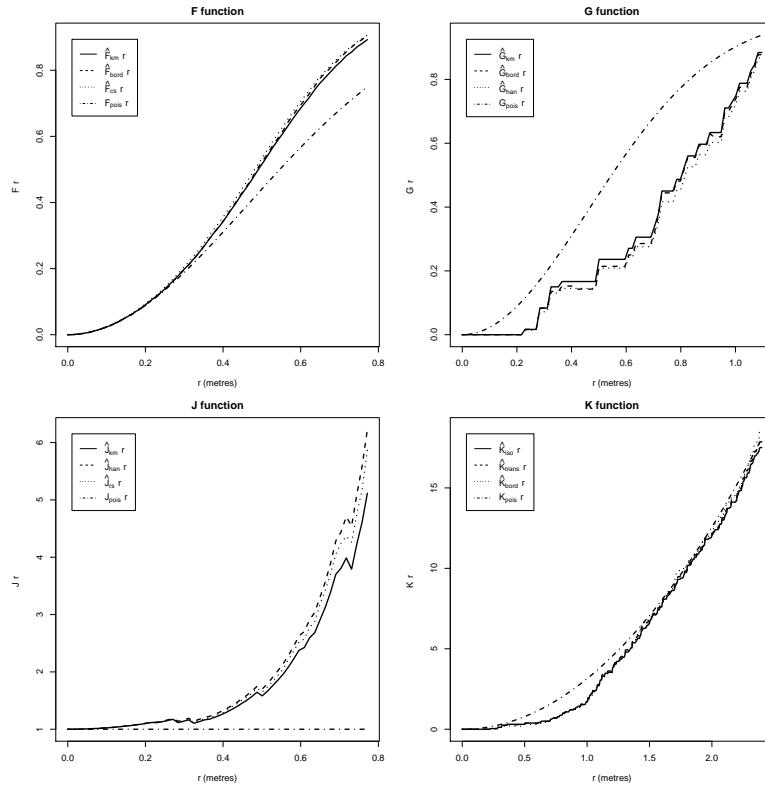


Figure 8.20: Result of `allstats()` for the Swedish Pines data.

The convenient function `allstats()` efficiently computes the F , G , J and K functions for a dataset. The result can be plotted directly. Figure 8.20 shows the graphic generated by `plot(allstats(Swedish))`. The plots of F and J in the left column are expected to respond to interpoint interaction in the opposite sense from the plots of G and K in the right column.

8.6 Inhomogeneous F , G and J functions

[321]

8.7 Distance to another spatial pattern

8.7.1 Empty space distance for a spatial pattern

[40, 176, 41, 175]

[191, 190, 217, 216, 218]

Extract from `help(Hest)`:

The spherical contact distribution function of a stationary random set \mathbf{X} is the cumulative distribution function H of the distance from a fixed point in space to the nearest point of \mathbf{X} , given that the point lies outside \mathbf{X} . That is, $H(r)$ equals the probability that \mathbf{X} lies closer than r units away from the fixed point u , given that \mathbf{X} does not cover u .

Let $D = d(u, \mathbf{X})$ be the shortest distance from an arbitrary point u to the set \mathbf{X} . Then the spherical contact distribution function is

$$H(r) = \mathbb{P}\{D \leq r \mid D > 0\}$$

For a point process, the spherical contact distribution function is the same as the empty space function F .

The argument \mathbf{X} may be a point pattern (object of class "ppp"), a line segment pattern (object of class "psp") or a window (object of class "owin"). It is assumed to be a realisation of a stationary random set.

The algorithm first calls `distmap()` to compute the distance transform of \mathbf{X} , then computes the Kaplan-Meier and reduced-sample estimates of the cumulative distribution following Hansen *et al.* [176].

If `conditional=TRUE` (the default) the algorithm returns an estimate of the spherical contact function $H(r)$ as defined above. If `conditional=FALSE`, it instead returns an estimate of the cumulative distribution function

$$H^*(r) = \mathbb{P}\{D \leq r\}$$

which includes a jump at $r = 0$ if \mathbf{X} has nonzero area.

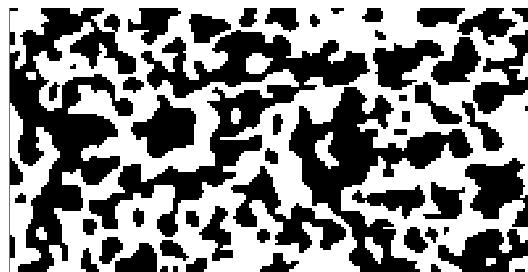


Figure 8.21: Diggle's heather data, high resolution. (**Coarse resolution image used for draft**) Land cover by *Calluna vulgaris* (black) in a 10 by 20 metre sampling plot. Image rotated by 90 degrees.

Figure 8.21 shows the spatial mosaic of vegetation of the heather plant *Calluna vulgaris* in a 10 by 20 metre sampling plot in a meadow near Jädraås, Sweden [269]. They were recorded and first analysed by Diggle [122] and subsequently studied in [284, pp. 121–122, 131–135], [169], [170, pp. 301–318], [108, pp. 763–770]. It is of interest to characterize the ‘spatial pattern’ of heather [162]. Diggle [122] fitted a ‘Boolean model’ in which the heather component is the union of discs of independent random radius centred at the points of a Poisson point process.

Several versions of this dataset are provided in `spatstat` in the object `heather`. Figure 8.21 shows the highest-resolution version, scanned from the original hand-drawn map and processed by Chris Jonker, Henk Heijmans and Adrian Baddeley at CWI, Amsterdam. The figure was generated by `plot(flipxy(heather$fine))`.

```
> HH <- with(heather, Hest(fine))
> SH <- Smooth(HH)
```

Figure 8.22 shows blah

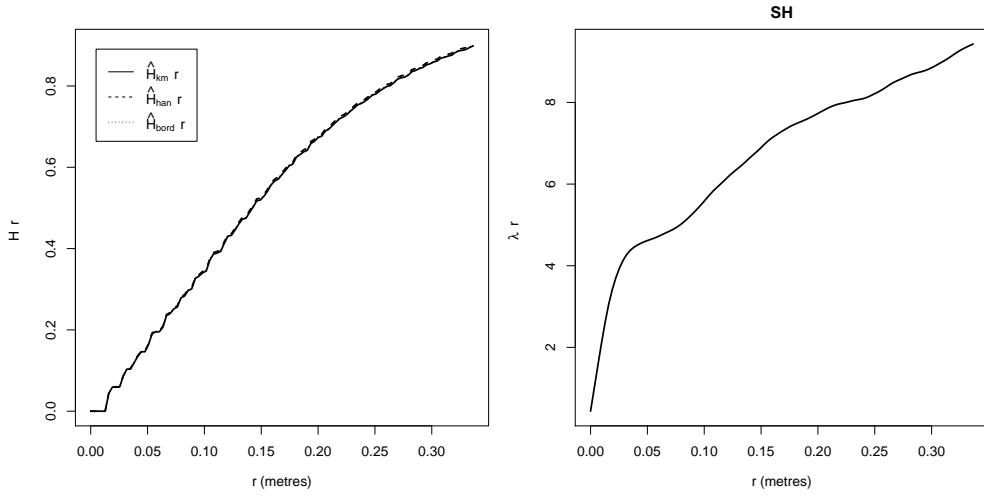


Figure 8.22: Conditional empty space function $H(r)$ (Left) and empty space hazard $h(r)$ (Right) for Diggle's heather data.

8.7.2 Distance from a point pattern to another spatial pattern

Foxall G and J function.

Extract from help(Jfox) :

Given a point pattern X and another spatial object Y , these functions compute two nonparametric measures of association between X and Y , introduced by Foxall and Baddeley [148].

Let the random variable R be the distance from a typical point of X to the object Y . Foxall's G -function is the cumulative distribution function of R :

$$G(r) = \mathbb{P}\{R \leq r\}$$

Let the random variable S be the distance from a *fixed* point in space to the object Y . The cumulative distribution function of S is the (unconditional) spherical contact distribution function

$$H(r) = \mathbb{P}\{S \leq r\}$$

which is computed by Hest.

Foxall's J -function is the ratio

$$J(r) = \frac{1 - G(r)}{1 - H(r)}$$

For further interpretation, see [148].

Figure 8.23 shows the result of

```
> GC <- with(copper, Gfox(SouthPoints, SouthLines))
> JC <- with(copper, Jfox(SouthPoints, SouthLines, eps=0.1))
```

Applications in geological prospectivity analysis [74, 148]. Simple interpretation: $J(r)$ is the intensity of copper deposits the region lying more than r units away from the nearest lineament, divided by the usual intensity. A decreasing J -function would indicate concentration of copper deposits around the lineaments. ASSUMES STATIONARITY but not very sensitive. Connected to rho-hat.

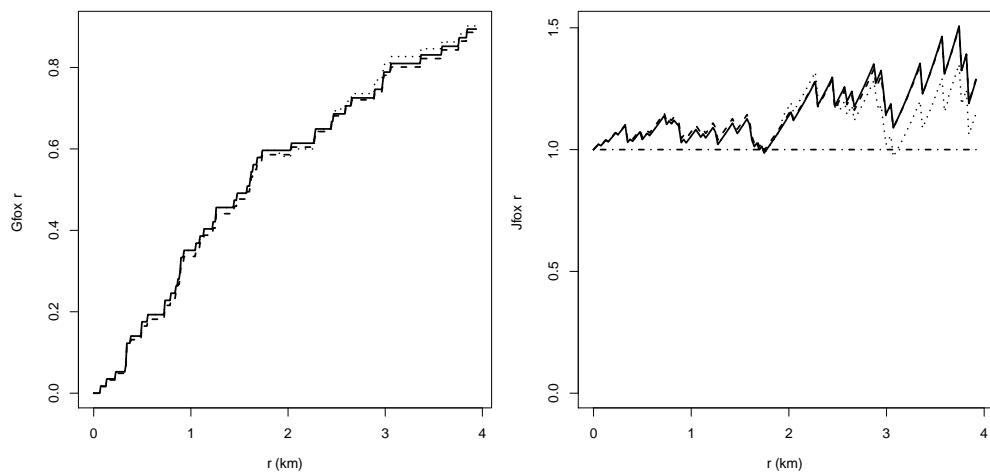


Figure 8.23: Foxall G -function (Left) and Foxall J -function (Right) from copper deposits to lineaments, in the southern half of the Queensland copper data of Figure 1.11.

8.8 Theory for edge corrections*

The estimation of $F(r)$ is hampered by edge effects, arising when we cannot see beyond the boundary of our study region: see Figure 8.24. Techniques for edge correction are needed.

Readers are warned that the software implementations of edge corrections, particularly for F , may be incorrect in some packages: trustworthy packages include `spatial`, `spatstat` and `splancs`. This section gives essential details of edge corrections in order to clarify them.

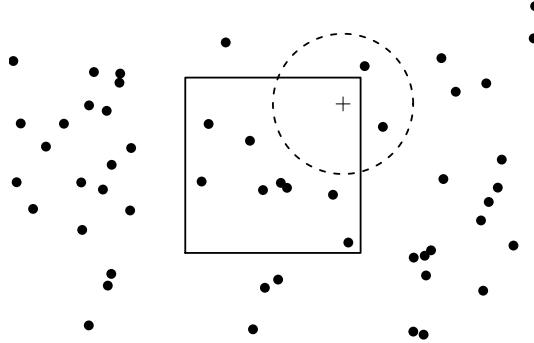


Figure 8.24: Edge effects in measuring empty space distances. Dots represent a stationary point process, which we observe only inside a window (black square). For a spatial location inside the window (+), the nearest neighbour may lie outside the window.

* Starred sections contain advanced material, and can be skipped by most readers.

8.8.1 Estimating F without edge effects

First we consider a different scenario in which edge effects do not arise. As in Section 7.4.1, visualise a sampling frame B placed in a homogeneous field of wildflowers. Taking any spatial location u inside the sampling frame, we measure the true distance $d(u, \mathbf{X})$ to the nearest wildflower, *whether it lies inside or outside the sampling frame*. Repeating this process for several spatial locations u_1, \dots, u_m we obtain the true empty space distances $d(u_j, \mathbf{X})$. The sampling locations u_j could be laid out in a grid inside B , or could be randomly-generated inside B , in any manner that is independent of \mathbf{X} . We compile the empirical cumulative distribution function

$$\hat{F}(r) = \frac{1}{m} \sum_{j=1}^m \mathbf{1}\{d(u_j, \mathbf{X}) \leq r\} \quad (8.20)$$

as a function of distance $r \geq 0$. The expectation of a sum is the sum of the expectations, so $\hat{F}(r)$ is an unbiased estimator of the true empty space function $F(r)$, for each distance r , regardless of the layout of the sampling locations u_j .

Some software packages such as `splancs` compute an estimate of $F(r)$ using relatively coarse grids of sampling locations u_j . Paradoxically, taking a finer grid does not necessarily increase the accuracy of the estimator (8.20), because there is strong positive correlation between the empty space distances [38]. However, the optimal choice of grid depends on the particular point process [292] and on the distance r . In `spatstat` we evaluate the empty space distances on a fine grid of pixels: this makes it easier to estimate related quantities such as the probability density or hazard rate of F .

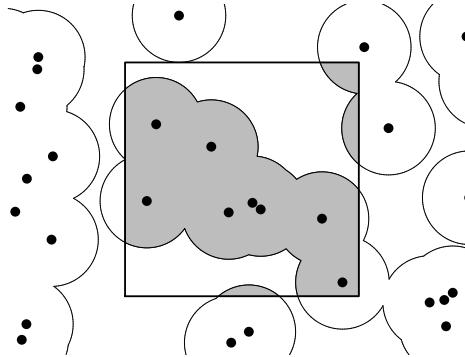


Figure 8.25: Estimation of the empty space function as the estimation of the area fraction of $\mathbf{X}_{\oplus r}$.

In geometrical terms, we saw that $F(r)$ is the average fraction of area covered by the dilation set $\mathbf{X}_{\oplus r}$, defined in equation (8.10) and sketched in Figure 8.9 on page 233. This fraction is estimated in (8.20) by visiting several test locations u_j and counting the fraction of them which are covered by $\mathbf{X}_{\oplus r}$. Taking the limit of (8.20) as the grid becomes infinitely fine, we get

$$\hat{F}(r) = \frac{1}{|B|} \int_B \mathbf{1}\{d(u, \mathbf{X}) \leq r\} du = \frac{|B \cap \mathbf{X}_{\oplus r}|}{|B|}, \quad (8.21)$$

the *fraction of area* of B occupied by the set $\mathbf{X}_{\oplus r}$. See Figure 8.25.

(In practice the integral is replaced by a discrete sum over a fine grid of pixels in W with pixel area du .)

8.8.2 Edge effects for F and G

The obvious practical problem is that if we only observe $\mathbf{X} \cap W$ whence the integrand in (??) is not observable. When u is a point close to the boundary of the window W , there may be points of \mathbf{X}

within distance r of u but lying outside of W so that they ‘can’t be seen’. More precisely, we have $d(u, \mathbf{X}) \leq r$ if and only if $n(\mathbf{X} \cap b(u, r)) > 0$. Since our data are a realisation of $\mathbf{X} \cap W$ we can only evaluate $n(\mathbf{X} \cap W \cap b(u, r))$ rather than the quantity we really need to evaluate.

Another way of putting this is to say that the window introduces a sampling bias. Recall that under the ‘standard model’ (Section ??) the point process \mathbf{X} extends throughout two dimensional space, but is observed only inside the window 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} .

It was once a common mistake to ignore this, and simply to replace \mathbf{X} by $\mathbf{X} \cap W$ in (??). This results in a negatively biased estimator of F . Call this estimator $\hat{F}_W(r)$. Since $n(\mathbf{X} \cap W \cap b(u, r)) \leq n(\mathbf{X} \cap b(u, r))$, we have

$$\mathbf{1}\{n(\mathbf{X} \cap W \cap b(u, r)) > 0\} \leq \mathbf{1}\{n(\mathbf{X} \cap b(u, r)) > 0\}$$

so that $\mathbb{E}\hat{F}_W(r) \leq F(r)$. The resulting bias is referred to as the *bias due to edge effects*. This bias can be substantial and has the potential to be misleading.

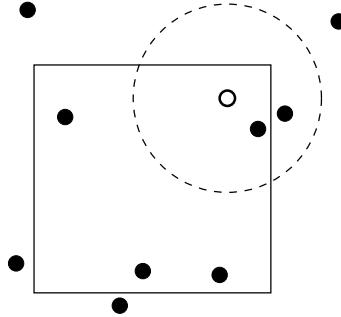


Figure 8.26: Edge effect problem for estimation of the empty space function F . If we can only observe the points of \mathbf{X} inside a window W (bold rectangle), then for some reference points u in W (open circle) it cannot be determined whether there is a point of \mathbf{X} within a distance r of u . This problem occurs if u is closer than distance r to the boundary of W .

The (raw) empirical distribution function of the observed empty space distances on a grid of locations u_j , $j = 1, \dots, m$ is

$$F^*(r) = \frac{1}{m} \sum_j \mathbf{1}\{d(u_j, \mathbf{x}) \leq r\} \quad (8.22)$$

where $d(u_j, \mathbf{x})$ is the minimum distance from the point u_j to a point of the pattern \mathbf{x} . This is a negatively biased estimator of $F(r)$, for reasons explained above.

Estimating the nearest neighbour function $G(r)$ is subject to similar difficulties. As is the case for empty space distances it is again not easy to interpret a histogram of observed nearest neighbour distances. Again the empirical distribution of the distances depends on the geometry of the window W as well as on characteristics of the point process \mathbf{X} . As before, 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} .

The (raw) empirical distribution function of the observed nearest-neighbour distances is

$$G^*(r) = \frac{1}{n(\mathbf{x})} \sum_i \mathbf{1}\{c_i \leq r\} \quad (8.23)$$

where c_i is the minimum distance from the point x_i of the observed pattern \mathbf{x} to the other points

of the pattern. (See page 221.) Again this is a negatively biased estimator of $G(r)$, for reasons we explained above.

8.8.3 Border correction for F

In most applications, a spatial point pattern is recorded inside a study region or ‘window’ W .

The border method of edge correction (Section 7.4.3) can be applied to the empty space function. When estimating the value of $F(r)$ for a particular distance r , we simply restrict attention to those test locations u for which the disc $b(u, r)$ lies entirely inside W . For such locations, the value of $d(u, \mathbf{X})$ is observed correctly, that is, $d(u, \mathbf{X} \cap W) = d(u, \mathbf{X})$. These are the locations u falling in the eroded window $W_{\ominus r}$ defined in (7.12) and sketched in Figure 7.16. In set theoretic terms

$$X_{\oplus r} \cap W_{\ominus r} = (X \cap W)_{\oplus r} \cap W_{\ominus r} \quad (8.24)$$

and the right-hand side is computable from the data $X \cap W$.

We can obtain unbiased estimators of $F(r)$ by taking $B = W_{\ominus r}$ in (8.20) or (8.21). In the first case, suppose u_1, \dots, u_m is a grid (or other arrangement) of test points in W . Let $b_j = d(u_j, W^c)$ be the distance from u_j to the boundary of the window. Then u_j belongs to $W_{\ominus r}$ if $b_j > r$. Taking $B = W_{\ominus r}$ in (8.20), the border corrected estimate of $F(r)$ is

$$\hat{F}_{bord}(r) = \frac{\sum_j \mathbf{1}\{d(u_j, \mathbf{X}) \leq r\} \mathbf{1}\{b_j > r\}}{\sum_j \mathbf{1}\{b_j > r\}} \quad (8.25)$$

the fraction of test points, amongst the test points falling in $W_{\ominus r}$, which have an empty-space distance less than or equal to r .

Similarly, taking $B = W_{\ominus r}$ in (8.21), the border corrected estimate of $F(r)$ is

$$\hat{F}_{bord}(r) = \frac{|\mathbf{X}_{\oplus r} \cap W_{\ominus r}|}{|W_{\ominus r}|} \quad (8.26)$$

the fraction of area of the eroded set $W_{\ominus r}$ covered by the dilated set $\mathbf{X}_{\oplus r}$. See Figure 8.27.

The estimator (8.25)–(8.26) is variously known as the “minus sampling” [247], “border correction” [284, chap. 3] or “reduced sample” [39] estimator, and equation (8.24) is sometimes said to be an instance of the “local knowledge principle” Serra [297, loc. cit.]. Note that F_r is only pointwise unbiased for F , and is not necessarily a distribution function.

The empty space distance can be computed for all locations u on a fine grid, using the `spatstat` function `distmap()`. The border correction can then be calculated rapidly using (8.25). However, it is statistically inefficient, because it discards a substantial amount of data. Although $\hat{F}_{bord}(r)$ is an unbiased estimator of $F(r)$ for each r , the function \hat{F}_{bord} may not be a distribution function: it may not be an increasing function of r .

8.8.4 Kaplan-Meier correction for F

[40, 176, 41, 175]

We proved in [?, Theorem 1] that for a stationary point process \mathbf{X} (in fact for any stationary random set) the empty space function F is absolutely continuous on $r > 0$, with density

$$f(r) = \frac{\mathbb{E}\text{length}(W \cap \partial(\mathbf{X}_{\oplus r}))}{\text{area}(W)} \quad (8.27)$$

for almost all $r > 0$ and any compact regular set $W \subset \mathbb{R}^2$. Thus $f(r)$ is the length density (expected length per unit area) of the boundary of $\mathbf{X}_{\oplus r}$.

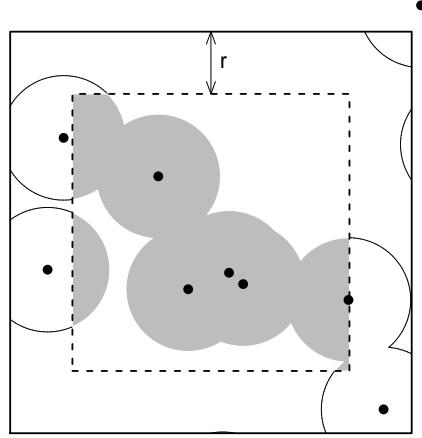


Figure 8.27: Geometry for the border correction estimator of $F(r)$.

The hazard rate of F equals

$$h(r) = \frac{\mathbb{E}\text{length}(W \cap \partial(\mathbf{X}_{\oplus r}))}{\mathbb{E}\text{area}(W \setminus \mathbf{X}_{\oplus r})}$$

for *almost every* $r > 0$.

The Kaplan-Meier estimator F_{km} of the empty space function F of \mathbf{X} , based on data $\mathbf{X} \cap W$, is

$$F_{km}(r) = 1 - \frac{\text{area}(W \setminus \mathbf{X})}{\text{area}(W)} \exp \left(- \int_0^r \frac{\text{length}(\partial(\mathbf{X}_{\oplus s}) \cap W_{\ominus s})}{\text{area}(W_{\ominus s} \setminus \mathbf{X}_{\oplus s})} ds \right). \quad (8.28)$$

Note that (8.28) is computable from the data $\mathbf{X} \cap W$ since $W \setminus \mathbf{X} = W \setminus (\mathbf{X} \cap W)$ and, by (8.24), we can replace \mathbf{X} by $\mathbf{X} \cap W$ in the numerator and denominator of the integrand.

To see that (8.28) deserves the epithet Kaplan-Meier, define for each $x \in W$

$$\begin{aligned} t(x) &= d(x, \mathbf{X} \cap W) \\ c(x) &= d(x, \partial W), \end{aligned}$$

respectively, the distance to ‘failure’ and the censoring distance. Define the ‘observed failure distance’

$$\begin{aligned} \tilde{t}(x) &= t(x) \wedge c(x) \\ &= d(x, \mathbf{X} \cap W) \wedge d(x, \partial W) \\ &= d(x, \mathbf{X}) \wedge d(x, \partial W) \end{aligned}$$

and the censoring indicator

$$\begin{aligned} d(x) &= \mathbf{1}\{t(x) \leq c(x)\} \\ &= \mathbf{1}\{d(x, \mathbf{X} \cap W) \leq d(x, \partial W)\} \\ &= \mathbf{1}\{d(x, \mathbf{X}) \leq d(x, \partial W)\} \end{aligned}$$

where in each case the last line follows by (??). Then the integrand of (8.28) has denominator

$$|W_{\ominus s} \setminus \mathbf{X}_{\oplus s}| = |\{x \in W : \tilde{t}(x) \geq s\}|,$$

the measure of the set of points ‘at risk’ at distance s , and numerator

$$\text{length}(\partial(X_{\oplus s}) \cap W_{\ominus s}) = \text{length}(\{x \in W : \tilde{t}(x) = s, d(x) = 1\}),$$

the measure of the set of points observed to ‘fail’ at distance s . Thus (8.28) is at least intuitively the analogue of the usual Kaplan-Meier estimator for the continuum of data $\{(\tilde{t}(x), c(x)) : x \in W\}$.

For an arbitrary closed set \mathbf{X} and regular compact set W , the statistic F_{km} is a distribution function (possibly defective), i.e. it is nondecreasing, right-continuous and bounded by 0 and 1. Indeed it is continuous, and absolutely continuous for $r > 0$, with hazard rate

$$h_{km}(r) = \frac{\text{length}(\partial(\mathbf{X}_{\oplus r}) \cap W_{\ominus r})}{\text{area}(W_{\ominus r} \setminus \mathbf{X}_{\oplus r})} \quad (8.29)$$

The estimator $h_{km}(r)$ of $h(r)$ is “ratio-unbiased” in the sense that $h_{km}(r) = U/V$ where U, V are such that $h(r) = \mathbb{E}U/\mathbb{E}V$.

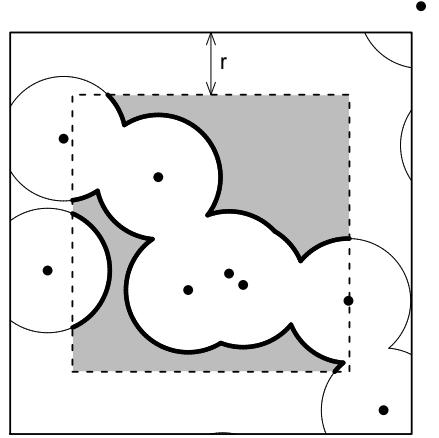


Figure 8.28: Geometry for the Kaplan-Meier estimator of $F(r)$ and its hazard rate.

In practice one would not actually compute the surface areas and volumes required in (8.28). Rather, the sampling window W will be discretised on a regular lattice (see e.g. [128]). A natural possibility is to calculate for each lattice point u_j the censored distance $d(u_j, \mathbf{X}) \wedge d(u_j, \partial W)$ and the indicator $\mathbf{1}\{d(u_j, \mathbf{X}) \leq d(u_j, \partial W)\}$. Then one would calculate the ordinary Kaplan-Meier estimator based on this finite dataset.

Our next result is that as the lattice becomes finer, the discrete Kaplan-Meier estimates converge to the ‘theoretical’ continuous estimator F_{km} .

Let $t_i = d(u_j, \mathbf{X} \cap W)$, $c_i = d(u_j, \partial W)$ and $\tilde{t}_i = t_i \wedge c_i$, $d_i = \mathbf{1}\{t_i \leq c_i\}$ be the observations at the points of $W \cap L$, where $L = \varepsilon M + b$ is a rescaled, translated copy of a fixed regular lattice M . Construct the discrete Kaplan-Meier estimator

$$F_{km}^L(r) = 1 - \prod_{s \leq r} \left(1 - \frac{\#\{i : \tilde{t}_i = s, d_i = 1\}}{\#\{i : \tilde{t}_i \geq s\}} \right) \quad (8.30)$$

and the discrete reduced-sample estimator

$$F_{rs}^L(r) = \frac{\#\{i : t_i \leq r \leq c_i\}}{\#\{i : c_i \geq r\}}. \quad (8.31)$$

Then as the lattice mesh ε converges to zero, $F_{km}^L(r) \rightarrow F_{km}(r)$ and $F_{rs}^L(r) \rightarrow F_{rs}(r)$ for any $r < R$, where

$$R = \inf\{r \geq 0 : W_{\ominus r} \cap \Phi_{\oplus r} = \emptyset\}.$$

The convergence is uniform on any compact interval in $[0, R)$.

8.8.5 Chiu-Stoyan correction for F

Using unpublished results by Baddeley and Gill [40], Chiu and Stoyan [82] developed

Cite Hanisch [173, 172]

8.8.6 Estimation for G

Likewise we can obtain an *approximately* unbiased estimator of the nearest neighbour function $G(r)$ as

$$\widehat{G}_b(r) = \frac{1}{n(\mathbf{x} \cap W_{\ominus r})} \sum_{x_i \in W_{\ominus r}} \mathbf{1}\{c_i \leq r\} \quad (8.32)$$

where c_i is the minimum distance from the point x_i of the observed pattern \mathbf{x} to the other points of the pattern.

Further discussion of edge corrections can be found in [312, 37].

NOT EDITED BEYOND THIS POINT

8.9 Conditioning*

In discussing nearest neighbour distances we had occasion to refer to a “typical point” of the process under investigation. In studying a point process we are often interested in properties relating to a “typical point”. When we say that we are dealing with a “typical point” we are in effect *conditioning* on there being a point of the process at a specified location. The event of there being a point of a process at a specified location is in general an event of probability zero and conditioning on such events is fraught with peril. (See [274].)

Dealing correctly with the problem of conditioning on events of probability zero requires the invocation of some fairly deep ideas from mathematical statistics. Applying these ideas leads to the concept of the *Palm distribution* of the point process, and the related *Campbell-Mecke formula* [312]. These tools allow us to define new characteristics of a point process, and in particular to put the definition of the nearest neighbour distance distribution function G on a sound and rigorous basis. A dual concept is the *conditional intensity* which provides many new results about point processes.

8.9.1 Motivation

The nearest neighbour distance function deals with the seemingly simple question: what is the probability distribution of the distance from a point of a process \mathbf{X} to its nearest neighbour (the nearest other point of \mathbf{X})?

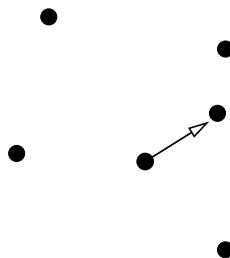


Figure 8.29: Concept of nearest neighbour distance.

As was emphasized in Section ?? the nearest neighbour is different and conceptually distinct from the ‘empty space function’ F introduced in Section ???. This latter function is the distribution of the distance $d(u, \mathbf{X})$ from a *fixed location* u to the nearest point of \mathbf{X} . In discussing the nearest neighbour function we are talking about the distance from a *point of the process* \mathbf{X} to the nearest other point of the process.

If x is known to be a point of \mathbf{X} , then the nearest neighbour distance is $R_x = d(x, \mathbf{X} \setminus x)$, and the value of $G(r)$ is the conditional probability

$$\mathbb{P}\{R_x \leq r \mid x \in \mathbf{X}\}.$$

* Starred sections contain advanced material, and can be skipped by most readers.

The problem is that this is not a conditional probability in the elementary sense, because the event $\{x \in \mathbf{X}\}$ has probability zero.

For some basic examples of point processes, this question can be resolved using classical methods.

Example 8.1. Consider the binomial process

$$\mathbf{X} = \{X_1, \dots, X_n\}$$

where n is fixed and where X_1, \dots, X_n are i.i.d. random points, uniformly distributed in $W \subset \mathbb{R}^2$.

Since X_1, \dots, X_n are exchangeable, it seems reasonable to write

$$\mathbb{P}\{R_x \leq r \mid x \in \mathbf{X}\} = \mathbb{P}\{R_x \leq r \mid X_1 = x\}$$

and use classical conditional probability for continuous random variables. Since X_1, \dots, X_n are independent,

$$(\mathbf{X} \mid X_1 = x) \equiv \mathbf{X}' \cup \{x\}$$

where

$$\mathbf{X}' = \{X_1, \dots, X_{n-1}\}$$

is the binomial process with $n - 1$ points. Thus

$$\begin{aligned} \mathbb{P}\{R_x \leq r \mid x \in \mathbf{X}\} &= 1 - \mathbb{P}\{R_x > r \mid x \in \mathbf{X}\} \\ &= 1 - \mathbb{P}\{\mathbf{X}' \cap b(x, r) = \emptyset\} \\ &= 1 - \left[\frac{\text{area}(b(x, r) \cap W)}{\text{area}(W)} \right]^{n-1}. \end{aligned}$$

For a general stationary point process \mathbf{X} , it is reasonable to address the question of interest by conditioning on the event that there is a point of \mathbf{X} at the origin 0. One simple way to define and calculate such probabilities would be to condition on the event that there is a point of \mathbf{X} in a small neighbourhood U of the origin 0, and then take the limit as U shrinks down to $\{0\}$.

Example 8.2. Suppose \mathbf{X} is a Poisson process in \mathbb{R}^2 with intensity λ . For $\varepsilon > 0$, let $U = b(0, \varepsilon)$ and define $R_{(\varepsilon)} = d(0, \mathbf{X} \setminus U)$, the distance from 0 to the nearest point of \mathbf{X} outside U . Clearly, $R_{(\varepsilon)} > r$ iff \mathbf{X} has no points in $b(0, r) \setminus U$. Since U and $b(0, r) \setminus U$ are disjoint,

$$\begin{aligned} \mathbb{P}\{R_{(\varepsilon)} > r \mid N(U) > 0\} &= \mathbb{P}\{R_{(\varepsilon)} > r\} \\ &= \exp\{-\lambda\pi(r^2 - \varepsilon^2)\}. \end{aligned}$$

As $\varepsilon \downarrow 0$, this conditional probability converges to the limit $\exp\{-\lambda\pi r^2\}$. Note also that

$$\mathbb{P}\{N(U) = 1 \mid N(U) > 0\} \rightarrow 1 \quad \text{as } \varepsilon \downarrow 0$$

so that, for small ε , we may effectively assume there is at most one point in U . Additionally, if $\mathbf{X} \cap U = \{x\}$, then

$$|d(x, \mathbf{X} \setminus x) - d(0, \mathbf{X} \setminus U)| \leq \varepsilon$$

so we have some confidence in formally writing

$$\mathbb{P}\{R_0 \leq r \mid 0 \in \mathbf{X}\} = \exp\{-\lambda\pi r^2\}.$$

8.9.2 Palm distribution

The ‘‘Palm distribution’’ formalises the concept of conditioning on a point of the process. The ideas underlying the Palm distribution were first formulated in the seminal paper by Palm [265]. These ideas were revisited, reworked and elaborated upon by several authors including Wold [342, 343], Bartlett [50], and Khinchin [201]. See Daley and Vere-Jones [113, pages 13, 14] for some discussion.

The Palm distribution \mathbb{P}^x of the point process \mathbf{X} at a location x is, intuitively speaking, the conditional probability measure given $x \in \mathbf{X}$. An elegant way to define it as follows. Let (Ω, \mathcal{A}, P) be the underlying probability space. Define the Campbell measure C on $S \times \Omega$ by

$$C(B \times A) = \mathbb{E}[N(B)1_A]$$

for all $A \in \mathcal{A}$ and $B \in \mathcal{B}(S)$, then by extension to $\mathcal{A} \otimes \mathcal{B}(S)$. Here 1_A is the indicator of the event A , equal to 1 if A occurs and equal to 0 otherwise and $\mathcal{B}(S)$ is the σ -algebra of Borel subsets of S .

Notice that

$$C(B \times A) \leq \mathbb{E}[N(B)] = v(B)$$

where v is the intensity measure of X (assumed to exist and to be locally finite).

For any fixed A , let $\mu_A(B) = C(B \times A)$ for all B . Then μ_A is a measure, and $\mu_A \leq v$, so certainly $\mu_A \ll v$. By the Radon-Nikodým Theorem,

$$\mu_A(B) = \int_B f_A(x) d\nu(x)$$

where $f_A : S \rightarrow \mathbb{R}_+$ is measurable (and unique up to equality almost everywhere). We shall interpret $f_A(x)$ as the Palm probability $\mathbb{P}^x(A)$.

Under certain conditions on (Ω, \mathcal{A}) , there exist ‘‘regular conditional probabilities’’ $\mathbb{P}^x(A)$ such that

- for all A , the function $x \mapsto \mathbb{P}^x(A)$ is a version of f_A , i.e.

$$\int_B \mathbb{P}^x(A) d\nu(x) = C(B \times A) = \mathbb{E}[N(B)1_A]$$

- for almost all x , the map $A \mapsto \mathbb{P}^x(A)$ is a probability measure on (Ω, \mathcal{A}) .

In these circumstances \mathbb{P}^x is called the **Palm distribution** of the process \mathbf{X} at the location x . We write \mathbb{E}^x for the expectation with respect to \mathbb{P}^x .

Example 8.3 (Poisson process). *Let \mathbf{X} be a uniform Poisson process in \mathbb{R}^2 . Consider the event*

$$A = \{N(K) = 0\}$$

where $K \subset \mathbb{R}^2$ is compact. For any closed U disjoint from K we have, by independence properties of the Poisson process,

$$\begin{aligned} C(U \times A) &= \mathbb{E}[N(U)1_A] \\ &= \mathbb{E}[N(U)]\mathbb{P}\{A\} \\ &= v(U)\mathbb{P}\{A\}. \end{aligned}$$

It follows that $\mathbb{P}^x(A) = \mathbb{P}\{A\}$ for almost all $x \in \mathbb{R}^2 \setminus K$. On the other hand, for $U \subseteq K$ we have $N(U) \leq N(K)$ so that

$$C(U \times A) = \mathbb{E}[N(U)1_A] = 0$$

so that $\mathbb{P}^x(A) = 0$ for almost all $x \in K$.

Now holding x fixed and varying K , and taking the complementary probabilities, we have $\mathbb{P}^x\{N(K) > 0\} = \mathbb{P}\{N(K) > 0\}$ if $x \notin K$, and $\mathbb{P}^x\{N(K) > 0\} = 1$ if $x \in K$. But this is the capacity functional of

$$\mathbf{X} \cup \{x\},$$

the Poisson process \mathbf{X} augmented by a fixed point at the location x .

In other words, under the Palm distribution \mathbb{P}^x , the process behaves as if it were a Poisson process superimposed with a fixed point at the location x .

Note that \mathbb{P}^x is a probability measure on the original space (Ω, \mathcal{A}) , giving a probability $\mathbb{P}^x A$ for any event $A \in \mathcal{A}$, and not just for events defined by the process \mathbf{X} .

Many writers consider only the Palm distribution of the point process \mathbf{X} itself, that is, the distribution \mathbf{P}^x on \mathcal{N} defined by

$$\mathbf{P}^x(A) = \mathbb{P}^x\{\mathbf{X} \in A\}$$

for $A \in \mathcal{N}$.

For the homogeneous Poisson process, we have just shown that

$$\mathbf{P}^x = \mathbf{P} * \delta_x$$

where $*$ denotes convolution (superposition of two point processes) and δ_x is the distribution of the point process consisting of a single point at x . We sometimes write \mathbf{X}^x for the process governed by the Palm distribution \mathbb{P}^x , so that the last equation can be expressed as

$$\mathbf{X}^x \equiv \mathbf{X} \cup \{x\}.$$

In fact, this property is characteristic of Poisson processes.

Theorem 8.1 (Slivnyak's Theorem). *Let \mathbf{X} be a point process with locally finite intensity measure v . Suppose*

$$\mathbf{P}^x = \mathbf{P} * \delta_x$$

Then \mathbf{X} is a Poisson process with intensity measure v .

It is often convenient to remove the point x .

Definition 8.1. *The reduced Palm distribution $\mathbf{P}^{!x}$ is the distribution of $\mathbf{X} \setminus x$ under \mathbb{P}^x :*

$$\mathbf{P}^{!x}(A) = \mathbf{P}^x\{\mathbf{X} \setminus x \in A\}$$

for $A \in \mathcal{N}$.

Thus Slivnyak's Theorem states that \mathbf{X} is a Poisson point process if and only if $\mathbf{P}^{!x} = \mathbf{P}$.

Example 8.4 (Binomial process). *Let $\mathbf{X}^{(n)}$ be the binomial process consisting of n independent random points X_1, \dots, X_n uniformly distributed in a domain W . It is easy to show that the reduced Palm distribution of $\mathbf{X}^{(n)}$ is identical to the distribution of $\mathbf{X}^{(n-1)}$.*

Example 8.5 (Mixed Poisson process). *Suppose Γ is a nonnegative real random variable defined on Ω and that, given $\Gamma = \gamma$, the point process \mathbf{X} is Poisson with intensity γ .*

The intensity measure of this mixture process is

$$\mathbb{E}[N(B)] = \mathbb{E}[\mathbb{E}[N(B) | \Gamma]] = \mathbb{E}[\Gamma] |B|.$$

Let $A = \{\Gamma \leq \gamma\}$ for some fixed $\gamma \geq 0$. Then for $B \subset \mathbb{R}^2$ we have

$$\begin{aligned} C(B \times A) &= \mathbb{E}[N(B) 1_A] \\ &= \mathbb{E}[\Gamma |B| 1_{\{\Gamma \leq \gamma\}}] \end{aligned}$$

so that

$$\mathbb{P}^x(A) = \frac{\mathbb{E}[\Gamma \mathbf{1}_{\{\Gamma \leq \gamma\}}]}{\mathbb{E}[\Gamma]}.$$

Thus, the distribution of Γ under \mathbb{P}^x is the Γ -weighted counterpart of its original distribution.

Theorem 8.2 (Campbell-Mecke formula). *For any function $Y : S \times \Omega \mapsto \mathbb{R}_+$ that is integrable with respect to the Campbell measure,*

$$\mathbb{E}\left[\sum_{x \in \mathbf{X}} Y(x)\right] = \int_S \mathbb{E}^x[Y(x)] d\nu(x) \quad (8.33)$$

In particular, if $Y(x) = f(x, \mathbf{X})$, that is, $Y(x, \omega) = f(x, \mathbf{X}(\omega))$, we get

$$\mathbb{E}\left[\sum_{x \in \mathbf{X}} f(x, \mathbf{X})\right] = \int_S \mathbb{E}^x[f(x, \mathbf{X})] d\nu(x). \quad (8.34)$$

8.9.3 Palm distribution for stationary processes

Lemma 8.1. *If \mathbf{X} is a stationary point process in \mathbb{R}^2 , then*

$$\mathbf{X}^x \equiv \mathbf{X}^0 + x$$

where \mathbf{X}^x again denotes a process governed by the Palm distribution \mathbb{P}^x . Equivalently,

$$\mathbf{P}^x = T_{-x} \mathbf{P}^0 \quad (8.35)$$

where T_{-x} denotes the effect of translation by the vector $-x$, that is, $\mathbf{P}^x(A) = \mathbf{P}^0\{X - x \in A\}$.

Proof. Apply the Campbell-Mecke formula to functions of the form

$$f(x, \mathbf{X}) = \mathbf{1}\{x \in B\} \mathbf{1}\{\mathbf{X} - x \in A\}$$

where $A \in \mathcal{N}$ is an event, $B \subset \mathbb{R}^2$, and $\mathbf{X} - x = \mathbf{X} + (-x)$ is the result of shifting \mathbf{X} by the vector $-x$. This yields

$$\mathbb{E}\left[\sum_{x \in \mathbf{X} \cap B} \mathbf{1}\{\mathbf{X} - x \in A\}\right] = \lambda \int_B \mathbb{P}^x\{\mathbf{X} - x \in A\} dx.$$

Since \mathbf{X} is stationary, \mathbf{X} has the same distribution as $\mathbf{X} + v$ for any vector v , so

$$\begin{aligned} \mathbb{E}\left[\sum_{x \in \mathbf{X} \cap B} \mathbf{1}\{\mathbf{X} - x \in A\}\right] &= \mathbb{E}\left[\sum_{x \in (\mathbf{X} + v) \cap B} \mathbf{1}\{(\mathbf{X} + v) - x \in A\}\right] \\ &= \mathbb{E}\left[\sum_{x \in \mathbf{X} \cap T_{-v} B} \mathbf{1}\{\mathbf{X} - x \in A\}\right]. \end{aligned}$$

Thus

$$\lambda \int_B \mathbb{P}^x\{\mathbf{X} - x \in A\} dx = \lambda \int_{T_{-v} B} \mathbb{P}^x\{\mathbf{X} - x \in A\} dx$$

which implies that for all B

$$\int_B \mathbb{P}^x\{\mathbf{X} - x \in A\} dx = c \text{ area}(B)$$

for some constant c , and hence that $\mathbb{P}^x \mathbf{X} - x \in A$ is constant. This proves (8.35). \square

One way to interpret this result is to construct a marked point process Y on \mathbb{R}^2 with marks in \mathfrak{N} by attaching to each point $x \in \mathbf{X}$ the mark $\mathbf{X} - x$. That is, the mark attached to the point x is a copy of the entire realisation of the point process, translated so that x is shifted to the origin. The result shows that this is a *stationary* marked point process. Hence the intensity measure of Y factorises,

$$v(B \times A) = \lambda |B| Q(A)$$

for $B \subset \mathbb{R}^2$, $A \in \mathcal{N}$ where Q is the ‘mark distribution’. Clearly Q can be interpreted as the Palm distribution given there is a point at 0. That is, $\mathbf{P}^0 = Q$, and $\mathbf{P}^x = T_{-x}Q$.

This gives us a direct interpretation of the Palm distribution \mathbf{P}^0 (but not \mathbb{P}^0) for a stationary point process in \mathbb{R}^2 . We have

$$\mathbb{E} \left[\sum_{x \in B} \mathbf{1}_{\{\mathbf{X} - x \in A\}} \right] = \lambda |B| \mathbf{P}^0(A) \quad (8.36)$$

for $B \subset \mathbb{R}^2$, $A \in \mathcal{N}$. Thus

$$\mathbf{P}^0(A) = \frac{\mathbb{E} [\sum_{x \in B} \mathbf{1}_{\{\mathbf{X} - x \in A\}}]}{\mathbb{E} N(B)} \quad (8.37)$$

for all $B \subset \mathbb{R}^2$ such that $0 < |B| < \infty$. On the right side of (8.37), the denominator is the expected number of terms in the numerator, so we can interpret $\mathbf{P}^0(A)$ as the ‘average’ fraction of points x satisfying $\mathbf{X} - x \in A$.

8.9.4 Conditional intensity

Return for a moment to the heuristic definition of the Palm probability $\mathbb{P}^x(A)$ as the limit of $\mathbb{P}\{A \mid N(U) > 0\}$ as $U \downarrow \{x\}$, where U is an open neighbourhood of x in \mathbb{R}^2 , and A is an event in \mathfrak{N} . Applying Bayes’ Theorem

$$\mathbb{P}\{N(U) > 0 \mid A\} = \frac{\mathbb{P}\{N(U) > 0\}}{\mathbb{P}\{A\}} \mathbb{P}\{A \mid N(U) > 0\}$$

so that, as $U \downarrow \{x\}$,

$$\frac{\mathbb{P}\{N(U) > 0 \mid A\}}{\mathbb{P}\{N(U) > 0\}} \rightarrow \frac{\mathbb{P}^x(A)}{\mathbb{P}\{A\}}.$$

Suppose \mathbf{X} has an intensity function $\lambda(u)$, $u \in \mathbb{R}^2$ which is continuous at x . Then asymptotically

$$\mathbb{P}\{N(U) > 0\} \sim \mathbb{E}[N(U)] = \int_U \lambda(u) du \sim \lambda(x)|U|$$

so that

$$\frac{\mathbb{P}\{N(U) > 0 \mid A\}}{|U|} \rightarrow \lambda(x) \frac{\mathbb{P}^x(A)}{\mathbb{P}\{A\}}. \quad (8.38)$$

Since we can also write

$$\frac{\mathbb{P}\{N(U) > 0\}}{|U|} \rightarrow \lambda(x),$$

the left side of (8.38) can be interpreted as a ‘conditional’ analogue of the intensity $\lambda(x)$ given the event A .

This motivates the following definitions.

Definition 8.2. Let \mathbf{X} be a point process on a space S . The reduced Campbell measure of \mathbf{X} is the measure $C^!$ on $S \times \mathfrak{N}$ such that

$$C^![B \times A] = \mathbb{E} \left[\sum_{x \in \mathbf{X}} \mathbf{1}_B(x) \mathbf{1}_A(\mathbf{X} \setminus x) \right]$$

for $B \subset \mathbb{R}^2$ and $A \in \mathcal{N}$.

Definition 8.3. Let \mathbf{X} be a point process on \mathbb{R}^2 , and suppose its reduced Campbell measure $C^!$ is absolutely continuous with respect to $\lambda_d \otimes \mathbf{P}$ (where \mathbf{P} is the distribution of \mathbf{X}).

Then the Radon-Nikodým derivative $\lambda^* : \mathbb{R}^2 \times \mathcal{N} \rightarrow \mathbb{R}_+$ of $C^!$ with respect to $\lambda_d \otimes \mathbf{P}$ is called the **conditional intensity** of \mathbf{X} . It is defined to satisfy

$$C^![B \times A] = \int_B \mathbb{E}[\lambda^*(u, \mathbf{X}) \mathbf{1}\{\mathbf{X} \in A\}] du \quad (8.39)$$

for $B \subset \mathbb{R}^2$ and $A \in \mathcal{N}$.

If \mathbf{X} has a conditional intensity then, by extension of the last equation, for any integrable $g : \mathbb{R}^2 \times \mathcal{N} \rightarrow \mathbb{R}_+$,

$$\mathbb{E}\left[\sum_{x \in \mathbf{X}} g(x, \mathbf{X} \setminus x)\right] = \int_{\mathbb{R}^2} \mathbb{E}[\lambda^*(x, \mathbf{X}) g(x, \mathbf{X})] dx. \quad (8.40)$$

If \mathbf{X} has an intensity function $\lambda(u)$, $u \in \mathbb{R}^2$, then the Campbell-Mecke formula gives

$$\mathbb{E}\left[\sum_{x \in \mathbf{X}} g(x, \mathbf{X} \setminus x)\right] = \int_{\mathbb{R}^2} \mathbb{E}^{!x}[g(x, \mathbf{X})] \lambda(x) dx \quad (8.41)$$

writing $\mathbb{E}^{!x}$ for the expectation with respect to $\mathbf{P}^{!x}$. Comparing the right sides of (8.40) and (8.41) shows that, for almost all $x \in \mathbb{R}^2$,

$$\mathbb{E}^{!x}[g(x, \mathbf{X})] = \mathbb{E}\left[\frac{\lambda^*(x, \mathbf{X})}{\lambda(x)} g(x, \mathbf{X})\right]. \quad (8.42)$$

Thus, $\lambda^*(x, \mathbf{X})/\lambda(x)$ is the Radon-Nikodým density of $\mathbf{P}^{!x}$ with respect to \mathbf{P} .

Example 8.6. If \mathbf{X} is a Poisson process on \mathbb{R}^2 with intensity function $\lambda(x)$, then we have $\mathbf{P}^{!x} = \mathbf{P}$ for all x , so $\lambda^*(x, \mathbf{X})/\lambda(x)$ is identically equal to 1, and the conditional intensity is $\lambda^*(x, \mathbf{X}) = \lambda(x)$.

Our heuristic argument says that

$$\lambda^*(x, \mathbf{X}) dx = \mathbb{P}\{N(dx) > 0 \mid \mathbf{X} \setminus x\};$$

that is, roughly speaking, $\lambda^*(x, \mathbf{X}) dx$ is the conditional probability that there will be a point of \mathbf{X} in an infinitesimal neighbourhood of x , given the location of all points of \mathbf{X} outside this neighbourhood.

Example 8.7. The binomial process $\mathbf{X}^{(n)}$ consists of n independent random points, uniformly distributed in a domain W . We saw above that the reduced Palm distribution of $\mathbf{X}^{(n)}$ is identical to the distribution of $\mathbf{X}^{(n-1)}$. In this case, $\mathbf{P}^{!x}$ and \mathbf{P} are mutually singular (since, for example, the event that there are exactly n points in the process has probability 1 under \mathbf{P} and has probability 0 under $\mathbf{P}^{!x}$). Hence, this process does not have a conditional intensity.

Example 8.8. The randomly translated grid in \mathbb{R}^2 was mentioned in Section 8.5. Intuitively, if we know that there is a point of the grid at the location x , this determines the position of the entire grid. The Palm distribution \mathbf{P}^x for this process is completely deterministic: with probability one, \mathbf{X}^x consists of points at the locations $x + (ks, ms)$ for all integers k, m . This can be proved using (8.37).

It follows that \mathbf{P}^x is not absolutely continuous with respect to \mathbf{P} , so this process does not possess a conditional intensity.

Example 8.9. The J -function of a stationary process can be written explicitly in terms of the con-

ditional intensity:

$$\begin{aligned}
 J(r) &= \frac{\mathbb{P}^0\{d(0, \mathbf{X} \setminus 0) > r\}}{\mathbb{P}\{d(0, \mathbf{X}) > r\}} \\
 &= \frac{\mathbf{P}^{10}\{d(0, \mathbf{X}) > r\}}{\mathbb{P}\{d(0, \mathbf{X}) > r\}} \\
 &= \frac{\mathbb{E}\left[\frac{\lambda^*(0, \mathbf{X})}{\lambda(0)} \mathbf{1}\{d(0, \mathbf{X}) > r\}\right]}{\mathbb{P}\{d(0, \mathbf{X}) > r\}} \\
 &= \mathbb{E}\left[\frac{\lambda^*(0, \mathbf{X})}{\lambda(0)} \mid d(0, \mathbf{X}) > r\right].
 \end{aligned}$$

This representation can often be evaluated, while F and G often cannot be evaluated explicitly.

8.10 FAQ's

- A plot of the G -function for my data suggests the pattern is regular, but the plot of the K -function suggests clustering. Which is right?

The results from G and K are not contradictory, because a summary function does not completely characterise the point process. The G and K functions are sensitive to different aspects of the point process. The G -function summarises information at a shorter scale than the K -function. If you are confident that the intensity is constant, the most likely explanation is that there is both short-scale regularity and larger-scale clustering.

- I can't seem to control the range of r values in `plot(Gest(X))`. How can I control it? How is the default plotting range determined?

To control the range of r values, use the argument `xlim` as in `plot(Gest(X), xlim=c(0, 7))`. See `help(plot.fv)`.

The default range of r values that is plotted depends on the ‘default plotting range’ of the object (of class “fv”) returned by `Gest()`.

- How are the r values determined in `Gest(X)`?

The steps to determine the range of distances are:

- Calculate a predetermined maximum distance to be considered. This is `rmax`. It is determined by `rmax.rule()` and it depends on both the intensity λ and the window geometry.
- Compute the relevant distances (up to `rmax`) and calculate the estimates.
- Look at the estimated function values and decide on the recommended range.

The ‘recommended range’ of an “fv” object is the attribute `alim`. For the F , G and J functions this is determined at the very last moment after the function has been computed, by looking at the tail. In `Fest()` the penultimate line is

```
> attr(Z, "alim") <- with(Z, range(.x[is.finite(.y) & .y <= 0.9]))
```

Since `.x` stands for the function argument and `.y` for the function estimate, this says that the recommended range is the range of r values for which the estimated $F(r)$ is less than or equal to 0.9.

This ensures that the 'interesting' part of the curve $F(r)$ occupies most of the plot region. If we didn't do this, then the graph of $F(r)$ would often look bad.

For `Gest()` the recommended range is the range where $G(r)$ is less than 0.9, which is of course different from the range for F (and different between different datasets).

First `rmax.rule` is called using intensity of the point pattern.

```
> rmaxdefault <- rmax.rule("F", as.owin(amacrine), intensity(unmark(amacrine)))
> rmaxdefault
```

Then `handle.r.b.args()` is called with `rmaxdefault = rmaxdefault`. Since '`r`' is `NULL`, this calls '`make.even.breaks()`' which generates a sequence of '`r`' values with a stepsizef '`eps`' determined by the geometry of the window. This results in a slightly higher value of '`rmax`'.

```
breaks <- handle.handle.r.b.args(r, breaks, dwin, eps, rmaxdefault=rmaxdefault)
breaks$rmax
```

— SUBSEQUENT SOFTWARE UPDATES —

This is a list of relevant changes to `spatstat` since the last version of the workshop notes in 2010. These changes may not yet have been covered in the text of this chapter, if the text was cut-and-pasted from existing sources. Chapter authors should try to merge this information into the chapter text. **DELETE ITEMS FROM THIS LIST IF THEY ARE COVERED IN THE CHAPTER TEXT.**

- Inhomogeneous F, G and J functions.
- `Finhom()`, `Ginhom()`, `Jinhom()` Inhomogeneous versions of the F, G and J functions
- `Gmulti()`, `Jmulti()`: The arguments `I`, `J` can now be any kind of subset index or can be functions that yield a subset index.
- `nndist()`, `nnwhich()`: New argument `by` makes it possible to find nearest neighbours belonging to specified subsets in a point pattern, for example, the nearest neighbour of each type in a multitype point pattern.
- `nncorr()`, `nnmean()`, `nnvario()`: These functions now handle data frames of marks.
- `distcdf()`: Cumulative distribution function of the distance between two independent random points in a given window.
- `nnfun.ppp()`, `distfun.ppp()`: New argument `k` allows these functions to compute k -th nearest neighbours.
- `minnndist()`, `maxnndist()`: Faster ways to compute `min(nndist(X))`, `max(nndist(X))`

Part III

STATISTICAL INFERENCE



9

Poisson models

This is the first chapter on (explicit) statistical modelling and inference for point pattern data.

Sections 9.1 and 9.2 explain the relevance and application of Poisson point process models. Section 9.3 explains how to fit Poisson models to point pattern data in `spatstat`, and Section 9.4 shows how to conduct formal inference including prediction, confidence intervals and hypothesis tests.

Sections 9.5–9.8 are “starred” because they contain advanced material. Section 9.5 gives basic theory of maximum likelihood estimation for Poisson models. Sections 9.6, 9.7 and 9.8 describe different techniques for model fitting.

Section 9.9 explains how to handle models which are not in the standard loglinear form. Frequently Asked Questions are answered in Section 9.10.

9.1 Introduction

Poisson point processes were introduced in Chapter 5. In this chapter we use Poisson point processes as statistical models for point pattern data. We show how to formulate a Poisson model appropriate to the data, fit the model to the data, interpret the results, and identify weaknesses in the analysis.

The key property of a Poisson process is that the random points are *independent* of each other. This property greatly simplifies the statistical analysis, and gives access to a wide variety of powerful statistical techniques.

A Poisson process is completely described by its intensity function $\lambda(u)$. Statistical models of Poisson point processes are easy to build: they are simply models of the intensity, that is, a model is just an equation stating the form of the function $\lambda(u)$. This also means that there is a close relationship between techniques for estimating an intensity function (Chapter 6) and methods for fitting a Poisson point process model.

Why Poisson?

The huge literature on spatial analysis contains many other techniques for analysing spatial point pattern data. Many of these techniques appear at first sight to have nothing to do with Poisson point processes — and even seem to have nothing to do with point processes at all.

In fact, several popular techniques are *equivalent* to fitting a Poisson point process model with a specific form of the intensity — namely, where the intensity is a loglinear function of the parameters. Such techniques include logistic regression [1, 319, 294, 179, 210, 65] and maximum entropy [136, 135, 140, 270].

Logistic regression is commonly used in GIS to predict the probability of occurrence of mineral deposits [1, 84], archaeological finds [294, 210], landslides [85, 158] and other events which can

be treated as points at the scale of interest. The study region is divided into pixels; in each pixel the presence or absence of any data points is recorded; then logistic regression [243, 130, 187] is used to predict the probability of the presence of a point as a function of predictor variables.

Logistic regression implicitly assumes that the presence/absence values for different pixels are independent [243, 130, 187]. This implies that the underlying point process is Poisson (as explained in Chapter 5). Furthermore, the *logistic* relationship implies that the intensity of the Poisson process is a *loglinear* function of the covariates and the parameters. Thus, logistic regression is essentially equivalent to fitting a Poisson point process model with *loglinear* intensity [20, 334].

The principle of maximum entropy [136] is often used in ecology, for example, to study the influence of habitat variables on the spatial distribution of animals or plants [135, 140, 270]. Conceptually we consider all possible spatial distributions, and find the spatial distribution which maximises a quantity called entropy, subject to constraints implied by the observed data. Surprisingly, the solution is a probability density which is a *loglinear* function of the covariates. An analogy could be drawn with the stretching of a string: a string may take on any shape, but if we demand that the string be stretched as tight as possible, it will take up a straight line. Thus, again, this analysis principle is equivalent to fitting a Poisson point process model with *loglinear* intensity [280].

Advantages of statistical modelling

Statistical modelling is a much more powerful way to analyse data than simply computing summary statistics. Formulating a statistical model, and fitting it to the data, allows us to adjust for effects that might otherwise distort the analysis (such as uneven distribution of survey effort, and spatially-varying population density) by including terms that represent these effects. By fitting different models that include or omit a particular term, and comparing the models, we can decide which variables have a statistically significant influence on the intensity.

A great advantage of statistical modelling is that the assumptions of the analysis are openly stated, rather than implicitly imposed. All model assumptions are “on the table” and are amenable to criticism. Chapter 11 provides tools for criticising each individual model assumption of a Poisson point process model, and for identifying weaknesses of the data analysis.

9.2 Poisson point process models

The Poisson point process was introduced in Sections ?? and ?? of Chapter 5. Here we give more detail about the Poisson process as a statistical model for data analysis.

9.2.1 Characteristic Properties of the Poisson Process

We start with a detailed statement of the characteristic properties of the Poisson point process. Apart from clarifying our understanding of the model, this is a useful checklist of properties that need to be verified before the analyst can be satisfied that a real point pattern dataset is well-described by a Poisson point process.

The *homogeneous Poisson process* with intensity $\lambda > 0$ (also called “*Complete Spatial Randomness*”, CSR) has the properties that

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

(PP2) homogeneous intensity: the expected number of points falling in B is $\mathbb{E}[n(\mathbf{X}_B)] = \lambda \cdot \text{area}(B)$;

(PP3) independence: if B_1, B_2, \dots are disjoint regions of space then $n(\mathbf{X}_{B_1}), n(\mathbf{X}_{B_2}), \dots$ are independent random variables;

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

We are justified in calling λ the “intensity” because, by virtue of (PP2), this point process has homogeneous intensity λ in the sense of Section 6.2. The parameter λ has dimension length $^{-2}$ and determines the average number of points per unit area. Scaling properties of λ (for example, the effect of changing the unit of length) are discussed in Section 6.2. Any value can be specified for λ as long as it is non-negative and finite. If $\lambda = 0$ then all realisations of the point process are empty.

The properties (PP1)–(PP4) provide the basis for many statistical techniques. For example, in quadrat counting, (PP3) implies that the counts in different quadrats are independent random variables; (PP1) implies that these counts have Poisson distributions; and (PP2) implies that the mean values of the counts are proportional to the areas of the quadrats. If the quadrats have equal area, then the quadrat counts are Poisson random variables with equal mean values, so they are identically distributed.

When we need to check whether a point process is CSR, the most important properties to check are independence (PP3) and homogeneity (PP2). As explained in Section ??, the other properties follow logically from these, under reasonable conditions.¹ However, if any one of these properties does not hold, then the process is not CSR. Statistical methods for checking CSR can focus, for example, on detecting a departure from the Poisson distribution (PP1) or from conditional independence and uniformity (PP4).

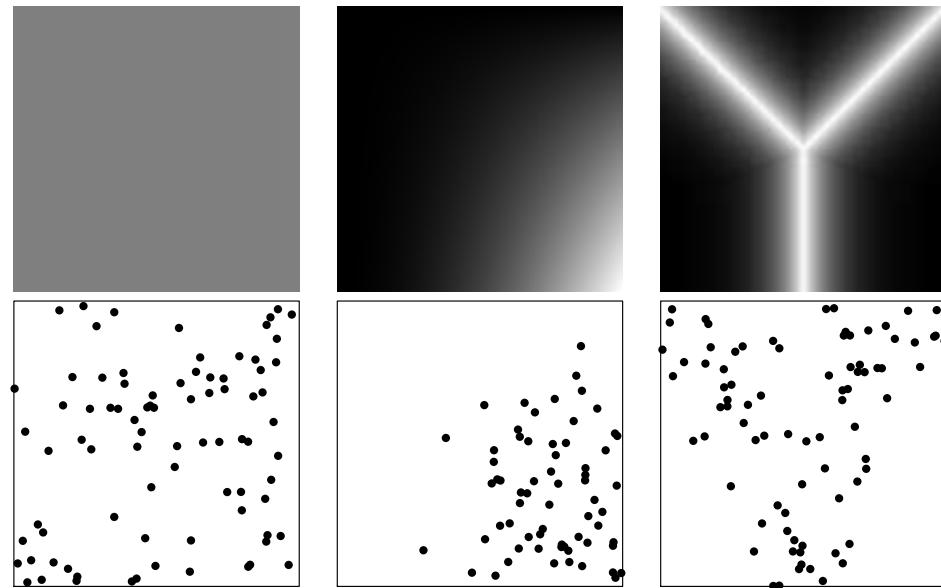


Figure 9.1: Examples of intensity functions (top) and simulated realisations (bottom) for Poisson processes.

¹The reasonable condition is that the probability of *more than one* random point falling in a region B , divided by the area of B , falls to zero as the area of B goes to zero. This excludes coincident points and fractal-like behaviour.

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 $n(\mathbf{X}_B)$ of points falling in a region B has expected value

$$\mathbb{E}[n(\mathbf{X}_B)] = \int_B \lambda(u) du. \quad (9.1)$$

(PP4'): given that $n(\mathbf{X}_B) = n$, the n points are independent and identically distributed, with common probability density

$$f(u) = \frac{\lambda(u)}{I} \quad (9.2)$$

where $I = \int_B \lambda(u) du$.

Again we are justified in calling $\lambda(u)$ the “intensity function” because, by virtue of (PP2’), it is the intensity function in the sense of Section 6.3. Its values have dimension length⁻² and determine the spatially-varying intensity (average number of points per unit area).

The intensity function encapsulates both the abundance of points (by equation (9.1)) and the spatial distribution of individual point locations (by equation (9.2)).

Any function $\lambda(u)$ can be used, as long as its values are non-negative and *integrable* (the right-hand side of (9.1) must be finite).

9.2.2 Fine-pixel limit

A useful way to think about statistical methodology for point processes, and to derive new methodology, is to suppose that space is divided into small squares or “pixels” of equal area a , and to count the number of random points falling in each pixel. See Figure 9.2.

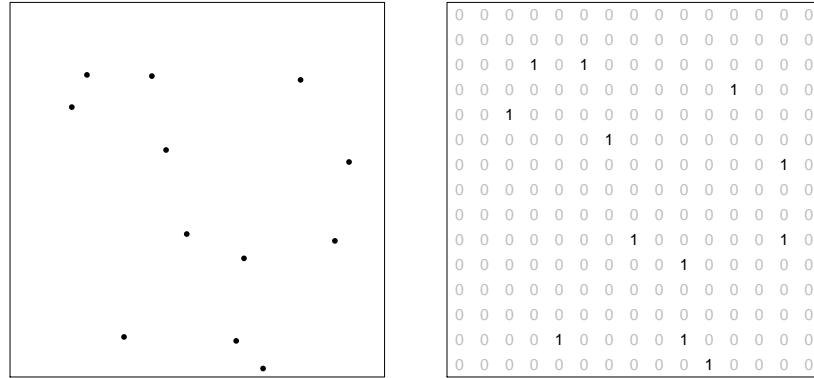


Figure 9.2: Fine-pixel limit. When space is divided into tiny pixels, a Poisson point process corresponds to assigning independent random values 0 or 1 to each pixel.

Suppose the point process is CSR. Property (PP3) implies that the counts in different pixels are independent random variables. Properties (PP1) and (PP2) tell us that the count in each pixel has a Poisson distribution with mean $\mu = \lambda a$ where again a is the pixel area. The probability of

getting no points in a pixel is $\mathbb{P}\{N = 0\} = e^{-\mu} = e^{-\lambda a}$. The probability of exactly one point is $\mathbb{P}\{N = 1\} = \mu e^{-\mu} = \lambda a e^{-\lambda a}$. The probability of more than one point is $\mathbb{P}\{N > 1\} = 1 - e^{-\mu} - \mu e^{-\mu} = 1 - (1 + \lambda a)e^{-\lambda a}$. When the pixel area a is small, we can approximate $e^{-\lambda a} \approx 1 - \lambda a$ so that $\mathbb{P}\{N = 0\} \approx 1 - \lambda a$ and $\mathbb{P}\{N = 1\} \approx \lambda a(1 - \lambda a) \approx \lambda a$ while $\mathbb{P}\{N > 1\} \approx 0$. That is, if space is divided into very fine pixels, the homogeneous Poisson process corresponds to a system of independent random variables, associating the values 0 or 1 with each pixel, with probability $p = \lambda a$ for getting a 1 in each pixel.

Explain interpretation of inhomogeneous Poisson process in terms of independent pixel presence/absence indicators.

9.2.3 Scenarios giving rise to a Poisson process

There are many realistic models of physical processes which give rise to a Poisson point process. We sketch some common examples below.

random trials on a very fine grid: Imagine a radioactive material, consisting of N atoms arranged in a regular grid, where N is very large, and the spacing between atoms is tiny. In a specified time interval, each atom has a very small probability p of undergoing random decay. Decay events of different atoms are independent. The total number of decay events in the specified period is the number of successes in N trials with success probability p . Since N is large and p is small, the number of decay events has a Poisson distribution (to a very good approximation) and the locations of the decay events approximately constitute a Poisson point process.

random thinning: Suppose that there is some initial population represented by a point process \mathbf{X} (not assumed to be a Poisson process) and that each point of \mathbf{X} is either deleted or retained, independently of other points. Suppose \mathbf{X} has intensity function $\beta(u)$ and the probability of retaining a point at the location u is $p(u)$. The resulting process \mathbf{Y} of retained points has intensity $\lambda(u) = p(u)\beta(u)$. If the original population density $\beta(u)$ is large and the retention probabilities $p(u)$ are small, then the resulting process \mathbf{Y} is approximately an inhomogeneous Poisson process.

For example, a model of plant propagation could assume that seeds are randomly dispersed according to some point 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 well described by an inhomogeneous Poisson process.

In spatial epidemiology, \mathbf{X} could represent the human population at risk of a rare disease, and \mathbf{Y} would be the disease cases. The key assumption is that the disease outcomes for different people are statistically independent.

Near-accidents and real accidents

Example where points are not always observed (eg astronomy)

random strewing: Suppose a large number N of points is scattered randomly in a large region D . Assume each point is uniformly distributed over D , and different points are statistically independent. If this random pattern is observed within a subregion W , where D is much larger than W , then the pattern is approximately a Poisson point process inside W .

random displacement: Again we imagine an initial population represented by any point process \mathbf{X} . Suppose that each point in \mathbf{X} is subjected to a random displacement, independently of other points. A point at the original location x_i is moved to the new location $y_i = x_i + V_i$ where V_i is a random vector. Assume that V_1, V_2, \dots are independent random vectors. Then under reasonable conditions, the resulting point process \mathbf{Y} is approximately an inhomogeneous Poisson process.

random birth-and-death: Consider a forest which evolves over time. In each small interval of

time Δt , each existing tree has probability $m\Delta t$ of dying, independently of other trees, where m is the ‘mortality rate’ per tree per unit time. In the same time interval, in any small region of area a , a new tree germinates with probability $ga\Delta t$, independently of any existing trees, where g is the ‘germination rate’ per unit area per unit time. No matter what the initial state of the forest, over long time scales this “spatial birth-and-death process” reaches an equilibrium in which any snapshot of the forest (the pattern of trees in existence at a fixed time) is a realisation of a Poisson process with intensity g/m .

9.2.4 Operations which preserve the Poisson property

There are several operations which, if applied to a Poisson process, yield another Poisson process.

random thinning of Poisson: suppose that a *Poisson* process with intensity $\beta(u)$ 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 exactly an inhomogeneous Poisson process, with intensity $\lambda(u) = p(u)\beta(u)$.

independent superposition: State independent marking property (needed for logistic likelihood)

transformation:

random displacement:

9.2.5 Common parametric models of intensity

The intensity function $\lambda(u)$ of the Poisson process can be assumed to take any form that is appropriate to the context, so long as the intensity values are non-negative and the integral of the intensity over the observation window is finite. Some common models of the intensity function, and their interpretation, are discussed below.

homogeneous intensity: the Poisson process with constant intensity $\lambda(u) \equiv \lambda$ is the simplest model, called the homogeneous Poisson process or “complete spatial randomness” (CSR).

homogeneous in different regions: Space is partitioned into non-overlapping regions B_1, \dots, B_m . Within region B_j , the process is Poisson with intensity β_j , where β_1, \dots, β_m are parameters of the model. Then the overall process is an inhomogeneous Poisson process where the intensity function $\lambda(u)$ takes the value β_j when u is in region B_j . See Figure 9.3.

intensity proportional to baseline:

$$\lambda(u) = \theta b(u) \quad (9.3)$$

where $b(u)$ is a known function (‘baseline’) and θ is an unknown parameter that must be estimated. Explain connection with “constant risk” model where b is density of population at risk. (Connection with thinning -- note that effect of thinning is multiplicative.)

exponential function of covariate:

$$\lambda(u) = \kappa e^{\beta Z(u)} = \exp(\alpha + \beta Z(u)) \quad (9.4)$$

where $Z(u)$ is a covariate. Example: prospectivity analysis, $Z(u)$ is distance to nearest fault. Explain why it’s exponential and not linear (e.g. to ensure positivity). Interpret parameters $\kappa = e^\alpha$ and β . Mention connection to Berman test.

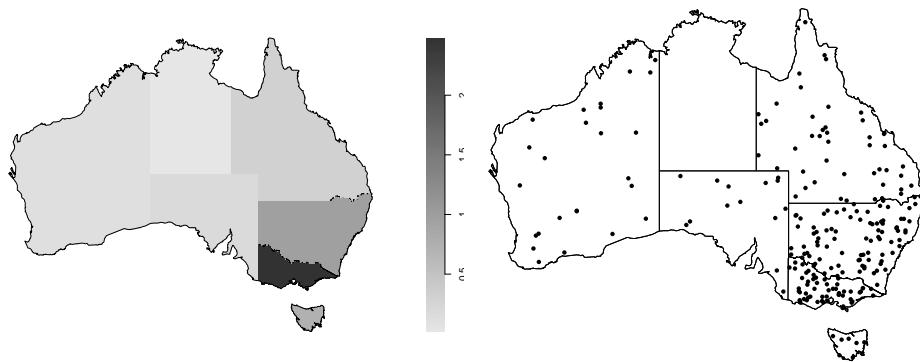


Figure 9.3: Intensity (*Left*) and simulated realisation (*Right*) of the Poisson process with a different, homogeneous intensity in each Australian state and territory. Intensity is proportional to average population density within the state or territory.

raised incidence model: A combination of the last two models is

$$\lambda(u) = b(u)e^{\alpha + \beta Z(u)} \quad (9.5)$$

where $b(u)$ is a known baseline function and $Z(u)$ is a spatial covariate. Explain about “raised incidence”. Refer to Diggle’s raised incidence models which are not loglinear.

loglinear model:

$$\lambda_\theta(u) = \exp(B(u) + \boldsymbol{\theta}^\top \mathbf{Z}(u)) = \exp(B(u) + \theta_1 Z_1(u) + \theta_2 Z_2(u) + \dots + \theta_p Z_p(u)) \quad (9.6)$$

where $B(u)$ and $Z_1(u), \dots, Z_p(u)$ are known functions, and we write $\mathbf{Z}(u) = (Z_1(u), \dots, Z_p(u))$ for the vector-valued function. Termed a *modulated* Poisson process by Cox [99].

Note that we often build models by multiplying terms which represent different effects. Hence a loglinear model is natural.

Note that we can include a term in the model that accounts for the probability of observing a point (eg. related to survey effort or visibility covariate)

9.3 Fitting Poisson models in spatstat

Fitting a statistical model to data means choosing values for the parameters governing the model, so that the model fits the data as well as possible. For example, to fit a line $y = ax + b$ through a scatter of data points, we find the values of the parameters a (slope) and b (intercept) that give the best-fitting line.

To fit a Poisson point process model to a point pattern dataset, we would specify the form of the intensity function — perhaps using one of the common models described above — and find the values of the parameters which best fit the point pattern dataset. The definition of “best fit” and the technical details of model-fitting are described in Sections 9.5–9.8. This section explains how to use `spatstat` to fit Poisson models to point pattern data in practice.

9.3.1 The ppm() function

Syntax

Fitting a Poisson point process model to a point pattern dataset is performed in `spatstat` by the function `ppm()` (for **point process model**). This is closely analogous to the standard model fitting functions in R such as `lm()` (for fitting linear models) and `g1m()` (for fitting generalized linear models). The essential syntax for specifying a Poisson point process model to be fitted by `ppm()` is

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

where the first argument “`X ~ trend`” is a `formula` in the R language.

The left hand side of the formula gives the name of the point pattern dataset `X`. Alternatively the left hand side may be an expression which can be evaluated to yield a point pattern.

The right hand side of the model formula specifies the form of the *logarithm* of the intensity function for the model. Thus, any *loglinear* intensity model (9.6) can be fitted.²

Simplest example

The simplest possible model for a Poisson planar point process is the constant intensity or homogeneous model, equivalent to Complete Spatial Randomness. A call to `ppm()` to fit this model employs the simplest possible formula, `X ~ 1`. We illustrate this using the *Beilschmiedia* data which are available in `spatstat` as the “`ppp`” object `bei`. This is a point pattern consisting of the locations of trees of the species *Beilschmiedia*.

```
> fit <- ppm(bei ~ 1)
> fit
Stationary Poisson process
Intensity: 0.007208
      Estimate   S.E. CI95.lo CI95.hi Ztest   Zval
log(lambda) -4.933 0.01666 -4.965    -4.9 *** -296.1
```

The result of `ppm()` is a fitted point process model, an object belonging to the class “`ppm`”. There are many facilities for handling such objects. For instance there is a print method `print.ppm()`, which produced the output above. This output tells us that the fitted intensity was $\lambda = 0.007208$ trees per square metre. For the *logarithm* of the intensity, the output gives an estimate, a standard error, and a 95% confidence interval. The fitted intensity can be recovered from this line of output by $\exp(-4.933) = 0.007208$.

Variables in a formula

A “variable name” in a formula is a name that plays the role of a variable rather than a function. For example, in the formula `Y ~ f(X)`, the symbols `X` and `Y` are recognised by R as playing the role of “variables”, while the symbol `f` is recognised as playing the role of a function.

In a call to `ppm()` of the form

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

all the variable names appearing in the formula should be the names of existing objects in the R session.

An alternative is to use the form

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

²Fitting Poisson models which are *not* loglinear is discussed in Section 9.9.

where the argument `data` is a list containing data needed to fit the model. In this case, for each variable name in the formula, `ppm()` will first try to find a matching entry in the list `data`; if it is not found, then `ppm()` will look for existing objects in the R session.

The left-hand side of the model formula should be either the name of a point pattern, or an expression which can be evaluated to yield a point pattern. On the left side of the formula, mathematical operators such as `+`, `-`, `*`, `/` and `^` have their usual mathematical meaning.

The right-hand side of the model formula is an expression involving the names of spatial covariates which affect the intensity of the point process. On the right-hand side of a model formula, the operators `+`, `-`, `*`, `/` and `^` have a special interpretation: they are model operators, which are used to combine terms in a model, as explained below.

The variable names `x` and `y` are reserved names which represent the Cartesian coordinates. Any other variable name in the formula should match the name of an existing object in the R session, or the name of an entry in the argument `data` if it is given. Each object should be either:

- a pixel image (object of class "im") giving the values of a spatial covariate at a fine grid of locations.
- a function which can be evaluated at any location (x, y) to obtain the value of the spatial covariate. It should be a `function(x, y, ...)` in the R language.
- a window (object of class "owin") which will be interpreted as a logical variable which is TRUE inside the window and FALSE outside it.
- a tessellation (object of class "tess") which will be interpreted as a factor covariate. For each spatial location, the factor value indicates which tile of the tessellation contains the location in question.
- a single number, indicating a covariate that is constant in this dataset.

In what follows we explain how to use the `ppm()` function and elaborate upon the use of formulas introduced in section 2.1.10. We illustrate the ideas by examples using point pattern datasets supplied with the `spatstat` package.

9.3.2 Models with a single numerical covariate

Recall that the interpretation of a formula depends on the types of variables involved: for example, numerical variables and categorical variables are interpreted differently. We start by dealing with models involving a single numerical covariate and proceed from there to more complicated models. Given a numerical covariate (predictor) Z , say, we could use a formula such as $X \sim Z$ to specify the model. This formula indicates that the intensity is a loglinear function of Z as given by (9.4).

9.3.2.1 The *Beilschmedia* data set

We illustrate the fitting procedure using the *Beilschmedia* data (the `spatstat` data set `bei`) previously referred to. These data are accompanied by auxiliary data which are bundled up in the list `bei.extra`. The entries of this list are numerical covariates presented as *images* (objects of class "im"). Explicitly these covariates are the terrain elevation `elev` and terrain slope (gradient) `grad`.

```
> bei.extra
elev :
  real-valued pixel image
  101 x 201 pixel array (ny, nx)
  enclosing rectangle: [-2.5, 1002.5] x [-2.5, 502.5] metres
```

```
grad :
real-valued pixel image
101 x 201 pixel array (ny, nx)
enclosing rectangle: [-2.5, 1002.5] x [-2.5, 502.5] metres
```

The following command fits a Poisson point process model in which the intensity of *Beilschmiedia* trees is a *loglinear* function of terrain slope:

```
> fit <- ppm(besi ~ grad, data=besi.extra)
> fit
Nonstationary Poisson process
Trend formula: ~grad
Fitted trend coefficients:
(Intercept)      grad
-5.391        5.022
```

The model fitted by the commands above is a Poisson point process with intensity

$$\lambda(u) = \exp(\beta_0 + \beta_1 s(u)) \quad (9.7)$$

where $s(u)$ is the terrain slope at location u . The argument to the `exp` function (in this case $\beta_0 + \beta_1 s(u)$) is referred to as the “*linear predictor*”. The component $\beta_1 s(u)$ is called the “*effect*” of the covariate s .

The printout above³ includes the fitted coefficients marked by the labels `(Intercept)` and `grad`. These are the maximum likelihood estimates of β_0 and β_1 respectively, the coefficients of the linear predictor, so the fitted model is

$$\lambda(u) = \exp(-5.391 + 5.022 s(u)). \quad (9.8)$$

These results tell us that the estimated intensity of *Beilschmiedia* trees on a flat surface (slope $s = 0$) is about $\exp(-5.391) = 0.004559$ trees per square metre, or 45.59 trees per hectare, and would increase by a factor of $\exp(5.022) = 151.7$ if the slope increased to 1.0. The largest slope value in the data is about 0.3, at which stage the predicted intensity has risen by a factor of $\exp(0.3 \times 5.022) = 4.511$ from its value on a flat surface.

Although it is possible to read off the fitted model from the printout and calculate intensity levels for various values of the predictors, this can be tedious for more than one or two values and becomes intricate when the model is more complicated. The generic R command `predict()` (explained in Section 9.4.3) has a method for point process models, which can be used to calculate fitted intensities and other properties of the model. In this simple model, the fitted intensity can also be plotted as a function of a covariate (in the current example, the covariate `grad`) using the `spatstat` utility `effectfun()`, as shown in the left panel of Figure 9.4.

It should be clearly understood that the formula `besi ~ grad` does not represent a flexible model in which the abundance of trees depends, in some unspecified way, on the terrain slope. On the contrary, the dependence is very tightly specified: the formula `besi ~ grad` corresponds to the *loglinear* relationship (9.7), that is, it corresponds to assuming intensity is exponentially increasing or decreasing as a function of terrain slope.

If a loglinear relationship is *not* an appropriate assumption, but the intensity is believed to depend on terrain slope somehow, several strategies are available.

One strategy is to transform the slope values and fit a loglinear relationship to the transformed values. Transformations of the original covariates can be specified directly in the model formula.

³To save space on the printed page, we have set `options(digits=4)` so that numbers are printed to four significant digits for the rest of this chapter. We have also set `spatstat.options(terse=2)` to shorten the output.

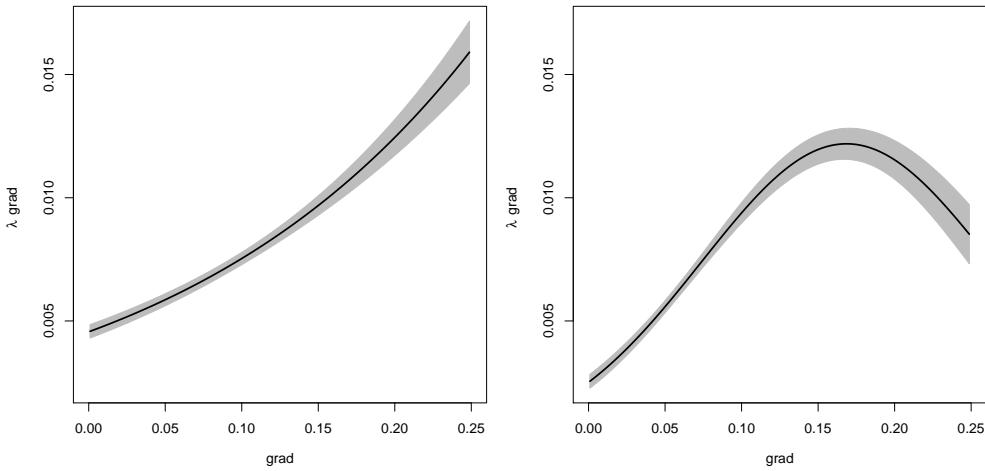


Figure 9.4: Fitted intensity of *Beilschmiedia* trees as a function of slope. Solid line: maximum likelihood estimate. Shading: pointwise 95% confidence interval. Generated by `plot(effectfun(fit, "grad", se.fit=TRUE))`. Left: `fit` is a "ppm" object containing the loglinear model (9.7). Right: `fit` is a "ppm" object containing the log-quadratic model (9.9).

For example, the covariate `grad` gives the gradient as the *tangent* of the angle of inclination (vertical elevation divided by horizontal displacement) so that a gradient of 1.0 corresponds to a 45-degree incline. If we think it is more appropriate to use the *angle* of inclination as the covariate in the loglinear model, we can convert the gradient values to angles using the arctangent, simply by typing

```
> ppm(bei ~ atan(grad), data=bei.extra)
```

Any mathematical function can appear in a model formula. Care however is required with expressions that involve the symbols `+`, `-`, `*`, `/` and `^`, which have special syntactic meanings in a formula, except inside a function call.

For example, `atan` calculates angles in radians; to convert them to degrees, we would need to multiply by $180/\pi$, so the covariate should be `atan(grad) * 180/pi`. To prevent the mathematical symbols `*` and `/` from being interpreted as model formula operators, this expression should be enclosed inside the function `I()` (`I` for 'identity'):

```
> ppm(bei ~ I(atan(grad) * 180/pi), data=bei.extra)
```

The function `I` returns its argument unchanged, and the rules for interpreting a formula ensure that the expression inside the parentheses is evaluated using the usual mathematical operations. The command above fits a loglinear model depending on the angle of inclination, expressed in degrees. It may be neater to write a separate R function which can then appear in the model formula:

```
> degrees <- function(x) { x * 180/pi }
> ppm(bei ~ degrees(atan(grad)), data=bei.extra)
```

Another strategy is to replace the first-order function in (9.7) by a quadratic function,

$$\lambda(u) = \exp(\beta_0 + \beta_1 s(u) + \beta_2 s(u)^2), \quad (9.9)$$

which results in a model which is *log-quadratic* as a function of slope. Here there are three parameters, β_0 , β_1 and β_2 to be estimated. This model can be fitted by specifying the `slope` and `slope^2` terms in the model formula:

```
> ppm(b ei ~ grad + I(grad^2), data=bei.extra)
Nonstationary Poisson process
Trend formula: ~grad + I(grad^2)
Fitted trend coefficients:
(Intercept)      grad    I(grad^2)
-5.987        18.745     -55.602
```

An advantage of this approach is that there is a formal mechanism for deciding (in statistical terms) whether the grad^2 term is needed, namely a test of the null hypothesis $H_0 : \beta_2 = 0$. We discuss this in Section 9.4. Higher order polynomials can be fitted in the same way. The command `effectfun()` can be used to visualise the fitted intensity: in this context `effectfun(fit, "grad")` would show the combined effect (linear and quadratic terms together) of the slope variable as shown in the right panel of Figure 9.4.

From a theoretical viewpoint, all the models fitted above are “loglinear” models, of the general form (9.6). It is useful to distinguish between “*original covariates*” (e.g. `grad` in the current example) and “*canonical covariates*”. The term “*canonical covariates*” refers to the variables Z_j which appear in the loglinear model (9.6). Canonical covariates may be transformations and combinations of the original covariates which were provided in the dataset.

The Murchison data

We continue to illustrate models involving a single numerical covariate. Our next example uses the Murchison geological survey data, shown in Figure 9.6.

The following introduction to the Murchison data should eventually be moved to its first occurrence in the book.

The Murchison geological survey data shown in Figure 9.5 record the spatial locations of gold deposits and associated geological features in the Murchison area of Western Australia. They are extracted from a large scale (1:500,000) study of the Murchison area by the Geological Survey of Western Australia [335]. The features recorded are the locations of gold deposits (point pattern `gold`); the locations of geological faults (line segment pattern `faults`); and the region that contains greenstone bedrock (window `greenstone`). The study region is contained in a 330×400 kilometre rectangle. At this scale, gold deposits are point-like, i.e. their spatial extent is negligible. Gold deposits in this survey occur only in greenstone bedrock, but the data may violate this rule, because the gold deposits lie below the surface, while the greenstone and faults were mapped only where they intersect the surface. Geological faults can be observed reliably only within the greenstone bedrock at the surface, but some faults have been extrapolated (by geological “interpretation”) outside the greenstone boundary.

These data were analysed in [148, 74]; see also [163, 203]. The main aim is to predict the intensity of the point pattern of gold deposits from the more easily observable fault pattern. End of moveable introduction.

Alternative short intro These data consist of the spatial locations of gold deposits (`gold`), a line segment pattern of geological faults (`faults`), and the polygonal boundary of greenstone outcrop (`greenstone`) in a 330 by 400 km survey region. End of alternative short intro

A 70 by 40 km detail from the middle right side of the survey is shown in Figure 9.7.

```
> mur <- lapply(murchison, rescale, s=1000, unitname="km")
```

A common model⁴ for such data is a Poisson process with intensity which is a loglinear function

⁴This is equivalent to the model customarily fitted in GIS software by logistic regression of presence-absence indicators on distance to nearest fault.



Figure 9.5: Murchison geological survey data. Gold deposits (+), geological faults (—) and green-stone outcrop (grey shading) in a survey region about 200 by 300 km across.

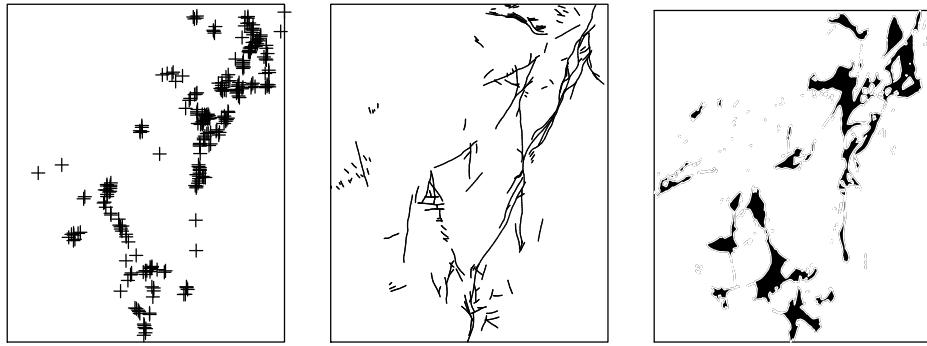


Figure 9.6: (**Alternative to previous Figure**) Murchison geological survey data, split into layers. *Left:* gold deposits. *Middle:* geological faults. *Right:* greenstone outcrop. Survey region about 200 by 300 km across.

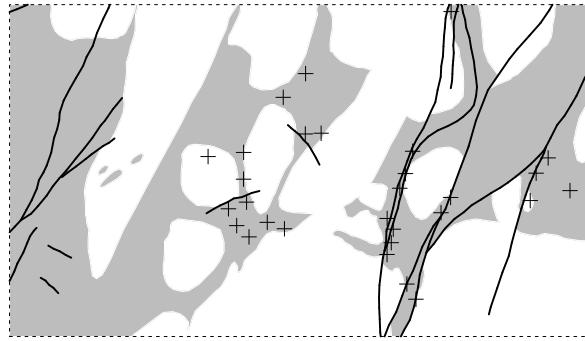


Figure 9.7: Detail of Murchison geological survey data in a 70 by 40 km region near the abandoned town of Reedy. Gold deposits (+), faults (solid lines) and greenstone outcrop (grey shading).

of distance to the nearest fault,

$$\lambda(u) = \exp(\beta_0 + \beta_1 d(u)) \quad (9.10)$$

where β_0, β_1 are parameters and $d(u)$ is the distance from location u to the nearest geological fault.

We compute the distance covariate $d(u)$ using `distfun()`, which has the advantage that distances will be computed exactly by analytic geometry. An alternative would be to compute the distance values at a grid of pixel locations using `distmap()`.

```
> dfault <- with(mur,distfun(faults))
```

Next we fit the loglinear model (9.10):

```
> fit <- ppm(gold ~ dfault,data=mur)
> fit
Nonstationary Poisson process
Trend formula: ~dfault
Fitted trend coefficients:
(Intercept) dfault
-4.3413     -0.2608
```

The fitted model is

$$\lambda(u) = \exp(-4.341 - 0.2608d(u)). \quad (9.11)$$

That is, the estimated intensity of gold deposits in the immediate vicinity of a geological fault is about $\exp(-4.341) = 0.01302$ deposits per square kilometre or 1.302 deposits per 100 square kilometres, and decreases by a *factor* of $\exp(-0.2608) = 0.7704$ for every additional kilometre away from a fault. At a distance of 10 kilometres, the intensity has fallen by a factor of $\exp(10 \times (-0.2608)) = 0.07368$ to $\exp(-4.341 + 10 \times (-0.2608)) = 0.0009596$ deposits per square kilometre. Figure 9.8 shows the effect of the distance covariate on the intensity function.

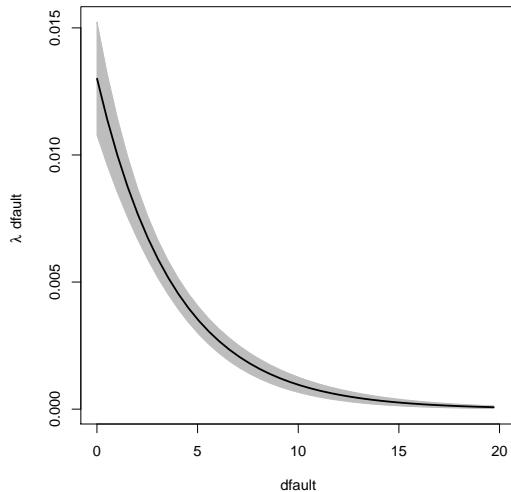


Figure 9.8: Fitted intensity of Murchison gold deposits as a function of distance to the nearest fault, assuming it is a loglinear function of distance (equivalent to logistic regression). Solid line: maximum likelihood estimate. Shading: pointwise 95% confidence interval. Generated by `plot(effectfun(fit, "dfault", se.fit=TRUE))`.

9.3.3 Models with a logical covariate

A logical covariate takes the values TRUE and FALSE. Logical covariates can arise in many ways, as we shall see in this chapter.

The Murchison data include the polygonal boundary of greenstone outcrop, given as a window called `greenstone`. Setting aside the information about geological faults for a moment, we could fit a simple Poisson model in which the intensity of gold deposits is constant inside the greenstone, and constant outside the greenstone, but with different intensity values inside and outside.

```
> ppm(gold ~ greenstone, data=mur)
Nonstationary Poisson process

Trend formula: ~greenstone

Fitted trend coefficients:
(Intercept) greenstoneTRUE
-8.103          3.980
```

	Estimate	S.E.	CI95.lo	CI95.hi	Ztest	Zval
(Intercept)	-8.103	0.1667	-8.430	-7.777	***	-48.62
greenstoneTRUE	3.980	0.1798	3.628	4.333	***	22.13

The function `ppm()` recognises that `greenstone` is a spatial window, and interprets it as a covariate with *logical* values, equal to `TRUE` inside the `greenstone` outcrop and `FALSE` outside.

Since the model formula does not explicitly exclude a constant term ('intercept'), this term is assumed to be present. The model fitted is

$$\lambda(u) = \exp(\alpha + \beta G(u)) \quad (9.12)$$

where G is the indicator covariate for the `greenstone`, taking the value $G(u) = 1$ if the location u is inside the `greenstone` outcrop, and $G(u) = 0$ if it is outside. The printed output shows the estimates for the parameters α (labelled `Intercept`) and β (labelled `greenstoneTRUE`) as -8.103 and 3.98 respectively. The estimated intensities are $\lambda_{\text{out}} = \exp(-8.103) = 0.0003026$ outside the `greenstone` outcrop and $\lambda_{\text{in}} = \exp(-8.103 + 3.98) = 0.0162$ inside. The last few lines of output give the estimates of α and β together with estimates of standard error, 95% confidence intervals, and a report that the coefficient β is significantly different from 0.

Slightly different output is obtained with the command

```
> ppm(gold ~ greenstone-1, data=mur)
Nonstationary Poisson process

Trend formula: ~greenstone - 1

Fitted trend coefficients:
greenstoneFALSE greenstoneTRUE
-8.103          -4.123

Estimate      S.E. CI95.lo CI95.hi Ztest   Zval
greenstoneFALSE -8.103 0.16667 -8.430 -7.777 *** -48.62
greenstoneTRUE    -4.123 0.06757 -4.255 -3.990 *** -61.01
```

The model formula now includes `-1` forbidding the use of a constant term ('intercept'). The model fitted here is

$$\lambda(u) = \exp(\gamma G_{\text{out}}(u) + \delta G_{\text{in}}(u)) \quad (9.13)$$

where $G_{\text{in}}(u) = G(u)$ is the indicator of the `greenstone` as before, and $G_{\text{out}}(u) = 1 - G(u)$ is the indicator of the complement of the `greenstone`. The estimates of γ and δ are -8.103 and -4.123 respectively. The estimates of the intensities are $\lambda_{\text{out}} = \exp(-8.103) = 0.0003026$ outside the `greenstone` outcrop and $\lambda_{\text{in}} = \exp(-4.123) = 0.0162$ inside. The two fitted models agree, but are parametrised differently. The last few lines of output give standard errors and confidence intervals for the parameters γ and δ , and indicate that these parameters are significantly different *from zero*. The first parameterisation (i.e. with an intercept included) is more useful for deciding whether the `greenstone` has an effect on the intensity of gold deposits, while the second parameterisation is more useful for directly reading off the estimated intensities.

The foregoing discussion depends upon the default "treatment contrasts" being used, as explained below.

9.3.4 Models with a factor covariate

A spatial covariate with *categorical* could arise from a map which divides the survey domain into regions of different type, according to criteria such as rock type, vegetation cover, land use, administrative region, socioeconomic level, government building zone type, or anatomical subdivision of

tissue. The map itself is a spatial tessellation of the survey domain. The associated spatial covariate $J(u)$ tells us which type or category applies to each given location u , so that J is a spatial covariate with categorical values.

In some cases the original data will be in ‘vector’ form, giving the spatial coordinates of the boundaries of the regions of each type. To preserve accuracy, vector data should not be discretised. The boundary polygons should be converted to a tessellation (object of class “`tess`”) which can then be passed directly as a covariate to `ppm()`.

If the categorical data are provided as a pixel image, it is important to ensure that the pixel values are recognised as categorical values. Printing or summarising the image in question should indicate that it is `factor`-valued.

A numerical covariate Z can also be converted into a factor by dividing the range of values into a few bands, and treating each band of values as a category. This is often a useful exploratory tool, as we saw in Section 6.6.2.

9.3.4.1 Gorilla nest data

The gorillas dataset comes from a study [151] of gorillas in the Kagwene Gorilla Sanctuary, Cameroon, by the Wildlife Conservation Society Takamanda-Mone Landscape Project (WCS-TMLP). A detailed description and analysis of the data is reported in [152]. The data were kindly contributed into `spatstat` by Dr Funwi-Gabga Neba. The collaboration of Prof Jorge Mateu is gratefully acknowledged.

The data include the marked point pattern of gorilla nest sites `gorillas`, and auxiliary data in a list `gorillas.extra` containing seven pixel images of spatial covariates. We rescale the data to kilometres:

```
> gor <- rescale(gorillas, 1000, unitname="km")
> gor <- unmark(gor)
> gex <- lapply(gorillas.extra, rescale,
+                 s=1000, unitname="km")
```

Figure 9.9 shows the spatial locations of the (unmarked) gorilla nests, and the covariate `vegetation` which reports the vegetation or cover type.

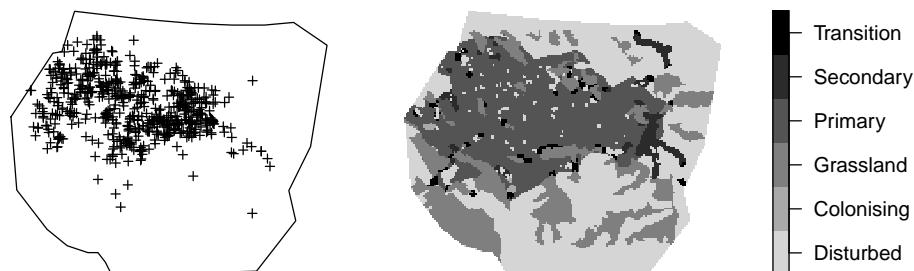


Figure 9.9: Gorillas data. *Left:* gorilla nest locations. *Right:* vegetation type.

To save space on the printed page, we will abbreviate the names of the covariates, and the names of the levels of factors. We use `substr()` to extract the first four characters of each name; alternatively we could have used `abbreviate()` to convert the names to shorthand.

```
> names(gex)
```

```
[1] "aspect"      "elevation"   "heat"          "slopeangle"  "slopetype"
[6] "vegetation" "waterdist"
> shorten <- function(x) substr(x, 1, 4)
> names(gex) <- shorten(names(gex))
> names(gex)
[1] "aspe" "elev" "heat" "slop" "slop" "vege" "wate"
> isfactor <- !unlist(lapply(lapply(gex, levels), is.null))
> for(i in which(isfactor))
  levels(gex[[i]]) <- shorten(levels(gex[[i]]))
> levels(gex$vege)
[1] "Dist" "Colo" "Gras" "Prim" "Seco" "Tran"
```

The following command fits a simple Poisson model in which the intensity is constant in each vegetation type, but may depend on vegetation type:

```
> ppm(gor ~ vege, data=gex)
Nonstationary Poisson process

Trend formula: ~vege

Fitted trend coefficients:
(Intercept)    vegeColo     vegeGras     vegePrim     vegeSeco     vegeTran
           2.3213      2.0842     -0.7719      2.1174      1.1367      1.6163

Estimate    S.E. CI95.lo CI95.hi Ztest   Zval
(Intercept) 2.3213 0.1060  2.1135  2.5291 *** 21.899
vegeColo    2.0842 0.5870  0.9337  3.2347 ***  3.551
vegeGras   -0.7719 0.2475 -1.2569 -0.2869 ** -3.119
vegePrim    2.1174 0.1151  1.8918  2.3430 *** 18.396
vegeSeco    1.1367 0.2426  0.6612  1.6122 ***  4.685
vegeTran    1.6163 0.2647  1.0976  2.1351 ***  6.107
```

For models which involve a factor covariate, the interpretation of the fitted coefficients depends on which *contrasts* are in force, as specified by `getOption("contrasts")`. We explain more about this below. By default the “treatment contrasts” are assumed. This means that — if an intercept term is present — the coefficient associated with the first level of the factor is taken to be zero, and the coefficients associated with the other levels are effectively differences relative to the first level. In the model above, the fitted log intensities for each vegetation type are calculated as follows:

TYPE	RELEVANT COEFFICIENTS	LOG INTENSITY
Disturbed	(Intercept)	2.321
Colonising	(Intercept) + vegeColo	2.321 + 2.084
Grassland	(Intercept) + vegeGras	2.321 - 0.7719
Primary	(Intercept) + vegePrim	2.321 + 2.117
Secondary	(Intercept) + vegeSeco	2.321 + 1.137
Transition	(Intercept) + vegeTran	2.321 + 1.616

Rather than relying on such interpretations, it would be prudent to use the command `predict()` to compute predicted values of the model, as explained in Section 9.4.1 below.

Another way to fit the same model is to remove the intercept, so that a single coefficient will be associated with each level of the factor.

```

> fitveg <- ppm(gor ~ vege - 1, data=gex)
> fitveg
Nonstationary Poisson process

Trend formula: ~vege - 1

Fitted trend coefficients:
vegeDist vegeColo vegeGras vegePrim vegeSeco vegeTran
  2.321    4.406    1.549    4.439    3.458    3.938

Estimate      S.E. CI95.lo CI95.hi Ztest   Zval
vegeDist     2.321 0.10600   2.114   2.529 *** 21.899
vegeColo     4.406 0.57735   3.274   5.537 *** 7.631
vegeGras     1.549 0.22361   1.111   1.988 *** 6.929
vegePrim     4.439 0.04486   4.351   4.527 *** 98.953
vegeSeco     3.458 0.21822   3.030   3.886 *** 15.846
vegeTran     3.938 0.24254   3.462   4.413 *** 16.235
> exp(coef(fitveg))
vegeDist vegeColo vegeGras vegePrim vegeSeco vegeTran
  10.189   81.900    4.709   84.662   31.752   51.298

```

For this simple model, where intensity is constant inside each region defined by vegetation type, an equivalent way to fit the same model is to estimate the intensities using quadrat counts. To check this, we convert the pixel image to a tessellation and apply quadrat counting:

```

> vt <- tess(image=gex$vege)
> intensity(quadratcount(gor, tess=vt))
tile
  Dist   Colo   Gras   Prim   Seco   Tran
 10.201 69.155  4.781 84.012 32.651 50.922

```

The slight discrepancies are due to discretisation effects.

9.3.4.2 Factor effects and contrasts

To fully understand the results obtained for the gorillas data above, we need to know more about the handling of factors by the model-fitting code in R. If f is a factor then $X \sim f$ specifies a point process model in which the intensity depends on the level of f . The model is

$$\lambda(u) = \exp(\mu + \alpha_{J(u)}) \quad (9.14)$$

where $J(u)$ is the level of the factor f at the location u . The parameters of this model are the *intercept* μ and the *effects* $\alpha_1, \dots, \alpha_L$ of the different factor levels. Here we are assuming the factor has L different levels numbered 1 to L . For the i th level of the factor, the corresponding value of the intensity is $e^{\mu + \alpha_i}$.

While the model (??) is conceptually useful, it has the practical drawback that it is *over-parameterised*. If there are L different levels of the factor, then there are $L + 1$ parameters to be estimated from data, but only L different values of the intensity on the right hand side of (??). In order to fit the model we need to reduce the number of parameters by 1.

One option is to remove the intercept parameter μ . The model is then

$$\lambda(u) = \exp(\alpha_{J(u)}) \quad (9.15)$$

so that the intensity value for the i th level of the factor is e^{α_i} . There are now only L parameters, the effects $\alpha_1, \dots, \alpha_L$ of the factor levels. The model `ppm(gor ~ vege - 1)` has this form, so the fitted coefficients are the fitted estimates of the log intensity for each type of vegetation.

The default behaviour in R is to retain the intercept and instead to constrain the effect of the first level to be zero, $\alpha_1 = 0$. This convention is called the “*treatment contrast*”; it makes sense when the first level of the factor is a baseline or control, while the other levels are different possible “treatments” that could be applied to a subject in an experiment. The model (9.14) then states that the intensity value for the first level is e^μ , while the intensity value for another level i is $e^{\mu+\alpha_i}$, so the coefficient α_i is the *difference* in effect between level i and the first level. The table on page 282 shows this calculation. An advantage of the treatment contrast is that we are often interested in whether there is any difference between the intensities associated with different factor levels: this can be assessed by testing the hypothesis $\alpha_i = 0$ for each $i > 1$.

In its linear modelling procedures, the commercial statistics package SAS® constrains α_L to be 0. A commonly used constraint is $\alpha_1 + \dots + \alpha_L = 0$. In R one can choose the constraint that is used by specifying the *contrast* or linear expression that will be set to zero. This is done using the `options()` function, with a call of the form:

```
options(contrasts=c(arg1,arg2))
```

where `arg1`, `arg2` are strings which specify the contrasts to be used with “unordered” and “ordered” factors respectively. All the factors discussed in this book are “unordered”. Note that if you do set the contrasts you have to specify both `arg1` and `arg2` even though you only care about `arg1`.

The possible values for either argument are the character strings “`contr.sum`”, “`contr.poly`”, “`contr.helmert`”, “`contr.treatment`” and “`contr.SAS`”. The default for `arg1` is “`contr.treatment`” which imposes the constraint $\alpha_1 = 0$ in the context of a single categorical predictor.

We shall use the default (“treatment contrasts”) throughout this book. We remark, just for the sake of interest, that although they are different contrasts “`contr.sum`” and “`contr.helmert`” both impose the same constraint, explicitly $\alpha_1 + \dots + \alpha_L = 0$ in the context of a single categorical predictor. They have other properties which are different however.

If you are unsure of what contrasts are currently in force (e.g. if you think you might have been using software which surreptitiously resets the contrasts) you can find out by typing `getOption("contrasts")`.

It is important to remember that all the models that one gets by employing different constraints are *precisely equivalent*. The *interpretation* of the parameter estimates is different, but the *information* obtained from the model fit is exactly the same.

The over-parameterised form (9.14) of the model allows one to conveniently specify models that are of genuine interest and that would be very hard to specify otherwise. We shall elaborate on this when we come to models with two (or more) categorical predictors.

The model-fitting software in R converts logical variables to categorical variables, so that the variable `greenstone` in the models above is actually converted to a factor with levels `TRUE` and `FALSE`. The intercept term in the model is associated with the first level of the factor, which is `FALSE`, since this comes alphabetically before `TRUE`. Happily this convention gives a sensible result here.

9.3.5 Additive models

We now start looking at models which involve more than one “original” covariate. The `Beilschmiedia` data include two (numerical) covariates, the terrain elevation and terrain slope. The simplest loglinear model depending on both covariates is the additive model

$$\lambda(u) = \exp(\beta_0 + \beta_1 E(u) + \beta_2 S(u)) \quad (9.16)$$

where $\beta_0, \beta_1, \beta_2$ are parameters to be estimated, $E(u)$ is the terrain elevation in metres at location u , and $S(u)$ is the terrain slope (a dimensionless gradient). Additive models are specified by joining the relevant model terms together using +:

```
> fitadd <- ppm(bei ~ elev + grad,data=bei.extra)
> fitadd
Nonstationary Poisson process
Trend formula: ~elev + grad
Fitted trend coefficients:
(Intercept)      elev        grad
-8.55862      0.02141     5.84104
```

The interpretation of the fitted model is straightforward, since both covariates are continuous numerical quantities. On a flat parcel of terrain (slope zero) at sea level (elevation zero) the intensity of *Beilschmiedia* trees would be $e^{-8.559} = 0.0001918$ trees per square metre. For each additional metre of terrain elevation, the intensity of increases by a factor $e^{1 \times 0.02141} = 1.022$ or about 2.2 percent. For each additional increase of (say) 0.1 units in slope, the intensity increases by a factor $e^{0.1 \times 5.841} = 1.793$. These two effects are *additive* on the scale of the linear predictor: elevation increases by 1 metre and slope increases by 0.1 units, then the *log* intensity increases by the elevation effect 1×0.02141 **plus** the slope effect 0.1×5.841 .

To assess the relative ‘importance’ of the slope and elevation variables in the model, we need to account for the range of values of each variable. Terrain slope varies from zero to $\max(\text{grad}) = 0.3285$ so the effect of slope varies by a factor of $e^{5.841 \times 0.3285} = 6.813$ between flattest and steepest slopes. Terrain elevation ranges between 119.8 and 159.5 metres, so the effect of elevation varies by a factor of $e^{0.02141(159.5 - 119.8)} = 2.34$ between lowest and highest elevations. Hence the slope effect is more ‘important’ in magnitude.

For the Murchison data we have two covariates, namely `dfault` (a continuous numerical variable) and `greenstone` (a logical covariate). The additive model with these two covariates is

$$\lambda(u) = \exp(\beta_0 + \beta_1 d(u) + \beta_2 G(u)) \quad (9.17)$$

where $\beta_0, \beta_1, \beta_2$ are parameters to be estimated, $d(u)$ is the distance to nearest fault, and $G(u)$ is the indicator which equals 1 inside the greenstone outcrop.

```
> ppm(gold ~ dfault + greenstone,data=mur)
Nonstationary Poisson process
Trend formula: ~dfault + greenstone
Fitted trend coefficients:
(Intercept)      dfault greenstoneTRUE
-6.6171       -0.1038      2.7540
```

The fitted model states that the intensity of gold deposits declines by a factor $e^{-0.1038} = 0.9014$ for every additional kilometre of distance from the nearest fault, and for a given distance, the intensity is $e^{2.754} = 15.71$ times higher inside the greenstone outcrop than it is outside the greenstone.

Note that adding a covariate twice has no effect in a model formula: `grad + grad` is equivalent to `grad`. (See item ?? in section ??.)

In fact we have already looked at an “additive model”, in section 9.3.2. The model `bei ~ grad + I(grad^2)` adds the effects of the two covariates `grad` and `I(grad^2)`. This is slightly different from the examples above, since the two covariates are related. Nevertheless this is formally an additive model.

9.3.6 Modelling spatial trend using Cartesian coordinates

The Cartesian coordinates x and y can also serve as spatial covariates. They are particularly useful for investigating spatial trend when there are no relevant covariate data, or perhaps when it is suspected that the intensity is not only dependent on the available covariates. A wide class of models can be built up by combining the Cartesian coordinates in convenient ways.

We illustrate this idea using the full Japanese Pines data,

```
> jpines <- residualspaper[["Fig1"]]
```

A plot of these data (Figure 1.3 on page 5) suggests that they exhibit spatial inhomogeneity. No auxiliary covariate data are available, so we resort to the Cartesian coordinates. For brevity we shall write the intensity function for such models in the form $\lambda((x,y))$ rather than the more long-winded form “ $\lambda(u)$ where u is the point with coordinates (x,y) ”. For instance an inhomogeneous Poisson model with an intensity that is first order loglinear in the Cartesian coordinates will be written as $\lambda_\theta((x,y)) = \exp(\theta_0 + \theta_1 x + \theta_2 y)$. To fit such a model to the Japanese Pines data simply type

```
> ppm(jpines ~ x + y)
Nonstationary Poisson process
Trend formula: ~x + y
Fitted trend coefficients:
(Intercept)           x           y
0.591840    0.014329   0.009644
```

Here the symbols x and y are reserved variable names that always indicate the Cartesian coordinates. The fitted intensity function is

$$\lambda_\theta((x,y)) = \exp(0.5918 + 0.01433x + 0.009644y).$$

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(jpines ~ polynom(x,y,2))
Nonstationary Poisson process
Trend formula: ~x + y + I(x^2) + I(x * y) + I(y^2)
Fitted trend coefficients:
(Intercept)           x           y           I(x^2)       I(x * y)       I(y^2)
0.130330    0.461497   -0.200945   -0.044738    0.001058    0.020399
```

Notice that the expression `polynom(x,y,2)` has been syntactically expanded into $x + y + I(x^2) + I(x*y) + I(y^2)$. This is a special spatstat trick and applies only to the model formulae passed to `ppm()`.

In the same vein we could fit a log-cubic polynomial `polynom(x,y,3)` and so on. An alternative to the full polynomial of order 3, which has 10 coefficients, is the *harmonic* polynomial `harmonic(x,y,3)` which has only 7 coefficients.

Other simple models which are expressed in terms of the Cartesian coordinates, but are not additive (polynomial) models can be fitted using `ppm()`. For example, to fit a model with constant but unequal intensities on each side of the diagonal line $x = 0.5$, we simply use the expression $x < 0.5$ to make a logical covariate:

```
> ppm(jpines ~ (x < 0.5))
Nonstationary Poisson process
Trend formula: ~(x < 0.5)
Fitted trend coefficients:
(Intercept) x < 0.5TRUE
0.7668     -1.8931
```

9.3.7 Models with offsets

An *offset* is a term in the linear predictor which does not involve any parameters of the model. Offsets are useful when the effect of one variable is already known, or when we want to fit the model relative to a known baseline.

For instance, in some cases it is appropriate to fit an inhomogeneous Poisson model with intensity that is *proportional* to the covariate,

$$\lambda(u) = \kappa Z(u) \quad (9.18)$$

where Z is the covariate and κ is the coefficient to be estimated. We called this a “baseline” model in Section 9.2.5. Taking logarithms, this model is equivalent to

$$\log \lambda(u) = \log \kappa + \log Z(u). \quad (9.19)$$

Note that there is no coefficient in front of the term $\log Z(u)$ in (9.19), so this term is an *offset*.

An important example of a baseline covariate $Z(u)$ is the spatially-varying density of human population. The spatial point pattern of cases of a rare disease could reasonably be expected to follow a Poisson point process with intensity (9.18), where κ is the (constant) disease risk per head of population.

The Chorley-Ribble data (Figure 1.12 on page 10) give the locations of cases of the rare cancer of the larynx, and a sample of cases of the much more common lung cancer. The smoothed intensity of lung cancer cases can serve as a surrogate for the spatially-varying density of the susceptible population.

```
> lung <- split(chorley)$lung
> larynx <- split(chorley)$larynx
> smo <- density(lung, sigma=0.15, eps=0.1)
> smo <- eval.im(pmax(smo, 1e-10))
```

In a model formula, we indicate an offset term by enclosing it in the function `offset()`. Accordingly we fit the constant risk model (9.18) by

```
> ppm(larynx ~ offset(log(smo)))
Nonstationary Poisson process
Trend formula: ~offset(log(smo))
Fitted trend coefficient:
(Intercept)
-2.938
```

The fitted coefficient (`Intercept`) is the constant $\log \kappa$ appearing in (9.19), so converting back to the form (9.18), the fitted model is

$$\lambda(u) = e^{-2.938} Z(u) = 0.05297 Z(u)$$

where in this case $Z(u)$ is the smoothed intensity of lung cancer cases.

In this example, note that the fitted parameter κ is not the estimated risk of laryngeal cancer per head of population, because the `lung` data are a subsample from the cancer registry, and lung cancer cases are a subset of the susceptible population. The best way to estimate the risk of laryngeal cancer assuming constant risk is to divide the total number of laryngeal cancer cases by an estimate of the total susceptible population, since the constant risk model does not involve spatial information. The model fitted above is useful mainly for comparison against alternative models where the risk of laryngeal cancer is spatially-varying.

Spatial transformations, such as geographic projections, also introduce offset terms. Suppose

that a point process \mathbf{X} has intensity function $\lambda_{\mathbf{X}}(u) = \exp(\boldsymbol{\beta}^T \mathbf{Z}(u))$. We change the coordinate system so that the points x_i are mapped to new coordinate positions $y_i = f(x_i)$. Changes of coordinates were discussed in Section 6.5.3: the transformed point process $\mathbf{Y} = f(\mathbf{X})$ has intensity function given by equation (6.14). The new model is

$$\lambda_{\mathbf{Y}}(u) = \exp(\log J(u) + \boldsymbol{\beta}^T \mathbf{Z}(f^{-1}(u))) \quad (9.20)$$

which includes the offset term $\log J(u)$, the log of the Jacobian of the change of coordinates.

When fitting a baseline model it is absolutely crucial to express the trend in the form `offset(log(baseline))`. If the `offset` is omitted, we get a completely different model, as discussed next.

9.3.8 Power law models

It is sometimes appropriate to fit a *power law* relationship

$$\lambda(u) = \alpha Z(u)^k \quad (9.21)$$

where the exponent k is not known, and must be estimated from data, along with the coefficient α . Taking logarithms, the power law (9.21) is equivalent to

$$\log \lambda(u) = \beta_0 + \beta_1 \log Z(u) \quad (9.22)$$

where $\beta_0 = \log \alpha$ and $\beta_1 = k$. This is a loglinear model with covariate $\log Z$.

In the *Beilschmiedia* data, suppose we believe that the forest density obeys a power law as a function of terrain slope. This can be fitted easily:

```
> ppm(bei ~ log(grad), data=bei.extra)
Nonstationary Poisson process
Trend formula: ~log(grad)
Fitted trend coefficients:
(Intercept)  log(grad)
-3.4797      0.5549
```

Note that there is no `offset` here. The printed output describes a model in which the intensity is proportional to the 0.55th power of the terrain slope, or roughly the *square root* of the slope.

Care needs to be taken with power models if the covariate data may include zero values. If the exponent $k = \beta_1$ is positive, then the power law (9.21) implies that, in places where the covariate value $Z(u)$ is zero, the intensity $\lambda(u)$ is also zero, so there is zero probability of observing a random point at such places. If a data point *is* observed at a place where the covariate is zero, this model is invalidated (it has likelihood zero). In the examples above, if there had been a *Beilschmiedia* tree growing on a perfectly flat patch of land (i.e. where the terrain slope is zero) then the data would have been inconsistent with the model (9.18) and (9.21). In some cases this may be detected by the `ppm()` function, but in many cases it will not be detected until the low-level code throws a rather undignified error. We can reproduce this scenario by artificially assigning a zero value to a pixel where there is a data point:

```
> G <- bei.extra[["grad"]]
> G[bei[42]] <- 0
> ppm(bei ~ log(G))

Error in glm.fit(x = structure(c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
:   NA/NaN/Inf in 'x'
```

If the fitted exponent $k = \beta_1$ is negative, then the power law implies that, in places where $Z(u) = 0$, the intensity is *infinite*. This means it is possible that the fitted model may be **improper**. In Section 9.2.1 we noted that, in order for a Poisson process to be well-defined, the integral of the intensity function over the window must be finite. Intensity functions of the form (9.21) may not be integrable if the exponent β_1 is negative.

For example with the Murchison data, the covariate $d(u)$, distance to nearest fault, takes zero values along the faults themselves. If we fit a power model,

```
> ppm(gold ~ log(dfault), data=mur)
Nonstationary Poisson process
Trend formula: ~log(dfault)
Fitted trend coefficients:
(Intercept) log(dfault)
-5.0576      -0.7188
```

the fitted model says that the intensity is the negative 0.72th power of $d(u)$. This function is not integrable so the model is improper, that is, it is not well-defined.

Improper models can occur in other contexts where some values of the covariate are infinite (in this case $Z = 0$ gives $\log Z = -\infty$). An improper model will lead to difficulties with simulation code and various other algorithms.

9.3.9 Models with interaction between covariates

A model which is not additive is said to have “interaction”. A statistical model involving two covariates A and B is additive if the linear predictor is the sum of a term depending only on A plus another term depending only on B. Otherwise, the model is said to exhibit *interaction* between the covariates A and B. Interaction means that the effect of a change in A depends on the current value of B, and *vice versa*.

Be warned that there is another use of the word “interaction” in spatial statistics. A point process which is not Poisson is said to exhibit “interaction” between the *points of the process*. This is a distinct and completely unrelated concept for which the same word is used. We will usually call this “interpoint interaction” to keep the distinction clear.

In the present context we are talking about *interaction between covariates* in a model for the intensity of the point process. In a model formula, the expression A:B represents a particular model term called “the” interaction between the covariates A and B. This is effectively a new covariate depending on both A and B. Its definition depends, as usual, on the type of variables involved: we explain further below.

Normally, a model that includes an interaction A:B should also include the “main effects” A and B. Usually it is *possible* to write a model which includes only an interaction term but this makes the results difficult to interpret and reduces their usefulness. How to interpret them, and whether they actually make any sense at all, depends on circumstances.

To include interaction between A and B in a model we would thus generally write the formula in the form X ~ A + B + A:B. The expression A + B + A:B can be abbreviated to A * B which is read as “A cross B” (and the operator * is referred to as the crossing operator).

9.3.9.1 Interaction between two numerical covariates

If A and B are both numerical covariates, the expression A:B is interpreted as simply the numerical product of A and B. This makes sense: there is “interaction” between A and B if the rate of change of the response with respect to A depends upon the value of B (and vice versa). This will be the case if the response depends on the product of A and B.

It should be noted that although it is thus possible to talk about (and that some people do

talk about) “interaction” between two numerical predictors, it is probably inadvisable to do so. It is less pretentious, and less confusing, if you just say “product”. In other words if both A and B are numerical covariates, then the formula $X \sim A * B = X \sim A + B + A:B$ is equivalent to $X \sim A + B + I(A*B)$. The latter expression, while more cumbersome to type, is much less likely to mislead the reader or to cause confusion.

For the *Beilschmiedia* data, we have already looked at an additive model involving the terrain elevation `elev` and terrain slope `grad`. Fitting a model with interaction between these covariates is just as easy:

```
> fit <- ppm(bei ~ elev + grad + I(elev*grad),
  data=bei.extra)
> fit
Nonstationary Poisson process
Trend formula: ~elev + grad + I(elev * grad)
Fitted trend coefficients:
(Intercept)          elev          grad I(elev * grad)
-4.389734       -0.007115     -36.608966      0.293532
```

9.3.9.2 Interaction between two factors

If A and B are both factors, the formula $X \sim A * B$ represents the (over-parameterised) mathematical model

$$\lambda(u) = \exp(\mu + \alpha_{A(u)} + \beta_{B(u)} + \gamma_{A(u),B(u)}) \quad (9.23)$$

where $A(u)$ is the level of factor A at the location u , and $B(u)$ is the level of B at u . This model has parameters μ (the intercept), $\alpha_1, \dots, \alpha_L$ (the main effects of the levels of A), β_1, \dots, β_M (the main effects of the levels of B) and $\gamma_{1,1}, \dots, \gamma_{L,M}$ (the interaction effects for each combination of levels of A and B). At a location u where $A(u) = i$ and $B(u) = j$, the predicted intensity is $\exp(\mu + \alpha_i + \beta_j + \gamma_{i,j})$. Compared with the additive model

$$\lambda(u) = \exp(\mu + \alpha_{A(u)} + \beta_{B(u)}), \quad (9.24)$$

the interaction model (9.23) has an additional “synergy” or “catalysis” term between the individual covariates (which could be negative; a “counter-synergy”) of the form $\gamma_{A(u),B(u)}$ which corresponds to the interaction term $A:B$.

The gorillas dataset (page 281) is supplied with factor covariates `vegetation` and `heat`. A model involving interaction between these two factors is:

```
> ppm(gor ~ vege * heat, data=gex)
Nonstationary Poisson process
Trend formula: ~vege * heat
Fitted trend coefficients:
(Intercept)      vegeColo      vegeGras      vegePrim
                2.6663      -13.9689      -1.1275      1.7212
      vegeSeco      vegeTran      heatMode      heatCool
                  0.6770       0.8777      -0.7430     -0.8727
vegeColo:heatMode vegeGras:heatMode vegePrim:heatMode vegeSeco:heatMode
                 16.5847       0.7808       0.8327      0.9812
vegeTran:heatMode vegeColo:heatCool vegeGras:heatCool vegePrim:heatCool
                 1.3610      17.9671     -11.9687      0.9337
vegeSeco:heatCool vegeTran:heatCool
                 -13.7731        NA
```

The result gives a fitted log intensity for every possible combination of vegetation type and heat load index. Note that one of the fitted coefficients is NA, because the corresponding combination of vegetation and heat does not occur in the data — at least, not at any of the quadrature points (sample locations used to fit the model).

9.3.9.3 Interaction between factor and numerical covariate

If A is a factor and Z is numerical, the additive model is

$$\lambda(u) = \exp(\mu + \alpha_{A(u)} + \beta Z(u)) \quad (9.25)$$

while the full interaction model $X \sim A * Z$ is

$$\lambda(u) = \exp(\mu + \alpha_{A(u)} + \beta Z(u) + \gamma_{A(u)} Z(u)). \quad (9.26)$$

In the additive model (9.25) the parameters $\alpha_1, \dots, \alpha_L$ are the effects of the different levels of the factor A, while the “slope” parameter β is the effect of a unit increase in the numerical covariate Z. If we were to plot the linear predictor (the log intensity) as a function of Z for different levels of the covariate, the graph would consist of parallel lines with the same slope β but with different intercepts $\mu + \alpha_i$.

In the interaction model (9.26) there is an extra term $\gamma_{A(u)} Z(u)$ which causes the effect of a unit increase in Z to depend on the level of the factor A. The plot of the linear predictor described above would show lines with *different* slopes $\beta + \gamma_{A_i}$.

The Murchison data provide a window `greenstone` (which effectively becomes a factor with levels FALSE and TRUE) and we have constructed a numerical covariate `dfault`. The model with interaction is:

```
> ppm(gold ~ dfault * greenstone, data=mur)
Nonstationary Poisson process
Trend formula: ~dfault * greenstone
Fitted trend coefficients:
  (Intercept)      dfault    greenstoneTRUE
  -6.0184       -0.2047      2.0013
  dfault:greenstoneTRUE
  0.1674
```

9.3.9.4 Nested interaction

It is sometimes appropriate to fit a model with the formula $X \sim A + A:B$. That formula can also be rendered as $y \sim A/B$. The operator / stands for a *nested interaction* and we read A/B as “B nested within A”.

For example, in a taxonomic classification of organisms, suppose G and S are the Genus and Species names, treated as factors. The Species name only makes sense when the Genus is specified. We may wish to compare models which depend only on Genus, of the form $y \sim G$, with models which depend on Species, of the form $y \sim G/S$.

Nesting a numerical covariate inside a factor is effectively the same as crossing the two covariates. For the Murchison data, we may try nesting the numerical covariate `dfault` inside the factor `greenstone`:

```
> ppm(gold ~ greenstone/dfault, data=mur)
```

```

Nonstationary Poisson process
Trend formula: ~greenstone/dfault
Fitted trend coefficients:
  (Intercept)      greenstoneTRUE greenstoneFALSE:dfault
  -6.01845          2.00131        -0.20466
greenstoneTRUE:dfault
  -0.03728

> ppm(gold ~ greenstone/dfault-1, data=mur)
Nonstationary Poisson process
Trend formula: ~greenstone/dfault - 1
Fitted trend coefficients:
  greenstoneFALSE      greenstoneTRUE greenstoneFALSE:dfault
  -6.01845            -4.01714        -0.20466
greenstoneTRUE:dfault
  -0.03728

```

These two models are exactly equivalent, except for the way in which they are parametrised. They are also equivalent to the interaction model with formula `gold ~ greenstone * dfault`. Different choices of parameterisation make it easier or harder to extract particular kinds of information.

Nesting a factor A inside a numerical covariate Z gives a model where the effect of a unit change in Z depends on the level of A, but the factor A has no effect on the intercept. In the plot of the linear predictor against Z described above, the lines are not parallel, but they all have the same intercept. For the Murchison data, the model `gold ~ dfault/greenstone` stipulates that the intensity of gold deposits very close to a geological fault ($dfault \approx 0$) must be the same inside and outside the greenstone, but as we increase distance from the faults, the intensity may decrease at different rates inside and outside the greenstone:

```

> ppm(gold ~ dfault/greenstone, data=mur)
Nonstationary Poisson process
Trend formula: ~dfault/greenstone
Fitted trend coefficients:
  (Intercept)      dfault dfault:greenstoneTRUE
  -4.3472         -0.4972        0.5020

```

Nested interactions also arise in connection with random effects, as discussed in Chapter 16.

9.3.10 Formulas involving many variables

A model can involve any number of covariates. In an additive model, the variable names are simply joined by the “+” operator:

```
> ppm(Y ~ X1 + X2 + X3 + ... + Xn)
```

This expression could be long and tedious to type out. A useful shortcut is the symbol “.” representing *all* the available covariates. This only works when the covariates are supplied in the argument `data` (see page 273). The additive model involving all covariates can be fitted using the formula “`Y ~ .`” as in

```
> ppm(gor ~ . , data=gex)
```

To fit the additive model involving all covariates *except* the `heat` covariate,

```
> ppm(gor ~ . - heat, data=gex)
```

Interactions between more than two predictors (“higher order interactions”) are easy to define. For three factors A, B and C the second order interactions are $A:B$, $A:C$ and $B:C$. The third order interaction is $A:B:C$. The mathematical expressions defining higher order interactions get increasingly cumbersome to write down, but the ideas are basically the same as for second order interactions. Higher order interactions can be difficult to interpret in practical terms.

In the model formula context the `+` and `:` operators are what the pure mathematicians (poor dears!) call “idempotent” operators. That is, $A+A$ is just equal to A , and likewise $A:A$ is equal to A .

The crossing operator `*` obeys the distributive and associative laws, so that $(A + B + C) * (A + B + C)$ expands to $A + B + A:B + A:C + B:C$, and $A * B * C$ expands to $A + B + C + A:B + A:C + B:C + A:B:C$.

We have now dealt with the model operators `+`, `-`, `:`, `*` and `/`. The last operator to be mentioned is the power operator `^`. If A is any model term, A^2 is equivalent to $A * A$, while A^3 is equivalent to $A * A * A$ and so on. This is particularly useful when we want to specify all main effects and interactions up to a certain order. For example $(A + B + C)^2$ is equivalent to $(A + B + C) * (A + B + C)$ which expands to $A + B + A:B + A:C + B:C$, containing all main effects and second order interactions between the covariates A, B and C. One could also use the “`.`” symbol, for example

```
> ppm(gor ~ .^2, data=gex)
```

would fit all main effects and all second-order interaction terms involving all the covariates in the list `gex`.

Tip: To expand a complicated formula `f`, try `update(f, . ~ .)` This uses the symbol “`.`” in another sense, explained in Section 9.4.4.

```
> update(Y ~ (A+B+C)^3 - B/(A:C), . ~ .)
Y ~ A + C + A:B + A:C + B:C
```

9.4 Statistical inference for Poisson models

9.4.1 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.

Table 9.1 lists some of the standard operations on fitted models in R which can be applied to point process models. That is, these generic operations have methods for the class “`ppm`”. For information on these methods, consult the help for `print.ppm()`, `summary.ppm()`, `plot.ppm()` etc.

Table 9.2 lists some non-generic functions in the base R system which work on “`ppm`” objects. For information on these functions, consult the help for the function itself.

The `print()` method prints important information about the model structure and the fitted

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 $\hat{\theta}$
vcov()	variance-covariance matrix of $\hat{\theta}$
anova()	analysis of deviance
logLik()	loglikelihood value
formula()	extract the model formula
terms()	extract the terms in the model
model.matrix()	compute the design matrix

Table 9.1: Generic operations which have methods for point process models (class "ppm").

confint()	confidence intervals for parameters
step()	stepwise model selection
stepAIC()	(package "MASS") stepwise model selection
drop1()	one step model deletion
add1()	one step model augmentation
AIC()	Akaike Information Criterion

Table 9.2: Functions in the base R system which work on "ppm" objects.

model parameters. For Poisson models it also gives a table of parameter estimates with standard errors, confidence intervals, and the result of a test that the parameter is zero. For example with the Beilschmiedia data we may do the following.

```
> beikm <- rescale(bei, 1000, unitname="km")
> bei.extrakm <- lapply(bei.extra, rescale, s=1000, unitname="km")
> fitkm <- ppm(beikm ~ x + y)
> fitkm
Nonstationary Poisson process

Trend formula: ~x + y

Fitted trend coefficients:
(Intercept)          x           y
```

```
 9.0910      -0.8031      0.6496
```

	Estimate	S.E.	CI95.lo	CI95.hi	Ztest	Zval
(Intercept)	9.0910	0.04306	9.0066	9.1754	***	211.128
x	-0.8031	0.05863	-0.9180	-0.6882	***	-13.698
y	0.6496	0.11571	0.4228	0.8764	***	5.614

This is the fitted model with intensity function

$$\lambda_\theta((x,y)) = \exp(\theta_0 + \theta_1 x + \theta_2 y) \quad (9.27)$$

where the spatial coordinates x and y are measured in kilometres, and for example, the estimate of θ_1 is $\hat{\theta}_1 = -0.8031$ with standard error $\text{se}(\hat{\theta}_1) = 0.05863$ and 95% confidence interval $[-0.918, -0.6882]$. The amount of information displayed, and the layout, depend on `spatstat.options('terse')` and `spatstat.options('print.ppm.SE')`.

The fitted coefficients of the model can be extracted by `coef.ppm()`, a method for the generic function `coef()`:

```
> coef(fitkm)
(Intercept)           x           y
 9.0910      -0.8031      0.6496
```

The `plot()` method is useful for initial inspection of a fitted model, because it offers a wide range of displays. Figure 9.10 shows the result of the command

```
> plot(fitkm, how="image", se=FALSE)
```

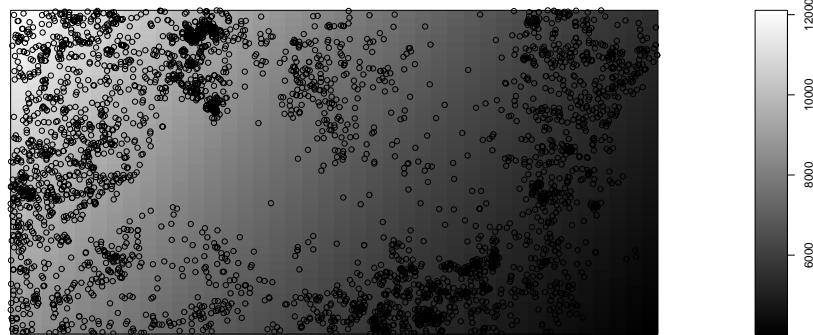


Figure 9.10: Result of plotting a fitted Poisson model.

Very detailed information about the model can be printed by typing

```
> summary(fitkm)
```

(which we do not reproduce, to save trees.) The result of `summary.ppm()` is an object of class `summary.ppm()` which can be printed and manipulated in other ways. For example the table of parameter estimates, standard errors and confidence intervals is obtained by

```
> coef(summary(fitkm))
            Estimate   S.E. CI95.lo CI95.hi Ztest    Zval
(Intercept) 9.0910 0.04306 9.0066 9.1754 *** 211.128
x          -0.8031 0.05863 -0.9180 -0.6882 *** -13.698
y          0.6496 0.11571 0.4228 0.8764 *** 5.614
```

9.4.2 Standard errors and confidence intervals for parameters

The accuracy of the fitted model coefficients can be analysed by standard asymptotic theory as explained in Section 9.5 and in [209, 278]. For sufficiently large samples from a loglinear Poisson model (9.6), the estimated parameters $\hat{\theta}$ have approximately a multivariate normal distribution with mean equal to the true parameters θ and variance-covariance matrix $\text{var}\hat{\theta} = I(\theta)^{-1}$, where $I(\theta)$ is the Fisher information matrix

$$I(\theta) = \int_W \mathbf{Z}(u) \mathbf{Z}(u)^\top \lambda_\theta(u) du. \quad (9.28)$$

The estimated variance-covariance matrix $\text{var}\hat{\theta} = I(\hat{\theta})$ is computed by `vcov.ppm()`, a method for the generic function `vcov()`.

```
> vcov(fitkm)
            (Intercept)      x          y
(Intercept)  0.001854 -1.491e-03 -3.528e-03
x           -0.001491  3.438e-03  1.208e-08
y           -0.003528  1.208e-08  1.339e-02
```

The diagonal of this matrix contains the estimated variances of the individual parameter estimates $\theta_0, \theta_1, \theta_2$, so the standard errors are:

```
> sqrt(diag(vcov(fitkm)))
            (Intercept)      x          y
                  0.04306  0.05863  0.11571
```

Confidence intervals for the parameters θ can be obtained from the standard function `confint()`:

```
> confint(fitkm, level=0.95)
            2.5 % 97.5 %
(Intercept) 9.0066 9.1754
x           -0.9180 -0.6882
y           0.4228  0.8764
```

and these could also be constructed manually from the parameter estimate `coef(fitkm)` and the standard error `sqrt(diag(vcov(fitkm)))`.

The estimated correlation between individual parameter estimates can be useful in detecting collinearity and confounding. Correlations can be computed using `vcov.ppm()`:

```
> co <- vcov(fitkm, what="corr")
> round(co, 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 suggests fairly strong negative correlation between the intercept parameter estimate $\hat{\theta}_0$ and the estimates of the coefficients of x and y . However, the correlation between an intercept estimate and a slope estimate depends on the origin for the covariate. We get a different answer if we place the coordinate origin in the centre of the study region:

```

> fitch <- update(fitkm, . ~ I(x-0.5) + I(y-0.25))
> co <- vcov(fitch, what="corr")
> round(co, 2)
      (Intercept) I(x - 0.5) I(y - 0.25)
(Intercept)     1.00      0.23     -0.09
I(x - 0.5)      0.23      1.00      0.00
I(y - 0.25)     -0.09     0.00      1.00

```

9.4.3 Prediction

For any Poisson model with intensity $\lambda(u) = \lambda_\theta(u)$ where θ is a parameter vector, the *fitted intensity* or *predicted intensity* is the function $\lambda_{\hat{\theta}}(u)$ obtained by substituting the fitted parameter estimates into the intensity formula.

The fitted intensity is computed by `predict.ppm()`, a method for the generic function `predict()`. By default it computes the fitted intensity at a regular grid of locations yielding a pixel image:

```
> lamhat <- predict(fitkm)
```

The result is shown in the left panel of Figure 9.11. Alternatively `predict.ppm()` can be used to obtain predicted values at any locations, including the original data points:

```
> lamB <- predict(fitkm, locations=beikm)
```

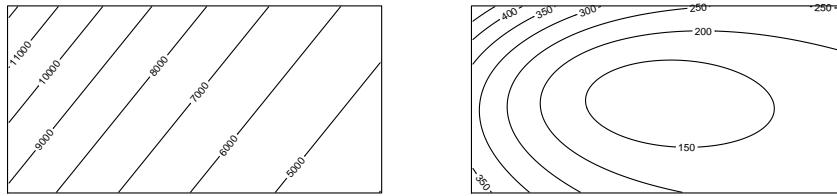


Figure 9.11: Contour plots of the images `predict(fitkm)` (Left) and `predict(fitkm, type="se")` (Right). Ribbon annotation is suppressed.

For a loglinear model (9.6), the asymptotic variance of the predicted intensity $\lambda_\theta(u)$ at a given location u is

$$\text{var} \hat{\lambda}(u) \sim \mathbf{Z}(u) I(\theta)^{-1} \mathbf{Z}(u)^\top \lambda_\theta(u). \quad (9.29)$$

The right panel of Figure 9.11 shows the standard error, i.e. the square root of this variance, which is computed by `predict(fitkm, type="se")`.

A confidence interval for the true value of $\lambda(u)$ at each location u can be computed by `predict(fitkm, interval="confidence")`. This yields a list of two images giving the lower and upper limits of the confidence interval. Figure 9.12 shows the result of plotting this.

It is also possible to plot the ‘effect’ of a single covariate in the model. The command `effectfun()` computes the intensity of the fitted model as a function of one of its covariates. An example was shown in Figure 9.4.

We may also be interested in predicting the number of points in a spatial region B . For a Poisson process, the expected number of points in B is the integral of the intensity over B , equation (9.1) on page 268. The fitted mean (expected) number of points in B is

$$\hat{\mu}(B) = \int_B \hat{\lambda}(u) du.$$

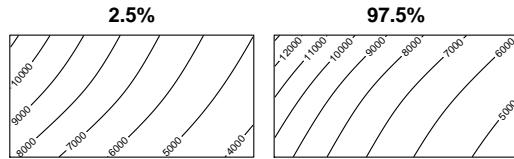


Figure 9.12: Contour plots of `predict(fitkm, interval="confidence")`.

This can be computed using the argument `total` to `predict.ppm()`. To find the expected number of trees at elevations below 130 metres:

```
> B <- levelset(bi.extrakk$elev, 130)
> predict(fitkm, total=B)
[1] 141.9
```

The asymptotic variance of $\hat{\mu}(B)$ is

$$\text{var } \hat{\mu}(B) \sim M(B)I(\boldsymbol{\theta})^{-1}M(B)^\top \quad (9.30)$$

where

$$M(B) = \int_B \mathbf{Z}(u)\lambda_\theta(u) du. \quad (9.31)$$

This allows us to compute the standard error for the estimate of the mean number of trees:

```
> predict(fitkm, total=B, type="se")
[1] 4.45
```

or a confidence interval for the true expected number (9.1):

```
> predict(fitkm, total=B, interval="confidence")
2.5% 97.5%
133.1 150.6
```

In this case it is also meaningful to compute a *prediction interval* for the random number of trees in the specified region:

```
> predict(fitkm, total=B, interval="prediction")
2.5% 97.5%
117    166
```

The generic function `model.matrix()` computes the “design matrix” of a model. In the case of a Poisson point process model, `model.matrix.ppm()` returns a matrix with one row for each quadrature point, and one column for each of the *canonical* covariates appearing in (9.6). Note that, if the model formula involves transformations of the original covariates, then `model.matrix(fitkm)` gives values of these transformed covariates.

The `spatstat` package also defines a new generic function `model.images()` with a method for “ppm” objects. This produces a list of *pixel images* of the canonical covariates.

```
> fitkm <- ppm(bi ~ sqrt(grad) + x, data=bi.extra)
> mo <- model.images(fitkm)
> names(mo)
[1] "(Intercept)" "sqrt(grad)"   "x"
```

9.4.4 Updating a model

In data analysis we typically fit several different candidate models to the same data. The generic function `update()` makes it easy to do this. It modifies a fitted model, for example by changing the model formula and re-fitting the model.

The method `update.ppm()` is provided in `spatstat` for updating a fitted point process model. The syntax is

```
> update(object, ...)
```

where `object` is a fitted point process model (class `ppm`) and the subsequent arguments “`...`”, if any, determine how the model should be changed. The result is another fitted model of the same kind.

The syntax `update(object)` makes sense, and causes the `object` to be re-fitted. The result is different from the original object if any of the data used to fit the original model have changed. For example

```
> X <- rpoispp(42)
> m <- ppm(X ~ 1)
> m
Stationary Poisson process
Intensity: 43
      Estimate   S.E. CI95.lo CI95.hi Ztest  Zval
log(lambda)  3.761 0.1525   3.462    4.06 *** 24.66
> X <- rpoispp(100)
> update(m)
Stationary Poisson process
Intensity: 107
      Estimate   S.E. CI95.lo CI95.hi Ztest  Zval
log(lambda)  4.673 0.09667  4.483    4.862 *** 48.34
```

The second argument of `update()` may be a formula, specifying the new model formula to be used in re-fitting the model.

```
> fitcsr <- ppm(besi ~ 1, data=besi.extra)
> update(fitcsr, besi ~ grad)
Nonstationary Poisson process

Trend formula: ~grad

Fitted trend coefficients:
(Intercept)      grad
-5.391        5.022

      Estimate   S.E. CI95.lo CI95.hi Ztest  Zval
(Intercept) -5.391 0.03002 -5.449 -5.332 *** -179.58
grad         5.022 0.24540  4.541  5.503 ***   20.46
```

Notice that we needed to provide the `data` argument to the first model, even though the first model does not depend on any covariates, in order for the covariate `besi` to be available for the updated model. This would have been unnecessary if `besi` had been an existing object in the R session.

The new formula may include the symbol “`.`” representing “*what was there before*”. To keep the same left-hand side of the formula, use “`.`” on the left-hand side:

```
> fitgrad <- update(fitcsr, . ~ grad)
```

To modify the right-hand side of the formula, use “.” on the right-hand side, modifying it with the model operators:

```
> fitall <- update(fitgrad, . ~ . + elev)
> fitall
Nonstationary Poisson process

Trend formula: ~grad + elev

Fitted trend coefficients:
(Intercept)      grad        elev
-8.55862      5.84104     0.02141

Estimate      S.E.  CI95.lo  CI95.hi Ztest   Zval
(Intercept) -8.55862 0.341101 -9.22717 -7.89008 *** -25.091
grad         5.84104 0.255861  5.33956  6.34252 ***  22.829
elev         0.02141 0.002288  0.01693  0.02589 ***   9.358
```

To remove the intercept, use -1 or +0:

```
> fitp <- update(fitall, . ~ . - 1)
```

The symbolic manipulation of formulas involving the symbol “.” is handled by the update method for formulas, `update.formula()`. This is quite useful in its own right as a quick way to see the effect of a change in a formula:

```
> update(gor ~ (heat + vege)^2, . ~ . - heat:vege)
gor ~ heat + vege
```

and simply to expand a complicated formula:

```
> update(gor ~ (heat + vege + slop)^3, . ~ .)
gor ~ heat + vege + slop + heat:vege + heat:slop + vege:slop +
heat:vege:slop
```

Warning: the operator “-” does not remove offset terms in model formulae. A command like `update(fit, . ~ . - offset(A))` does not delete the term `offset(A)`.

9.4.5 Model selection

Often we want to decide whether a particular covariate has an effect on the point pattern, after allowing for the effects of other covariates. For example, for the Chorley-Ribble data , the key question is whether distance to the incinerator has an effect on the intensity of laryngeal cancer, after allowing for the fact that the intensity of laryngeal cancer cases will depend on the spatially-varying density of the population at risk.

This is an example of *model selection* — the task of choosing the “best fitting” statistical model from amongst several competing models for the same dataset.

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.

```

> fit1 <- ppm(bei ~ grad,data=bei.extra)
> fitnull <- ppm(bei ~ 1)
> anova(fitnull, fit1, test="Chi")
Analysis of Deviance Table

Model 1: ~1           Poisson
Model 2: ~grad         Poisson
Df Deviance Pr(>Chi)
1
2 1      382   <2e-16 ***
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

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 (9.7). The *p*-value (under the heading *Pr(>Chi)*) is extremely small, indicating rejection of CSR in favour of the alternative. Note that 2e-16 or 2×10^{-16} is the smallest detectable difference between “real numbers”⁵ on the 32-bit computer which produced this output, so the output says that the *p*-value is effectively zero.

The LR test of the null hypothesis that the slope coefficient is zero, is equivalent to Berman’s Z_1 test (Section 6.7.3) based on slope covariate.

Note that standard Analysis of Deviance requires the null hypothesis to be a sub-model of the alternative. Unfortunately the model (9.18), 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 (9.18).

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

```

> fitprop <- ppm(bei ~ offset(log(grad)),data=bei.extra)
> fitnull <- ppm(bei ~1)
> AIC(fitprop)
[1] 42497
> AIC(fitnull)
[1] 42764

```

The smaller AIC favours the model (9.18) with intensity proportional to slope.

Automatic model selection can be performed using `step()` or `stepAIC()` (a more fully-functional version of `step()` available in the MASS package). By default, this performs stepwise deletion. Starting from the fitted model, the procedure considers each term in the model, and determines whether the term should be deleted (according to AIC). The deletion giving the biggest improvement in AIC is carried out. This is applied recursively until no more terms can be deleted.

```

> X <- rpoispp(100)
> fit <- ppm(X ~x + y)
> step(fit, trace=0)
Stationary Poisson process
Intensity: 106
      Estimate    S.E. CI95.lo CI95.hi Ztest  Zval
log(lambda)  4.663  0.09713   4.473   4.854 *** 48.01

```

⁵.Machine\$double.eps gives the smallest double-precision floating-point number x that satisfies 1 + x != 1.

(Putting `trace=1` or `2` gives more information about the sequence of models considered.) The conclusion in this case is that CSR is the most preferred sub-model of `fit`.

9.4.6 Simulating the fitted model

A fitted Poisson model can be simulated automatically using the function `simulate.ppm()`. The result of the commands `X <- simulate(fitprop); plot(X[[1]])` is shown in Figure 9.13.

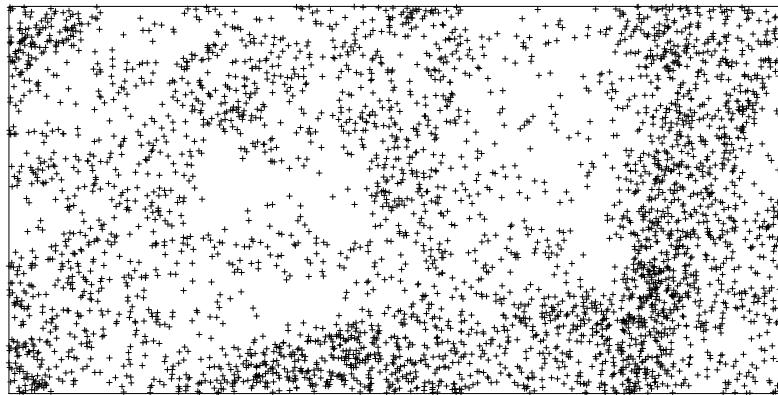


Figure 9.13: Simulated realisation of a fitted Poisson model

It is also possible to perform conditional simulation (conditional on the number of points, on the configuration of points in a particular subregion, or on the presence of certain points). See `rmhcontrol()` for details.

9.4.7 Non- and semi-parametric estimation

```
ppm(~ covariates + offset(log(smoothed baseline)))
rhohat.ppm
```

Material from [23]

Mention formulas involving `gam` terms, and `use.gam=TRUE`

9.5 Theory*

This section covers the theory which underpins the available methods for fitting a Poisson point process model to a point pattern dataset.

Since there has been some confusion in the literature, we shall go to some lengths to explain maximum likelihood estimation for Poisson point processes. Readers who are not interested could skip this section.

* Starred sections contain advanced material, and can be skipped by most readers.

9.5.1 Introduction to maximum likelihood

In mainstream statistical methodology, a standard way to fit models to data is by maximising the *likelihood* of the model for the data. For any choice of parameter values, the likelihood value is defined as the probability (or probability density) that the given observations would have been obtained, from the model with these parameter values. For different possible values of the model parameters, the likelihood gives a measure of relative plausibility of these values. The parameter values which maximise the likelihood are the ‘most plausible’ or ‘best fitting’ values, and are called the *maximum likelihood estimates (MLE)*.

As an example, suppose that we have counted the number n of insects caught in a trap, and we believe this number follows a Poisson distribution with some mean μ . Our goal is to infer (‘estimate’) the value of μ from the data. Refer formula for Poisson distribution. The likelihood is

$$L(\mu) = e^{-\mu} \frac{\mu^n}{n!}$$

where n is the observed number of insects. When we want to stress that this is the likelihood for λ given the data n we may write $L(\mu) = L(\mu; n)$. We need to find the value of μ which gives the maximum value of $L(\mu)$. It is easier to work with the natural logarithm of the likelihood,

$$\log L(\mu) = -\mu + n \log \mu - \log(n!)$$

and maximising $\log L$ is equivalent to maximising L . Take the derivative of $\log L(\mu)$ with respect to μ (known as the score function or simply the *score*):

$$U(\mu) = U(\mu; n) = \frac{d}{d\mu} \log L(\mu) = -1 + \frac{n}{\mu}.$$

The likelihood is maximised when $U(\mu) = 0$, which gives $\mu = n$. That is, the *maximum likelihood estimate* of μ is $\hat{\mu} = n$, the observed number of insects.

Note that the term $\log(n!)$ disappeared in the score. This always happens to additive terms in the loglikelihood (or equivalently multiplicative factors in the likelihood) that only depend on data and not the parameters. Since such terms do not effect the location of the maximum of the (log)likelihood it is common practice to leave them out and still call the function the (log)likelihood.

9.5.2 Maximum likelihood for CSR

For simplicity we start with the homogeneous Poisson point process (CSR). This model has a single parameter λ , the intensity of the process. Assume we have observed a point pattern \mathbf{x} inside a spatial window W . The likelihood function is

$$L(\lambda) = L(\lambda; \mathbf{x}) = \lambda^{n(\mathbf{x})} e^{(1-\lambda)|W|} \quad (9.32)$$

where $n(\mathbf{x})$ is the number of points of \mathbf{x} . The right hand side is the probability density of observing the pattern \mathbf{x} if the true intensity is λ . Notice that this probability density depends only on the number of points, and not on their location, because all spatial locations are equally likely under CSR.

We shall not go into the technical derivation of point process likelihoods, but it is useful to appreciate why the likelihood (9.32) consists of two terms, one associated with the data \mathbf{x} and the other with the window W . Imagine that space is divided into pixels of area a , as illustrated in Figure ?? on page ???. If a is small, there is negligible chance that any pixel will contain more than one point, so that each pixel contains either 1 point (with probability λa) or 0 points (with probability $1 - \lambda a$). Pixels are independent of each other, so the probability of a particular configuration of zeroes and ones is obtained by multiplying together the probabilities of the outcomes for each separate pixel.

If there are n pixels which contain random points, the presence probability λa will appear n times, giving a factor of $\lambda^n a^n$. For the remaining pixels, which do not contain random points, the contribution to the probability is $(1 - \lambda a)^m$ where m is the number of these pixels. Since m is large and a is small, the second term is close to $e^{-\lambda |W|}$. Thus, ignoring some rescaling,⁶ the first term in the likelihood (9.32) is the probability of observing the data points in \mathbf{x} , and the second term is the probability of **not** observing any other points in the window W .

The likelihood (9.32) is conventionally used in theoretical work, but as mentioned at the end of Section 9.5.1 the constant factor $e^{|W|}$ can be omitted. Thus the likelihood we use in practice is

$$L(\lambda) = \lambda^{n(\mathbf{x})} e^{-\lambda |W|} \quad (9.33)$$

The maximum likelihood estimate (MLE) of the intensity λ is the value $\hat{\lambda}$ which maximises $L(\lambda)$ defined in (9.33). As before it is easier to work with the logarithm of the likelihood, which in this case is

$$\log L(\lambda) = n(\mathbf{x}) \log \lambda - \lambda |W|. \quad (9.34)$$

The score (derivative of the loglikelihood) is

$$U(\lambda) = U(\lambda; \mathbf{x}) = \frac{d}{d\lambda} \log L(\lambda) = \frac{n(\mathbf{x})}{\lambda} - |W|. \quad (9.35)$$

The maximum of the likelihood is attained when this derivative is zero, so the MLE is

$$\hat{\lambda} = \frac{n(\mathbf{x})}{|W|}. \quad (9.36)$$

That is, the maximum likelihood estimate $\hat{\lambda}$ is the average intensity of \mathbf{x} , a good “commonsense” estimate of the intensity λ .

Another way to estimate the intensity λ would have been to use the “*method of moments*”. We would equate the observed number of points, $n(\mathbf{x})$, to the theoretically expected number of points, $\lambda |W|$, and solve the equation $\lambda |W| = n(\mathbf{x})$ for λ , yielding $\hat{\lambda} = n(\mathbf{x})/|W|$. In this case, the method-of-moments estimate agrees with the MLE.

9.5.3 Maximum likelihood for general Poisson process

Refer standard literature [278, 209]

9.5.3.1 General theory

Now consider a general, inhomogeneous Poisson process model, governed by a parameter θ . This model states that the intensity is $\lambda_\theta(u)$ where the value of θ is to be estimated. The likelihood for θ is

$$L(\theta) = L(\theta; \mathbf{x}) = \lambda_\theta(x_1) \lambda_\theta(x_2) \dots \lambda_\theta(x_n) \exp\left(\int_W (1 - \lambda_\theta(u)) du\right) \quad (9.37)$$

where x_1, \dots, x_n are the points of \mathbf{x} . This can be derived by pixel approximation as described above. This probability density *does* depend on the locations of the data points x_i , because the intensity function $\lambda_\theta(u)$ makes some locations more likely than others. Therefore the data on spatial locations x_i , and not just the total number of points, are needed for model-fitting.

The loglikelihood for θ is

$$\log L(\theta) = \sum_{i=1}^n \log \lambda_\theta(x_i) - \int_W \lambda_\theta(u) du \quad (9.38)$$

⁶The point process likelihood is conventionally measured relative to the probability for a Poisson process of rate 1. To do this we divide the value obtained above, $\lambda^n a^n e^{-\lambda |W|}$, by the corresponding value for a Poisson process of intensity $\lambda = 1$, namely $a^n e^{-|W|}$ (which is a constant only depending on data, not the parameter), finally yielding $\lambda^n e^{(1-\lambda)|W|}$.

where we have omitted the constant term $\int_W 1 \, du = |W|$.

The MLE $\hat{\theta}$ is usually not a simple function of the data, and must be computed by maximising the likelihood numerically. In general, with no assumptions on the way $\lambda_\theta(u)$ depends on θ , the likelihood function might behave poorly, and might not have a unique maximum. Further analysis depends on assuming more about the intensity model.

9.5.3.2 Maximum likelihood for baseline model

Another Poisson model that is easy to analyse is the one in which the intensity is an unknown multiple of a known baseline (9.3). The loglikelihood (9.38) is

$$\begin{aligned} \log L(\theta) &= \sum_{i=1}^n \log(\theta b(x_i)) - \int_W \theta b(u) \, du \\ &= n \log \theta - \sum_{i=1}^n \log b(x_i) - \theta \int_W b(u) \, du. \end{aligned} \quad (9.39)$$

The loglikelihood is differentiable with respect to θ for fixed \mathbf{x} , even if $b(u)$ is not a continuous function. The score is

$$U(\theta) = \frac{d}{d\theta} \log L = \frac{n(\mathbf{x})}{\theta} - \int_W b(u) \, du. \quad (9.40)$$

If there are no constraints on θ , the maximum likelihood estimate (MLE) of θ is the solution of the score equation $U(\theta) = 0$, which is

$$\hat{\theta} = \frac{n(\mathbf{x})}{\int_W b(u) \, du}. \quad (9.41)$$

This is also the method-of-moments estimate, because under the baseline model, the expected total number of points is

$$\mathbb{E}_\theta[n(\mathbf{X})] = \int_W \lambda_\theta(u) \, du = \theta \int_W b(u) \, du$$

so that $\hat{\theta}$ is the solution of $n(\mathbf{x}) = \mathbb{E}_\theta[n(\mathbf{X})]$.

9.5.4 Maximum likelihood for loglinear Poisson models

9.5.4.1 Loglinear models

For the vast majority of Poisson models treated in this book, the intensity is a **loglinear** function of the parameters:

$$\lambda_\theta(u) = e^{B(u) + \boldsymbol{\theta}^\top \mathbf{Z}(u)} \quad (9.42)$$

where $B(u)$ is a known function (the ‘offset’ or ‘baseline’), $\boldsymbol{\theta} = (\theta_1, \dots, \theta_p)$ is the vector of parameters, $\mathbf{Z}(u) = (Z_1(u), \dots, Z_p(u))$ is a vector of covariate functions, and

$$\boldsymbol{\theta}^\top \mathbf{Z}(u) = \theta_1 Z_1(u) + \dots + \theta_p Z_p(u).$$

Note that the model implies that the *logarithm* of the intensity is a linear function of the parameters:

$$\log \lambda_\theta(u) = B(u) + \boldsymbol{\theta}^\top \mathbf{Z}(u) \quad (9.43)$$

The functions B and Z_1, \dots, Z_p could be spatially-varying in any fashion, so this is a very wide and flexible class of models.

The loglinear intensity model has several advantages. The intensity of a point process must be greater than or equal to zero, and this is always satisfied by the loglinear model, regardless of the value of $\boldsymbol{\theta}$ and the values of the functions B and Z_1, \dots, Z_p , because of the exponent in (9.42). In statistical theory the logarithm is the “canonical” transformation of the mean for a Poisson model, and this confers many advantages in theory and practice.

9.5.4.2 Likelihood for loglinear model

For the loglinear intensity the loglikelihood takes the form

$$\log L(\boldsymbol{\theta}) = \sum_{i=1}^n B(x_i) \boldsymbol{\theta}^\top \sum_{i=1}^n \mathbf{Z}(x_i) - \int_W \exp(B(u) + \boldsymbol{\theta}^\top \mathbf{Z}(u)) du. \quad (9.44)$$

This model is a canonically parametrised exponential family [49, p. 113], [223, pp. 23–24]. The loglikelihood (9.44) is a concave function of the parameter $\boldsymbol{\theta}$, and is differentiable with respect to $\boldsymbol{\theta}$, even if the functions B and Z_j are not continuous. If the matrix

$$M = \int_W \mathbf{Z}(u) \mathbf{Z}(u)^\top du$$

is positive definite, then the model is identifiable. If the data are such that $\sum_i Z_j(x_i) \neq 0$ for all j , the MLE exists and is unique. Unless there are further constraints on $\boldsymbol{\theta}$, the MLE is the solution of the *score equations* $\mathbf{U}(\boldsymbol{\theta}) = 0$, where the score function is

$$\mathbf{U}(\boldsymbol{\theta}) = \mathbf{U}(\boldsymbol{\theta}; \mathbf{x}) = \sum_{i=1}^{n(\mathbf{x})} \mathbf{Z}(x_i) - \int_W \mathbf{Z}(u) \lambda_{\boldsymbol{\theta}}(u) du. \quad (9.45)$$

The score is a vector $\mathbf{U}(\boldsymbol{\theta}; \mathbf{x}) = (U_1(\boldsymbol{\theta}; \mathbf{x}), \dots, U_p(\boldsymbol{\theta}; \mathbf{x}))$ with components

$$U_j(\boldsymbol{\theta}; \mathbf{x}) = \sum_{i=1}^{n(\mathbf{x})} Z_j(x_i) - \int_W Z_j(u) \lambda_{\boldsymbol{\theta}}(u) du$$

for $j = 1, \dots, p$.

9.5.4.3 Accuracy of maximum likelihood estimate

In many statistical models, as the sample size increases, the maximum likelihood estimator $\hat{\boldsymbol{\theta}}$ follows a standard pattern of behaviour. It is a consistent estimator (it converges in probability to the true answer $\boldsymbol{\theta}$), and it asymptotically follows a normal distribution, with mean equal to the true parameter vector $\boldsymbol{\theta}$, and variance $\mathcal{I}_{\boldsymbol{\theta}}^{-1}$, where $\mathcal{I}_{\boldsymbol{\theta}}$ is the *Fisher information* matrix. The Fisher information is defined as the variance-covariance matrix of the score:

$$\mathcal{I}_{\boldsymbol{\theta}} = \text{var}_{\boldsymbol{\theta}} [\mathbf{U}(\boldsymbol{\theta}; \mathbf{X})] = \mathbb{E}_{\boldsymbol{\theta}} [\mathbf{U}(\boldsymbol{\theta}; \mathbf{X}) \mathbf{U}(\boldsymbol{\theta}; \mathbf{X})^\top] = \mathbb{E}_{\boldsymbol{\theta}} \left[-\frac{\partial}{\partial \boldsymbol{\theta}} \mathbf{U}(\boldsymbol{\theta}; \mathbf{X}) \right] \quad (9.46)$$

where $\mathbb{E}_{\boldsymbol{\theta}}$ and $\text{var}_{\boldsymbol{\theta}}$ denote the mean and variance when the true parameter value is $\boldsymbol{\theta}$. The negative Hessian matrix

$$H(\boldsymbol{\theta}; \mathbf{x}) = -\frac{\partial}{\partial \boldsymbol{\theta}} \mathbf{U}(\boldsymbol{\theta}; \mathbf{x}) \quad (9.47)$$

is called the *observed information*, while the Fisher information is sometimes called the *expected information*.

For the loglinear Poisson point process model, asymptotic distribution theory is available [278, 209, 20] and the results conform to the pattern described above. Under suitable conditions,⁷ the MLE $\hat{\boldsymbol{\theta}}$ is consistent, asymptotically normal and asymptotically efficient in a ‘large domain’ limiting regime [278, Theorem 11, pp. 135–136], [209, Thm. 2.4, p. 51], [206]. The observed information is

$$H(\boldsymbol{\theta}; \mathbf{x}) = \int_W \mathbf{Z}(u) \mathbf{Z}(u)^\top \lambda_{\boldsymbol{\theta}}(u) du. \quad (9.48)$$

⁷The “suitable conditions” are essentially that every entry in the Fisher information matrix should be large, and that regularity conditions hold.

Since the observed information does not depend on the data \mathbf{x} , it is equal to the Fisher information. The Fisher information is a matrix with entries (on row i , column j)

$$(\mathcal{I}_\theta)_{ij} = \int_W Z_i(u) Z_j(u) \lambda_\theta(u) du.$$

This is the basis of calculations of standard error and confidence intervals for Poisson models in `spatstat`. Asymptotic theory is also available [278, 209] to support the likelihood ratio test.

The integral in the loglinear Poisson process likelihood (9.44) is the Laplace transform of the covariate function \mathbf{Z} , which is not usually available in closed form. Consequently, it is not usually possible to find an exact analytic solution for the maximum likelihood estimate. Some form of numerical approximation is required. Strategies are presented in the next sections.

9.6 Coarse quadrature approximation*

The likelihood function (9.38) of a Poisson point process involves an integral over the spatial window. Except in special cases, this means that the likelihood cannot be computed exactly, but must be approximated numerically.

A good strategy is to set up the approximation so that the approximate likelihood function is equivalent to the likelihood of another, simpler, statistical model which we know how to handle. Then using statistical techniques for the simpler model, we can compute approximate parameter estimates for the original, complex model. This strategy has been used in statistical science since earliest times.

For a point process model, approximation of the likelihood converts the point process into a *regression* model. Lewis [224] and Brillinger [70, 72, 71] showed that the likelihood of a general point process in one-dimensional time, or a Poisson point process in higher dimensions, can be usefully approximated by the likelihood of logistic regression for the discretised process. Asymptotic equivalence was established in [59]. This makes it practicable to fit spatial Poisson point process models of general form to point pattern data [56, 87, 31, 32] by enlisting efficient and reliable software already developed for generalized linear models. Approximation of a stochastic process by a generalized linear model is now commonplace in applied statistics [226, 227, 229, 228].

9.6.1 Berman-Turner device

Numerical quadrature is a simple and efficient computational strategy for numerical integration, in which the integral $\int_W f(u) du$ of some function f is approximated by a weighted sum $\sum_j w_j f(u_j)$ of values of the function at a finite list of “quadrature points” u_j which have “quadrature weights” w_j .

Berman & Turner [56] developed a numerical quadrature method for approximate maximum likelihood estimation for an inhomogeneous Poisson point process. Suppose we approximate the integral in (9.38) by a finite sum using any quadrature rule,

$$\int_W \lambda_\theta(u) du \approx \sum_{j=1}^m \lambda_\theta(u_j) w_j \tag{9.49}$$

where u_j , $j = 1, \dots, m$ are points in W and $w_j > 0$ are quadrature weights summing to $|W|$. This

* Starred sections contain advanced material, and can be skipped by most readers.

yields an approximation to the loglikelihood,

$$\log L(\theta) \approx \sum_{i=1}^{n(\mathbf{x})} \log \lambda_\theta(x_i) - \sum_{j=1}^m \lambda_\theta(u_j) w_j. \quad (9.50)$$

Berman and Turner observed that if the list of points $\{u_j, j = 1, \dots, m\}$ includes all the data points $\{x_i, i = 1, \dots, n\}$, then we can rewrite (9.50) as a sum over quadrature points:

$$\log L(\theta) \approx \sum_{j=1}^m (I_j \log \lambda_j - w_j \lambda_j) \quad (9.51)$$

where $\lambda_j = \lambda_\theta(u_j)$ and

$$I_j = \begin{cases} 1 & \text{if } u_j \text{ is a data point, } u_j \in \{x_1, \dots, x_n\} \\ 0 & \text{if } u_j \text{ is a dummy point, } u_j \notin \{x_1, \dots, x_n\}. \end{cases} \quad (9.52)$$

Berman and Turner [56] re-expressed this as

$$\log L(\theta) \approx \sum_{j=1}^m (y_j \log \lambda_j - \lambda_j) w_j \quad (9.53)$$

where $y_j = I_j/w_j$, and noted that the right side of (9.53), for fixed \mathbf{x} , is formally equivalent to the weighted loglikelihood of independent Poisson variables $Y_j \sim \text{Poisson}(\lambda_j)$ taken with weights w_j . The expression (9.53) can therefore be maximised using standard software for fitting Generalised Linear Models [243].

Later it was pointed out [244] that (9.51) is the *unweighted* loglikelihood of independent Poisson variables $I_j \sim \text{Poisson}(w_j \lambda_j)$.

The key reason for adopting this approach is that the use of standard statistical packages rather than *ad hoc* software confers great advantages in applications. Modern statistical packages have a convenient notation for statistical models [3, 77, 324] which makes it very easy to specify and fit a wide variety of models. Algorithms in the package may allow one to fit very flexible model terms such as the smooth functions in a generalised additive model [180]. Interactive software allows great freedom to reanalyse the data. The fitting algorithms are typically more reliable and stable than in home-grown software.

In summary, the procedure is as follows.

1. Generate a set of dummy points, and combine it with the data points x_i to form the set of quadrature points u_j ;
2. Compute the quadrature weights w_j ;
3. Form the indicators I_j as in (9.52) and calculate $y_j = I_j/w_j$;
4. Compute the (possibly vector) values $\mathbf{z}_j = \mathbf{Z}(u_j)$ of the covariates at each quadrature point;
5. Invoke standard model-fitting software, specifying that the model is a loglinear Poisson regression

$$\log \lambda_j = \boldsymbol{\theta}^\top \mathbf{z}_j$$

to be fitted to the responses y_j and covariate values \mathbf{z}_j , with weights w_j .

In our implementation, step 5 is performed by the model-fitting function `glm()` in R.

The coefficient estimates returned by `glm()` give the (approximate) MLE $\hat{\boldsymbol{\theta}}$ of $\boldsymbol{\theta}$.

The estimates of standard errors returned by `glm()` are also valid, because both the weighted

loglinear Poisson regression and the loglinear Poisson point process model have the property that the Fisher information is equal to the negative Hessian of the loglikelihood, and these are approximately equal by (9.53).

Additionally `glm()` returns the deviance D of the fitted model; this is related to the loglikelihood of the fitted model by

$$-\log L(\hat{\theta}; \mathbf{x}) = \frac{D}{2} + \sum_{j=1}^m I_j \log w_j + n(\mathbf{x}); \quad (9.54)$$

where the sum is effectively over data points only.

Conveniently, the null model $\lambda_j \equiv \lambda$ in the weighted loglinear Poisson regression corresponds to the uniform Poisson point process with intensity λ . The MLE is $\hat{\lambda} = n(\mathbf{x})/\sum_j w_j = n(\mathbf{x})/|W|$ with corresponding loglikelihood $\log L(\hat{\lambda}) = n(\mathbf{x})[\log n(\mathbf{x}) - \log |W| - 1]$.

Note that this formulation assumes $\lambda(u)$ is positive everywhere. Zero values are also permissible, provided the set of zeroes does not depend on θ . Thus we formally allow negative infinite values for $Z(u)$. In the approximation (9.50) all points u_j with $\lambda(u_j) = 0$ will be dummy points. Their contribution is zero and so they should be omitted in all contexts.

9.6.1.1 Design of quadrature schemes

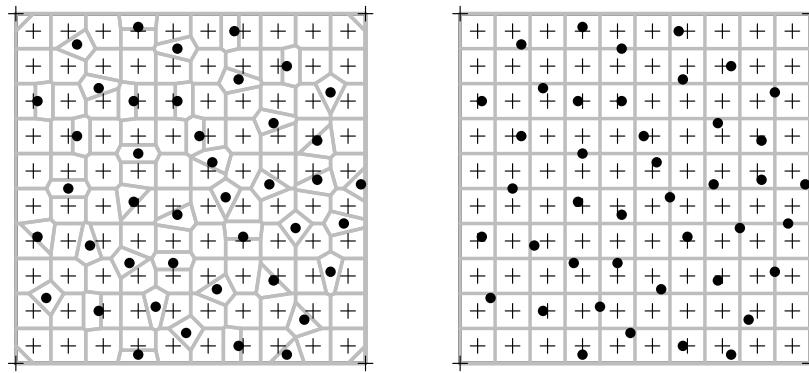


Figure 9.14: Quadrature schemes for a point pattern dataset. Data points (\bullet), dummy points (+) and tile boundaries (grey lines). *Left:* Dirichlet tiles and weights. *Right:* rectangular tiles, “counting weights”. Coarsely-spaced example for illustration only.

Berman & Turner [56] used the Dirichlet tessellation or Voronoi diagram [262] to generate quadrature weights. The data points are augmented by a list of dummy points, then the Dirichlet tessellation of the combined set of points is computed as sketched in the left panel of Figure 9.14. The quadrature weight w_j associated with a (data or dummy) point u_j is the area of the corresponding Dirichlet tile.

It is computationally cheaper to use the counting weights proposed in [31]. In the simplest form we assign the same proportion of the total window area to all the points, i.e. $w_j = |W|/m$. As a refinement of the method we may e.g. split the window in two and divide the area $a = |W|/2$ equally among the m_1 points in the first half, so $w_j = a/m_1$ and likewise for the second half. This naturally generalises to any subdivision of the window. The right panel of Figure 9.14 shows an example where W is partitioned into tiles T_j of equal area a . In this case each tile contains only one or two points so the weights will be either $a/1$ or $a/2$, but more generally the weight would be $w_j = a/m_j$ where m_j is the number of (dummy or data) points in the tile T_j . We call these the *counting weights*.

Discuss the case where covariate values are only available at certain sample points.

9.6.1.2 Quadrature schemes in spatstat

The `spatstat` function `ppm()` fits a point process model to an observed point pattern. By default, it uses Berman-Turner quadrature approximation.

In normal use, the quadrature scheme is generated automatically from the data point pattern, so the user does not need to think about it. However, the default quadrature scheme may give unsatisfactory results in some cases. The default rules for building a quadrature scheme are designed so that `ppm()` will execute quickly, rather than guaranteeing a highly accurate result.

Users who wish to override the defaults may modify the rules for building a quadrature scheme, or may even provide their own quadrature scheme.

Quadrature schemes are created by the function `quadscheme`:

```
> quadscheme(data, dummy, ..., method="grid")
```

The arguments `data` and `dummy` specify the data and dummy points, respectively.

<code>quadscheme(X)</code>	default for data pattern X
<code>quadscheme(X, nd=128)</code>	dummy points in 128×128 grid
<code>quadscheme(X, eps=0.1)</code>	dummy point spacing 0.1 units
<code>quadscheme(X, random=TRUE)</code>	stratified random dummy points
<code>quadscheme(X, quasi=TRUE)</code>	quasirandom dummy points
<code>quadscheme(X, D)</code>	data X, dummy D

Table 9.3: Typical options for controlling the dummy points in a quadrature scheme (options are passed to `default.dummy()`).

For the **dummy points**, there is a sensible default, provided by `default.dummy(X)`. Table 9.3 shows some of the important options for modifying the default dummy pattern. By default, the dummy points are arranged in a rectangular grid; recognised arguments include `nd` (the number of grid points in the horizontal and vertical directions) and `eps` (the spacing between dummy points). If `random=TRUE`, a systematic random (also called stratified random) pattern of dummy points is generated instead. If `quasi=TRUE`, a quasirandom pattern of dummy points is generated.

Alternatively the dummy point pattern may be specified arbitrarily and given in any format recognised by `as.ppp`. There are also functions for creating dummy patterns including `corners`, `gridcentres`, `stratrand` and `spokes`.

The **quadrature weights** are determined by further arguments to `quadscheme()`. If `method = "grid"` (the default) the window is divided into an `ntile[1]` by `ntile[2]` grid of rectangular tiles, and the “counting weights” are applied: the weight for each quadrature point is the area of a tile divided by the number of quadrature points in that tile. By default the values `ntile` and `nd` are the same so all tiles (except the corners) contain exactly one dummy point as illustrated in the Right panel of Figure 9.14. If `method="dirichlet"`, the quadrature points (both data and dummy) are used to construct the Dirichlet tessellation. The quadrature weight of each point is the area of its Dirichlet tile inside the quadrature region.

<pre>quadscheme(X, ..., method="d") specify Dirichlet weights quadscheme(X, ..., ntile=8) 8 × 8 array of counting tiles</pre>

Table 9.4: Typical options for controlling the quadrature weights.

A quadrature scheme (consisting of the original data point pattern, an additional pattern of dummy points, and a vector of quadrature weights for all these points) is represented by an object of class "quad". In principle, the user could create one of these objects from scratch, using the creator function `quad()`.

9.6.1.3 Gorilla nests example

In Section 9.3.4.1 we fitted a simple model to the `gorillas` data in which the intensity is constant inside each region defined by vegetation type:

```
> ppm(gor ~ vege, data=gex)
Nonstationary Poisson process

Trend formula: ~vege

Fitted trend coefficients:
(Intercept)    vegeColo    vegeGras    vegePrim    vegeSeco    vegeTran
              2.3213     2.0842    -0.7719     2.1174     1.1367     1.6163

                               Estimate   S.E. CI95.lo CI95.hi Ztest   Zval
(Intercept)    2.3213 0.1060  2.1135  2.5291 *** 21.899
vegeColo       2.0842 0.5870  0.9337  3.2347 *** 3.551
vegeGras      -0.7719 0.2475 -1.2569 -0.2869 ** -3.119
vegePrim       2.1174 0.1151  1.8918  2.3430 *** 18.396
vegeSeco       1.1367 0.2426  0.6612  1.6122 *** 4.685
vegeTran       1.6163 0.2647  1.0976  2.1351 *** 6.107
```

Fitting this model by maximum likelihood is equivalent to estimating the intensities using quadrat counts. To check this, we convert the pixel image to a tessellation and apply quadrat counting:

```
> vt <- tess(image=gex$vege)
> intensity(quadratcount(gor, tess=vt))
tile
  Dist  Colo  Gras  Prim  Seco  Tran
10.201 69.155 4.781 84.012 32.651 50.922
```

The discrepancies are due to discretisation of the integral in `ppm()` in the Berman-Turner technique. Better agreement can be obtained by increasing the density of dummy points, for example, using the parameter `nd`:

```
> fitveg2 <- ppm(gor ~ vege-1, data=gex, nd=256)
> exp(coef(fitveg2))
```

Exact agreement (up to numerical rounding error) can be obtained by using a quadrature scheme with one point at each pixel of the covariate image. This is constructed by the command `pixelquad()`:

```
> Q <- pixelquad(gor, gex$vege)
> fitveg3 <- ppm(Q ~ vege-1, data=gex)
> exp(coef(fitveg3))
vegeDist vegeColo vegeGras vegePrim vegeSeco vegeTran
 10.201    69.155     4.781    84.012    32.651    50.922
```

9.7 Pixel approximation*

9.7.1 Pixel counts

Another quadrature strategy for approximating the Poisson process likelihood is to divide the window W into small pixels of equal area a . The integral over the window W is then approximated by a sum over pixels:

$$\int_W \lambda_\theta(u) du \approx \sum_j \lambda_\theta(u_j) a \quad (9.55)$$

where u_j is the centre of the j th pixel. We also discard the exact locations of the data points, and effectively move each data point to the centre of the pixel which contains it. Thus we approximate the sum over data points by a sum over pixels,

$$\sum_i \log \lambda_\theta(x_i) \approx \sum_j n_j \log \lambda_\theta(u_j) \quad (9.56)$$

where n_j is the number of data points falling in the j th pixel. Collecting (9.55) and (9.56), we approximate the true loglikelihood (9.38) by

$$\begin{aligned} \log L(\theta) &\approx \sum_j [n_j \log \lambda_\theta(u_j) - \lambda_\theta(u_j) a] \\ &= \sum_j (n_j \log \lambda_j - \lambda_j a) \end{aligned} \quad (9.57)$$

where $\lambda_j = \lambda_\theta(u_j)$.

Readers familiar with standard regression models will notice that the right-hand side of (9.57) has the same form as the loglikelihood of independent Poisson random variables N_j with means $a\lambda_j$. This was to be expected, because the pixel counts N_j are independent Poisson random variables (by standard properties of the Poisson process) and the value $a\lambda_j$ is an approximation to the true mean of N_j .

This observation is important because it means that we can maximise the *approximate* loglikelihood (9.57) using standard statistical software for fitting Poisson regression models. The approximation has effectively reduced the problem to a standard statistical model-fitting task.

* Starred sections contain advanced material, and can be skipped by most readers.

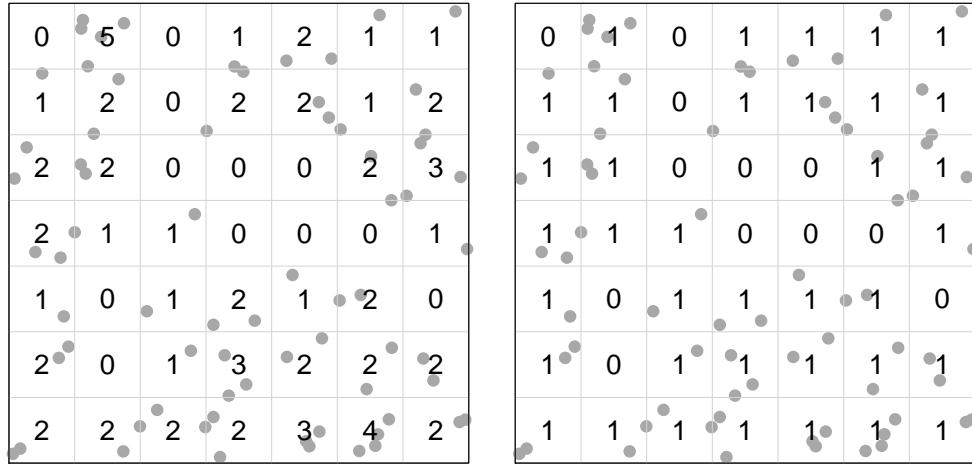


Figure 9.15: Pixel counts (*Left*) and pixel presence-absence indicators (*Right*) for the same point pattern.

In the important case of a loglinear Poisson point process model (9.42), we have

$$\lambda_j = \lambda_{\theta}(u_j) = \exp(B(u_j) + \theta^{\top} \mathbf{Z}(u_j)) = \exp(b_j + \theta^{\top} \mathbf{z}_j) \quad (9.58)$$

where $b_j = B(u_j)$ and $\mathbf{z}_j = \mathbf{Z}(u_j)$. So the right hand side of (9.57) is the loglikelihood of independent Poisson random variables N_j with means

$$\mu_j = a\lambda_j = \exp(\log a + b_j + \theta^{\top} \mathbf{z}_j) = \exp(\log o_j + \theta^{\top} \mathbf{z}_j) \quad (9.59)$$

where $o_j = \log a + b_j$. This is, *loglinear Poisson regression* with regression covariates \mathbf{z}_j and offset $o_j = b_j + \log a$.

A practical strategy for fitting a loglinear Poisson point process model is therefore:

1. divide the window W into a fine grid of pixels of area a ;
2. count the number n_j of data points falling in each pixel j ;
3. evaluate the offset term $o_j = \log a + B(u_j)$ and the covariate vector $\mathbf{z}_j = \mathbf{Z}(u_j)$ at the centre u_j of each pixel j ;
4. use standard statistical software (such as the `glm()` function in R) to fit (by maximum likelihood) a loglinear Poisson regression model with responses n_j , regression covariates \mathbf{z}_j and offsets o_j ;
5. the fitted coefficients $\hat{\theta}$ for the loglinear Poisson regression are the approximate maximum likelihood estimates $\hat{\theta}$ for the loglinear Poisson point process model (9.42).

The adequacy of this approximation depends on the spatial regularity of the function \mathbf{Z} . In the most optimistic case, \mathbf{Z} is *constant within each pixel*,

$$\mathbf{Z}(u) = \mathbf{z}_j \text{ for } u \in S_j, \quad (9.60)$$

so that (9.55) and (9.56) are exact equations rather than approximations, and (9.57) is the exact loglikelihood. Covariates of this kind include indicators of broad geological classification, and national boundaries. In the next best case, \mathbf{Z} is a smooth function⁸. Then (9.55), (9.56) and (9.57) are

⁸ \mathbf{Z} should be a Lipschitz function, i.e. $\|\mathbf{Z}(u) - \mathbf{Z}(v)\| \leq C\|u - v\|$ where $C < \infty$ is constant.

good approximations when the pixels are small. Examples include distance transforms (e.g. $\mathbf{Z}(u)$ is the distance from u to the nearest geological fault), geographical coordinates, and kernel-smoothed geochemical assay values. In the least optimistic case, \mathbf{Z} is a discontinuous function, such as the indicator of a very irregular spatial domain such as a rock outcrop. The approximation (9.57) can then give rise to considerable bias, even for quite small pixel sizes.

9.7.2 Pixel presence-absence indicators

A further simplification is to replace the counts n_j by the *presence-absence indicators*

$$I_j = \begin{cases} 1 & \text{if } n_j \geq 1 \\ 0 & \text{if } n_j = 0 \end{cases}$$

which tell us whether data points were present ($I_j = 1$) or absent ($I_j = 0$) in each pixel. See the right panel of Figure 9.15.

If p_j denotes the probability that there are any data points in pixel j , we have $p_j = \mathbb{P}\{I_j = 1\} = \mathbb{P}\{N_j \geq 1\} = 1 - e^{-a\lambda_j}$. The presence-absence indicators are independent random variables, so their loglikelihood is

$$\sum_j (I_j \log p_j + (1 - I_j) \log(1 - p_j)).$$

For a loglinear Poisson point process model (9.42) substituting (9.59) we get

$$p_j = 1 - \exp(-e^{o_j + \boldsymbol{\theta}^\top \mathbf{z}_j})$$

and inverting this relationship gives

$$\log(-\log(1 - p_j)) = o_j + \boldsymbol{\theta}^\top \mathbf{z}_j \quad (9.61)$$

where $o_j = b_j + \log a$. The transformation $\log(-\log(1 - p))$ is called the *complementary log-log link*. Thus, the presence-absence indicators follow a *complementary log-log regression* with offset o_j and regression covariates \mathbf{z}_j as before.

This model is an exponential family, but is not canonically parametrised. The conditions for uniqueness of the MLE are the standard conditions for this discrete model [243].

9.7.3 Logistic regression

There is a further simplification when the pixels are so small that they have negligible chance of containing more than one data point. If we impose the condition that $N_j \leq 1$ for all pixels j , then the conditional probability that $N_j = 1$ is

$$p_j^* = \mathbb{P}\{N_j = 1 \mid N_j \leq 1\} = \frac{\mathbb{P}\{N_j = 1\}}{\mathbb{P}\{N_j \leq 1\}} = \frac{\mathbb{P}\{N_j = 1\}}{\mathbb{P}\{N_j = 0\} + \mathbb{P}\{N_j = 1\}}$$

Using the formula for the Poisson probabilities (REF) this is

$$p_j = \frac{\mu_j \exp(-\mu_j)}{\exp(-\mu_j) + \mu_j \exp(-\mu_j)} = \frac{\mu_j}{1 + \mu_j}$$

and similarly the conditional probability that $N_j = 0$ is

$$1 - p_j^* = \frac{1}{1 + \mu_j}.$$

We find that

$$\frac{p_j^*}{1-p_j^*} = \mu_j$$

so that the *odds* of presence (that is, the ratio of the presence probability divided by the absence probability) is equal to μ_j . For a loglinear Poisson process, substituting (9.59) and taking the logarithm gives

$$\log\left(\frac{p_j^*}{1-p_j^*}\right) = o_j + \boldsymbol{\theta}^\top \mathbf{z}_j \quad (9.62)$$

saying that the *log odds* of presence is equal to a linear function of the parameters. The transformation $\log(p/(1-p))$ is called the *logistic link*, and (9.62) states that the presence-absence indicators follow a *logistic regression* with offset o_j and covariates \mathbf{z}_j as before.

Thus, *conditional on $N_j \leq 1$ for all j , the responses Y_j satisfy a logistic regression with the same linear predictor*. One consequence is that spatial logistic regression and complementary log–log regression will be approximately equivalent if $\mathbb{P}\{N_j \geq 2 \text{ for some } j\}$ is negligible.

A rule-of-thumb for the relative efficiency of the MLE of $\boldsymbol{\theta}$ based on the presence/absence indicators Y_j , relative to the MLE based on the pixel counts N_j was stated in [20]. If there is only one real covariate, the relative efficiency is approximately

$$\frac{I_b(\boldsymbol{\theta})}{I_d(\boldsymbol{\theta})} \approx \frac{\bar{\mu}}{e^{\bar{\mu}} - 1}$$

where $\bar{\mu} = \frac{1}{N} \sum \mu_j$ is the average expected number of points per pixel. Thus the loss of efficiency should be tolerable as long as $\bar{\mu} < 0.2$, say. For pixels of equal area, $\bar{\mu} = \frac{1}{N} \int_W \lambda_{\boldsymbol{\theta}}(u) du$ can be estimated by $n(\mathbf{X})/N$, the average number of data points per pixel.

9.7.4 Interpretation of pixel logistic regression

Pixel-based spatial logistic regression for point events was pioneered in geology by F.P. Agterberg [1] on the suggestion of J.W. Tukey [319]. It was later independently rediscovered in archaeology [294, 179, 210, 211] and is now a standard technique in GIS applications [65].

Despite its popularity, pixel logistic regression does not seem to be universally well understood. Some writers describe it as a “nonparametric” technique [212, p. 24]. The interpretation of the fitted parameters is widely held to be obscure [338, p. 175] and is typically based only on the sign of the slope parameters, that is, parameters other than the intercept [158, pp. 405–407].

Contrary to these statements, there is a clear physical meaning for the model parameters of (pixel) logistic regression. When pixels are sufficiently small, the model is approximately a Poisson point process with loglinear intensity (9.6). The fitted parameters of the logistic regression have a direct interpretation as the parameters of the Poisson point process (after adjusting the intercept term by subtracting the logarithm of pixel area).

Theoretical understanding of the Poisson model is so deep that we are able to make numerous predictions about quantities of interest, such as the expected number of points in a target region, the probability of exactly k points in a target region, the distribution of distance from a fixed starting location to the nearest random point, and so on. See Sections 9.2.1 and 9.4.3.

Need to say something about “weights of evidence”

9.7.5 Floating fragments of text

Text cut-and-pasted from [20].

Of course, counting points inside pixels is a special case of aggregating spatial data into discrete

geographical areas. This has been studied extensively in epidemiology and in ecological and environmental statistics [141, 331, 329, 328]. However, most epidemiological research deals with geographical areas that are predetermined regions of appreciable size. In pixel-based regression, the pixel grid is chosen arbitrarily, and is usually very fine, which leads to a different set of methodological and practical problems.

When the pixel grid can be chosen arbitrarily, important questions include the equivalence of models fitted using different pixel grids (the “modifiable area unit problem” [263] or “change-of-support” [159, 46, 104]), the relation between discrete and continuous spatial models (“ecological fallacy” [287]) and bias due to aggregation over pixels (“ecological bias” [328, 329] or aggregation bias [116, 9]). GIS literature seems unclear on these questions, except to acknowledge that the interpretation of the fitted probabilities and model parameters clearly depends on the pixel size. The interpretation of fitted parameters is often said to be obscure [338, p. 175] and is typically based only on the sign of the slope parameters [158, pp. 405–407].

One of our unexpected findings is that it may be *impossible* to reconcile two spatial logistic regression models that were fitted to the same spatial point pattern data using different pixel grids (Section ??). Two such models are logically incompatible except in simple cases. Thus, there is no physical process which satisfies a logistic regression model whenever it is discretised on any pixel grid. The implication is that two research teams who apply spatial logistic regression to the same data, but using different pixel sizes, may obtain results that cannot be reconciled. A spatial logistic regression, fitted to data from a fully-explored survey region, cannot always be extrapolated to make predictions in a prospective exploration region.

In the GIS community the accepted practical remedy for such inconsistency is to take a very fine pixel grid. As the pixel size tends to zero, spatial logistic regressions with the same form of linear predictor are asymptotically equivalent, provided the linear predictor includes an intercept term.

However, the limit of spatial logistic regressions as pixel size tends to zero is a Poisson point process. Essentially this follows from the assumption of independent responses in logistic regression. A more precise result is given in [20] using distributional approximation. Furthermore, the use of the logistic link implies that the intensity of the limiting Poisson point process is a *loglinear* function of the spatial covariates, termed a *modulated* Poisson process by Cox [99]. Thus, contrary to the general view in the GIS literature, there *is* a clear physical meaning for the model parameters of spatial logistic regression, when pixels are small (cf. [33]). The Poisson model allows prediction of several quantities of interest, as we discuss in a companion paper on applications [17].

To avoid technical problems, the pixels should be very small. However, for small pixel size, the logistic regression model is approximately a Poisson point process [20]. Essentially this follows from the assumption of independent responses in logistic regression. ref chapter 5. Furthermore, the use of the *logistic* link implies that the intensity of the limiting Poisson point process is a *loglinear* function of the spatial covariates. Thus, contrary to the general view in the GIS literature, logistic regression implies a very specific model, and there is a clear physical meaning for the model parameters. The Poisson model allows prediction of several quantities of interest.

Asymptotic normality

Note that complementary log–log regressions on different pixel grids are compatible. Under the assumptions of Section ??, complementary log–log regressions on different pixel grids satisfy the multiplicative property (??) exactly.

A popular technique for analysing point pattern data in Geographical Information Systems is *spatial logistic regression* [319, 1, 65]. The spatial domain is divided into a fine grid of pixels;

each pixel is assigned the value $y = 1$ if it contains at least one data point, and $y = 0$ otherwise. Then logistic regression is used to model the presence probability $p = P(Y = 1)$ as a function of a covariate x in the form

$$\log \frac{p}{1-p} = \beta_0 + \beta_1 x$$

where β_0, β_1 are parameters to be estimated. Similarly for multiple covariates and so on.

This is asymptotically equivalent to fitting a Poisson point process with loglinear intensity (11.18) [20, 334].

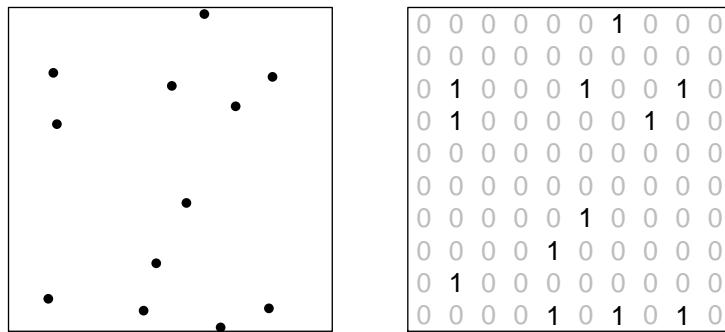


Figure 9.16: Another try at the presence-absence picture

9.7.6 Implementation in spatstat

The `spatstat` package provides facilities for performing spatial logistic regression, for comparison purposes. Models are fitted using the command `slrm()`.

```
> X <- rotate(copper$SouthPoints, pi/2)
> L <- rotate(copper$SouthLines, pi/2)
> D <- distfun(L)
> fit <- slrm(X ~ D)
> fit
Fitted spatial logistic regression model
Formula: X ~ D
Fitted coefficients:
(Intercept) D
-4.72338 0.07811
```

`slrm()` produces a “fitted spatial logistic regression model” object of class “`slrm`” (*pronounced “SLURM”*). Methods for this class include `print()`, `plot()`, `predict()`, `coef()`, `fitted()`, `update()`, `terms()`, `formula()`, `anova()` and `logLik()`. You can also use `step()` for model selection.

Less well-developed than `ppm`

9.7.7 Technical problems

Approximation error can be controlled using a fine discretisation, but this leads to numerical instability and the failure of the delta-method approximation [181], arising because the overwhelming majority of pixels do not contain a data point.

This leads to next strategy...

9.8 Conditional logistic regression*

9.8.1 Logistic regression with random dummy points

Instead of using a dense grid of dummy points or pixels in an effort to achieve accurate approximation of the likelihood, another strategy is to generate a smaller number of dummy points at *random* locations. Random sampling can produce unbiased estimates of the score using much fewer function evaluations, at the expense of increased variability due to the randomisation.

Assume, as before, that the data points come from a Poisson process \mathbf{X} with intensity function $\lambda_\theta(u)$ where θ is to be estimated. Suppose that we generate dummy points at random, according to a Poisson process \mathbf{D} with known intensity function $\delta(u) > 0$, independently of the data points. Combining these two types of points, the superposition $\mathbf{Y} = \mathbf{X} \cup \mathbf{D}$ is again a Poisson point process with intensity function

$$\kappa(u) = \lambda_\theta(u) + \delta(u).$$

For each point $y_i \in \mathbf{Y}$, let $I_i = \mathbf{1}\{y_i \in \mathbf{X}\}$ be the indicator that equals 1 if y_i was a data point, and 0 if it was a dummy point.

Let us now *condition on* \mathbf{Y} , so that we regard the points y_i as fixed locations. The data now consist only of the data/dummy indicators I_1, \dots, I_m where $m = n(\mathbf{X}) + n(\mathbf{D})$ is the total number of (data and dummy) points. Conditional on location, the indicators I_i are independent random variables, with probabilities

$$\begin{aligned}\mathbb{P}\{I_i = 1\} &= \frac{\lambda_\theta(y_i)}{\lambda_\theta(y_i) + \delta(y_i)}. \\ \mathbb{P}\{I_i = 0\} &= \frac{\delta(y_i)}{\lambda_\theta(y_i) + \delta(y_i)}\end{aligned}$$

so that the odds are

$$\frac{\mathbb{P}\{I_i = 1\}}{\mathbb{P}\{I_i = 0\}} = \frac{\lambda_\theta(y_i)}{\delta(y_i)}.$$

If the intensity of \mathbf{X} is loglinear, $\lambda_\theta(u) = \exp(B(u) + \theta^\top \mathbf{Z}(u))$, then

$$\frac{\mathbb{P}\{I_i = 1\}}{\mathbb{P}\{I_i = 0\}} = \frac{\exp(B(y_i) + \theta^\top \mathbf{Z}(y_i))}{\delta(y_i)}$$

so that the log odds are

$$\log \frac{\mathbb{P}\{I_i = 1\}}{\mathbb{P}\{I_i = 0\}} = B(y_i) + \theta^\top \mathbf{Z}(y_i) - \log \delta(y_i).$$

The indicators I_i therefore satisfy a *logistic regression*. The loglikelihood is

$$\begin{aligned}\log L(\theta; \mathbf{x}) &= \sum_i I_i \log(p_i) + (1 - I_i) \log(1 - p_i) \\ &= \sum_{u \in \mathbf{x}} \log \frac{\lambda_\theta(u)}{\lambda_\theta(u) + \delta(u)} + \sum_{u \in \mathbf{d}} \log \frac{\delta(u)}{\lambda_\theta(u) + \delta(u)}.\end{aligned}\tag{9.63}$$

* Starred sections contain advanced material, and can be skipped by most readers.

This can be used to fit the model

Commonly used in GIS

This connection has many advantages. Estimation can be implemented straightforwardly using standard software for generalized linear models. The loglikelihood (9.63) is a concave function of θ , and conditions for existence and uniqueness of the maximum are well known [299][24].

9.8.2 Conditional logistic regression as a point process method

The score is

$$\mathbf{U}(\theta; \mathbf{x}, \mathbf{d}) = \sum_{u \in \mathbf{x}} \frac{\delta(u)\mathbf{Z}(u)}{\lambda_\theta(u) + \delta(u)} - \sum_{u \in \mathbf{d}} \frac{\lambda_\theta(u)\mathbf{Z}(u)}{\lambda_\theta(u) + \delta(u)}. \quad (9.64)$$

By Campbell's formula (REF) the expectation of the first sum in (9.64) over all outcomes of \mathbf{X} is

$$\mathbb{E} \sum_{u \in \mathbf{X}} \frac{\delta(u)\mathbf{Z}(u)}{\lambda_\theta(u) + \delta(u)} = \mathbb{E} \int_W \frac{\lambda_\theta(u)\delta(u)\mathbf{Z}(u)}{\lambda_\theta(u) + \delta(u)} du \quad (9.65)$$

and for the second sum in (9.64) the expectation over all outcomes of \mathbf{D} is

$$\mathbb{E} \sum_{u \in \mathbf{D}} \frac{\lambda_\theta(u, X)\mathbf{Z}(u)}{\lambda_\theta(u, X) + \delta(u)} = \mathbb{E} \int_W \frac{\lambda_\theta(u)\delta(u)\mathbf{Z}(u)}{\lambda_\theta(u) + \delta(u)} du \quad (9.66)$$

It follows that $\mathbb{E}_\theta \mathbf{U}(\theta; \mathbf{X}, \mathbf{D}) = 0$ where the expectation is taken over both \mathbf{X} and \mathbf{D} . The logistic score (9.64) is an unbiased estimating function.

What to say about asymptotics and variance?

See [24].

If we rearrange (9.64) as

$$\mathbf{U}(\theta; \mathbf{x}, \mathbf{d}) = \sum_{u \in \mathbf{x}} \mathbf{Z}(u) - \sum_{u \in \mathbf{x} \cup \mathbf{d}} \frac{\lambda_\theta(u)}{\lambda_\theta(u) + \delta(u)} \mathbf{Z}(u) \quad (9.67)$$

and apply Campbell's formula to the last term in (9.67), we obtain

$$\mathbb{E} \sum_{u \in \mathbf{x} \cup \mathbf{d}} \frac{\lambda_\theta(u)}{\lambda_\theta(u) + \delta(u)} \mathbf{Z}(u) = \int_W \mathbf{Z}(u) \lambda_\theta(u) du. \quad (9.68)$$

Thus, if the last term in (9.67) is replaced by its expectation, the score of the full point process likelihood is obtained. Hence the score of the conditional logistic regression may be viewed as a Monte Carlo approximation of the full maximum likelihood score to which it converges (in mean square) when $\inf_{u \in W} \delta(u) \rightarrow \infty$.

9.8.3 Logistic method in spatstat

Explain `method=logi`

This technique is roughly equivalent in speed to maximum pseudolikelihood, but is believed to be less biased. Because it is less biased, the default settings for `method='logi'` specify a relatively small number of dummy points, so that this method is the fastest, in practice.

Note that `method='logi'` and `method='ho'` involve randomisation, so that the results are subject to random variation.

Explain about options for `quadscheme.logi()`

9.8.4 Logistic regression in case-control studies

The Chorley-Ribble data are an example of a spatial case-control study. We have two types of points: cases (of the disease of interest) and controls (points which serve as a sample from the susceptible population).

As explained in Section 9.2.5 the susceptible population has some unknown spatial density $s(u)$ (people per square kilometre) and the null model of constant risk assumes that the cases are a Poisson process with intensity $\lambda(u) = ps(u)$ (cases per square kilometre) where p is the disease risk per head of population. Common alternative models assume that the cases are Poisson with intensity $\lambda(u) = r(u, \theta)s(u)$ where $r(u, \theta)$ is a spatially-varying risk function.

Explain that logistic regression estimates $r(u, \theta)/q$ where q is the sampling fraction of controls.

This is a well-known principle in epidemiology. Diggle and Rowlingson [129] argue advantages of this approach in spatial context include not having to estimate the population density.

9.9 Non-loglinear models

Until this point in the chapter, we have been concerned exclusively with fitting *loglinear* Poisson point process models. These models have a specially advantageous structure.

However there are many applications where we need to fit a model which does not have this loglinear form (with respect to the parameters!). An important example is in spatial epidemiology (refer Chorley-Ribble example) where the effect of a pollution source may not be easily expressible in loglinear form.

This section covers methods for fitting a *general* (not loglinear) Poisson point process model.

It describes the model-fitting functions `ippm()` (Newton method for differentiable models) and `profilepl()` (brute force for general models eg thresholds).

9.9.1 Profile likelihood

In many intensity models, *some* of the parameters appear in loglinear form, while other parameters do not. Such a model is of the form

$$\lambda_{\theta}(u) = \exp(\varphi^{\top} \mathbf{Z}(u, \psi)) \quad (9.69)$$

where $\theta = (\varphi, \psi)$ is a partition of the entries of the parameter vector θ into *regular parameters* φ which appear in loglinear form in (9.69) and *irregular parameters* ψ which do not appear in loglinear form.

If we fix the values of the irregular parameters ψ , then equation (9.69) is loglinear in the remaining parameters φ . Considered as a model with parameters φ only, this is a loglinear Poisson point process model, which can be fitted using the methods described in previous Sections.

Thus, for any chosen value of ψ , the likelihood $L(\theta) = L(\varphi, \psi)$ can easily be maximised over all possible values of φ , giving us the *profile maximum likelihood estimate*

$$\hat{\varphi}(\psi) = \operatorname{argmax}_{\varphi} L(\varphi, \psi). \quad (9.70)$$

The achieved maximum value of the likelihood for a given value of ψ is called the *profile likelihood*:

$$pL(\psi) = \max_{\varphi} L(\varphi, \psi). \quad (9.71)$$

The maximum likelihood estimate of θ can be obtained by maximising the *profile* likelihood over

ψ . That is, if $\widehat{\psi} = \operatorname{argmax}_{\psi} pL(\psi)$ is the value of the irregular parameters that maximises the profile likelihood, then $\widehat{\theta} = (\widehat{\varphi}(\widehat{\psi}), \widehat{\psi})$ is the maximum likelihood estimate of θ .

9.9.2 Maximising profile likelihood by brute force

A simple strategy for maximising the profile likelihood is to evaluate $pL(\psi)$ at a grid of test values of ψ and simply choose the value of ψ which yields the maximum.

The function `profilepl` performs this “brute force” maximisation. Its syntax is

```
> profilepl(s, f, ...)
```

The argument `s` is a data frame containing values of the irregular parameters over which the profile likelihood will be computed. The argument `f` should be set to `Poisson()` for fitting Poisson processes. Additional arguments `...` are passed to `ppm()` to fit the model.

For example, the `nztrees` dataset appears to have a line of trees (perhaps a planted avenue) at right boundary. We fit a threshold model where the intensity is different on either side of the line $x = a$. The threshold value a can be found using `profilepl()`:

```
> thresh <- function(x,y,a) { x < a }
> df <- data.frame(a=1:152)
> nzfit <- profilepl(df, Poisson, nztrees ~ thresh,
  eps=0.5)

> nzfit
Profile log pseudolikelihood
for model: ppm(nztrees ~ thresh, eps = 0.5, interaction = Poisson)
fitted with rbord = 0
Interaction: Poisson
Irregular parameter: a in [1, 152]
Optimum value of irregular parameter: a = 150
```

The result, `nzfit`, is an object of class "profilepl". Printing the object gives information about the fitting procedure, the fitted value of the irregular parameter, and the fitted values of the regular parameters. The `plot()` method generates a plot of the profile likelihood, as shown in Figure 9.17. The plot shows a clear preference for a threshold near the right-hand edge of the field.

Figure 9.17 shows the typical behaviour of profile likelihood in many applications. The profile likelihood is not continuous as a function of a : it has discrete jumps at the values a_i which coincide with the horizontal coordinates of data points. Between these jumps, the profile likelihood is continuous and differentiable.

The fitted model can be extracted as `as.ppm(nzfit)`.

9.9.3 Maximising profile likelihood by Newton’s method

If the likelihood is differentiable with respect to the irregular parameters, then a more efficient technique for maximisation is Newton’s method of root-finding, applied to the derivative. If $\lambda_{\theta}(u)$ is differentiable with respect to (all components of) θ , the score is

$$\mathbf{U}(\theta; \mathbf{x}) = \sum_{i=1}^n z_{\theta}(x_i) - \int_W z_{\theta}(u) \lambda_{\theta}(u) du \quad (9.72)$$

where $z_{\theta}(u) = (\partial/\partial\theta) \log \lambda_{\theta}(u)$, and the *observed* information is

$$H(\theta; \mathbf{x}) = - \sum_{i=1}^n \kappa_{\theta}(x_i) + \int_W \kappa_{\theta}(u) \lambda_{\theta}(u) du + \int_W z_{\theta}(u) z_{\theta}(u)^{\top} \lambda_{\theta}(u) du \quad (9.73)$$

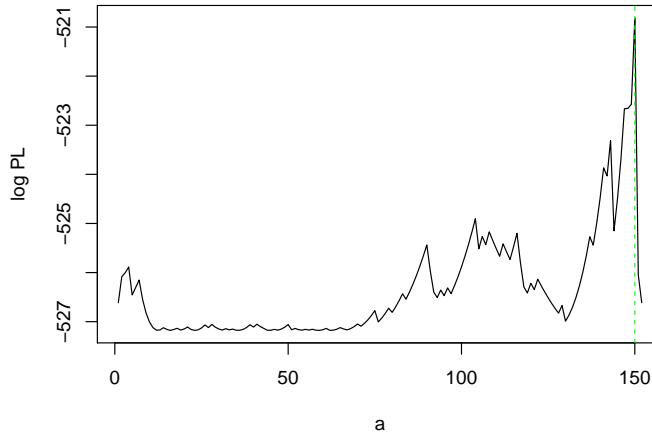


Figure 9.17: Profile likelihood for changepoint model of NZ Trees data.

where $\kappa_\theta(u) = (\partial/\partial\theta)z_\theta(u) = (\partial^2/\partial\theta^2)\log\lambda_\theta(u)$. In the Newton-Raphson method we repeatedly update our current estimate of θ by

$$\theta_{m+1} = \theta_m - H(\theta_m; \mathbf{x})^{-1}\mathbf{U}(\theta_m; \mathbf{x}). \quad (9.74)$$

The `spatstat` function `ippm()` performs this iterative maximisation.

From help file for `ippm`: For example, consider a Poisson point process with intensity function $\lambda(u)$ at u such that

$$\lambda(u) = \exp(\alpha + \beta Z(u))f(u, \gamma)$$

where α, β, γ are parameters to be estimated, $Z(u)$ is a spatial covariate function, and f is some known function. Then the parameters α, β are called *regular* because they appear in a loglinear form; the parameter γ is called *irregular*.

To fit this model using `ippm()`, we specify the model formula in the same way as usual for `ppm()`. Recall that the right-hand side of the model formula is a representation of the *log* of the intensity. In the above example the log intensity is

$$\log \lambda(u) = \alpha + \beta Z(u) + \log f(u, \gamma)$$

So the model above would be encoded with the trend formula `~Z + offset(log(f))`. Note that the irregular part of the model is an *offset* term, which means that it is included in the log trend as it is, without being multiplied by another regular parameter.

The optimisation runs faster if we specify the derivative of $\log f(u, \gamma)$ with respect to γ . We call this the *irregular score*. To specify this, the user must write an R function that computes the irregular score for any value of γ at any location (x, y) .

Thus, to code such a problem,

1. The model formula should define the log intensity, with the irregular part as an `offset`;
2. The argument `start` should be a list containing initial values of each of the irregular parameters;
3. The argument `iScore`, if provided, must be a list (with one entry for each entry of `start`) of functions with arguments x, y, \dots , that evaluate the partial derivatives of $\log f(u, \gamma)$ with respect to each irregular parameter.

An example is given below.

For a general concave loglikelihood, with arbitrary data, the existence and uniqueness of the maximum likelihood estimate (MLE) are not guaranteed, and depend on the absence of “directions of recession” [288, 337, 299, 6, 155].

See also [126].

9.9.3.1 Chorley-Ribble example

```
> lung <- split(chorley)$lung
> larynx <- split(chorley)$larynx
> Q <- quadscheme(larynx, eps=0.1)
```



Figure 9.18: Chorley-Ribble data. *Left:* Spatial locations of cases of cancer of the larynx (black solid dots), and the lung (grey crosses) and a disused industrial incinerator (crosshairs in circle). *Right:* kernel-smoothed intensity estimate of lung cancer cases.

The left panel of Figure 9.18 shows the Chorley-Ribble cancer data of Diggle [126] giving the residential locations of new cases of cancer of the larynx (58 cases) and cancer of the lung (978 cases) in the Chorley and South Ribble Health Authority of Lancashire, England, between 1974 and 1983. The location of a disused industrial incinerator is also given. The aim is to assess evidence for an increase in the incidence of laryngeal cancer close to the incinerator. The lung cancer cases serve as a surrogate for the spatially-varying density of the susceptible population. Data analysis in [126, 129, 34] concluded there is significant evidence of an incinerator effect.

Here we follow Diggle [126] in treating the cases of laryngeal cancer as the response, and taking the lung cancers as a covariate. We applied kernel smoothing [124] to the lung cancer locations to obtain an unnormalised estimate ρ of the spatially varying population density of susceptibles, shown in the right panel of Figure 9.18. This approach is open to critique [126, 129, 34] but is shown here for demonstration purposes.

```
> smo <- density(lung, sigma=0.15, eps=0.1)
> smo <- eval.im(pmax(smo, 1e-10))
```

The null model, constant relative risk, postulates that the larynx cases are a Poisson process with intensity $\lambda(u) = \kappa \rho(u)$ at location u . The parameter κ adjusts for the relative abundance of the two types of data points; it is the baseline relative risk of larynx and lung cancer, multiplied by the ratio of sampling fractions used when these data were sampled from the cancer registry. This *could* be used to infer the absolute risk of laryngeal cancer if these sampling fractions were known.

We fit the null model:

```
> ppm(Q ~ offset(log(smo)))
Nonstationary Poisson process

Trend formula: ~offset(log(smo))

Fitted trend coefficient:
(Intercept)
-2.825

Estimate    S.E. CI95.lo CI95.hi Ztest   Zval
(Intercept) -2.825  0.1313 -3.082  -2.567 *** -21.51
```

Comment: K-S test (etc) is not very sensitive because all the action occurs in a very small area.

Diggle [126] considered alternative ‘raised incidence’ models that include an effect due to proximity to the incinerator. Assume the intensity of laryngeal cancer cases at a location u is

$$\lambda_\theta(u) = \kappa \rho(u) b_{\alpha,\beta}(d(u)) \quad (9.75)$$

where $d(u)$ is the distance in kilometres to the incinerator from location u . Here $\theta = (\kappa, \alpha, \beta)$ is the parameter vector and

$$b_{\alpha,\beta}(d) = 1 + \alpha \exp(-\beta d^2) \quad (9.76)$$

is the raised risk ratio at a distance d from the incinerator [126, eq. (6)]. The parameters α, β control the magnitude and dropoff rate, respectively, of the incinerator effect. The log intensity is

$$\log \lambda_\theta(u) = \log \kappa + \log \rho(u) + \log(1 + \alpha \exp(-\beta d^2)); \quad (9.77)$$

this is not a linear function of the parameters α and β , so these parameters are irregular, and need to use another algorithm such as `ippm()` or `profilepl()` to fit the model. Since (9.76) is differentiable with respect to α and β , we can use `ippm()`.

We start by defining the squared distance to the incinerator, as this function will be re-used frequently.

```
> d2incin <- function(x, y, xincin=354.5, yincin=413.6) {
  (x - xincin)^2 + (y - yincin)^2
}
```

The arguments `xincin`, `yincin` are the coordinates of the incinerator. By specifying default values for them, we do not have to handle these values again.

Next we define the raised incidence term $b_\theta(u)$:

```
> raisin <- function(x,y, alpha, beta) {
  1 + alpha * exp( - beta * d2incin(x,y))
}
```

We then fit the model using `ippm()`:

```
> chorleyDfit <- ippm(Q ~offset(log(smo) + log(raisin)),
  start=list(alpha=5, beta=1))
```

The argument `start` lists the irregular parameters over which the likelihood is to be maximised, and gives their starting values for the iterative algorithm. The parameter names should match the names of arguments of the function `raisin()`. As intended, the arguments `xincin`, `yincin` are held fixed, because they are not listed in `start`.

Explain that iterative procedure can fail in non-convex situations. Tweak arguments passed to ‘`nlm`’.

```
> chorleyDfit
Nonstationary Poisson process

Trend formula: ~offset(log(smo) + log(raisin))

Fitted trend coefficient:
(Intercept)
-2.897

Irregular parameters (covfunargs) fitted by 'ippm':
alpha = 22.25
beta = 0.8889
Estimate   S.E. CI95.lo CI95.hi Ztest   Zval
(Intercept) -2.897 0.1313 -3.154 -2.639 *** -22.06
```

The MLE's are $\hat{\alpha} = 22.3$ (dimensionless) and $\hat{\beta} = 0.89 \text{ km}^{-2}$.

9.10 FAQ's

- I get different parameter estimates from different versions of `spatstat`.
- I read in the literature that “`spatstat` is biased” (e.g. [327])
- what are the relative merits of/relationship with logistic regression?
- what are the relative merits of/relationship with maxent?
- when I use pixel logistic regression, the fitted presence probabilities are tiny numbers like 10^{-8} . Is this physically meaningful?
- Search email correspondence for questions



10

Hypothesis Tests and Simulation Envelopes

This chapter explains how to conduct a variety of tests of statistical significance, and how to use simulation envelopes for hypothesis testing. It also attempts to correct some of the common misconceptions about these methods.

10.1 Introduction

Often the main purpose of statistical analysis is to decide whether a claim is true or false. For example, the Chorley-Ribble data (page 323) were studied to investigate whether there is, or is not, evidence that a now-disused industrial incinerator may have increased the risk of laryngeal cancer for residents living close to the incinerator. A standard tool for deciding on the truth or falsity of a claim is *hypothesis testing* or *significance testing*, which is covered in this Chapter.

Hypothesis tests and their associated p -values are often used in scientific publications to reach a *formal, definitive conclusion* about the scientific question that is under study. Over the years many polemics have been published, railing against hypothesis tests, condemning their use as being misguided and asserting that they lead to unsound scientific conclusions. Two such polemics, published in reasonably reputable journals are [88] and [45]. A relatively recent and amusingly quirky example is [275].

It is our opinion that, although such papers often have valid points to make, the blanket dismissal of the whole concept of hypothesis testing falls in the category of crank writing. Hypothesis testing involves abstraction and idealization of the true scientific question that is being addressed. However such idealization permits the investigator to separate the various factors that have an impact upon the analysis and thereby deal with each such factor in an appropriate manner. Correctly applied and used with judicious attention to the underlying scientific issues, hypothesis testing provides as reliable a guide to scientific decision making as is realistically possible. Some caveats and advice about the appropriate use of hypothesis testing can be found in section 10.3.5.

Hypothesis tests also have several other roles in statistical analysis. They can be used to *guide the choice of strategy* when we start analysing a dataset. For example in the analysis of spatial point patterns, a hypothesis test can be used to decide whether we can provisionally treat the pattern as homogeneous (and then use the K -function, etc) or whether spatial variation in intensity must be investigated.

Hypothesis testing is also one of the tools available for *model selection*, that is, for choosing the best among several competing models for the same data. This is useful at intermediate stages of data analysis, where we need to adopt (at least provisionally) a statistical model in order to perform detailed analysis. Model selection is also part of the formal process of assessing evidence for an “incinerator effect” in the Chorley-Ribble data, for example. We again emphasize that hypothesis testing must be applied appropriately and judiciously. Model selection via hypothesis testing is most

appropriate when the number of competing models is *small*; in other circumstances other related techniques may yield better results.

In the analysis of spatial point patterns, an important role is played by statistical tests based on *simulation envelopes*, such as the envelope of simulations of Ripley's K -function (Chapter 7).

A good general reference on hypothesis testing is [222].

10.2 Terminology

Following is a brief summary of standard terminology for testing.

In most scientific literature, the convention for reporting a statistical test is actually a muddled compromise between two different procedures, Fisherian *significance testing* and Neyman-Pearson *hypothesis testing*. For lack of space we will not delve into the details: instead we will offer pragmatic warnings about the misinterpretation of test outcomes as they arise in examples.

Formal hypothesis testing is structured in terms of two hypotheses, the *null* hypothesis, usually denoted by H_0 and the alternative hypothesis, often denoted by H_1 . The null hypothesis is so called because it is in effect the hypothesis that “nothing is happening” — the treatment has no effect, or there is no difference amongst a number of treatments, or that there is no “interesting” structure to the data. In the context of the Neyman-Pearson paradigm there is also an alternative hypothesis which may simply be that *something* is happening. The alternative could also specify that “something is happening” in a particular way. It is important, if one is following the Neyman-Pearson paradigm, to select an appropriate alternative hypothesis so as to have the best chance of making the appropriate decision.

The actual test is conducted in terms of a test statistic T calculated from observed data. The distribution of T given that H_0 is true (the “null distribution”) must be available (either analytically or via simulation). Ideally the distribution of T when H_0 is not true should be quite different and readily distinguishable from the null distribution.

In the Neyman-Pearson context the test is conducted at a given *significance level* or “size” (often denoted by “ α ”) which is chosen by the investigator. The significance level is a (small) probability, often taken to be 0.05 (also 0.01, and occasionally 0.10). The significance level together with the null distribution determine a *critical* region (or rejection region) in the set of possible values of T . The critical region has the property that if H_0 is true then probability that T falls in the critical region is equal to the significance level.

When the data have been observed and the value of T calculated we reject H_0 if that value is in the critical region. Otherwise we *do not reject* H_0 . Strictly speaking we *never* accept a hypothesis, but adhering to this level of propriety often leads to awkward phraseology so it is common (and “mostly harmless”) to speak of accepting H_0 if we do not reject it. We shall use this slightly unsound terminology in what follows. One really shouldn't (and we won't) ever talk about accepting the *alternative* hypothesis.

To a very large extent the results of hypothesis tests are reported, in practice, in terms of *p-values*. This concept originates in the Fisherian paradigm, i.e. in “significance testing”. The *p-value* of a test is the probability of a random variable T that arises from the null distribution having a value “at least as extreme as t_{obs} ” where t_{obs} is the value of the test statistic calculated from the observed data. The underlying idea is that if the *p-value* is small then we are very unlikely to observe data as “peculiar” as what we actually observed given that H_0 is indeed true. So either H_0 is not true or a highly improbable event has occurred. Thus small *p-values* cast doubt upon the null hypothesis.

Under the strict Fisherian paradigm no decision about whether to reject H_0 is made. The level of doubt about H_0 , as reflected by the *p-value* is simply reported. In practice the *p-value* is usually used as a convenient summary of the results of the hypothesis test. If the *p-value* is less than α then

H_0 is rejected at significance level α . Also, in practice, the “extremeness” of T is measured in terms of a specified Neyman-Pearson style alternative hypothesis. For instance in the elementary scenario of testing the hypothesis $H_0: \mu = \mu_0$, about the mean of a population from which an i.i.d. sample X_i arises, then values of \bar{X} which are a good deal *less* than μ_0 would not be considered extreme if the alternative were $H_1: \mu > \mu_0$.

Clearly there are two ways things can go wrong when a hypothesis is being tested. One is to reject H_0 when H_0 is in fact true. This is referred to as a “Type I” error. The other way is to fail to reject H_0 when H_0 is in fact false. This is referred to as a “Type II” error. The probability of making a Type I error when H_0 is true is by definition α , the significance level of the test. The probability of making a Type II error when H_0 is false is often denoted by β . This quantity is much more complicated than the probability of a Type I error since it depends on “just how false” H_0 is.

The discussion is often phrased in terms of the probability of *not* making a Type II error, i.e. $1 - \beta$, probability of “getting it right” when H_0 is false. This is called the *power* of the test and is the fundamental consideration when choosing a particular test to use. Note that power is a function of “just how false” H_0 is, and ought to get larger (closer to 1) as H_0 grows “more false”. It is obviously to have some numeric measure of the falseness of H_0 . The ultimate desideratum is to find a “uniformly most powerful” (UMP) test, i.e. one which is at least as powerful as any other test for all values of the “falseness” of H_0 .

Finding an UMP test is in general more easily said than done. However there is one old and famous important positive result. According to the Neyman-Pearson lemma [255] the likelihood-ratio test is UMP for testing simple (point) hypotheses.

10.3 Testing for a covariate effect in a parametric model

10.3.1 General problem

A common task for analysis is to decide whether a point pattern \mathbf{x} depends on a covariate Z , after adjusting for variables that are known to influence \mathbf{x} . For example, in the Chorley-Ribble data, we wish to decide whether the rate of laryngeal cancer depends on distance from the incinerator, after allowing for spatial variation in the population density. In the Murchison gold data, we might wish to determine whether the abundance of gold deposits depends on distance from the nearest geological fault, after adjusting for rock type.

This is effectively a choice between two models, which differ only in that one of the models includes a term for the effect of the covariate Z in question, and the other does not. In order to “adjust” for other variables that are known to influence \mathbf{x} , these other variables are included in both the models. For example in the Murchison gold survey,

```
> mur <- lapply(murchison, rescale, s=1000, unitname="km")
> mur$dfault <- with(mur, distfun(faults))
```

we might compare the two Poisson point process models

```
> mfit0 <- ppm(gold ~ greenstone, data=mur)
> mfit1 <- ppm(gold ~ greenstone + dfault, data=mur)
```

or equivalently

```
> mfit1 <- update(mfit0, . ~ . + dfault)
```

If ψ is the vector of parameters for the first model `mfit0`, and φ is the vector of parameters corresponding to the extra covariate term Z , then the augmented model `mfit1` has parameters $\theta = (\psi, \varphi)$. We can regard `mfit0` as a special case of `mfit1` in which the extra parameters φ are equal to zero.

Effectively we wish to decide whether $\varphi = 0$ or not. A standard tool for this purpose is a *hypothesis test* of the *null hypothesis* $H_0 : \varphi = 0$ against the *alternative hypothesis* $H_1 : \varphi \neq 0$. Note that the null hypothesis corresponds to the *absence* of an effect due to the covariate Z .

Another example is a formal test of homogeneous intensity against the alternative that the intensity depends on a covariate Z . For example, for the Queensland copper data,

```
> copper$dist <- with(copper, distfun(SouthLines))
> cfit0 <- ppm(SouthPoints ~ 1, data=copper)
> cfit1 <- ppm(SouthPoints ~ dist, data=copper)
```

Again the task is to compare the two models cfit0 and cfit1 where the null hypothesis, fitted by the smaller model cfit0 , corresponds to the *absence* of an effect due to the covariate.

10.3.2 Likelihood ratio test

The **likelihood ratio test** of $H_0 : \varphi = 0$ against $H_1 : \varphi \neq 0$ in a statistical model with parameters $\theta = (\psi, \varphi)$ is based on the test statistic¹

$$\Gamma = 2 \log \frac{L_1}{L_0} = 2(\log L_1 - \log L_0) \quad (10.1)$$

where L_0 and L_1 are the maximum values of the likelihood achieved under H_0 and H_1 respectively. For loglinear Poisson point process models, under regularity conditions, the large-sample asymptotic distribution of Γ under H_0 is χ^2 with d degrees of freedom, where d is the dimension of φ .

The likelihood ratio test for Poisson models can be carried out by `anova.ppm`, a method for the generic function `anova`. For the Murchison gold data:

```
> anova(mfit0, mfit1, test="Chi")
Analysis of Deviance Table
Model 1: ~ greenstone
Model 2: ~ greenstone + dfault
  Df Deviance Pr(>Chi)
1
2  1      67.8   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Although the name of the generic function is `anova`, the extension of Analysis of Variance to more general models is Analysis of Deviance, as indicated by the printed output. The value of Γ is reported under the heading `Deviance`, and the number of degrees of freedom is `Df`. The *p*-value given under the heading `Pr(>Chi)` is obtained by referring Γ to the χ^2 distribution with 1 degree of freedom. The symbol ******* refers to the significance codes printed at the bottom of the output, and indicates that $p < 0.001$. The null hypothesis is emphatically rejected, indicating strong evidence that the abundance of gold deposits depends on distance to the nearest fault. However, see the caveats in Section 10.3.5 below.

For the Queensland copper data we use a similar command:

```
> anova(cfit0, cfit1, test="Chi")
Analysis of Deviance Table
Model 1: ~ 1
```

¹The likelihood ratio test statistic is usually written as Λ in statistical literature, but in this book Λ and λ always refer to point process intensity.

```
Model 2: ~ dist
  Df Deviance Pr(>Chi)
1
2 1      0.34     0.56
```

In this case the null hypothesis is accepted, indicating a lack of evidence that the abundance of copper deposits depends on distance to the nearest lineament.

The Chorley-Ribble data were introduced on page 323. Diggle's [126, 129] raised incidence model was fitted on page 324:

```
> chorleyDfit
Nonstationary Poisson process
Trend formula: ~offset(log(smo) + log(raisin))
Fitted trend coefficient:
(Intercept)
-2.9
Irregular parameters (covfunargs) fitted by 'ippm':
alpha = 22.3
beta = 0.889
      Estimate  S.E. CI95.lo CI95.hi Ztest  Zval
(Intercept) -2.9  0.131   -3.15   -2.64 *** -22.1
```

To test whether the raised incidence term `raisin` is significant, we fit the model without this term:

```
> chorley0fit <- update(chorleyDfit, . ~ offset(log(smo)))
```

Warning: The operator “-” is *not supported for offsets* in model formulae. The command
`update(chorleyDfit, . ~ . - offset(log(raisin)))`
would **not** delete the offset term.

Then we invoke `anova.ppm`:

```
> anova(chorley0fit, chorleyDfit, test="Chi")
Analysis of Deviance Table
Model 1: ~ offset(log(smo))
Model 2: ~ offset(log(smo) + log(raisin))
  Df Deviance Pr(>Chi)
1
2 2      9.32    0.0095 **
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The result indicates strong evidence against H_0 in favour of the raised incidence alternative.

Note that some special tricks are happening in the last call to `anova.ppm` above. At first glance, it appears to compare two `ppm` objects defined by formulas that contain only offset terms. This would not be possible if the models were both fitted by `ppm`, because two such models are not nested: one ‘pure offset’ model is not a special case of another. However, in this case, the alternative model `chorleyDfit` was fitted by `ippm` and includes two irregular parameters `alpha` and `beta`, the parameters of the function `raisin` appearing in the second offset term. The models *are* nested, because if we set `alpha=0` then the `log(raisin)` term is identically equal to zero. The likelihood ratio test is therefore valid, and `anova.ppm` correctly counts the degrees of freedom as $d = 2$. Note that `anova.ppm` is not smart enough to verify that the models are nested: it relies on the user to check this.

10.3.3 Wald test for single parameter

The Wald test of $H_0 : \varphi = 0$ for a *one-dimensional* parameter φ is based on the statistic

$$V = \frac{\hat{\varphi}}{\text{se}(\hat{\varphi})} \quad (10.2)$$

where $\hat{\varphi}$ is the maximum likelihood estimate of φ under the alternative hypothesis H_1 , and $\text{se}(\hat{\varphi})$ is the plug-in estimate of standard error of $\hat{\varphi}$. The asymptotic null distribution of V is standard normal. The likelihood ratio test is asymptotically equivalent to the Wald test, under regularity conditions.

For a loglinear Poisson point process model fitted by `ppm`, the Wald test for each coefficient is performed as part of the `print` method. It can be recovered using `coef(summary(...))`.

```
> coef(summary(mfit1))
      Estimate   S.E. CI95.lo CI95.hi Ztest   Zval
(Intercept) -6.617 0.2171 -7.043 -6.1916 *** -30.48
greenstoneTRUE 2.754 0.2066  2.349  3.1588 *** 13.33
dfault       -0.104 0.0179 -0.139 -0.0686 *** -5.78
```

The column headed `Ztest` reports the outcomes of the Wald tests for each of the model coefficients, using the customary asterisk convention. In particular the covariate effect for `dfault`, distance to the nearest fault, is significant at the 0.001 level.

The output includes the value of the Wald statistic, under the heading `Zval`. This makes it possible to perform a *one-sided* test of $H_0 : \varphi = 0$ against $H_2 : \varphi < 0$, which is the alternative of interest for the Murchison data, corresponding to a concentration of gold deposits *near* the geological faults rather than *away* from the faults:

```
> V <- coef(summary(mfit1))["dfault", "Zval"]
> pnorm(V, lower.tail=TRUE)
[1] 3.69e-09
```

The use of character indices helps to ensure we are extracting the correct entry from the data frame. The call to `pnorm` computes the p -value corresponding to the left-sided test, because we requested the lower tail probability. The result gives strong evidence against H_0 in favour of H_2 .

The Wald test for *irregular* parameters is not yet supported in `spatstat`. For example it is not yet straightforward to apply the Wald test to the parameters `alpha` and `beta` in the raised-incidence model for the Chorley-Ribble data.

10.3.4 Score test

The likelihood ratio test and Wald test require us to estimate the parameters under the alternative hypothesis — that is, to fit the full model including covariate effects — which can be difficult and computationally costly. It may be convenient to apply the Rao [276] *score test* (also known as the Lagrange multiplier test) which only requires us to fit the *null* model, that is, the model without the covariate effect. The score test statistic of $H_0 : \varphi = 0$ against $H_1 : \varphi \neq 0$ in a model with parameter $\theta = (\psi, \varphi)$ is

$$S = U(\dot{\theta})^\top I(\dot{\theta})^{-1} U(\dot{\theta}) \quad (10.3)$$

where $\dot{\theta} = (\dot{\psi}, 0)$ is the maximum likelihood estimate of θ under H_0 , and U and I are the score function and Fisher information of the full model. Under H_0 the test statistic has a χ^2 distribution with d degrees of freedom where d is the dimension of φ . The score test is less powerful than the likelihood ratio test or Wald test in small samples, but asymptotically equivalent to them in large samples, under regularity conditions.

An important example of the score test in spatial statistics is the Berman-Waller-Lawson test (Section 6.7.3). Suppose we want to test the null hypothesis that the point process intensity is constant (or more generally, proportional to a known baseline) against the alternative that the intensity depends on a specified covariate Z . Suppose $b(u)$ is a known baseline function, $Z(u)$ is a specified covariate function, and assume a Poisson process with intensity

$$\lambda_\theta(u) = b(u) \exp(\psi + \varphi Z(u)) \quad (10.4)$$

where $\theta = (\psi, \varphi)$ are parameters. The null hypothesis $H_0 : \varphi = 0$ states that the intensity is proportional to the baseline $b(u)$, and does not depend on Z . The alternative $H_1 : \varphi \neq 0$ means that the intensity is *loglinear* in $Z(u)$. The model (10.4) can be fitted using `ppm` and the likelihood ratio test can be performed using `anova.ppm`, but only using numerical approximations, as described in Chapter 9. There is no analytically exact formula for the likelihood ratio test statistic Γ or the Wald test statistic Z in this model.

The score test for a covariate effect in (10.4) is much easier to describe. The null hypothesis is a Poisson process with intensity proportional to $b(u)$. Straightforward calculation gives the score test statistic

$$S = \frac{(T - nM_1)^2}{nM_2}$$

where $n = n(\mathbf{x})$ is the number of data points, $T = \sum_i Z(x_i)$ is the sum of covariate values over all data points, and

$$M_k = \frac{\int_W Z(u)^k b(u) du}{\int_W b(u) du}, \quad k = 1, 2$$

is the mean value of $Z(U)^k$ when U is a random point in the window W with probability density proportional to $b(u)$. To perform the score test, we refer the observed value of S to the χ^2 distribution with 1 degree of freedom. Equivalently, the signed square root of S is referred to the standard normal distribution; this is the Z_1 test of Berman [54], and is closely related to the tests of Waller et al [330] and Lawson [220].

The Pearson χ^2 goodness-of-fit test is also a score test [303], and this clarifies its relationship to the theoretically ‘optimal’ likelihood ratio test.

The score test for irregular parameters (such as the parameters `alpha`, `beta` in the raised incidence model of the Chorley-Ribble data) is not yet supported in `spatstat` at the time of writing.

10.3.5 Caveats

Scientific articles sometimes mis-state the conclusion from a hypothesis test, exaggerate the strength of the conclusion, or overlook potential weaknesses of a test. Here we list some of the most important caveats.

The conclusion from a hypothesis test

Hypothesis testing focuses on the *truth or falsity of the null hypothesis* H_0 . The possible conclusions from the test are either to “accept H_0 ” or “reject H_0 ”. If the test accepts the null hypothesis, we are effectively reporting there is insufficient evidence against it. If the test rejects H_0 , we are reporting that there is sufficient evidence against H_0 . Concluding that H_0 is false does not necessarily imply that H_1 is true.

The *p*-value, as discussed in Section 10.1, is a measure of the strength of evidence *against* the null hypothesis, with smaller values indicating stronger evidence. It is defined as the probability of obtaining data at least as extreme as the data that were observed, *assuming the null hypothesis is true*. ‘Extreme’ is defined below. The logic of significance testing is that, if the *p*-value is small,

then “either an exceptionally rare chance has occurred, or the theory [null hypothesis] is not true” [146, p. 39]. The *p*-value measures how rare is this “rare chance”.

A common fallacy is to interpret a *p*-value of 0.03 as meaning we are 97% confident that the alternative hypothesis is true (or equally bad, that there is an 0.97 probability that the null hypothesis is false). A hypothesis test never accepts the alternative; and the *p*-value is not a measure of confidence about the alternative. The *p*-value is a probability calculated under the assumption that the *null* hypothesis is true. *A hypothesis test does not evaluate the strength of evidence for the alternative hypothesis.*

Role of the alternative hypothesis

The alternative hypothesis serves mainly to define the term ‘extreme’ used in calculating the *p*-value. ‘Extreme’ observations are those which are more consistent with the alternative hypothesis than the null hypothesis. For example, in the Wald test for the Murchison gold example (Section 10.3.3) the null hypothesis is $H_0 : \varphi = 0$. Specifying the alternative $H_1 : \varphi \neq 0$ implies that extreme values of the test statistic V are those which are large and positive, or large and negative. Specifying $H_1 : \varphi < 0$ implies that only large *negative* values of V are considered extreme. Essentially the alternative hypothesis determines the *direction* of departures from the null hypothesis that are relevant to the test.

Changing the alternative hypothesis can change the outcome of the test. Making the alternative hypothesis more restrictive will reduce the *p*-value, strengthening the apparent evidence against the null hypothesis.

Tests depend on assumptions

A hypothesis test involves assumptions, and the conclusions may be highly sensitive to violations of these assumptions.

In all the examples in Sections 10.3.2–10.3.4 above, we assumed that the point process was a Poisson process, and entertained various possibilities for the intensity function. The calculated *p*-values therefore assume that the point process is Poisson, among other assumptions.

For the Murchison gold data, the null model is a Poisson process with two different, constant intensities inside and outside the greenstone. The *p*-value is calculated under these assumptions. The *p*-value close to zero indicates that, assuming this null model to be true, it would be virtually impossible to obtain a value of Γ as large as the value obtained. As discussed above, this does not imply the truth of the alternative model, in which the intensity of gold deposits is an exponential function of distance to the nearest fault. Essentially we have not evaluated the strength of evidence for an exponential relationship. Even if we are confident that the gold deposits are a Poisson process (which seems implausible from a geological standpoint) the best we can conclude from the Likelihood Ratio Test is that the intensity is not constant inside the greenstone *and/or* is not constant outside the greenstone.

The null hypothesis

Hypotheses are often written in shorthand — for example, “ $H_0 : \mu_1 = \mu_2$ ” or “no difference between groups”. However, this is dangerous if the user does not appreciate the full meaning of the shorthand. Does “no difference between groups” signify that the two groups have the same population means, or the same population distributions, or something else appropriate to the scientific context?

Strong [317] gave an important example related to the study of species diversity on islands. The scientific null hypothesis is that small and large island communities are governed by the same ecological processes. Researchers have tested this by measuring species diversity on large and small islands, and testing whether the diversity index values are equal. However, in a smaller population

chosen at random from a larger population, less species diversity is to be expected. A test of “no difference in species diversity” would be inappropriate.

The null hypothesis represents what we expect to see if “nothing is happening”. It can be very important to spell out exactly what we expect to see under the null hypothesis.

Asymptotic distributions

A technical point about the tests described in Sections 10.3.2–10.3.4 above is that they use the *asymptotic* (large sample limit) distributions of the test statistics, namely the χ^2 and normal distributions. In small samples, these are only approximations to the true distribution of the test statistic, and therefore the p -value is an approximation.

More accurate p -values can be obtained by simulation from the null hypothesis. This may or may not be easy to do, depending on circumstances. There is a reasonably wide range of simulation tools in `spatstat` to help with this task. For instance if the null hypothesis is “CSR” then `rpoispp()` does the necessary, and does it quickly. More complicated null hypotheses may be handled by `rmh()`. In any case it pays to think carefully about what you are doing, in particular to make sure that *composite* null hypotheses (see section 10.7.3.2) are being handled correctly.

10.4 Model selection using AIC

When there are many candidate models to choose from, some technique for “model selection” is needed. (The reader should be aware that there are lurking perils in “automatic” model selection techniques. Some salutary comments on this issue may be found in [249].) In “forward stepwise selection” we start with a minimal acceptable model, and add new terms to the model one-by-one. In “backward stepwise selection” we start with a maximal model which is believed to be adequate, and delete terms from the model one-by-one.

We could use the Likelihood Ratio Test as the criterion for deciding whether to add a term (or to delete a term) at each step. However, this often results in models which are smaller than they should be: the customary significance level $\alpha = 0.05$ is an excessively stringent standard of evidence for including a proposed term in the model. A better approach is to compare models using the *Akaike Information Criterion* [5]

$$\text{AIC} = -2 \log L_{\max} + 2p$$

where $L_{\max} = L(\hat{\theta})$ is the maximised likelihood for the model in question, and p the number of parameters for this model. The model with the *lowest* value of AIC is preferred.

In stepwise model selection, choosing the model with the smaller AIC is equivalent to applying the likelihood ratio test, but taking the critical value to be $2d$ where d is the number of degrees of freedom, i.e. the number of added or deleted parameters. The size of this test is $\alpha = 0.157, 0.135, 0.112$ when $d = 1, 2, 3$ respectively, dropping below 0.05 when $d = 8$.

The `stats` package (which is included in a standard installation of R) provides functions `AIC`, `add1`, `drop1` and `step` to perform stepwise model selection. They can be applied to many kinds of models, including fitted point process models. The function `AIC` evaluates the AIC. The function `drop1` compares a model with all the sub-models obtained by deleting a single term, and evaluates the AIC for each sub-model.

```
> fitxy <- ppm(swedishpines ~ x + y)
> drop1(fitxy)
Single term deletions
```

```
Model:
~x + y
      Df AIC
<none>  844
x       1 843
y       1 842
```

The output indicates that the lowest AIC (i.e. 842) would be achieved by deleting the x term. Similarly add1 compares a model with all the “super-models” (!!) obtained by adding a single term:

```
> fitcsr <- ppm(swedishpines ~ 1)
> add1(fitcsr, ~x+y)
Single term additions

Model:
~1
      Df AIC
<none>  841
x       1 842
y       1 843
```

The output indicates that CSR has lower AIC than the models with formulae $\sim x$ and $\sim y$.

The function step performs stepwise model selection using AIC. By default it performs backward stepwise selection starting from a given ‘maximal’ model. Starting from the maximal model, the procedure considers each term in the model, and decides whether the term should be deleted to reduce the AIC. The deletion giving the biggest reduction in AIC is carried out. This is applied recursively until no more terms can be deleted.

```
> fitxy <- ppm(swedishpines ~ x + y)
> fitopt <- step(fitxy)
Start: AIC=844
~x + y

      Df AIC
- y     1 842
- x     1 843
<none>  844

Step: AIC=842
~x

      Df AIC
- x     1 841
<none>  842

Step: AIC=841
~1
> fitopt
Stationary Poisson process
Intensity: 0.0074
```

	Estimate	S.E.	CI95.lo	CI95.hi	Ztest	Zval
log(lambda)	-4.91	0.119	-5.14	-4.67	***	-41.3

At each step in the procedure, the possible deletions are ranked in ascending order of AIC: the deletion giving the greatest reduction in AIC is at the top of the table. The output shows that the term x was first deleted from the model, and then the term y was deleted, leaving only the formula ~ 1 corresponding to CSR. In future we shall set `trace=0` to suppress the progress reports, so the output will show only the final result. For brevity one can use `formula.ppm()` to extract the model formula:

```
> bigfit <- ppm(swedishpines ~ polynom(x,y,3))

> formula(bigfit)

~x + y + I(x^2) + I(x * y) + I(y^2) + I(x^3) + I(x^2 * y) + I(x * y^2)
+ I(y^3)

> formula(step(bigfit, trace=0))
~x + y + I(x * y) + I(y^2)
```

Note that the AIC of the selected model is slightly larger than that of the constant model in `fitopt`. This demonstrates that (not too surprisingly) dropping terms one at a time is *not* guaranteed to produce the submodel with minimal AIC.

The likelihood ratio test requires the null hypothesis to be a sub-model of the alternative. An advantage of AIC and similar criteria is that they can be used to compare models which are not “nested”. For example, suppose we wish to compare the model (9.18) in which the intensity of *Beilschmiedia* trees is proportional to slope, against CSR.

```
> fitprop <- ppm(bei ~ offset(log(grad)), data=bei.extra)
> fitnull <- ppm(bei ~ 1)
> AIC(fitprop) - AIC(fitnull)
[1] -267
```

The smaller AIC favours the model `fitprop`, that is, the model (9.18) in which the intensity is proportional to slope. Note that we cannot use `drop1` here because the update mechanism does not work with offset terms, as explained on page 331.

This is primitive -- can implement other methods on top of `spatstat` -- open to further research.

10.5 Goodness-of-fit tests for an intensity model

A “goodness-of-fit” test of a statistical model is a test of the null hypothesis that the model is true, against the very general alternative hypothesis that the model is not true [111, 279]. Classical tests of goodness-of-fit for probability distributions can be adapted to test the goodness-of-fit of a model for the intensity of a point process.

10.5.1 Goodness-of-fit of a probability distribution

Suppose we wish to test whether some numerical data z_1, \dots, z_n follow a specified probability distribution (such as the standard normal distribution) with cumulative distribution function (cdf) $F_0(z)$. This is a goodness-of-fit problem for which the standard, generic tools are the Kolmogorov-Smirnov, Cramér-von Mises and Anderson-Darling tests. Assume the observations are independent, and come from a common distribution with cdf F . The null hypothesis is $H_0 : F_0 \equiv F$ and the alternative is $H_1 : F_0 \not\equiv F$.

First form the *empirical* cdf of the data,

$$\hat{F}(z) = \frac{1}{n} \sum_{i=1}^n \mathbf{1}\{z_i \leq z\} \quad (10.5)$$

so that $\hat{F}(z)$ is the fraction of observations less than or equal to z .

A test can be based on the discrepancy between the functions \hat{F} and F_0 . The Kolmogorov-Smirnov test statistic is the maximum vertical separation between the graphs of $\hat{F}(z)$ and $F_0(z)$:

$$D = \max_z |\hat{F}(z) - F_0(z)|. \quad (10.6)$$

The Cramér-von Mises test statistic is the average squared separation:

$$\omega^2 = n \int_{-\infty}^{\infty} [\hat{F}(z) - F_0(z)]^2 dF_0(z) \quad (10.7)$$

and the Anderson-Darling test statistic is a weighted average:

$$A = n \int_{-\infty}^{\infty} \frac{(\hat{F}(z) - F_0(z))^2}{F_0(z)(1 - F_0(z))} dF_0(z) \quad (10.8)$$

Assuming F_0 is differentiable (which importantly implies that tied values $z_i = z_j$ cannot occur), the null distributions of the test statistics D , ω^2 and A are known exactly, so that a test can be conducted. The Anderson-Darling test is typically the most powerful of the three tests, but also the most computationally intensive.

The Kolmogorov-Smirnov test is implemented in the function `ks.test` in the standard `stats` package. Efficient methods for performing the Cramér-von Mises and Anderson-Darling tests were developed recently [109, 238] and are implemented as `cvm.test` and `ad.test` in the contributed package `goftest`.

10.5.2 Spatial CDF test for point process

The goodness-of-fit tests described above can be adapted to point patterns. The test is performed by comparing the observed distribution of the values of a spatial covariate at the data points, and the predicted distribution of the same covariate under the model, using one of the classical goodness-of-fit tests.

Suppose we wish to test whether our point pattern dataset `x` came from a Poisson process with intensity $\lambda_0(u)$ on the window W . Let $Z(u)$ be a spatial covariate function, with real values. Consider the values $z_i = Z(x_i)$ of the covariate evaluated at the data points $x_i \in \mathbf{x}$. If the null hypothesis is true, then using properties of the Poisson process, the values z_1, z_2, \dots are a Poisson process on the real line; individual values z_i are independent and identically distributed, with cdf

$$F_0(z) = \frac{\int_W \mathbf{1}\{Z(u) \leq z\} \lambda_0(u) du}{\int_W \lambda_0(u) du}. \quad (10.9)$$

A *spatial cdf test* of the null hypothesis of a Poisson process with intensity $\lambda_0(u)$ extracts the values

$z_i = Z(x_i)$ of the covariate Z at the data points x_i , and tests the goodness-of-fit of the cdf F_0 to the observed values z_i . This technique is implicit in the work of Kolmogorov, but for spatial point processes it was apparently first mentioned formally by Berman [54].

The spatial cdf test requires the user to specify a covariate function Z . Any function Z can be used, but different choices of Z change the sensitivity of the test to different types of departure from the null hypothesis.

10.5.3 Implementation in spatstat

The `spatstat` generic function `cdf.test` performs a spatial cdf test. It has methods for point patterns ("`ppp`") and fitted point process models ("`ppm`" and "`lppm`"). The Kolmogorov-Smirnov, Cramér-von Mises and Anderson-Darling test statistics are selected by the argument `test="ks"`, `test="cvm"`, and `test="ad"` respectively.

If X is a point pattern dataset, then `cdf.test(X, covariate, ...)` performs a goodness-of-fit test of the uniform Poisson point process (Complete Spatial Randomness, CSR) for the pattern X using the specified `covariate`. For a multitype point pattern, the uniform intensity is assumed to depend on the type of point (Complete Spatial Randomness and Independence, CSRI). The `covariate` can be given as a function, a pixel image, a list of pixel images for each type in a multitype pattern, or one of the strings "`x`" or "`y`" indicating the Cartesian coordinates.

```
> X <- copper$SouthPoints
> D <- distfun(copper$SouthLines)
> cdf.test(X, D, test="ad")
   Spatial Anderson-Darling test of CSR in two dimensions

data: covariate 'D' evaluated at points of 'X'
      and transformed to uniform distribution under CSR
An = 0.764, p-value = 0.5074
```

If `model` is a fitted point process model (object of class "`ppm`" or "`lppm`") then `cdf.test(model, covariate, ...)` performs a test of goodness-of-fit for this fitted model. In this case, `model` should be a Poisson point process.

```
> cdf.test(mfit0, mur$dfault, test="ad")
   Spatial Anderson-Darling test of inhomogeneous Poisson process
   in two dimensions

data: covariate 'mur$dfault' evaluated at points of 'gold'
      and transformed to uniform distribution under 'mfit0'
An = 15.3, p-value = 2.353e-06
```

The return value is an object of class "`htest`" containing the results of the hypothesis test. The `print` method for this class gives an informative summary of the test outcome.

The return value also belongs to the class "`cdftest`" for which there is a plot method `plot.cdftest`. The plot method displays the empirical cumulative distribution function of the covariate at the data points, and the predicted cumulative distribution function of the covariate under the model, plotted against the value of the covariate.

10.5.4 Caveats

- The spatial cdf test assumes a Poisson process, and may be sensitive to clustering.

- Could use simulation
 - For a fitted point process model, the computed p -values may be slightly incorrect. The theory for the classical tests presented in Section 10.5.1 assumes that the distribution function F_0 is fixed and known in advance. If the null model involves parameters which must be estimated, then (10.9) involves estimated parameters, and the theory is not strictly applicable. By failing to account for the effect of estimating the parameters, the test typically becomes slightly *conservative*, meaning that the computed p -value is slightly too high.
-

10.6 Goodness-of-fit tests of independence between points

GOF for point process models with non-Poisson alternative: χ^2 , Clark-Evans, etc

- `quadrat.test`
 - `clarkevans.test` without simulation (or with extensive simulation)
 - not very informative
 - often assume known intensity eg uniform
-

10.7 Monte Carlo tests

Monte Carlo methods use random simulation to replace complicated calculations in algebra and calculus. The use of Monte Carlo methods typically involves large numbers of simulations, in order to achieve high accuracy.

However, a *Monte Carlo test* is different: it uses a relatively small number of simulations from the null hypothesis, and appeals to a symmetry principle instead of the law of large numbers.

Monte Carlo tests were developed independently by Barnard [48] and Dwass [137], and further elaborated by Hope [183]. They were first applied in spatial statistics by Ripley [281, 283] and Besag [58, 62]. Monte Carlo tests are related to the *randomisation tests* which are commonly used in nonparametric statistics.

10.7.1 Basic principle

The principle of a Monte Carlo test can be explained by a simple example. Suppose we wish to test whether the `cells` point pattern was generated by Complete Spatial Randomness. Figure 10.1 shows the `cells` data plotted side-by-side with 19 simulated point patterns, each containing the same number of points, generated according to CSR. If the null hypothesis of CSR were true, then these 20 point patterns should be statistically equivalent, and we should not be able to guess which of the 20 patterns was the original data.

In order to make a formal test, we need to reduce each point pattern to a single number, and compare the different numbers obtained. For argument's sake we use the Clark-Evans [86] index. The value for the `cells` data is 1.672 while the (sorted) values for the simulated patterns are

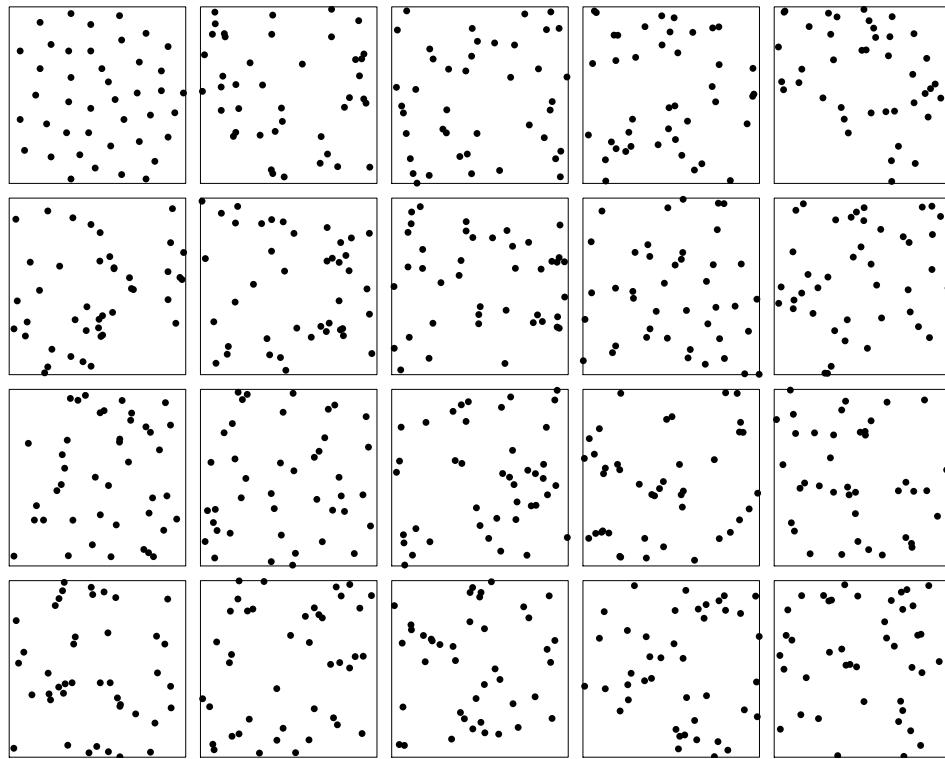


Figure 10.1: The `cells` data (*Top Left*) and 19 simulated realisations of Complete Spatial Randomness with the same number of points.

```
[1] 0.834 0.977 0.983 0.992 1.006 1.012 1.034 1.038 1.039 1.053 1.058
[12] 1.065 1.066 1.085 1.154 1.168 1.169 1.200 1.233
```

We find that the Clark-Evans index for the data is greater than all of the simulated values. This suggests that the `cells` data is not a realisation of CSR.

The formal rationale for the Monte Carlo test is as follows. If the null hypothesis were true, then these 20 point patterns (the `cells` data and the 19 simulated patterns) would be statistically equivalent, because they would be independent random outcomes of the *same* point process. The 20 values of the Clark-Evans index would be statistically equivalent, because they would be independent random numbers with the same probability distribution. By symmetry, there would be 1 chance in 20 that the index value for the `cells` data is the largest of the 20 numbers. Therefore, the result we obtained is statistically significant at the level $1/20 = 0.05$ and the p -value is equal to 0.05.

The main advantage of a Monte Carlo test is that it is cheap — only a small number of simulations are needed — and the p -value is exactly correct, rather than an approximation. The main disadvantages are that the test involves randomisation, so the outcome may change if the test is repeated; there is some loss of power (reduced sensitivity to alternatives); and very strong evidence may be understated, because the smallest possible p -value is $1/(1 + m)$ where m is the number of simulations.

10.7.2 Details of Monte Carlo test

10.7.2.1 Simple form

In its simplest form, a Monte Carlo test for spatial point patterns involves the following steps:

- (M1)** Generate m simulated random point patterns $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}$ from the null hypothesis, where typically m is 19, 39 or 99. These are computer-generated random point patterns, similar to the observed point pattern \mathbf{x} , but generated under the assumption that the null hypothesis is true. The random patterns should be independent of each other (in the sense of probability) and independent of the observed data \mathbf{x} .
- (M2)** Reduce each point pattern to a single numerical value using a test statistic T . The observed point pattern \mathbf{x} is reduced to the number $t_{obs} = T(\mathbf{x})$. The simulated patterns $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}$ are reduced to the numbers t_1, \dots, t_m where $t_i = T(\mathbf{x}^{(i)})$ for each i .
- (M3)** Assuming that larger values of T are more favorable to the alternative hypothesis, the test rule is to reject H_0 at significance level $1/(m+1)$ if the observed value t_{obs} is larger than all of the simulated values t_1, \dots, t_m .

The basic rationale for Monte Carlo tests is **symmetry**. Assume the null hypothesis is true (and that any parameters of the null model are known). Then the original data and the m simulated patterns are statistically equivalent, so the test statistic value t_{obs} calculated for the original data, and the test statistic values t_1, t_2, \dots, t_m calculated for the simulated patterns, are statistically equivalent. By symmetry, there is a 1 in $(m+1)$ chance that the test statistic value t_{obs} is the largest of these $m+1$ numbers — that is, that t_{obs} is larger than (each of) the other m values t_1, t_2, \dots, t_m . If this happens, the result is statistically significant at level $\alpha = 1/(m+1)$.

For example, if $m = 19$, the test procedure is to reject H_0 if the value t_{obs} for the observed data is larger than the maximum of the 19 simulated values $t_{max} = \max\{t_1, \dots, t_{19}\}$. The significance level of the test is $\alpha = 1/(19+1) = 0.05$ (often reported in terms of the p -value as “ $p < 0.05$ ” although in this case the p -value is exactly equal to 0.05).

There is considerable freedom in choosing the test statistic T . Ideally it should be a continuously-varying quantity (so that tied values are impossible) and one which tends to give different values under the null and alternative hypotheses.

There are several modifications of this basic form of the test. Instead of a “one-sided” test which rejects the null hypothesis when t_{obs} is large, we could have a “two-sided” test which rejects the null hypothesis when t_{obs} is either the largest or smallest of the $m+1$ numbers. This has a significance level of $\alpha = 2/(m+1)$. If we want the standard significance level of 0.05, then we would typically choose $m = 39$ simulations giving a significance level of $2/40 = 0.05$.

Instead of taking the maximum and minimum of the simulated values, a Monte Carlo test can be performed using the k -th largest and k -th smallest values out of m simulations, where k is a chosen rank. For a one-sided test we reject the null hypothesis if the observed value t_{obs} is larger than the k -th largest simulated value. This test has significance level $\alpha = k/(m+1)$. For a two-sided test, we reject the null hypothesis if the observed value t_{obs} is either larger than the k -th largest simulated value, or smaller than the k -th smallest simulated value. This test has significance level $\alpha = 2k/(m+1)$.

Note that many choices of m and k will give a test of significance level $\alpha = 0.05$. Examples include a one-sided test using the 5th largest simulated value out of 99 simulations, and a two-sided test using the 5th largest and 5th smallest simulated values out of 199 simulations. Researchers are free to make their own choices of m and k with an eye to computational cost, standards of evidence, performance, and other factors.

If we prefer to compute a p -value instead of specifying a fixed significance level α , the test procedure is as follows. Count the number of simulated values t_i which are greater than the observed value t_{obs} . If there are j such values then the p -value is $(j+1)/(m+1)$ for a one-sided test, or $2\min(j+1, m+1-j)/(m+1)$ for a two-sided test.

For integer k , the Monte Carlo test with (nominal) significance level $\alpha = k/(m+1)$ rejects H_0 if $t_{obs} > t_{[k]}$, where $t_{[k]}$ is the k th largest value amongst t_1, \dots, t_m . Equivalently, the Monte Carlo p -value

is, if large positive values of t are favorable to H_0 ,

$$p_+ = \frac{1 + \sum_{j=1}^m \mathbf{1}\{t_j \geq t_{obs}\}}{m+1}. \quad (10.10)$$

The numerator is the rank of t_{obs} in the set $\{t_1, \dots, t_m\} \cup \{t_{obs}\}$. If small values of t are favorable to H_0 , the p -value is

$$p_- = \frac{1 + \sum_{j=1}^m \mathbf{1}\{t_j \leq t_{obs}\}}{m+1} \quad (10.11)$$

and for two-sided tests

$$p = 2 \min\{p_+, p_-\}. \quad (10.12)$$

10.7.2.2 Code examples

Monte Carlo tests are relatively straightforward to encode in the R language. For example, to test the null hypothesis of CSR for the `cells` data using the Clark-Evans statistic, we would first evaluate

```
> (Tobs <- clarkevans(cells, correction="none"))
[1] 1.67
```

The following code generates 19 simulated point patterns, each containing the same number of points as the `cells` data, and generated according to CSR, and computes the Clark-Evans index for each simulated pattern:

```
> Tsim <- numeric(19)
> for(i in 1:19) {
  X <- runifpoint(npoints(cells), Window(cells))
  Tsim[i] <- clarkevans(X, correction="none")
}
```

An important point is illustrated by the foregoing code. It might seem “more natural” to use `X <- rpoispp(npoints(cells), Window(cells))` However this would be wrong; see section 10.7.3.2.

If the alternative is a regular (inhibited) point process, large values of the Clark-Evans index are favorable to the alternative, so the p -value is, from (10.10),

```
> (preg <- (1 + sum(Tsim > Tobs))/(1 + length(Tsim)))
[1] 0.05
```

For the alternative of a clustered process the p -value is

```
> (pclus <- (1 + sum(Tsim < Tobs))/(1 + length(Tsim)))
[1] 1
```

and for the two-sided alternative

```
> (peither <- 2 * min(pclus, preg))
[1] 0.1
```

For convenience, a few functions in `spatstat` can perform a Monte Carlo test as an option, instead of using asymptotic approximations. These include `quadrat.test` and `clarkevans.test`, which perform tests of CSR.

```
> quadrat.test(redwood, nx=5, alternative="clustered",
  method="MonteCarlo", nsim=999)
> clarkevans.test(redwood, alternative="clustered", nsim=999)
```

10.7.2.3 General form

The astute reader will have noticed that the code example in Section 10.7.2.2 above does not strictly follow the procedure laid out in step **(M1)**. That would require the simulated point patterns $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}$ to be independent of the observed pattern \mathbf{x} . However in Section 10.7.2.2, the simulated patterns were chosen to have the same number of points as the observed data, so the simulated patterns are not independent of the data.

The symmetry principle nevertheless applies to this example. A list of random numbers (Y_1, \dots, Y_n) is called *exchangeable* if any permutation of the list has the same statistical characteristics.² We apply the same definition to a list of random point patterns. Then the Monte Carlo test scheme above can be modified by replacing **(M1)** by

(M1') Given the data pattern \mathbf{x} , generate m simulated random point patterns $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}$ such that, if the null hypothesis is true, then $\mathbf{x}, \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}$ are exchangeable.

The test is still valid with this modification because, if the data and simulations are exchangeable, then the summary statistic values t, t_1, \dots, t_m are exchangeable, and by symmetry, the probability that t is greater than all the simulated values t_1, \dots, t_m is $1/(m+1)$.

In the example in Section 10.7.2.2, given the observed data, the simulated patterns were generated by placing the same number of points independently and uniformly in the same window. If the null hypothesis of CSR is true, then the data also satisfy this description (by property **(PP4)** of the Poisson process stated on page 267), so the data and the simulated patterns are exchangeable. Thus, the example in Section 10.7.2.2 is a valid Monte Carlo test.

One can extend the scope of Monte Carlo tests even further, replacing **(M2)** by

(M2') The observed point pattern \mathbf{x} and simulated patterns $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}$ are reduced to the numbers t and t_1, \dots, t_m which are exchangeable if the null hypothesis is true.

This allows the test statistic T to depend on both the observed and simulated point patterns,

$$t_{obs} = T(\mathbf{x}; \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}). \quad (10.13)$$

For example, in Section 10.7.2.2, the test statistic t_{obs} could be the Clark-Evans statistic for the `cells` data *minus the average for the simulated patterns*. To preserve symmetry, the statistic t_i for the i th simulated pattern must be calculated using the same rule with $\mathbf{x}^{(i)}$ and \mathbf{x} exchanged.

10.7.3 Caveats

10.7.3.1 Dependence on assumptions

e.g. Clark-Evans is fooled by inhomogeneity.

10.7.3.2 Conservatism when parameters are estimated

An important caveat about Monte Carlo tests, which is often overlooked, is that they are strictly invalid, and usually *conservative*, if parameters have been estimated from the data [127, p. 89], [25]. The problem arises when the null hypothesis is a model depending on a parameter or parameters θ that must be estimated from the data (known as a “composite” hypothesis). The usual procedure is to fit the model to the observed point pattern, obtaining an estimate $\hat{\theta}$ of the parameter value, and then to generate the simulated point patterns from the model using this fitted parameter value

²The joint distribution of (Y_1, \dots, Y_n) must be the same as that of $(Y_{i_1}, \dots, Y_{i_n})$ where (i_1, \dots, i_n) is any permutation of $(1, 2, \dots, n)$.

$\hat{\theta}$. But this violates the essential requirement of symmetry. Under the procedure just described, the simulated point patterns have been generated from the null model with parameter value $\hat{\theta}$, while (if the null hypothesis is true) the observed point pattern came from the null model with unknown parameter value θ . See Figure 10.2. Since the estimate $\hat{\theta}$ is usually not exactly equal to the true parameter value θ , the simulated and observed point patterns do not come from the same random process, so they are not exchangeable, and the Monte Carlo test is invalid, strictly speaking.

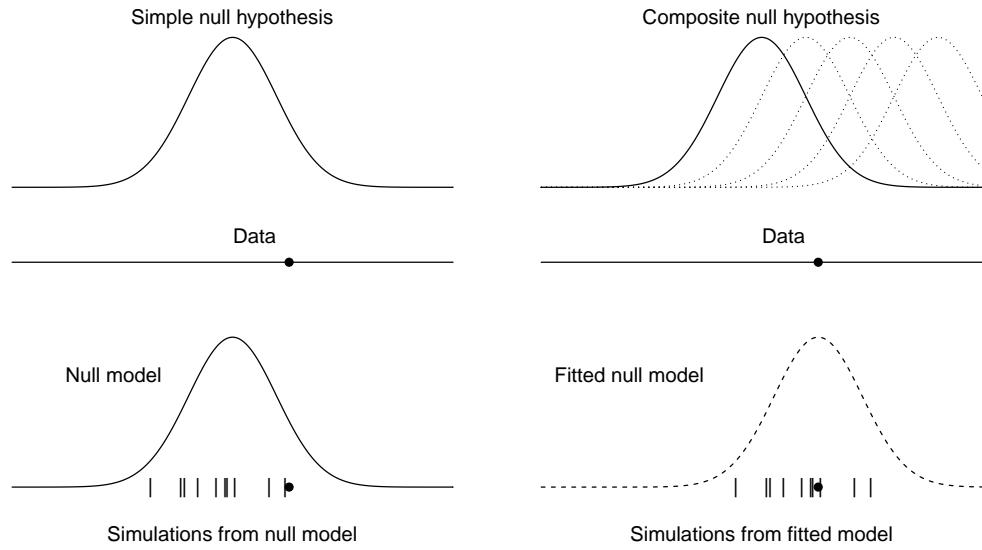


Figure 10.2: Why Monte Carlo tests are typically conservative when the null hypothesis is composite. *Left column:* simple null hypothesis; *Right column:* composite null hypothesis. The Monte Carlo test compares the observed value (dot) with simulated values (vertical bars). For a simple null hypothesis, the simulated values are generated from the null distribution, and the test is exact. For a composite null hypothesis, simulated values are generated from the *fitted* distribution, which is typically centred around the observed value, and the test is typically conservative.

Figure 10.2 suggests that, if we generate random values from a model fitted to the observed data, the random values are likely to scatter *around* the observed value, so that the observed value is unlikely to lie in the extremes of the simulated values. This causes the test to be *conservative*. A test is called conservative if the true significance level — the true probability of Type I error — is smaller than the reported significance level — the significance level which we quote in reporting the outcome of the test. Equivalently, the reported *p*-value is higher (less significant) than the true *p*-value. A conservative test may conclude that the data are not statistically significant when in fact they should be declared statistically significant.

Experiments [25] show that the conservatism effect can be severe in spatial examples, with a test of nominal significance 0.05 having true significance level 0.02 or lower. Conservatism is a small-sample problem in the sense that it disappears in large datasets: if the spatial point pattern dataset is large enough, the parameter estimate $\hat{\theta}$ will be close to the true parameter value θ , and the conservatism effect will be small. However, for small spatial datasets, conservatism is *not* reduced by taking a larger number of simulations m .

This problem arises even in the simplest case of testing CSR. The null hypothesis of CSR has one free parameter, namely the intensity λ (the average number of points per unit area). Typically we estimate λ from our observed data by $\hat{\lambda} = n_{obs}/A$ where n_{obs} is the number of points in the

observed point pattern and A is the area of the survey region. If we were to generate the simulated point patterns according to a Poisson process with intensity $\hat{\lambda}$, the Monte Carlo test would be strictly invalid and probably conservative.

The effect can be measured by an experiment like the following. First we define a function that conducts one Monte Carlo test of CSR using the Clark-Evans statistic, with simulations generated according to CSR.

```
> dotest <- function(X, nsim=19) {
  Tobs <- clarkevans(X, correction="none")
  Tsim <- numeric(nsim)
  for(i in 1:nsim) {
    Xsim <- rpoispp(intensity(X), win=Window(X))
    Tsim[i] <- clarkevans(Xsim, correction="none")
  }
  return(Tobs > max(Tsim))
}
```

The result of `dotest` is a logical value equal to TRUE if the test rejects the null hypothesis of CSR. Taking `nsim=19` this test has nominal significance level (=probability of rejecting the null hypothesis when it is true) equal to 0.05. To estimate the true significance level, we generate simulated data according to CSR:

```
> N <- 10000
> lambda <- 50
> rejected <- logical(N)
> for(i in 1:N) {
  X <- rpoispp(lambda)
  rejected[i] <- dotest(X)
}
> alpha <- mean(rejected)
```

This gives an estimated true significance level of `alpha = 0.046` with standard error `sqrt(alpha * (1-alpha)/N) = 0.002` so the Monte Carlo test is only slightly conservative.

In the case of CSR, a solution to the problem of conservatism is to hold the number of points fixed. We generate the simulated patterns with the same number of points as the observed pattern. This exploits the conditional property (PP4) defined on page 267. If the number of points $n(\mathbf{x})$ is known, the locations of the points are independent and uniformly distributed, whatever the value of the intensity λ . That is, conditional on the observed number of points $n(\mathbf{x}) = n$, the joint spatial distribution of the points x_1, \dots, x_n does not depend on the intensity parameter. It follows that the Monte Carlo test of CSR *conditional on the number of points* is exact (non-conservative).

In the more realistic scenario where the null hypothesis is a model involving spatial trend and interpoint interaction, requiring several parameters to be fitted, the conservatism effect may be greater and yet harder to handle. In general we would need to generate simulations conditionally on the value of the sufficient statistic for the model. This may be quite difficult. The statistic $n(\mathbf{x})$ is typically one component of the sufficient statistic, so conditioning on $n(\mathbf{x})$ may partially alleviate the conservatism effect.

A conservative test may be tolerable in some applications, since it effectively applies a standard of statistical significance that is more stringent than intended. The current usage of Monte Carlo tests in spatial statistics is defensible for some purposes, such as exploratory data analysis. However, for goodness-of-fit testing, a very conservative test is problematic, since a “non-significant” outcome (where the null hypothesis is not rejected) is often misinterpreted as confirmation of the fitted model. For definitive formal inference and for goodness-of-fit tests, some remedy is needed.

The correct handling of p -values for composite null hypotheses is regarded as an unresolved research problem in statistical inference. See, for example, [73, 52, 286]. Various strategies have been suggested, including adjusting the test statistic so that its mean value is less sensitive to the parameter [286]; using a summary function which is unrelated to the model fitting procedure [127, p. 89]; and adjusting the p -value itself by performing additional simulations [73, 114]. Some of these remedies are discussed below.

10.8 Monte Carlo tests based on summary functions

Monte Carlo tests can be based on the envelopes of summary functions such as the K -function.
Introduced in chap 7

Monte Carlo tests for spatial point patterns, based on simulation envelopes of summary functions, were pioneered by Ripley [281] and their statistical rationale was set out in detail in [58, 282, 283]; see also [237].

The test may be performed using any summary function for the point pattern. Typical choices would be the K -function, L -function or pair correlation function g (Chapter 7), the empty space function F or the nearest-neighbour distance distribution function G (Chapter 8).

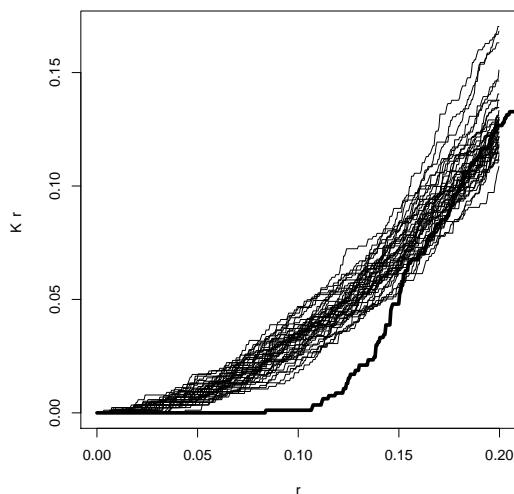


Figure 10.3: Estimated K -function for cells data (thick line) and K -functions of 39 simulated datasets (thin lines).

We will use the letter S to stand for any chosen summary function. To perform the test, we first calculate the S -function estimate for the observed data, $S_{obs}(r)$. Next we generate m random simulated point patterns, and obtain their S -function estimates, $S^{(1)}(r), \dots, S^{(m)}(r)$.

For example, Figure 10.3 shows the K -function estimated from the `cells` data (thick line), and also the estimated K -functions from $m = 39$ simulated realisations of a binomial process with the same number of points (thin lines). Clearly, the K -function estimated from the `cells` data lies outside the typical range of values of the estimated L -function for a completely random pattern. To conclude formally that there is a ‘significant’ difference between \hat{K} and K_{pois} , we apply a Monte Carlo test.

The Monte Carlo test principle of Section 10.7 cannot be applied to the function $S(r)$ directly,

because functions cannot be ranked from smallest to largest. We need to *reduce the function $S(r)$ to a single number T* that will serve as the test statistic. (This is a **HUGELY** important point, and one over which many users appear to get confused. Take careful note!) Four possible strategies are described in Sections 10.8.1, 10.8.3, 10.8.4 and ?? below.

For simplicity of exposition we will take the null hypothesis to be Complete Spatial Randomness in this Section.

This can be useful in initial investigation [127, p. 12], [19] where CSR serves as a “dividing hypothesis” which is often *known to be false* [100, pp. 51–52]. However, for definitive formal inference, CSR is not usually the appropriate null hypothesis. The null hypothesis should not be chosen naively, but should correspond to a scientifically meaningful scenario in which “nothing is happening” [317]. In plant ecology, the appropriate null hypothesis might involve spatial inhomogeneity due to known environmental factors, and spatial clustering and regularity due to known processes of dispersal and competition [233, p. 1929].

10.8.1 Pointwise envelope test

One strategy for reducing the summary function $S(r)$ to a single numerical value T , is to evaluate $S(r)$ at a prespecified, fixed distance $r = r_0$. For example, using the K -function and choosing $r_0 = 0.1$ units, the test statistic would be the numerical value $T = K(0.1)$ of the K -function at the prespecified distance $r = 0.1$.

One can imagine drawing a vertical line in Figure 10.3 at the position $r = 0.1$ units. Each K -function curve crosses this vertical line at a single point. The heights of these crossing-points serve as the test statistic values for a Monte Carlo test.

The two-sided Monte Carlo test with significance level $\alpha = 2/(m+1)$ rejects the null hypothesis if the observed value $t_{obs} = S_{obs}(r_0)$ lies outside the range of all the simulated values $t_1 = S^{(1)}(r_0), \dots, t_m = S^{(m)}(r_0)$, that is, if either $t_{obs} > \max_j t_j$ or $t_{obs} < \min_j t_j$. This is equivalent to drawing the graph of the *upper and lower envelopes*

$$\begin{aligned} U(r) &= \max_j S^{(j)}(r) = \max\{S^{(1)}(r), \dots, S^{(m)}(r)\} \\ L(r) &= \min_j S^{(j)}(r) = \min\{S^{(1)}(r), \dots, S^{(m)}(r)\}, \end{aligned} \quad (10.14)$$

slicing this graph at the position $r = r_0$, and rejecting the null hypothesis if the observed function $S_{obs}(r)$ lies outside this envelope *at the prespecified distance $r = r_0$* . More generally the upper and lower boundaries of the envelope are, for each value of r , the k th largest and k th smallest of the simulated values of $S(r)$.

Figure 10.4 shows the upper and lower envelopes of the simulated K -functions in Figure 10.3. The region between the envelopes is shaded. The plot was generated by the command

```
> plot(envelope(cells, Kest, nsim=39, fix.n=TRUE))
```

Picking the value $r_0 = 0.1$ units, for example, the null hypothesis of CSR is rejected by the two-sided test with significance level 0.05.

The outcome of the pointwise Monte Carlo test depends only on the relative ordering of the values $S_{obs}(r_0)$ and $S^{(1)}(r_0), \dots, S^{(m)}(r_0)$. Therefore the same test outcome will be obtained if we apply any monotone increasing transformation to the summary function. For example, the K -function could be replaced by the L -function $L(r) = \sqrt{K(r)/\pi}$, the centred functions $K(r) - \pi r^2$ or $L(r) - r$, and so on, without affecting the outcome of the pointwise test.

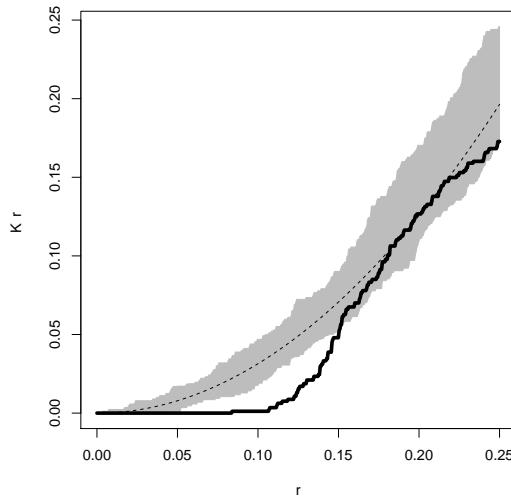


Figure 10.4: “Pointwise envelope” corresponding to Figure 10.3. Grey shaded region is bounded by the maximum and minimum values of simulated K -functions at each distance r .

10.8.2 Misinterpretation of pointwise envelopes

10.8.2.1 Spurious significance

An important caveat about pointwise envelopes is that their formal interpretation as a significance test requires the distance value r to have been fixed in advance of the experiment and the analysis.

In Figure 10.4, the K -function for the data point pattern wanders outside the envelope of the K -functions of 39 simulated patterns generated under CSR. Many writers attach a statistical significance of $2/(39 + 1) = 0.05$ to this outcome, using the Monte Carlo test principle. However, this reasoning would be valid only if we had decided in advance to inspect the K -function at one particular interpoint distance. The usual practice is to plot the simulation envelope without any pre-conceived idea, and look for deviations from the envelope at *any* position. This practice leads to invalid statistical inferences. The probability that the K function will wander outside the envelope *somewhere* is very much higher than 0.05, so that the results in Figure 10.4 cannot then be declared statistically significant at the 0.05 level.

This caveat was mentioned in Ripley’s pioneering paper [281, p. 181] and has been re-emphasised by leading expositors of spatial statistics [123, p. 12]. An influential paper by Loosmore and Ford [233, p. 1926] also draw attention to this common error. However, this has been widely misinterpreted as meaning that simulation envelopes do not have *any* valid interpretation as a significance test. Many recent writers have advised that “envelope tests should not be thought of as formal tests of significance” [219, p. 619] and have drawn a distinction between invalid “envelope tests” and valid “deviation tests” [161].

This opinion is at odds with the viewpoint widely held in statistical science, especially in spatial statistics [127, 193, pp. 455-459] and nonparametric statistics [69, 79], that simulation envelopes do have a valid statistical interpretation. Indeed “envelope tests” and “deviation tests” have the *same* statistical rationale, that of a Monte Carlo test, so it cannot be true that envelope tests are invalid while deviation tests are valid.

The pointwise envelope does give a valid significance test when it is correctly interpreted and applied. If prior information indicates that the random points are likely to interact over distances in the range from 0 to r_0 , then a sensible and valid test of statistical significance is to reject the null hypothesis if the K -function lies outside the simulation envelope at distance $r = r_0$.

Different values of the distance variable r correspond to different Monte Carlo tests. Plotting the pointwise envelope over a range of r values (cf. Figure 10.4) effectively displays the outcomes of these different tests. Each test is valid if performed on its own, but it is generally not valid to perform all the tests and to combine their results in some arbitrary fashion. In particular it is not valid to scan the plot searching for values of r where the empirical function $S_{obs}(r)$ falls outside the envelope. This is a problem of *multiple testing* or *multiple comparison* [182, 188, 298].

One benefit of graphically displaying a summary function like the K -function is that it contains information from different spatial scales. Plotting the pointwise envelope over a range of r values enables us to assess, for different distance values r , what the result of the test *would have been* if we had chosen that distance value to perform the test. This is a useful diagnostic since it indicates the sensitivity of the test outcome to the choice of distance value r . The use of such “significance traces” is standard practice in other fields of statistics [69, 79].

There are other types of simulation envelopes which do not suffer from the same risk of misinterpretation; see Section 10.8.3.

10.8.2.2 Confidence bands

Many writers describe the simulation envelope as a “confidence envelope”, “confidence interval” or similar ([197, p. 1020], [233, p. 1926]). Such phrasing constitutes incorrect use of statistical terminology and may be a source of confusion.

Confidence intervals for the K and L summary functions were discussed in Chapter 7, Section 7.7. In brief, a **confidence interval** is designed to contain the *true* value of the target quantity with a specified degree of confidence. A confidence interval is usually centred around an *estimated* value of the target quantity. By contrast, an **acceptance interval** (or “non-rejection interval”) represents a statistical hypothesis test; it is the range of values that are not significantly different from the null value. An acceptance interval is usually centred around the *hypothesised* value of the target quantity.

True confidence intervals or confidence bands for the K -function can be constructed using several well-established methods. Figure 10.5 shows a pointwise 95% confidence band for the true K -function of the natural process that generated the real (`cel1s`) data shown in the top left hand corner of Figure 10.1. The confidence band was obtained using a bootstrap method proposed by Loh [232]. Note that the bands are centred around the K -function of the data. Such confidence bands might be useful in some applications of statistical ecology; they may in some instances constitute a better approach to statistical inference than does formal hypothesis testing [100, 345].

[See Section 7.7 for ways of making confidence intervals for $K(r)$.]

10.8.3 Simultaneous envelopes and MAD test

Another strategy for reducing the summary function $S(r)$ to a single numerical value T , is to compute the maximum deviation between the S -function of the observed data and the theoretical (‘expected’) S -function of the null model.

In some cases the theoretical value of the S -function under the null model is known. For example under CSR, the K -function takes the form $K_{theo}(r) = \pi r^2$ and so $L_{theo}(r) = r$. In such cases we may take the test statistic T to be the maximum deviation, in absolute value, between $S(r)$ and its theoretical value $S_{theo}(r)$ under the null model, where the maximum is taken over the range of distances from 0 to R units, say, where R is a chosen upper limit on the interaction distance (to be discussed below). That is, we choose

$$T = \max_{0 \leq r \leq R} |S(r) - S_{theo}(r)|. \quad (10.15)$$

Then T is the maximum vertical separation between the graphs of $S(r)$ and $S_{theo}(r)$ over the chosen

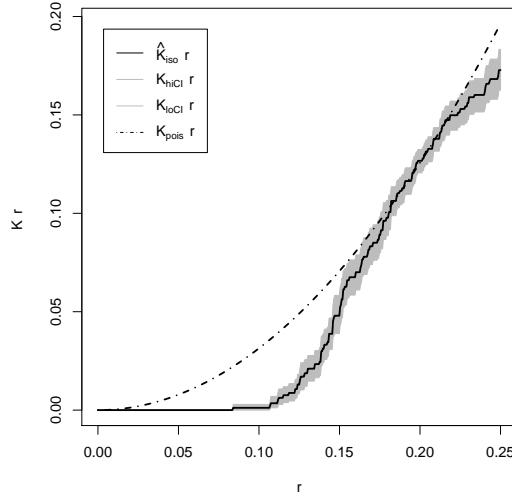


Figure 10.5: Pointwise **confidence bands** for the true K -function of the `cells` data, using the bootstrap method of Loh [232]. Generated by `plot(lohboot(cells, "Kest"))`.

range of distances. This procedure reduces the data to a single number T called the “maximum absolute deviation” (MAD), analogous to the Kolmogorov-Smirnov test statistic (10.6).

To apply a Monte Carlo test we would compute the observed value t_{obs} of the MAD for the point pattern data, and the MAD values t_1, \dots, t_m for the simulated point patterns, then reject the null hypothesis if t_{obs} is greater than the maximum $t_{max} = \max\{t_1, \dots, t_m\}$ of all the simulated values. This is the *MAD test* with significance level $\alpha = 1/(m+1)$.

The `spatstat` function `mad.test()` performs this test. It calls `envelope()` to perform the simulations and to compute the summary functions; the syntax of `mad.test()` is essentially the same as that of `envelope()`.

```
> mad.test(cells, Lest, nsim=99, ginterval=c(0, 0.2),
  fix.n=TRUE, verbose=FALSE)
Maximum absolute deviation test of CSR
Monte Carlo test based on 99 simulations with fixed number of
points
Summary function: L(r)
Reference function: sample mean
Alternative: two.sided
Interval of distance values: [0, 0.25]

data: cells
mad = 0.0866, rank = 1, p-value = 0.01
```

The result of `mad.test()` is a hypothesis test object (class "htest") with additional attributes storing information about the simulations.

The MAD test rule is equivalent to plotting an envelope of *constant width* $2t_{max}$ centred on the theoretical curve $S_{theo}(r)$, that is, the region bounded by the limits

$$\begin{aligned} L(r) &= S_{theo}(r) - t_{max} \\ U(r) &= S_{theo}(r) + t_{max} \end{aligned} \tag{10.16}$$

and rejecting the null hypothesis if the observed S -function $S_{obs}(r)$ ever wanders outside this envelope. This is called the *global* or *simultaneous* envelope. Global envelopes were also proposed by Ripley [281, 282, 283], but many writers do not appear to be aware of them.

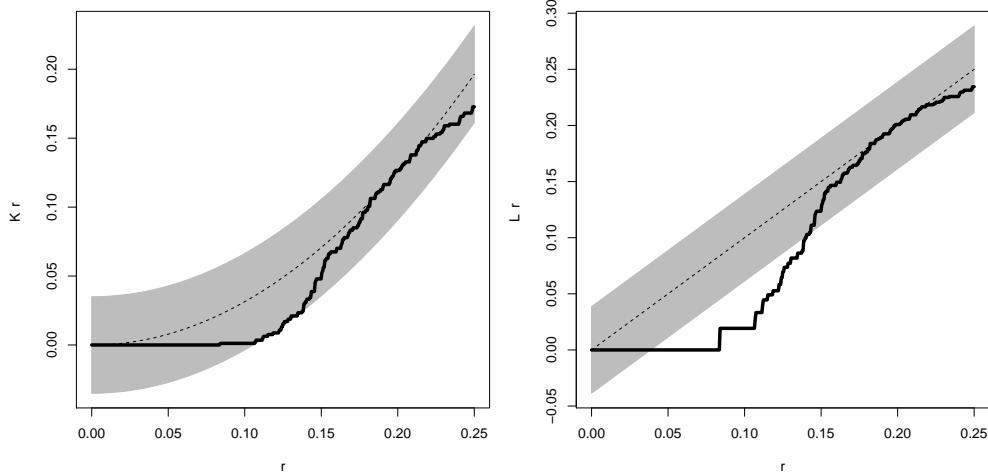


Figure 10.6: “Global envelopes” based on the K -function (Left) or L -function (Right). Grey shading shows regions of constant vertical width determined by the maximum deviation (from the expected value for CSR) of the summary function estimates for 19 simulations of CSR.

See Figure 10.6 for an example with $m = 19$ so that $\alpha = 0.05$. This was generated by calling `envelope()` with the argument `global=TRUE`. As distinct from the “pointwise envelope” in Figure 10.4, the “global envelope” in each panel of Figure 10.6 is a zone of *constant width*. The width is determined by finding the most extreme deviation (from the theoretical curve) achieved by any of the 39 simulated curves, at any distance r along the horizontal axis. The centred L -function of the data point pattern wanders outside the global envelope; this is statistically significant at the level $1/(1+19) = 0.05$, often reported in terms of the p -value as “ $p < 0.05$ ” although the p -value is actually equal to 0.05. The global envelope test is statistically valid; the problem of multiple testing has been avoided.

Note that the outcome of the MAD test is affected by transformations of the summary function. In Figure 10.6, different test outcomes were obtained using the K and L functions on the same data and the same simulated patterns.

In the general case the MAD test rejects the null hypothesis if t_{obs} is greater than the k -th largest of the simulated values, and this has significance level $\alpha = k/(m+1)$.

If the theoretical value $S_{theo}(r)$ is not known for every r , then it could be estimated from a separate set of simulations of the null model [123], which guarantees the basic requirement of symmetry. Alternatively, using only a single set of simulations, we can replace $S_{theo}(r)$ by

$$\bar{S}(r) = \frac{1}{m+1} (S^{(1)}(r) + \dots + S^{(m)}(r) + S_{obs}(r)), \quad (10.17)$$

the average of all the simulated and observed S -functions. This preserves symmetry and ensures that the test has significance level $k/(m+1)$.

10.8.4 DCLF test

A third strategy for reducing the summary function $S(r)$ to a single numerical value T , is to compute the integrated squared deviation from the theoretical value,

$$T = \int_0^R (S(r) - S_{\text{theo}}(r))^2 dr \quad (10.18)$$

as proposed by Diggle [125, p. 122], [127, eq. (2.7), p. 12]. Here R is a chosen upper limit on the range of distances of interest. The statistic T was subsequently advocated by Cressie [108, eq. (8.5.42), p. 667] and Loosmore and Ford [233]. A Monte Carlo test based on T has come to be known as the *Diggle-Cressie-Loosmore-Ford (DCLF) test*.

The motivation is that in many statistical applications it is better to use an integrated squared error than a maximum absolute error. For example in goodness-of-fit testing, the Kolmogorov-Smirnov test, based on the maximum absolute difference (10.6), typically has less power than the Cramér-von Mises test, based on a weighted integral of squared difference (10.7). This heuristic argument is borne out by experiments [25].

If the theoretical value S_{theo} has to be estimated then (10.18) is replaced by

$$T = \int_0^R (S(r) - \bar{\bar{S}}(r))^2 dr, \quad (10.19)$$

where again $\bar{\bar{S}}$ is the average of the simulated and observed S -functions as in (10.17).

The test is performed in `spatstat` using `dclf.test()`, analogous to `mad.test()`. The function `dclf.test()` calls `envelope()` to perform the simulations and to compute the summary functions; the syntax of `dclf.test()` is essentially the same as that of `envelope()`.

```
> dclf.test(cells, Lest, nsim=99, fix.n=TRUE, verbose=FALSE)
   Diggle-Cressie-Loosmore-Ford test of CSR
   Monte Carlo test based on 99 simulations with fixed number of
   points
   Summary function: L(r)
   Reference function: sample mean
   Alternative: two.sided
   Interval of distance values: [0, 0.25]

   data: cells
   u = 5e-04, rank = 1, p-value = 0.01
```

The result of `dclf.test()` is a hypothesis test object (class "htest") with additional attributes storing information about the simulations. In the printed output, the line `reference function: sample mean` indicates that (10.19) was computed rather than (10.18).

10.8.5 Choice of test

The three strategies outlined in described in Sections 10.8.1, 10.8.3, 10.8.4 and ?? above are equally valid. The choices of summary function S and test statistic T materially affect the *performance* of the test, but not its basic validity. It will often be sensible to transform the summary function S (for example, transforming the K -function to the L -function) or to weight the values of S according to their variances under the null hypothesis [108, p. 642] in order to improve performance. Factors that affect performance are discussed in Section 10.11.

10.9 Envelopes in spatstat

The `spatstat` function `envelope()` performs the calculations required for envelopes. It computes the summary function for a point pattern dataset, generates simulated point patterns, computes the summary functions for the simulated patterns, and computes the envelopes of these summary functions. The result is an object of class "envelope" and "fv" which can be printed and plotted immediately. For example, Figure 10.7 was generated by the commands

```
> plot(envelope(redwood, Lest, nsim=39))
> plot(envelope(redwood, Lest, nsim=39, global=TRUE))
```

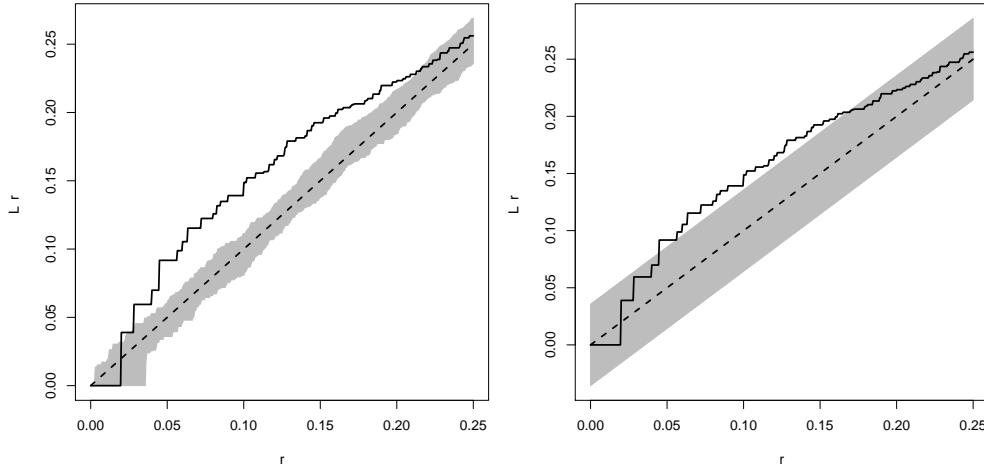


Figure 10.7: Example envelopes. Pointwise (*Left*) and global (*Right*) envelopes of Besag's L -function based on 39 simulations of CSR, and observed L -function for the `redwood` data.

The function `envelope()` is generic, with methods for class "ppp", "ppm" and several other classes. If X is a point pattern, then by default `envelope(X, ...)` constructs pointwise envelopes for the null hypothesis of CSR. If M is a fitted point process model, then by default `envelope(M, ...)` constructs pointwise envelopes for the null hypothesis represented by the model M . The default behaviour is to construct a pointwise envelope from the K -functions of 99 simulations. Different types of envelopes, different null hypotheses, different alternative hypotheses, and different simulation procedures, are selected using additional arguments listed in Table 10.1.

10.9.1 Choice of the null hypothesis

The most important consideration, and also the most prone to mistakes, is the choice of null hypothesis for the envelope test.

10.9.1.1 Envelopes for testing CSR

Envelopes for testing Complete Spatial Randomness are generated (by default) if the first argument of the `envelope()` command is a point pattern.

If Y is an *unmarked* point pattern, by default the simulated point patterns are realisations of Complete Spatial Randomness (the uniform Poisson point process) with the same intensity as the

ARGUMENT	DEFAULT	MEANING
Y		point pattern data, or fitted model
fun	Kest	summary function
nsim	99	number of simulations m for test
nrank	1	rank for statistical significance
global	FALSE	global envelope
VARIANCE	FALSE	envelope based on sample variance
simulate		procedure for generating simulated patterns
alternative	two.sided	alternative hypothesis
fix.n	FALSE	fix number of points
fix.marks	FALSE	fix number of points of each type
correction		edge correction for summary function
transform		transformation of summary function
ginterval		range of distance values for global envelope
nsim2	nsim	number of simulations to estimate mean
savepatterns	FALSE	store point patterns
savefuns	FALSE	store summary functions

Table 10.1: Common arguments for envelope

pattern Y and in the same window as Y. Each simulated pattern has a random number of points, according to a Poisson distribution.

If fix.n=TRUE, simulations are generated from CSR conditionally on having the same number of points as the observed data. (This argument can be used to avoid the conservatism inherent in the test provided by unconditional CSR, as discussed in Section ??). By property (PP4) of the Poisson process (page 267), this means that given the dataset x, each simulated pattern consists of $n = n(\mathbf{x})$ independent random points distributed uniformly in the same window as x. In either case, the resulting envelopes support a test of CSR.

If Y is a multitype point pattern, then the simulated patterns are also given independent random marks. The simulated patterns are realisations of Complete Spatial Randomness and Independence (the uniform Poisson point process with independent random marks). By default the probability distribution of the random marks is determined by the relative frequencies of marks in Y. If fix.marks=TRUE, the number of points of each type is held fixed.

In either case, the resulting envelopes support a test of CSRI.

10.9.1.2 Envelopes for any fitted model

If the first argument of envelope is a fitted point process model (object of class "ppm", "kppm" or "lppm"), 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.

If fix.n=TRUE the number of points in each simulated realisation is held fixed; for multitype point process models, if fix.marks=TRUE, the number of points of each type is held fixed. Other arguments are available to control the simulation procedure: for a full list see help(envelope).

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

```
> numata <- residualspaper$Fig1
> fit <- ppm(numata ~ polynom(x,y,3))
```

```
> E <- envelope(fit, Lest, nsim=19, global=TRUE, correction="border")
> plot(E)
```

10.9.1.3 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, when evaluated, yields a point pattern. Typically the expression should involve a random generator, so that the result of evaluating the expression is different each time it is evaluated. For example if we type

```
> e <- expression(rpoispp(100))
```

then each time the expression `e` is evaluated, it will yield a different random outcome of the Poisson process with intensity 100 in the unit square. Try this by typing `eval(e)` several times.

The expression should be passed to the `envelope` function as the argument `simulate`. Then simulated point patterns will be generated by evaluating the expression repeatedly.

This makes it possible to perform a *randomisation test* in which each simulated point pattern is a random alteration of the original data. For example, the function `rlabel()` randomly permutes the mark values of a marked point pattern. The following code generates and plots simultaneous envelopes for the cross-type L function based on random labellings of the `amacrine` data.

```
> e <- expression(rlabel(amacrine))
> E <- envelope(amacrine, Lcross, nsim=19, global=TRUE, simulate=e)
> plot(E)
```

The corresponding null hypothesis is that the pattern has the “*random labelling property*”: given the locations of the points, the marks are independent random variables with a common distribution.

10.9.1.4 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. The argument `simulate` can be a list of point patterns:

```
> Xlist <- list()
> for(i in 1:99) Xlist[[i]] <- runifpoint(42)
> envelope(cells, Kest, nsim=99, simulate=Xlist)
```

The argument `simulate` can also be an “envelope” object. This improves efficiency and consistency if, for example, we are going to calculate the envelopes of several different summary statistics.

```
> EK <- envelope(cells, Kest, nsim=99, savepatterns=TRUE)
> Ep <- envelope(cells, pcf, nsim=99, simulate=EK)
```

In the first call to `envelope`, the argument `savepatterns=TRUE` indicates that we want to save the simulated point patterns. These are stored in the object `EK`. Then in the second call to `envelope`, the simulated patterns are extracted from `EK` and used to compute the envelopes of the pair correlation function.

The method `envelope.envelope` allows envelope calculations to be based on an existing “envelope” object. The previous example could have been done by typing

```
> EK <- envelope(cells, Kest, nsim=99, savepatterns=TRUE)
> Ep <- envelope(EK, pcf)
```

The same principle applies to the functions `mad.test()` and `dclf.test()`, which have essentially the same syntax as `envelope()`. For example, to perform both the MAD and DCLF tests of CSR using the same simulations for both tests:

```
> A <- mad.test(cells, Lest, nsim=99, savefun=TRUE)
> B <- dclf.test(A)
```

10.9.2 The summary function and its arguments

The second argument of `envelope()` specifies the summary function `fun` to be used in constructing the envelopes. The default is `fun=Kest` giving envelopes of the K -function.

Typically `fun` is one of the standard `spatstat` summary functions such as `Kest`, `Gest`, `Fest`, `Jest`, `pcf` or one of their variants. It may also be a character string containing the name of one of these functions. Optionally, `fun` could be a function created by the user.

Additional arguments to `fun` may be given in the call to `envelope`. For example, to compute the translation edge correction estimate of the K -function, we would call `Kest` with the argument `correction="trans"`. To compute envelopes based on the translation-corrected K -function, we simply type

```
> envelope(X, Kest, correction="trans")
```

Note that `correction` is not a formal argument of `envelope` or `envelope.ppp`.

The list of available edge corrections is different for each summary function, and may also depend on the kind of window in which the point pattern is recorded. However, all the summary functions in `spatstat` recognise the options `correction="best"` and `correction="good"` which indicate respectively the most accurate edge correction, and the most accurate *fast* edge correction, respectively. In a call to `envelope`, if `fun` is one of the summary functions provided in `spatstat`, then the default is `correction="best"`. This means that *by default, the envelope will be computed using the “best” available edge correction*. To increase speed, the user could specify `correction="good"` or choose a specific edge correction.

Envelopes based on a summary function like `Kinhom` need careful attention. Recall that `Kinhom(X, lambda)` computes an estimate of the inhomogeneous K -function of the process which generated `X`, assuming that the intensity function is given by `lambda`. Typically `lambda` should be estimated from the data `X` somehow. If `lambda` is missing, it is estimated by kernel smoothing, so that `Kinhom(X)` produces a valid estimate. The following is therefore a valid way to build envelopes:

```
> D <- density(X)
> envelope(X, Kinhom, simulate=expression(rpoispp(D)))
```

The null hypothesis is an inhomogeneous Poisson process; the intensity of the process is estimated by kernel smoothing the data to yield the image `D`; realisations of this fitted model are generated by evaluating the expression. For each simulated pattern, since `lambda` is not given, the intensity will be estimated by kernel smoothing the simulated pattern, and the inhomogeneous K -function for the simulated pattern will be estimated. The procedure used to compute an inhomogeneous K -function is exactly the same for the simulated patterns as it is for the data.

However, the following is **not valid**:

```
> D <- density(X)
> envelope(X, Kinhom, lambda=D, simulate=expression(rpoispp(D)))
```

The null hypothesis and the simulation procedure are the same as above, but in this case the intensity function estimate `D` obtained from the data will be used to compute the inhomogeneous K -functions

of the simulated patterns as well as the original data. The simulated patterns are not treated in the same way as the data. The corresponding envelopes do not support a valid Monte Carlo test of the null hypothesis. The `spatstat` package will usually detect this kind of problem and issue a warning.

For a parametric model, the following is **valid**, but depends on a special feature of `Kinhom`:

```
> fit <- ppm(.....)
> envelope(fit, Kinhom, lambda=fit)
```

Here the null hypothesis is the Poisson point process model specified by the trend formula in `fit`. The simulated point patterns will be generated from this Poisson process, using the parameters estimated in `fit`. The special feature is that when `lambda` is a fitted point process model, `Kinhom(XX, lambda)` will re-fit the model to the new point pattern `XX`, before computing the predicted intensity function, and using this intensity to compute the inhomogeneous K -function estimate. The procedure is exactly the same for the simulated patterns as it is for the data.

10.9.3 Types of envelope

Pointwise envelopes

By default, `envelope()` computes pointwise envelopes of the summary function. The argument `nsim` controls the number m of simulated point patterns. The argument `nrank` specifies the rank k of the critical value amongst the simulated values: the envelope is based on the k th largest and k th smallest simulated values. The default is $k = 1$, meaning that the minimum and maximum simulated values will be used. For a two-sided test, the significance level is $2k/(m+1)$.

Simultaneous envelopes

To obtain simultaneous (global) envelopes, set `global=TRUE`:

```
> envelope(redwood, Lest, global=TRUE)
```

The significance level is $k/(m+1)$.

As noted in Section 10.8.3, the outcome of the test is affected by transforming the summary function. In order to achieve the maximum power of the test, it is usually recommended to apply a *variance-stabilising* transformation to the summary function, that is, a transformation $H(r) = f(S(r))$ such that the variance of $H(r)$ is approximately constant as a function of r . The global measure of deviation (10.15) treats all fluctuations of the summary function S equally; this could be inappropriate if fluctuations of $S(r)$ at some distance r are more informative than at other distances. After a variance-stabilising transformation, fluctuations in the summary function at different distances r have equal scale, so can be treated equally.

An approximate variance-stabilising transformation for the K -function is the square root. Indeed this was one of the motivations for Besag's [61] original proposal of the L -function, $L(r) = \sqrt{K(r)/\pi}$. Thus we would typically use `Lest` rather than `Kest` for global envelopes.

For the summary functions F and G , an approximate variance-stabilising transformation is Fisher's arcsine transformation $f(x) = \arcsin\sqrt{x}$. Rather than writing a new function for the variance-stabilised versions of `Fest` and `Gest`, we can specify the transformation using the argument `transform`:

```
> fisher <- function(x) { asin(sqrt(x)) }
> envelope(redwood, Fest, global=TRUE,
            transform=expression(fisher(.)))
```

To allow very flexible specification of the transformation, `envelope()` uses `with.fv()` to evaluate the `transform` expression. In the syntax of `with.fv()`, the symbol “.” represents the function value, in this case the value of the estimate $\hat{F}(r)$.

10.9.3.1 Envelopes based on sample mean & variance

Envelopes can be constructed using the sample mean and sample variance of the simulations. By default the envelope is the sample mean plus or minus 2 times the sample standard deviation. This is useful for understanding the variability of the summary function. Be aware that these envelopes do not have the same significance interpretation.

```
> envelope(cells, Kest, nsim=100, VARIANCE=TRUE)
```

10.9.3.2 One-sided envelopes

In `envelope` methods, the argument `alternative` is a character string determining whether the envelope corresponds to a two-sided test (`alternative="two.sided"`, the default) or a one-sided test with a lower critical boundary (`alternative="less"`) or a one-sided test with an upper critical boundary (`alternative="greater"`).

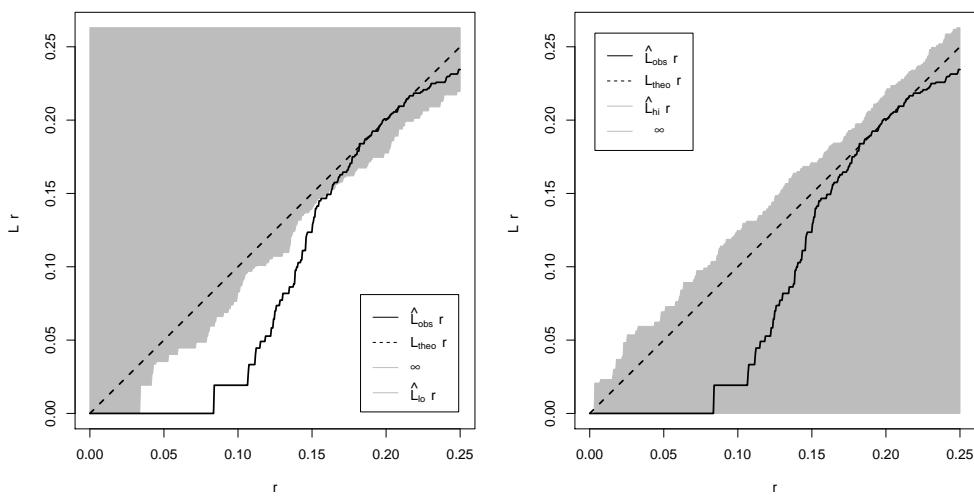


Figure 10.8: One-sided envelopes. Grey shading shows the pointwise acceptance region for a one-sided test of CSR against the alternative of regularity (Left) or clustering (Right) based on the L -functions of 19 simulations of CSR. Thick line shows empirical L -function for the `cells` data.

10.9.4 Re-using envelope data

Computing new envelopes

The method `envelope.envelope` allows new `envelope()` commands to be applied to a previously-computed "envelope" object, provided it contains the necessary data.

In the original call to `envelope()`, if the argument `savepatterns=TRUE` was given, the resulting "envelope" object contains all the simulated point patterns. Alternatively if the argument `savefuns=TRUE` was given, the resulting object contains the individual summary functions for each of the simulated patterns. This information is not saved, by default, for efficiency's sake.

Envelopes created with `savepatterns=TRUE` allow any kind of new envelope to be computed using the same simulated point patterns:

```
> E1 <- envelope(redwood, Kest, savepatterns=TRUE)
```

```
> E2 <- envelope(E1, Gest, global=TRUE,
                  transform=expression(fisher(.)))
```

Envelopes created with `savefun`=`TRUE` allow the user to switch between pointwise and global envelopes of the same summary function, to apply different transformations of the summary function, and to change some parameters:

```
> A1 <- envelope(redwood, Kest, nsim=39, savefun=TRUE)
> A2 <- envelope(A1, global=TRUE, nsim=19,
                  transform=expression(sqrt(./pi)))
```

Pooling several envelopes

It is also possible to combine the simulation data from several envelope objects and to compute an envelope based on the combined data. This is done using `pool.envelope()`, a method for the `spatstat` generic `pool()`. The envelopes must be compatible, in that they are envelopes for the same function, and were computed using the same options. The individual summary functions must have been saved.

```
> E1 <- envelope(cells, Kest, nsim=10, savefun=TRUE)
> E2 <- envelope(cells, Kest, nsim=20, savefun=TRUE)
> E <- pool(E1, E2)
```

10.10 Hahn tests?

10.11 Performance

The performance of a test is measured by its *power*, the probability of making the correct decision if the null hypothesis is false. Here we discuss various strategies that can improve the power of a Monte Carlo test, and we measure the power of tests that are based on summary functions.

10.11.1 Factors that affect power

The power of a test is defined as the probability of rejecting the null hypothesis when the null hypothesis is false and a specified alternative hypothesis is true (i.e. when the data were actually generated in some other specified way). The test power depends on this alternative hypothesis and can be regarded as a measure of the sensitivity of the test to the specified alternative. A test may have strong power against one alternative and weak power against another alternative.

All the choices involved in designing a Monte Carlo test will affect the power of the test. The power typically increases if we increase the number of simulations m , for a fixed level of significance α . The power is affected by the choice of summary function S and the test statistic T .

The choice of summary function is pivotal in determining the sensitivity of the test to different types of spatial pattern. For example an envelope test based on the nearest-neighbour distance distribution G is usually very sensitive to the presence of inhibition between points, but insensitive to clustering. In extreme cases, two different spatial point process models may have exactly the same summary function. [44] constructed a point process which has the same K -function as CSR, but is

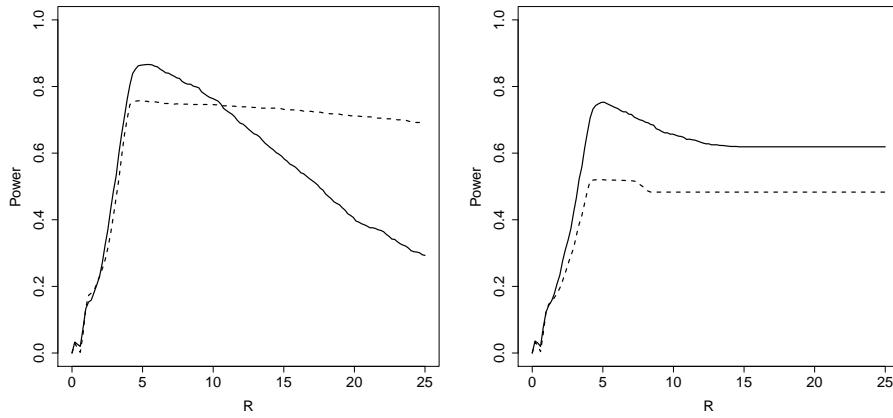


Figure 10.9: Performance curve for the DCLF test (solid lines) and the MAD test (dashed lines) for detecting spatial inhibition. The power is plotted against the length of the interval of distance values used in the test. Null hypothesis: CSR. Alternative hypothesis: inhibitory point process with interaction range $s = 4$ metres. *Left:* test based on the L -function. *Right:* test based on the variance-stabilised G -function.

manifestly different from CSR. A test of CSR based on the K -function has no utility against this alternative.

The power of a test conducted in terms of an estimate of a summary function S depends partly on the sampling variability of that estimate. For the K -function, pointwise envelopes tend to have a “funnel” shape, because the mean and variance of $K(r)$ for a random point pattern are approximately proportional to r^2 . This means that the MAD test statistic (10.15) and the DCLF test statistic (10.19) tend to be more influenced by fluctuations of $K(r)$ occurring at larger distances r . Besag [61] (in the discussion of [281]) proposed transforming $K(r)$ to $L(r) = \sqrt{K(r)/\pi}$ before applying the MAD test. This transformation *stabilises* the variance, i.e. it renders the variance of $L(r)$ approximately constant as a function of r . Variance stabilisation substantially improves the power of the MAD test and of the DCLF test.

The nearest-neighbour distance distribution function G and the empty space function F [283, 127, 193] have greatest variability for intermediate values of r . The variance of $G(r)$ for random point patterns is approximately proportional to $G(r)(1 - G(r))$. The appropriate variance-stabilising transformation is due to Fisher [145] (see [4]) and involves replacing $G(r)$ by $G^\dagger(r) = \arcsin(\sqrt{G(r)})$. Similar comments apply to F .

10.11.2 Measurements of power

We have compared the power of the DCLF test and the MAD test, based on different summary functions, in a series of simulation experiments. Following is a summary of our findings; more detail will be reported in a forthcoming paper [26]. A detailed description of experiments to measure the power of a test has been given by [205, sec. 3.5].

In all our experiments, the null hypothesis was CSR, while a range of alternative hypotheses was studied. For each choice of alternative hypothesis, we measured the power of the test by generating 1000 simulated realisations of the alternative hypothesis, performing the test on each simulated pattern, and counting the proportion of correct outcomes of the test. For comparison, both the DCLF

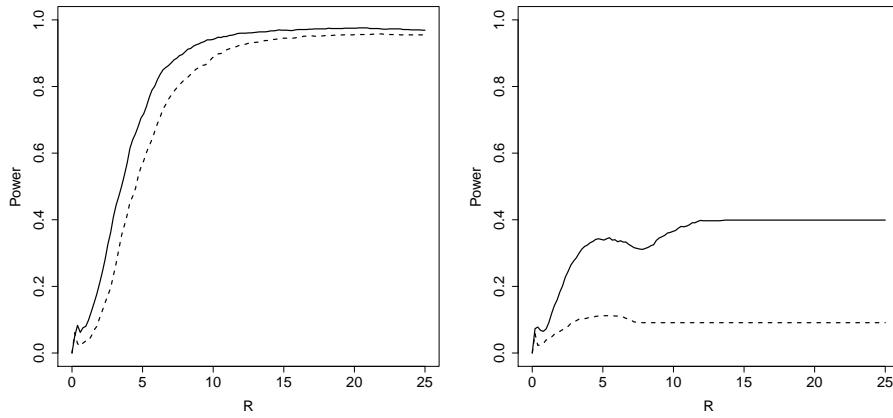


Figure 10.10: Performance curve for the DCLF test (solid lines) and the MAD test (dashed lines) for detecting spatial clustering. Null hypothesis: CSR. Alternative hypothesis: clustered process of [240] with cluster radius $s = 14$ metres. *Left:* test based on L -function. *Right:* test based on variance stabilised G -function.

and MAD tests were performed, using different choices of summary statistic, and different choices of the length of the interval R . The estimated power was plotted against R .

Two examples are shown in Figures 10.9 and 10.10. Figure 10.9 is the power curve for an alternative hypothesis that is an inhibitory (regular) point process, namely the *Strauss* process [316, 196], with intensity parameter $\beta = 0.025$, interaction parameter $\gamma = 0.5$ and interaction range $s = 4$ metres. For further information, see [127, p. 75]; [193, pp. 141, 147]. Figure 10.10 is the power curve when the alternative is the cluster point process of [240, pp. 46–47] as described in Section ??.

We found that the power of each test is maximised when the interval length R is slightly larger than the range of spatial interaction. Additionally, the power of a test based on the L function is greater than the power of the corresponding test based on the variance-stabilised G function $G^\dagger(r) = \arcsin \sqrt{G(r)}$.

If the alternative hypothesis is a point process with *inhibition* at distance s , and we use the L -function, then for $R > s$ the power of the DCLF test declines sharply as R increases, while the power of the MAD test is (essentially) constant, as shown in the left panel of Figure 10.9. Thus, the DCLF test is more powerful than the MAD test when R is close to s , but less powerful when R is large.

If we use instead the variance-stabilised G -function with the same alternative hypothesis, then the DCLF is always more powerful than the MAD test, as shown in the right panel of Figure 10.9. The power of each test rises sharply from $R = 0$ to $R = s$, then declines slightly before reaching a plateau.

If the alternative hypothesis is a point process with *clustering* at distance s , and we use either the L -function or variance-stabilised G -function, then for $R > s$ the power of both tests is (essentially) constant, and the DCLF test is more powerful, as shown in Figure 10.10.

Our power analysis showed that (1) the DCLF test is more powerful than the MAD test in most cases, (2) optimal power is achieved when the interval of distance values used for the test is slightly larger than the range of spatial interaction, and (3) power is improved by variance-stabilisation. An exception to (1) occurs when the data are spatially inhibited and the interval of distance values is much greater than the range of spatial interaction, as shown in the left panel of Figure 10.9. We expect the findings of this study to hold more widely, because they are consistent with the general pattern found in classical goodness-of-fit testing [308]: the DCLF and MAD tests are analogous,

respectively, to the Cramér–von Mises and Kolmogorov-Smirnov tests of goodness-of-fit, and the former is usually more powerful than the latter.

Consequently our recommendation is to use the DCLF test, provided the range of spatial interaction is known approximately (and the interval length is chosen accordingly). If there is no information about the range of spatial interaction, then it may be prudent to use the MAD test, choosing the interval length to be as large as practicable.

10.12 Conclusions

Statistical tests of spatial pattern based on simulation envelopes, and those based on measures of “deviation”, have the same statistical rationale, namely the Monte Carlo test principle. Both are valid statistical tests under appropriate conditions (including conditions on the way they are applied and interpreted). Simulation envelopes can be of several different kinds, including pointwise envelopes (Figure ??) and global envelopes (Figure ??), all of which are statistically valid when used appropriately. Pointwise envelopes carry a higher risk of misinterpretation or misuse, due to the problem of multiple testing.

If model parameters must be estimated from the data, then both “envelope tests” and “deviation tests” are strictly invalid, and probably conservative, so that significance will be under-reported. In the examples studied, this effect was quite substantial.

The Diggle-Cressie-Loosmore-Ford test generally performs better than competing methods, and is recommended, provided there is some prior information about the range of spatial interaction. If no such information is available, the global envelope test (maximum deviation test) should be used.

Envelopes are not confidence intervals. True confidence intervals for the K -function of a spatial pattern can be constructed using bootstrap techniques. Confidence intervals may be preferable to significance tests in many investigations. Envelopes and other graphical devices are useful for investigation of data but cannot be used directly to identify scales of spatial interaction without a parametric model.

10.13 FAQ's

- I applied two tests to the same data and obtained different answers: one test says the result is significant and the other says it is not. What is wrong? What do I do?

There's probably nothing wrong. Two different tests of the same null hypothesis will sometimes give different answers for the same data. (If they didn't sometimes disagree, they would be the same test.) Each test is typically designed to have high power (sensitivity) against a particular alternative hypothesis, at the expense of lower power against other alternatives.

- I'm confused about how to report the results of a hypothesis test. In this chapter you speak of statistical significance at the level 0.05, or a p -value less than 0.05, whereas some journals seem to report results as being “95% significant” or having “95% confidence”.
- “Which null model do you recommend: CSR or restricted randomisation?”
- I have heard that “envelope tests” are invalid and should be replaced by “deviation tests”.

“Envelope tests” and “deviation tests” have the *same* statistical rationale, that of a Monte Carlo test [137, 48, 183], so it cannot be true that envelope tests are invalid while deviation tests are valid. More explanation..

- A referee said that it is naive to perform a test of CSR, when we already know that CSR is false. Referee says that ecologists “long ago abandoned naive choices of the null model”, quoting Strong [317].

This depends on the scientific objectives and the stage of analysis. In initial investigation, CSR often serves as a “dividing hypothesis” although it is *known to be false* [100, pp. 51–52]: the test outcome guides the direction of the subsequent data analysis [127, p. 12], [19] and serves as a cross-check that the data are roughly consistent with our cherished theoretical assumptions.

In the final, definitive stages of analysis, the null hypothesis should correspond to a scientifically meaningful scenario in which “nothing is happening” [317]. Rejection of the null hypothesis allows us to conclude that “something is happening”, for example, that a particular variable does have an effect. In this context, CSR is often not the most appropriate scientific hypothesis. For example, in ecological applications, the appropriate null hypothesis might involve spatial inhomogeneity due to known environmental factors, and spatial clustering and regularity due to known processes of dispersal and competition.

Discuss Strong [317]

- Editorial policy of the journal requires that the conclusions should be summarised in a hypothesis test, but I don’t see why this is necessary for my case.
- Editorial policy of the journal is that hypothesis tests are invalid or unreliable, and that everyone should now be quoting confidence intervals for the *effect size*, rather than *p*-values.
- Editorial policy of the journal is that frequentist inferential methods like hypothesis tests are unreliable, and that everyone should now be using Bayesian inference.
- Why are there different tests for the same purpose? Which is best?

Because there is no universally optimal test. Some tests are better (more sensitive, higher power) against some alternatives. It may depend on what you’re willing to assume (since assumptions restrict both H_0 and H_1) or what you suspect about the type of departure from H_0 . [Universally *sub-optimal* tests are discarded!]

- When I do `envelope(X, nsim=99)` the printout tells me that the significance level is $\alpha = 0.02$, but I want significance level $\alpha = 0.01$. According to the help file for `envelope`, the significance level is $\alpha = 2k/(n + 1)$ where n is the number of simulations specified by the argument `nsim`, and k is the rank specified by the argument `nrank`. So by this formula I should set $k = 1/2$ to get $\alpha = 0.01$. But when I do `envelope(X, nsim=99, nrank=1/2)` it gives an error. Why?

Fractional ranks are meaningless. Rank 1 means the largest value, rank 2 means the second largest, etc.

Only some values of α are achievable (namely multiples of $2/(m + 1)$ in this case).

If you want $\alpha = 0.01$ you could set `nsim=199`, and leave `nrank` at its default value of 1.

- Why is hypothesis testing such a confusing topic?
 - Oh for the good old days when the guilt or innocence of an accused could be settled by simple procedures like drowning.
 - Like legal argument: fair hearing required; no-one expects court proceedings to be simple.
 - Expectation that it is easy
 - Wide application — all fields of research
 - To people who have been correctly trained, hypothesis testing is not confusing.
 - Many people who teach statistics or data analysis at universities are not qualified in statistics.



11

Validation of Poisson models

After fitting a point process model to a spatial point pattern dataset, we need to check that the model is faithful to the data. Techniques for checking or ‘validating’ a Poisson point process model are introduced in this Chapter.

Model validation is important because a fitted model is not like a fitted shoe. A shoe must conform to the shape of the foot reasonably well; but a model that we have “fitted” to data does not necessarily conform to the pattern of the real data *at all*.

For example, linear regression software, which “fits” a straight line through a cloud of data points, will perform this task even if the data follow a non-linear relationship, or follow no pattern at all. The software effectively assumes that the linear regression model is true. Software outputs such as confidence intervals and p -values also assume the model is true, and may give us a false sense of confidence about the validity of the model.

Similarly when we fit a Poisson point process model to a spatial point pattern dataset as described in Chapter 9, the model makes several assumptions which could be challenged. It assumes that the points are independent of each other, and often assumes that the intensity is a loglinear function of the specified covariates. The fitting procedure does not examine whether these assumptions are appropriate. This is a problem because an incorrect model could undermine the validity of the entire analysis [108, Section 8.5], [19].

A test of *goodness-of-fit* can be used to decide whether the data appear to follow the fitted model. The χ^2 test based on quadrat counts (Sections 6.4.2 and 6.7.1) is an example. Unfortunately, goodness-of-fit tests are rather uninformative: if the test finds significant evidence of “lack of fit”, we do not know which of the model assumptions is incorrect (Section 6.4.3).

A more searching way to validate the model is by using a suite of *diagnostics*, each designed to focus on one specific component (“assumption”) of the model. Diagnostics are well-developed for linear regression and for generalized linear models [16, 89, 94, 178]. Diagnostics are mostly built from the *residuals*

$$(\text{residual}) = (\text{observed}) - (\text{fitted})$$

which contain essentially all information about the discrepancies between the model and the data. The residuals can be difficult to interpret on their own, but we can extract diagnostic information by combining and plotting the residuals in particular ways.

A great advantage of model diagnostics is that they help us to *improve* the model. Each specialised diagnostic assesses whether a particular assumption of the model appears to be correct, and if not, suggests how it may be corrected. This supports a cyclic process of statistical modelling in which we formulate tentative models, fit them to data, criticise the fit and update the model [102, 101].

Diagnostics for spatial point process models are an active area of research [34, 28, 21, 22, 30].

Section 11.2 discusses goodness-of-fit tests for a fitted Poisson point process model. Section 11.3 discusses the use of relative intensities and the function `rhohat()` for diagnostic purposes. The key concept of *residuals* for point processes is introduced in Section 11.4 along with several diagnostic plots based on residuals. As is the case in other branches of statistics, it is sometimes

difficult to interpret residuals directly, and it is better to use a carefully-designed plot of the residuals. Sections 11.5–11.7 present several kinds of diagnostic plots, based on the residuals, which have their origins in diagnostics for linear regression. Section 11.5 presents *partial residual plots* for spatial point process models, a diagnostic for the form of a covariate effect. Section 11.6 presents *added variable plots*, a diagnostic for the existence of a covariate effect. Section 11.7 presents *leverage* and *influence* for point process models. All the above techniques are diagnostics for the *intensity* of the process: Section 11.8 discusses how to validate the assumption of independence between points.

11.1 Overview of techniques

A palatable overview of the available techniques and when to use them

11.2 Goodness-of-fit tests of a fitted model

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 (Sections 6.4.2 and 6.7.1) can also be applied to a fitted inhomogeneous Poisson model. The null hypothesis is the inhomogeneous Poisson process of the form specified (with unknown values of the parameters). Under the null hypothesis, the quadrat counts n_j in quadrats B_j are realisations of independent Poisson variables with different mean values μ_j . The means μ_j have to be *estimated* using the intensity of the fitted model:

$$\hat{\mu}_j = \int_{B_j} \hat{\lambda}(u) du$$

where $\hat{\lambda}(u) = \lambda_{\hat{\theta}}(u)$ is the fitted intensity obtained by substituting the parameter estimate $\hat{\theta}$ into the formula for the intensity in the model. The test statistic is

$$X^2 = \sum_j \frac{(\text{observed} - \text{expected})^2}{\text{expected}} = \sum_j \frac{(n_j - \hat{\mu}_j)^2}{\hat{\mu}_j}. \quad (11.1)$$

To perform the test, the observed value of X^2 is referred to the χ^2 distribution with $m - p$ degrees of freedom, where m is the number of quadrats and p is the number of model parameters that were estimated. Subtracting p can be regarded as an adjustment for the fact that the parameters were estimated.

This calculation is performed by the method `quadrat.test.ppm()`.

```
> fit2e <- ppm(besi ~ polynom(elev,2), data=besi.extra)
> M <- quadrat.test(fit2e, nx=4, ny=2)
> M
Chi-squared test of fitted Poisson model 'fit2e' using quadrat
counts
Pearson X2 statistic

data: data from fit2e
X2 = 474.1716, df = 5, p-value < 2.2e-16
```

```
alternative hypothesis: two.sided
```

```
Quadrats: 4 by 2 grid of tiles
```

Notice that the degrees of freedom for χ^2 are $m - p = 4 \times 2 - 3 = 5$.

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 — that is, in what way the data appear to depart from the predictions of the model. To do this we can inspect the residual counts.

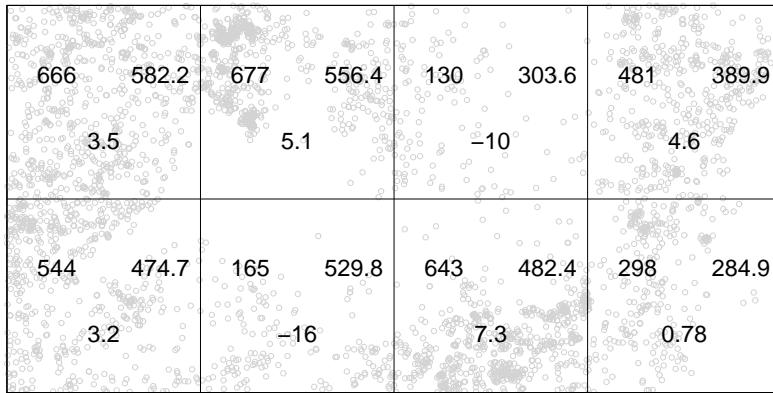


Figure 11.1: Plot of quadrat counting test of CSR for *Beilschmiedia* data

Figure 11.1 shows the result of `plot(M)`. 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

$$\text{Pearson residual} = \frac{\text{(observed)} - \text{(expected)}}{\sqrt{\text{expected}}} = \frac{n_j - \hat{\mu}_j}{\sqrt{n_j}}. \quad (11.2)$$

If the original data were Poisson, this transformation approximately standardises the residuals, so that they have mean zero and variance 1 (approximately) when the model is true. A Pearson residual of -16 or 7 is a gross departure from the fitted model.

Cite [96].

Goodness-of-fit tests have weak power (low sensitivity) for detecting some kinds of departure from the fitted model, so a non-significant result is not always reassuring. On the other hand, if there is found to be significant evidence of a lack-of-fit, then we can only conclude that at least one of the model assumptions is false, without knowing which assumption it is.

When a spatial covariate Z is available, we could divide the range of Z values into several bands, divide the data correspondingly, and apply a χ^2 test to this grouping of the data. This was done in Section 6.6.2 when the null hypothesis is CSR. Similar code works when the null hypothesis is a fitted inhomogeneous Poisson model, using `quadrat.test.ppm()`:

```
> elev <- bei.extra$elev
> grad <- bei.extra$grad
> b <- quantile(elev, probs=(0:4)/4)
> Zcut <- cut(elev, breaks=b, labels=1:4)
> V <- tess(image=Zcut)
> quadrat.test(fit2e, tess=V)
```

```

Chi-squared test of fitted Poisson model 'fit2e' using quadrat
counts
Pearson X2 statistic

data: data from fit2e
X2 = 125.7821, df = 1, p-value < 2.2e-16
alternative hypothesis: two.sided

Quadrats: 4 tiles (levels of a pixel image)

```

Notice that there is only one degree of freedom remaining.

For a continuous spatial covariate it is probably better to avoid discretisation, and to use one of the tests based on the spatial distribution of the covariate. The Kolmogorov-Smirnov test (Section 6.7.2 on page 146) can be applied to a fitted Poisson model, with homogeneous or inhomogeneous intensity.

```

> cdf.test(fit2e, grad, test="ks")
Spatial Kolmogorov-Smirnov test of inhomogeneous Poisson
process in two dimensions

data: covariate 'grad' evaluated at points of 'bei'
      and transformed to uniform distribution under 'fit2e'
D = 0.1868, p-value < 2.2e-16
alternative hypothesis: two-sided

```

This uses the method `cdf.test.ppm()` for the generic function `cdf.test()`. Here we used the slope covariate `grad` to check goodness-of-fit. A significant lack of fit in this test may prompt us to try including the slope covariate in the model, but does not guarantee that this will remedy the lack of fit.

If additional covariates had not been available, we could have used the *x*-coordinate, by typing `cdf.test(fit2e, "x")`, or some user-defined function of *x* and *y* if it were appropriate.

Note that the Kolmogorov-Smirnov test is *not* adjusted for the fact that the parameters were estimated. It can be a conservative test in this context, if the sample size is small.

Similar comments apply to the Cramér-Von Mises and Anderson-Darling tests which are performed using `cdf.test()` with the argument `test="cvm"` or `test="ad"` respectively.

The Berman-Lawson-Waller test (Section 6.7.3) can also be applied to a fitted Poisson point process model:

```

> berman.test(fit2e, grad)
Berman Z1 test of inhomogeneous Poisson process in two
dimensions

data: covariate 'grad' evaluated at points of 'bei'
Z1 = 11.3883, p-value < 2.2e-16
alternative hypothesis: two-sided

```

Recall that the Berman-Lawson-Waller Z_1 test is the likelihood ratio test for presence of a loglinear covariate effect. In this context, it is a test of $H_0 : \phi = 0$ against $H_1 : \phi \neq 0$ in the model $\lambda(u) = \lambda_0(u) \exp(\phi Z(u))$ where $\lambda_0(u)$ is the intensity of the model under scrutiny, and $Z(u)$ is the additional spatial covariate whose effect is being tested. We could alternatively use `anova.ppm()` to perform the same test:

```

> fit2e1g <- update(fit2e, ~ . + grad)
> anova(fit2e, fit2e1g, test="Chi")
Analysis of Deviance Table

Model 1: ~ elev + I(elev^2)
Model 2: ~ elev + I(elev^2) + grad
Df Deviance Pr(>Chi)
1
2 1    418.18 < 2.2e-16 ***
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

However, our implementation of `berman.test()` does not adjust for the fact that the parameters were estimated, whereas `anova.ppm()` does take this into account.

11.3 Relative intensity

11.3.1 Inverse-lambda weighting

If a point process \mathbf{X} with true intensity function $\lambda(u)$ is modelled by a point process with intensity $\lambda_0(u) > 0$, the *relative intensity* $r(u) = \lambda(u)/\lambda_0(u)$ at spatial location u can be estimated by kernel smoothing. One estimator is

$$\hat{r}(u) = \frac{1}{e(u)} \sum_i \frac{1}{\lambda_0(x_i)} \kappa(u - x_i) \quad (11.3)$$

where κ is a smoothing kernel on the two-dimensional plane and $e(u)$ is the edge correction factor (6.10) defined on page 132. This is a weighted kernel intensity estimate, with weights equal to the reciprocal of the reference intensity λ_0 . By Campbell's formula, the expected value of $\hat{r}(u)$ is

$$\mathbb{E}\hat{r}(u) = \frac{1}{e(u)} \int_W \frac{1}{\lambda_0(v)} \kappa(u - v) \lambda(v) dv = \frac{1}{e(u)} \int_W r(v) \kappa(u - v) dv,$$

a smoothed version of the function $r(u)$.

The estimate $\hat{r}(u)$ can be computed in `spatstat` by `density.ppp()` using the argument `weights`. The weights $w_i = 1/\lambda_0(x_i)$ are computed from the fitted model using `fitted.ppm()`. For the *Beilschmiedia* data:

```

> lam0 <- fitted(fit2e, dataonly=TRUE)
> rel2e <- density(besi, weights=1/lam0)

```

The estimate is plotted in Figure 11.2. Values of $\hat{r}(u)$ have quite a wide range:

```

> range(rel2e)
[1] 0.1047864 6.1245612

```

which suggests the model `fit2e` is a poor fit.

11.3.2 Relative intensity as function of covariate

There are several goodness-of-fit tests (such as the Kolmogorov-Smirnov test and Berman's tests) which compare the observed distribution of the values of a spatial covariate Z to the expected distribution under a model. Another way to perform this comparison is to assume that the true intensity

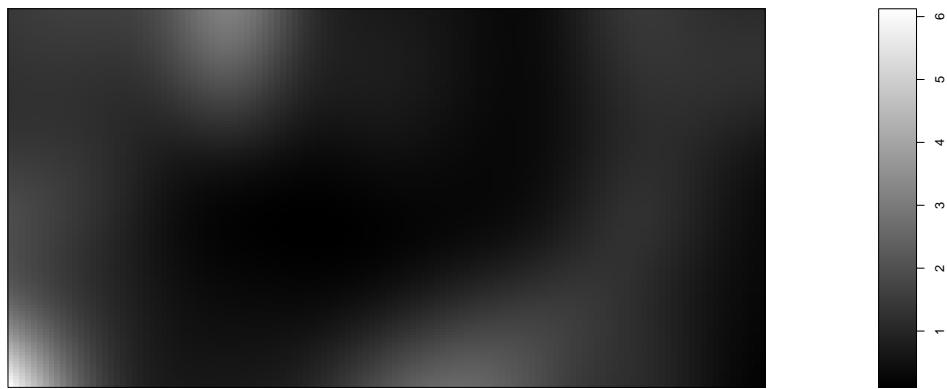


Figure 11.2: Estimate of relative intensity r of *Beilschmiedia* data against a fitted model in which the intensity is a log-quadratic function of terrain elevation.

of the point process is

$$\lambda(u) = \rho(Z(u))\lambda_0(u) \quad (11.4)$$

where $\lambda_0(u)$ is the intensity under the fitted model, and ρ is an unknown function; then we may estimate the function ρ using “relative distribution” methods (Section 6.6.3).

Equation (11.4) is a special case of equation (6.20) on page 143, where the “baseline” function $B(u)$ is the intensity of the fitted model $\lambda_0(u)$. The function ρ can be estimated using the methods described there.

A direct way to perform this calculation is to use `predict.ppm()` to compute the fitted intensity function $\lambda_0(u)$, then apply `rhohat()` to the point pattern using the argument `baseline`:

```
> lambda0 <- predict(fit2e)
> rh1 <- rhohat(bi, grad, baseline=lambda0)
```

This uses the method `rhohat.ppp()`. There is an alternative method `rhohat.ppm()` for fitted models, which performs the same calculation:

```
> rh2 <- rhohat(fit2e, grad)
```

The results `rh1` and `rh2` are identical up to numerical error, and are shown in Figure 11.3. The value $\rho = 1$ corresponds to agreement with the fitted model depending only on terrain slope, so Figure 11.3 suggests that this model overestimates the intensity at low elevations, and underestimates it at high elevations.

11.4 Residuals for Poisson processes

Residuals from the fitted model are a fundamental building block for diagnostics in applied statistics. They are ubiquitous because they are closely connected to the likelihood. Residuals can be examined for diagnostic purposes in their own right, although they can be difficult to interpret. More often, the residuals are used to construct specialised diagnostic plots [16, 178].

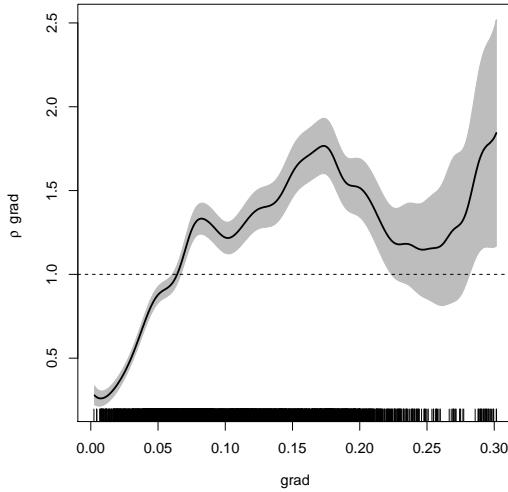


Figure 11.3: Estimated relative density for *Beilschmiedia* trees against terrain slope, relative to the fitted model in which forest density is a log-quadratic function of terrain elevation. Dashed line at $\rho \equiv 1$ corresponds to agreement with fitted model.

11.4.1 Residual measure

In spatial statistics, residuals for point processes have only recently been developed [257, 310], [291, pp. 49–50], [34]. (Probably well-known for Poisson case!?!)

Maybe start explanation with case of CSR

For a fitted Poisson process model, with fitted intensity $\hat{\lambda}(u)$, the predicted number of points falling in any region B is $\int_B \hat{\lambda}(u) du$. The **point process residual** [34] for the region B is defined to be the *observed minus expected* number of points falling in B :

$$\mathcal{R}(B) = n(\mathbf{x}_B) - \int_B \hat{\lambda}(u) du \quad (11.5)$$

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

This residual is 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 $n(\mathbf{x}_B)$. The ‘expected’ quadrat count is $\hat{\lambda} \text{ area}(B)$ if the model is CSR, or more generally $\int_B \hat{\lambda}(u) du$ if the model is an inhomogeneous Poisson process. Hence the ‘raw residual’ for quadrat B is (11.5).

However, unlike the residuals associated with quadrats, the residuals (11.5) are defined for **any** region B . The equation (11.5) is the definition of a *measure* \mathcal{R} . As explained on page 126, a measure is a function M which assigns a value $M(B)$ to *any* region B , intuitively representing the amount of “stuff” contained in B .

There has been some confusion over this in the literature, so we want to emphasise that the definition (11.5) does not force anyone to choose a particular set B or choose a particular size of set B . It is incorrect to describe $\mathcal{R}(B)$ as a “pixel residual” requiring a choice of pixel size. On the contrary, (11.5) is meant to be applied to *all* possible regions B , large and small. This is the mathematician’s roundabout way of defining a spatial distribution of mass: when we know the amount of ‘stuff’ contained in each region B , we effectively know the exact spatial distribution of the ‘stuff’.

A good physical analogy is to think of the residual measure as a spatial distribution of “electric charge”. Each of the data points carries a positive electric charge of 1 unit. The background space carries a diffuse negative charge, with a spatially-varying density of $\hat{\lambda}(u)$ units of negative charge per unit area. The total electric charge in a region B is the sum of the positive and negative charges as shown in (11.5). If the fitted model is true, then the positive and negative charges should approximately cancel and the total charge should be approximately zero.

The residual measure of a fitted model is computed by `residuals.ppm()`:

```
> res2e <- residuals(fit2e)
> res2e
Scalar-valued measure
Approximated by 20508 quadrature points
window: rectangle = [0, 1000] x [0, 500] metres
3604 atoms
Total mass:
discrete = 3604    continuous = -3604    total = 3.713e-09
```

Note that *the total residual $\mathcal{R}(W)$ is zero*, up to numerical error. The same holds for any loglinear model that contains an intercept term, because the score equation for the intercept is equivalent to saying that the total raw residual must be zero. Similarly, the residuals in linear regression always sum to zero.

To visualise the electric charge in a direct way, we plot the positive charges as points, and the negative charge density as a colour map and contour plot, as shown in Figure 11.4, produced by the plot method `plot.msr()` for measures. The black dots represent positive unit charges. The background greyscale image represents a density of negative charge: lighter shades of grey represent higher density of negative charge.



Figure 11.4: Point process residuals for model in which *Beilschmiedia* intensity is a log-quadratic function of terrain elevation. Generated by `plot(res2e)`.

11.4.2 Smoothed residual field

The usual way to interpret Figure 11.4 is that, if the model fits well, the darker and lighter colours should balance out. The darker colours are either positive charges or small negative charge densities, while the lighter colours are large negative charge densities. If we were to smear the ink in Figure 11.4, the result should be a neutral grey if the model fits well.

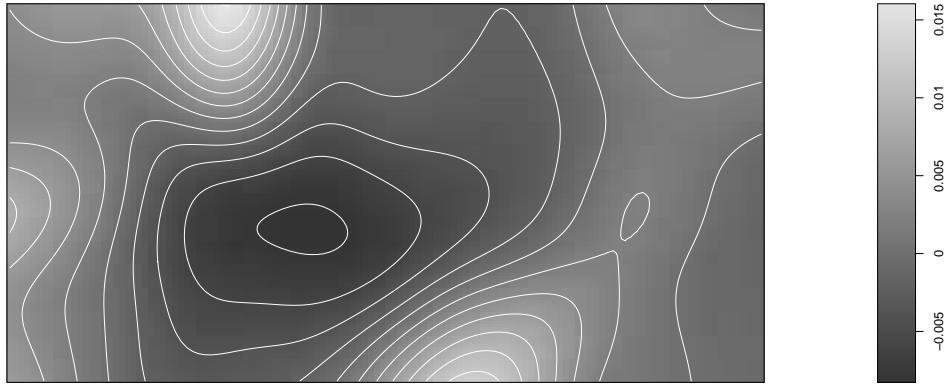


Figure 11.5: Smoothed residual field for model in which *Beilschmiedia* intensity is a log-quadratic function of terrain elevation. Generated by `plot(Smooth(res2e))`.

This suggests that we *smooth* the residuals: a smoothed version of Figure 11.4 is shown in Figure 11.5. If we apply a two-dimensional smoothing kernel κ to the positive and negative charges, we obtain the ‘smoothed residual field’

$$s(u) = \frac{1}{e(u)} \left[\sum_{i=1}^{n(x)} \kappa(u - x_i) - \int_W \kappa(u - v) \lambda_{\hat{\theta}}(v) dv \right] \quad (11.6)$$

where $e(u)$ is the edge correction defined in equation (6.10) on page 132. The smoothed residual field is a spatially-weighted average of the residuals. We can recognise (11.6) as the difference of two terms

$$s(u) = \tilde{\lambda}(u) - \lambda^{\dagger}(u) \quad (11.7)$$

where $\tilde{\lambda}(u)$ is a *nonparametric* estimate of intensity, namely the uniformly-corrected kernel estimate defined in equation (6.8) on page 131, and

$$\lambda^{\dagger}(u) = \frac{1}{e(u)} \int_W \kappa(u - v) \lambda_{\hat{\theta}}(v) dv.$$

is a correspondingly-smoothed version of the *parametric* estimate of the intensity according to the fitted model. The difference (11.7) should have mean value close to zero if the model is true. Positive values of $s(u)$ suggest that there is an overabundance of data points relative to the prediction of the model, that is, they suggest that the model underestimates the true intensity. Negative values of $s(u)$ suggest that the model overestimates the true intensity.

Figure 11.5 shows the smoothed residual field for the *Beilschmiedia* example, calculated by `Smooth(res2e)`. There are several regions of over- and under-estimation of the intensity.

11.4.3 Scaled or weighted residuals

Interpreting the magnitudes of the residuals can be complicated. In regression analysis, it is common to rescale the residuals so that they satisfy some desired statistical property.

For a Poisson point process we can define the *weighted* residual measure with any desired ‘weight’ function w ,

$$\mathcal{R}_w(B) = \sum_{x_i \in \mathbf{X}_B} w(x_i) - \int_B w(u) \hat{\lambda}(u) du \quad (11.8)$$

for each spatial region B . Choosing the constant weight $w \equiv 1$ gives the raw residual measure \mathcal{R} defined above. Other common choices are the *Pearson residual* measure with $w(u) = \hat{\lambda}(u)^{-1/2}$,

$$\mathcal{R}^{(P)}(B) = \sum_{x_i \in \mathbf{X}_B} \hat{\lambda}(x_i)^{-1/2} - \int_B \hat{\lambda}(u)^{1/2} du \quad (11.9)$$

and the *inverse-lambda residual* measure with $w(u) = 1/\hat{\lambda}(u)$,

$$\mathcal{R}^{(I)}(B) = \sum_{x_i \in \mathbf{X}_B} \frac{1}{\hat{\lambda}(x_i)} - |B|. \quad (11.10)$$

If the fitted intensity $\hat{\lambda}(u)$ can take zero values, then the sums above must be restricted to data points x_i where $\hat{\lambda}(x_i) > 0$, and the region B in (11.9)–(11.10) must be replaced by the subregion of B where the fitted intensity $\hat{\lambda}(u)$ is nonzero.

Note that there is a big difference between the Pearson residual measure defined in (11.9) above, and the Pearson residuals used in quadrat counting, defined in equation (11.2) on page 369. The Pearson residuals for quadrat counting are defined only for the quadrats used in the counting; these residuals are scaled to have mean 0 and variance 1, approximately. The Pearson residual measure $\mathcal{R}^{(P)}(B)$ is defined for *any* region B , and is scaled to have mean 0 and variance $|B|$ if the model is true. The Pearson residual measure for a region with *unit area* has variance 1.

Something about measures being summable

In the fine-pixel limit of logistic regression, the raw residuals of logistic regression converge to the raw residual measure. The Pearson residuals of logistic regression, rescaled by dividing by the square root of pixel area, converge to the Pearson residual measure. These facts can be used to compute the residual measures approximately from pixel data.

A diagnostic similar to the inverse-lambda residuals was first proposed in [310]. The residual measures were developed in [34, 28]. The inverse-lambda residuals have the computational advantage that they only require calculation at the data points. The Pearson residuals have the desirable statistical property that the variance of $\mathcal{R}^{(P)}(B)$ is approximately $|B|$ when the model is true.

The smoothed inverse-lambda residual field is identical to $\hat{r}(u) - 1$ where $\hat{r}(u)$ is the smoothing estimate (11.3) of the relative intensity $r(u)$ defined in Section 11.3.1.

For a loglinear Poisson point process model (9.6) we can also define the vector-valued *score residual* measure with $w(u) = \mathbf{Z}(u)$,

$$\mathcal{R}^{(s)}(B) = \sum_{x_i \in B} \mathbf{Z}(x_i) - \int_B \mathbf{Z}(u) \hat{\lambda}(u) du. \quad (11.11)$$

The score residuals are intimately related to the maximum likelihood estimation procedure, because (assuming regularity conditions) the MLE is the parameter value for which the total score residual $\mathcal{R}^{(s)}(\mathbf{W})$ is zero.

In `spatstat` the weighted residual measures are computed using `residuals.ppm()` with the argument `type` set to "pearson", "inverse" or "score". Figure 11.6 shows the Pearson residual measure for the example, computed by

```
> pres2e <- residuals(fit2e, type="pearson")
```

In the left panel, the sizes of the "+" signs are proportional to the positive charges $\hat{\lambda}(x_i)^{-1/2}$ associated with the data points x_i , and again the greyscale image represents the negative charge density $\hat{\lambda}(u)^{1/2}$.

When reading the smoothed Pearson residual field, we must remember that the Pearson residuals are scaled so that the total Pearson residual in a region of *unit area* has unit variance. In Figure 11.6 the numerical scale suggests that the deviations are small, with smoothed Pearson residuals of the

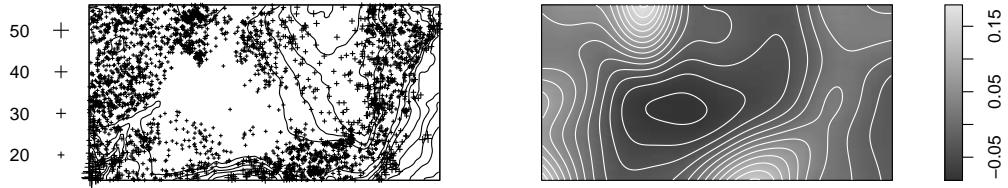


Figure 11.6: Pearson residuals for model in which *Beilschmiedia* intensity is a log-quadratic function of terrain elevation. *Left*: Pearson residual measure `residuals(fit2e, type="pearson")`; *Right*: smoothed Pearson residual field `Smooth(residuals(fit2e, type="pearson"))`.

order of ± 0.15 . However, the *Beilschmiedia* coordinates are given in *metres*; the survey area is 500,000 square metres. Let us rescale to kilometres:

```
> beikm <- rescale(bi, s=1000, unitname="km")
> beikm.extra <- lapply(bi.extra, rescale,
+                         s=1000, unitname="km")
> fit2eKM <- ppm(beikm ~ polynom(elev, 2), data=beikm.extra)
> pres2eKM <- residuals(fit2eKM, type="Pearson")
```

The result (plotted in Figure 11.7) now gives smoothed Pearson residuals of the order of ± 150 (as expected, since the intensity values were rescaled by 1000^2 so the square root of intensity is rescaled by 1000). These are substantial deviations (because the study area is 0.5km^2), and indicate that the model is a poor fit.

11.4.4 Lurking variable plot

If there is a spatial covariate $Z(u)$ that could play an important role in the analysis, it may be useful to display a *lurking variable plot* [34] of the residuals against Z . For each possible covariate value z plotted on the horizontal axis, the lurking variable plot shows on the vertical axis the total residual for the region of space where Z is less than or equal to z . That is, the lurking variable plot is a line plot of $C(z) = \mathcal{R}(B(z))$ against z , where

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

is the sub-level set, the region of space where the covariate value is less than or equal to z .

The left panel of Figure 11.8 shows the lurking variable plot for the terrain slope variable, generated by the command

```
> lurking(fit2e, grad, type="raw")
```

For each value of terrain slope s on the horizontal axis, the solid line on the graph shows the residual for the parts of the survey with slopes no steeper than s , that is, the cumulative number of *Beilschmiedia* trees growing on slopes less than or equal to s , minus the expected number according to the model.

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.

The plot also shows approximate 5% significance bands for the cumulative residual $C(z)$, obtained from the asymptotic variance under the model [28].

It can be helpful to display the derivative $C'(z)$, which often indicates which values of z are associated with a lack of fit. This is shown in the right panel of Figure 11.8 and is generated by

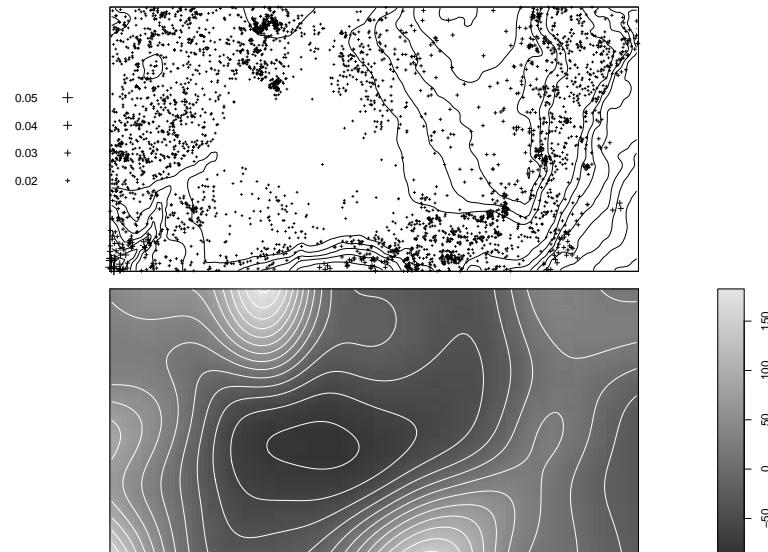


Figure 11.7: Counterpart of previous Figure after converting the spatial coordinates to kilometres.

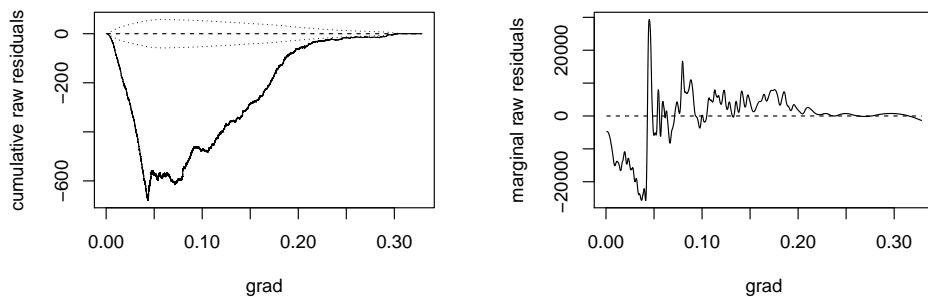


Figure 11.8: Lurking variable plot for terrain slope, for model depending only on terrain elevation. *Left:* cumulative residual. *Right:* derivative.

```
> lurking(fit2e, grad, type="raw", cumulative=FALSE)
```

The derivative is estimated using a smoothing spline and you may need to tweak the smoothing parameters (argument `splineargs`) to get a useful plot. The package currently does not plot significance bands for $C'(z)$.

Both panels of Figure 11.8 suggest that the model substantially overestimates the intensity on flat terrain (slope less than 0.05) and underestimates intensity on steeper slopes. To assess

whether this discrepancy is important, we inspect the distribution of slope values (for example using `hist(grad)` or `plot(ecdf(grad))` or `ecdf(grad)(0.05)`) to find that slopes less than 0.05 constitute about 40% of the survey region. This indicates either that terrain slope is an important variable in determining intensity, or at the very least, that another unknown variable spatially correlated with terrain slope is important in determining intensity; and that this effect has not been captured by the current model depending only on terrain elevation.

11.4.5 Four-panel residual plot

Especially when there are no spatial covariates available, it is often convenient to use the command `diagnose.ppm()` to generate four different plots of the residuals, as shown in Figure 11.9.

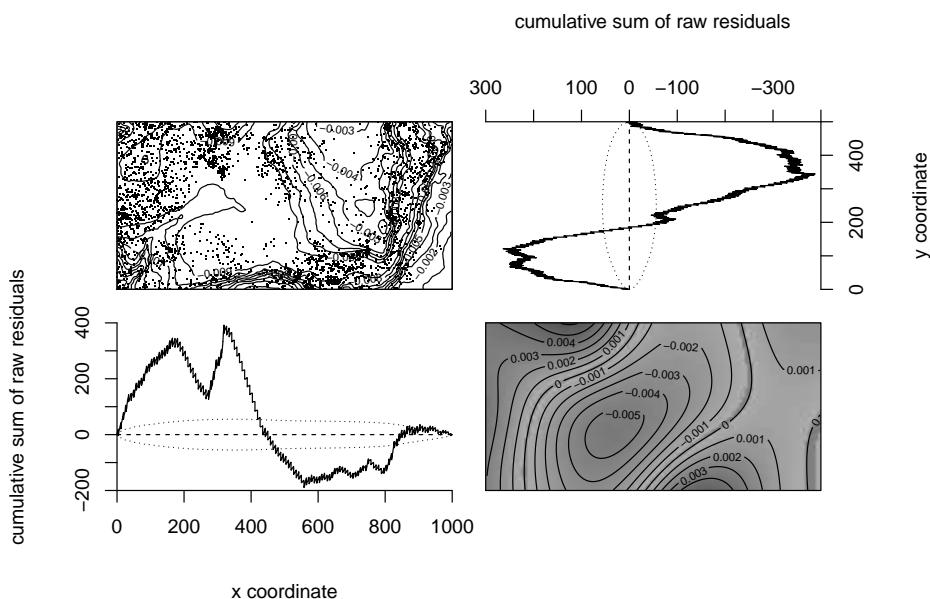


Figure 11.9: Four-panel diagnostic plot generated by `diagnose.ppm()`.

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 or contour map representing the fitted intensity (negative residuals). However, it is often difficult to interpret.

The two other panels are lurking variable plots against one of the cartesian coordinates. 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.

As is the case in other branches of statistics, it is sometimes difficult to interpret residuals directly, and it is better to use a carefully-designed plot of the residuals. These are discussed below.

11.5 Partial residual plots

[Improve motivation here](#)

A loglinear Poisson point process model is closely related to logistic regression. Standard diagnostics for validating a logistic regression include the smoothed partial residual (component-plus-smoothed-residual) plot [213] and the added variable plot [15, 332]. These diagnostics were recently adapted to point process models in [21].

11.5.1 Partial residuals in regression

In regression models, a simple diagnostic for nonlinearity of covariate effects is the *partial residual plot* (component-plus-residual plot) [143, 214, 213], [89, pp. 135–137], [178, sec 10.4, p. 230 ff.]. Suppose we fit a linear relationship $\mathbb{E}y = \beta^\top \mathbf{x} + \gamma^\top \mathbf{z}$ between responses y_i and vector-valued covariates \mathbf{x}_i and \mathbf{z}_i for $i = 1, \dots, n$, but we suspect the true relationship is $\mathbb{E}y = h(\mathbf{x}) + \gamma^\top \mathbf{z}$ where $h(\mathbf{x})$ is a nonlinear function of \mathbf{x} . The partial residual [143] for \mathbf{x} is the fitted effect for \mathbf{x} plus the residual:

$$s_i = \hat{\beta}^\top \mathbf{x}_i + r_i$$

where $r_i = y_i - \hat{y}_i = y_i - (\hat{\beta}^\top \mathbf{x}_i + \hat{\gamma}^\top \mathbf{z}_i)$ is the residual. If $h(\mathbf{x})$ is close¹ to a linear function, the delta method gives $\mathbb{E}s_i \approx h(\mathbf{x}_i)$, so a scatterplot of s_i against \mathbf{x}_i suggests the true shape of the function h .

Similarly, in a generalised linear model (such as logistic regression), suppose we fit a relationship $\mathbb{E}(y) = g(\beta^\top \mathbf{x} + \gamma^\top \mathbf{z})$ where g is the inverse link function, but we suspect the true relationship is $\mathbb{E}(y) = g(h(\mathbf{x}) + \gamma^\top \mathbf{z})$. The partial residual is defined on the scale of the linear predictor $g^{-1}(\mathbb{E}(y))$ as

$$s_i = \hat{\beta}^\top \mathbf{x}_i + r_i^W \quad (11.12)$$

where r_i^W is the *working residual*

$$r_i^W = \frac{y_i - \hat{y}_i}{g'(\hat{y}_i)}. \quad (11.13)$$

The derivative g' is related to the *variance* of the response.

If the function h is close to a linear function, then $\mathbb{E}s_i \approx h(\mathbf{x}_i)$. A scatterplot smoother of s_i against \mathbf{x}_i should suggest the correct shape of $h(\mathbf{x})$. See [213, 178].

11.5.2 Partial residual for Poisson point process

Partial residual plots for spatial point processes were developed in [21]. Suppose we have fitted a loglinear Poisson point process model with intensity

$$\lambda(u) = \exp(\beta_0 + \beta_1 Z(u) + \gamma \mathbf{V}(u)) \quad (11.14)$$

where $Z(u)$ is a real-valued spatial covariate function and $\mathbf{V}(u)$ is a vector-valued covariate (possibly missing); however we suspect that the true relationship may be

$$\lambda(u) = \exp(h(Z(u)) + \gamma \mathbf{V}(u)) \quad (11.15)$$

where h is an unknown function.

If we approximate the model by a logistic regression, then in the fine-pixel limit, the smoothed

¹To be precise, for some value of β , the discrepancy $\varepsilon(\mathbf{x}) = h(\mathbf{x}) - \beta \mathbf{x}$ should be small enough to justify a Taylor expansion of $\mathbb{E}s_i$ in terms of $\varepsilon(\mathbf{x}_i)$.

partial residual curve (using Nadaraya-Watson [253, 336] smoothing, Section 6.9) converges to the *smoothed partial residual* for the point process,

$$\hat{h}(z) = \hat{\beta}_1 z + \frac{\sum_i \frac{1}{\lambda(x_i)} \kappa(Z(x_i) - z)}{\int_W \kappa(Z(u) - z) du} - 1 \quad (11.16)$$

for all real z where the denominator is positive. Here $(\hat{\beta}_0, \hat{\beta}_1, \hat{\gamma})$ are the MLE's of the parameters of (11.14), $\hat{\lambda}(u) = \exp(\hat{\beta}_0 + \hat{\beta}_1 Z(u) + \hat{\gamma} V(u))$ is the fitted intensity, and κ is a smoothing kernel on the real line. The numerator is a sum over all points of the point pattern dataset \mathbf{x} .

A plot of $\hat{h}(z)$ against z is a diagnostic for nonlinearity of the covariate effect of Z . Kernel smoothing is not obligatory; other alternatives could be applied. Standard error calculations for $\hat{h}(z)$ are described in [21].

To understand why (11.16) is a suitable diagnostic, notice that the terms in the sum have weights $1/\hat{\lambda}(x_i)$. In brief, this is because the working residuals (11.13) are inversely scaled by the variance of the response, and a Poisson random variable has variance equal to its mean. The ratio term in (11.16) can be recognised as an inverse-lambda weighted estimate (Section 11.3) of the relative intensity, that is, an estimate of the ratio of the true intensity to the fitted model intensity. If the model fits well, this ratio should be approximately 1, so that (11.16) should be approximately equal to the fitted effect $\hat{\beta}_1 z$.

The partial residual is also intimately connected to the inverse-lambda point process residuals of Section 11.4. We may rewrite (11.16) as

$$\hat{h}(z) = \hat{\beta}_1 z + \frac{\int_W \kappa(Z(u) - z) d\mathcal{R}^{(I)}(u)}{\int_W \kappa(Z(u) - z) du} \quad (11.17)$$

where $\mathcal{R}^{(I)}$ is the inverse lambda residual measure (11.10). Note that the residuals are smoothed before the covariate effect is added.

11.5.3 Implementation in spatstat

The `spatstat` command `parres()` computes a partial residual plot. Its syntax is `parres(model, covariate, ...)` where `model` is a fitted Poisson point process model and `covariate` is a character string giving the name of a covariate used in the model.

The result of `parres()` is an object that belongs to the classes "parres" and "fv". Plot this object to generate the partial residual plot.

In straightforward use, `covariate` is the name of one of the *canonical* covariates in the model. These are the functions Z_j that appear in the loglinear intensity (9.6). Type `names(coef(model))` to see the names of the canonical covariates in `model`. If the selected canonical covariate is Z_j , then the diagnostic plot concerns the model term $\beta_j Z_j(u)$. The plot shows a smooth estimate of a function $h(z)$ that should replace this linear term, that is, $\beta_j Z_j(u)$ should be replaced by $h(Z_j(u))$. The linear function is also plotted as a dotted line.

Alternatively `covariate` may be one of the *original* covariates that were supplied as data when fitting the model, and from which the canonical covariates are derived. There may be several canonical covariates which depend on the specified `covariate`. Then the covariate effect is computed using all these canonical covariates. For example in a log-quadratic model which includes the terms `x` and `I(x^2)`, the effect involving both these terms will be computed.

By default, the bandwidth for the smoothing kernel κ is selected automatically (by the one-dimensional bandwidth selection rule `bw.nrd0()` from the `stats` package). This can be overridden by specifying the argument `bw`.

To scrutinize outliers, the analyst may *restrict* the domain of computation to a subregion $V \subset W$, by restricting the sum in (11.16) to the points of $\mathbf{y} \cap V$ and restricting the integrals in (11.16)–(11.17) to the domain V . The argument `subregion` allows the user to specify the subregion V .

11.5.4 Beilschmiedia data example

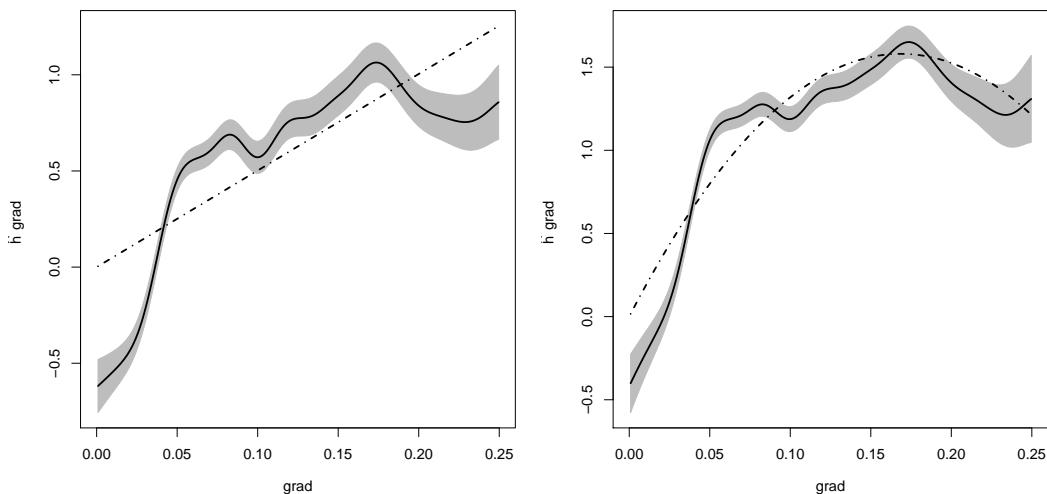


Figure 11.10: Partial residual plots for the terrain slope covariate, for models where the intensity is a loglinear function (*Left*) or log-quadratic function (*Right*) of terrain slope. Shading shows pointwise 95% confidence intervals based on asymptotic standard errors.

Suppose we fit a model to the *Beilschmiedia* data in which the intensity is a loglinear function of terrain slope:

```
> fit1g <- ppm(bei ~ grad)
> coef(fit1g)
(Intercept)      grad
-5.390553     5.022021
```

The partial residual plot for the slope covariate in this model, computed by `parres(fit1g, "grad")`, is shown in the left panel of Figure 11.10. The shape strongly suggests that the loglinear model is inadequate, and suggests that a quadratic function of slope would be more appropriate. We fit the log-quadratic model:

```
> fit2g <- update(fit1g, ~ polynom(grad, 2))
> coef(fit2g)
(Intercept)      grad    I(grad^2)
-5.987136    18.744888   -55.602328
```

The right panel of Figure 11.10 shows the corresponding partial residual plot for the log-quadratic model, `parres(fit2g, "grad")`.

In Figure 11.10, the plots have been restricted to slope values less than 0.25 because the plots beyond this limit show much greater deviations and inflated standard errors. This is attributable to the paucity of data with slopes greater than 0.25 (as shown by `hist(grad)` or `plot(ecdf(grad))` or `quantile(grad, 0.99)` or `plot(eval.im(grad >= 0.25))`). This is a common artefact.

The quadratic term is also significant according to the likelihood ratio test:

```
> anova(fit2g, test="Chi")
```

```

Analysis of Deviance Table
Terms added sequentially (first to last)

Df Deviance Pr(>Chi)
NULL
grad      1    382.25 < 2.2e-16 ***
I(grad^2) 1    197.97 < 2.2e-16 ***
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

11.6 Added variable plots

Added Variable Plots [98, sec. 4.5]; [332] are commonly used in linear models and generalized linear models, to decide whether a model with response y and predictors \mathbf{x} would be improved by including another predictor z . Essentially the same technique can be used for point processes [21].

11.6.1 Added variable plot for regression

In a linear model with response y , predictors \mathbf{x} and i.i.d. Normal errors, the *added variable plot* [98, sec. 4.5] for a new covariate z is a scatterplot of the raw residuals $r = y - \hat{y}$ from the original model against the residuals $t = z - \hat{z}$ from a regression of z on \mathbf{x} . If this plot suggests a linear relationship between r and t with nonzero slope, then the new covariate z is needed. The slope of the added variable plot is equal to the coefficient of z in a regression of y on (\mathbf{x}, z) and is related to the score test [15, 92].

Added variable plots for generalised linear models [332, 333] are similar, based on the approximating linear model [94], [147, p. 412 ff.], [89, pp. 133–134]. The Added Variable Plot for a new covariate z is a plot of the smoothed *Pearson* residuals from the original model against the scaled residuals from a *weighted* linear regression of z on x . If this plot has nonzero slope, then the new covariate z is needed [94, 178].

11.6.2 Added variable plot for point processes

Consider a Poisson point process \mathbf{X} with intensity of the form

$$\lambda(u) = \exp(A(u) + \boldsymbol{\beta}^\top \mathbf{Z}(u)) \quad (11.18)$$

where $A(u)$ is a known baseline, \mathbf{Z} is a *vector-valued* covariate function and $\boldsymbol{\beta}$ is a vector parameter. Let $Y(u)$ be an additional, *real-valued* covariate function.

First consider weighted linear regression of $Y(u)$ on $\mathbf{Z}(u)$ with weights proportional to $\lambda(u)$. The Pearson residuals for this linear regression are

$$T(u) = \hat{\lambda}(u)^{1/2} \left(Y(u) - \mathbf{Z}(u)^\top \mathbf{I}(\boldsymbol{\beta})^{-1} \int_W \mathbf{Z}(v) Y(v) \hat{\lambda}(v)^{1/2} dv \right) \quad (11.19)$$

where $\hat{\lambda}(u) = \lambda_{\hat{\beta}}(u)$ is the fitted intensity and $\mathbf{I}(\boldsymbol{\beta})$ is the Fisher information matrix (9.28).

The Pearson residual measure of the point process, equation (11.9) on page 376, contains discrete atoms and a continuous component; it should be smoothed for visualisation. Accordingly our proposal for the added variable plot is to smooth the Pearson residual measure as a function of the linear regression residual T .

Definition 11.1. The added variable plot, for adding the real-valued covariate $Y(u)$ to the model (11.18), is the plot of the smoothed Pearson point process residual

$$\tilde{r}(t) = \sum_i \hat{\lambda}(x_i)^{-1/2} \kappa(T(x_i) - t) - \int_W \hat{\lambda}(u)^{1/2} \kappa(T(u) - t) du \quad (11.20)$$

against t , where $\hat{\lambda}(u)$ denotes the fitted intensity for the model (11.18), $T(u)$ is the linear regression residual (11.19), and κ is a smoothing kernel on \mathbb{R} .

Pointwise standard error calculations are described in [21].

In linear or generalised linear regression, the added variable plot is not expected to be accurate at estimating the nonlinear effect of Y ; it should only be used to decide whether or not to add the covariate Y to the model [16]. Similar comments apply here.

11.6.3 Implementation in spatstat

The `spatstat` function `addvar()` computes an added variable plot for a fitted point process model. Its syntax is `addvar(model, covariate, ...)`. The argument `model` is again a fitted point process model.

The argument `covariate` identifies the covariate that is to be considered for addition to the model. It should be either a pixel image (object of class "im") or a `function(x, y)` giving the values of the covariate at any spatial location. Alternatively `covariate` may be a character string, giving the name of a covariate that was supplied (in the `data` argument to `ppm()`) when the model was fitted, but was not used in the model.

The result of `addvar(model, covariate)` is an object belonging to the classes "addvar" and "fv". Plot this object to generate the added variable plot. The plot method shows the pointwise significance bands for a test of the *null* model, i.e. the null hypothesis that the new covariate has no effect.

11.6.4 Beilschmiedia example

Continuing the analysis of the *Beilschmiedia* data from Section 11.5.4, we consider adding the terrain elevation covariate into the current model. The added variable plot is calculated by `addvar(fit2g, elev)` and is plotted in Figure 11.11. The broadly increasing trend in the plot is a strong suggestion that the terrain elevation should be added, but is not a reliable estimate of the form of the effect for this covariate.

After adding a loglinear effect of terrain elevation, the partial residual plot for elevation (not shown) suggests adding a quadratic term in terrain elevation. The diagnostics finally support a model which is log-quadratic in both the terrain slope and the terrain elevation.

11.7 Leverage and influence

Standard procedures for fitting parametric models, including least squares and maximum likelihood, can be excessively sensitive to anomalies in the data. An important step in model validation is to check whether the fitted model was likely to have been influenced by such anomalies. Two standard tools for checking this are *leverage* and *influence*.

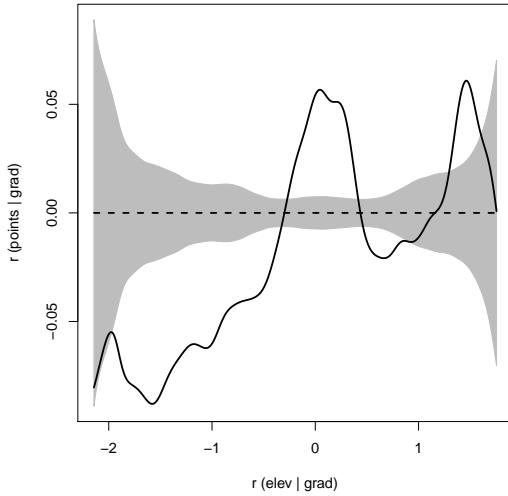


Figure 11.11: Added-variable plot for terrain elevation covariate, for the model with log-quadratic effect of terrain slope.

11.7.1 Leverage and influence in regression models

Leverage and influence are standard tools in applied statistics for validating fitted regression models [16, 93, 178]. These tools measure the sensitivity of the model-fitting procedure to changes in the input data. The *leverage* of a data point is the change in the predicted response caused by a change in the observed response, expressed in standard deviation units. The *influence* of a data point is a measure of the change in the fitted model if we delete the data point in question.

Leverage is a measure of the *potential* ability of each data point to “pull” the regression line toward itself. The leverage of a data point depends only on its covariate value (and the fitted parameters), not on the response value. The data points with highest leverage are those with the most extreme values of the covariate.

Influence is a measure of the actual impact of the data point on the final result. The data points with greatest influence are those which are both anomalous in their response value (have a large residual) and extreme in their covariate values (have a relatively high leverage).

Figure 11.12 shows an example for linear regression of a response y on a single covariate x . The observation with greatest influence is point A: the two lines are the linear regressions fitted to the data with and without point A. The observation with greatest leverage is point B, simply because it has the most extreme value of x (furthest away from the mean of x).

Suppose we have data $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)$ where \mathbf{x}_i is the covariate value, and y_i is the observed response, for the i th observation. We fit the regression relationship $\mathbb{E}y = f(\boldsymbol{\beta}^T \mathbf{x})$ where $\boldsymbol{\beta}$ is the parameter vector to be estimated. The *influence* (or likelihood influence) of the i th observation is

$$s_i = \frac{2}{p} \log \frac{L(\hat{\boldsymbol{\beta}})}{L(\hat{\boldsymbol{\beta}}_{(-i)})} \quad (11.21)$$

where L is the likelihood function (based on all the data), $\hat{\boldsymbol{\beta}}$ is the fitted parameter value, and $\hat{\boldsymbol{\beta}}_{(-i)}$ is the fitted parameter value obtained by fitting the model to all of the data except (\mathbf{x}_i, y_i) . That is, s_i is a measure of how much the model changes if we delete the i th observation.

Applied statisticians also use the *parameter influence* (charmingly nicknamed “*dfbeta*”) which

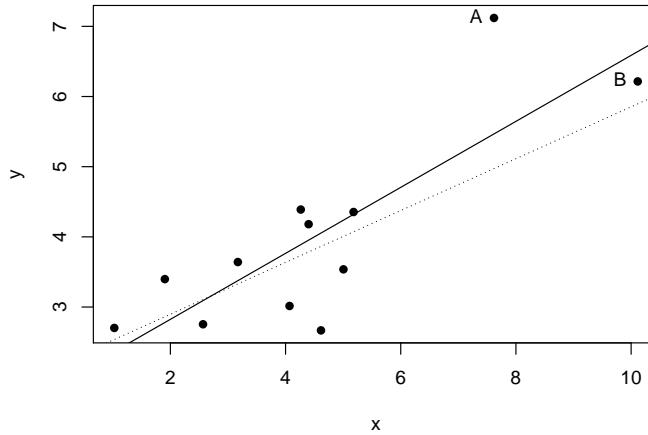


Figure 11.12: Leverage and influence in linear regression. Point B has highest leverage; point A has highest influence. Solid line: fitted linear regression using all data points. Dotted line: fitted linear regression excluding point A.

is the effect on the parameter estimate $\hat{\beta}$ of deleting the i th observation,

$$\Delta_i \hat{\beta} = \hat{\beta} - \hat{\beta}_{(-i)}. \quad (11.22)$$

In practice, these influence measures are not computed exactly, but are approximated using first-order Taylor expansions. The key to these approximations is the “leverage equation”,

$$V^{-1/2}(\hat{\mu} - \mu) \approx H^* V^{-1/2}(\mathbf{y} - \mu), \quad (11.23)$$

where $\mathbf{y} = (y_1, \dots, y_n)$ is the vector of responses, V is the variance matrix of the responses, $\mu = (\mu_1, \dots, \mu_n)$ is the vector of true mean responses $\mu_i = f(\beta^\top \mathbf{x}_i)$, and $\hat{\mu} = (\mu_1(\hat{\beta}), \dots, \mu_n(\hat{\beta}))$ is the vector of fitted mean responses $\hat{\mu}_i = f(\mathbf{x}_i^\top \hat{\beta})$.

The matrix $H^* = H^*(\hat{\beta})$ appearing in (11.23) is called the (standardised) *leverage matrix*. The scaling by $V^{-1/2}$ in (11.23) means that, in the words of McCullagh and Nelder [243, p. 397] the leverage matrix “measures the influence, in Studentized units, of changes in \mathbf{y} on $\hat{\mu}$.”

The *leverage* of observation i is defined to be the diagonal entry h_{ii}^* of H^* . That is, the leverage of observation i measures the effect that a change in the observed response y_i will have on the fitted response $\hat{\mu}_i$ for the same observation, where the measurement unit is the standard deviation of the response. The leverage value h_{ii}^* depends only on the covariate values and the fitted model parameters, and not directly on the observed response. A large leverage value can be interpreted as meaning that the observation has a strong *ability* to influence the fitted model. Since $\sum_i h_{ii}^* = \text{tr } H^* = p$, the number of model coefficients, observations i with $h_{ii}^* > p/n$ are interpreted as having relatively large leverage. See [117, 149, 341].

11.7.2 Leverage for Poisson point process model

Leverage for Poisson point processes was developed in [22]. Given an observed point pattern $\mathbf{x} = \{x_1, \dots, x_n\}$ in a bounded domain $W \subset \mathbb{R}^d$, suppose we model \mathbf{x} as a realisation of a Poisson point process \mathbf{x} with loglinear intensity function

$$\lambda_\beta(u) = \exp(A(u) + \beta^\top \mathbf{Z}(u)) \quad (11.24)$$

where $A(u)$ is a known offset function, $\mathbf{Z}(u)$ is a known, p -dimensional-vector-valued covariate function and β is again the parameter vector. If $\hat{\beta}$ is the maximum likelihood estimate of β , a first-order Taylor approximation gives $\hat{\beta} - \beta \approx \mathcal{I}_\beta^{-1} \mathbf{U}(\beta)$ where $\mathbf{U}(\beta)$ is the score (9.45) defined on page 306 and \mathcal{I}_β is the Fisher information (9.28) defined on page 296. This gives

$$\begin{aligned}\hat{\lambda}(u) - \lambda(u) &= \lambda_{\hat{\beta}}(u) - \lambda_\beta(u) \\ &= [e^{(\hat{\beta}-\beta)^\top \mathbf{Z}(u)} - 1] \lambda_\beta(u) \\ &\approx (\hat{\beta} - \beta)^\top \mathbf{Z}(u) \lambda_\beta(u)\end{aligned}\tag{11.25}$$

$$\approx \lambda_\beta(u) \mathbf{Z}(u)^\top \mathcal{I}_\beta^{-1} \left[\sum_i \mathbf{Z}(x_i) - \int_W \mathbf{Z}(v) \lambda_\beta(v) dv \right]\tag{11.26}$$

where $\lambda(u) = \lambda_\beta(u)$ denotes the true intensity and $\hat{\lambda}(u) = \lambda_{\hat{\beta}}(u)$ is the fitted intensity based on the point pattern \mathbf{x} . Dividing both sides by $\lambda(u)^{1/2}$ to “standardise”, we get the analogue of the leverage equation (11.23)

$$\lambda(u)^{-1/2} [\hat{\lambda}(u) - \lambda(u)] \approx \sum_i h(u, x_i) \lambda(x_i)^{-1/2} - \int_W h(u, v) \lambda(v)^{1/2} dv\tag{11.27}$$

where

$$h(u, v) = \lambda(u)^{1/2} \lambda(v)^{1/2} \mathbf{Z}(u) \mathcal{I}_\beta^{-1} \mathbf{Z}(v)^\top$$

is the “leverage kernel”, the analogue of the leverage matrix H^* . In practice β and $\lambda(u)$ will be replaced by their estimates $\hat{\beta}$ and $\hat{\lambda}(u) = \lambda_{\hat{\beta}}(u)$. The *leverage* is the diagonal of this estimated kernel, the function

$$\hat{h}(u) = \hat{\lambda}(u) \mathbf{Z}(u) \mathcal{I}_{\hat{\beta}}^{-1} \mathbf{Z}(u)^\top.\tag{11.28}$$

A relatively large value of $\hat{h}(\cdot)$ indicates a part of the space where the data may have a strong effect on the result.

The leverage of a spatial point process model is a function of spatial location, and is typically displayed as a colour pixel image or perspective plot. The leverage value $\hat{h}(u)$ at a spatial location u represents the (standardised) change in the fitted trend of the fitted point process model that would have occurred if a data point were to have occurred at the location u . A relatively large value of $\hat{h}(\cdot)$ indicates a part of the space where the data have a *potentially* strong effect on the fitted model (specifically, a strong effect on the intensity or trend of the fitted model) due to the values of the covariates.

Notice that the leverage function of a point process has dimensions length⁻² while the leverage matrix of linear regression is dimensionless. The trace of the leverage kernel is again

$$\int_W \hat{h}(u) du = \int_W \mathbf{Z}(u) \mathcal{I}_\beta^{-1} \mathbf{Z}(u)^\top \hat{\lambda}(u) du = p.\tag{11.29}$$

Locations u where $\hat{h}(u) > p/|W|$ can be considered to have relatively large leverage.

The preceding calculations are for the canonical case of a Poisson point process whose intensity is a loglinear function (11.24) of the parameter β .

The calculations can be generalised to models fitted by maximising a criterion other than the likelihood, including robust maximum likelihood [346, 13, 14]. See [22] for the generalisation.

For a general model with intensity $\lambda_\theta(u)$, the same results hold with $\mathbf{Z}(u)$ replaced by

$$\mathbf{Z}_\theta(u) = \frac{\partial}{\partial \theta} \log \lambda_\theta(u)\tag{11.30}$$

and the Fisher information \mathcal{I}_{β} replaced by the negative Hessian of the loglikelihood

$$J(\theta) = \int_W \mathbf{Z}_\theta(u) \mathbf{Z}_\theta(u)^\top \lambda_\theta(u) du + \int_W \lambda_\theta(u) \frac{\partial}{\partial \theta} \mathbf{Z}_\theta(u) du - \sum_i \frac{\partial}{\partial \theta} \mathbf{Z}_\theta(x_i). \quad (11.31)$$

11.7.3 Influence for Poisson point process models

The *influence* is a discrete measure on the data points x_i with masses

$$m_i = \frac{1}{p} \mathbf{Z}(x_i) \mathcal{I}_{\hat{\beta}}^{-1} \mathbf{Z}(x_i)^\top. \quad (11.32)$$

Since the influence measure turns out to be an atomic measure on the data points only, it would be sufficient to compute the values $\log(L(\hat{\beta})/L(\hat{\beta}_{-i}))$ exactly for each data point x_i . Taylor series approximation is not necessary on computational grounds unless the dataset is very large. Similar comments apply in classical generalized linear models [213, 341]. However, the result (11.32) provides greater insight into the causes of influence.

The influence of a point process model is a value attached to each data point (i.e. each point of the point pattern to which the model was fitted). The influence value $s(x_i)$ at a data point x_i represents the change in the maximised log-likelihood that occurs when the point x_i is deleted. A relatively large value of $s(x_i)$ indicates a data point with a large influence on the fitted model.

The *effect measure* (“dfbeta”) is

$$\mathbb{D}\tilde{\beta}(B) = \mathcal{I}_{\hat{\beta}}^{-1} \left[\sum_{x_i \in B} \mathbf{Z}(x_i) - \int_B \mathbf{Z}(u) \lambda_{\hat{\beta}}(u) du \right]. \quad (11.33)$$

Since $\mathbf{Z}(u)$ is a vector-valued covariate function, with j th component $Z_j(u)$ corresponding to the j th model parameter θ_j , the effect measure $\mathbb{D}\tilde{\beta}$ is a vector-valued measure, or equivalently it is a list of signed measures, with the j th measure corresponding to the j th model parameter. The measure has atomic and continuous components. The atomic component has masses $\mathcal{I}_{\hat{\beta}}^{-1} \mathbf{Z}(x_i)$ at the data points x_i . The continuous component has density $-\mathcal{I}_{\hat{\beta}}^{-1}(u) \lambda_{\hat{\beta}}(u)$ at location $u \in W$.

The j th component of the influence measure, corresponding to the j th parameter θ_j , is a signed measure in two-dimensional space. It consists of a discrete mass on each data point (i.e. each point in the point pattern to which the model was originally fitted) and a continuous density at all locations. The mass at a data point represents the *negative* change in the fitted value of the parameter θ that would occur if this data point were to be deleted. The density at other non-data locations represents the *negative* effect (on the fitted value of θ) of deleting these locations (and their associated covariate values) from the input to the fitting procedure.

Give better explanation

The effect measure $\mathbb{D}\tilde{\beta}$ is a weighted point process residual, equation (11.8), with weight $w(u) = \mathcal{I}_{\hat{\beta}}^{-1}(u) \lambda_{\hat{\beta}}(u)$. This can be visualised using methods described in Section 11.4.

11.7.4 Implementation in spatstat

For a fitted point process model `fit`, the commands

```
> lev <- leverage(fit)
> inf <- influence(fit)
> dfb <- dfbetas(fit)
```

calculate the leverage, influence and parameter influence respectively. The functions `leverage()`, `influence()` and `dfbetas()` are generic. The commands above invoke the methods

`leverage.ppm()`, `influence.ppm()` and `dfbetas.ppm()` respectively for objects of class "ppm" representing point process models.

The result of `leverage.ppm()` is an object belonging to the special class "leverage.ppm" representing the leverage function $\hat{h}(u)$. It can be plotted (by the plot method `plot.leverage.ppm()`) or converted to a pixel image by `as.im()`.

The result of `influence.ppm()` is an object of class "influence.ppm" representing the influence masses s_i . It can be plotted (by `plot.influence.ppm()`), or converted to a marked point pattern by `as.ppp()` (see `as.ppp.influence.ppm`).

The result of `dfbetas.ppm()` is a signed measure (object of class "msr").

If the point process intensity has irregular parameters ("covariate function arguments" that were fitted using `ippm()`) then the leverage/influence calculation requires the first and second derivatives of the log intensity with respect to the irregular parameters. The argument `iScore` should be a list, with one entry for each irregular parameter, of R functions that compute the partial derivatives of the log intensity with respect to each irregular parameter. The argument `iHessian` should be a list, with p^2 entries where p is the number of irregular parameters, of R functions that compute the second order partial derivatives of the log intensity with respect to each pair of irregular parameters.

11.7.5 Murchison example

```
> mur <- lapply(murchison, rescale, s=1000, unitname="km")
> attach(mur)
> green <- greenstone
> dfault <- distfun(faults)

> ## spatial resolution in km for pixels, dummy points
> EPS <- 0.5
```

Here we consider the full loglinear model with interaction between the variables `green` and `dfault`. That is, we allow the log intensity to depend on distance to the nearest fault according to two completely different linear relationships holding inside and outside the greenstone.

```
> murfit1x <- ppm(gold ~ green * dfault, eps=EPS)
> murlev1x <- leverage(murfit1x)
> murinf1x <- influence(murfit1x)
> murdfb1x <- dfbetas(murfit1x)
```

Figure 11.13 shows a perspective plot of the leverage function for this model, generated by `persp(as.im(murlev1x))`. Sharp peaks indicate small regions of space (in fact, small regions of greenstone outcrop) with very high leverage. The tallest peak in this plot, at upper left, is associated with a small region of greenstone outcrop at a relatively large distance from the nearest fault. The high leverage indicates that the presence or absence of gold deposits inside this region has a relatively large effect on the fitted intensity.

Figure 11.14 shows the influence measure including a few strikingly large values, indicating that these points had a substantial effect on the fitted model. To investigate more closely, we selected small neighbourhoods of influential points (by displaying Figure 11.14 in a graphics window and using `clickbox()`) and displayed the detailed survey data.

Figure 11.15 shows the neighbourhood of the most highly influential point, in the top right corner of Figure 11.14. The survey data are shown in the left panel, and the corresponding influence values in the right panel. Similarly Figure 11.16 shows the neighbourhood of another highly influential point in the middle left of the survey. It is clear why these gold deposits are so influential:

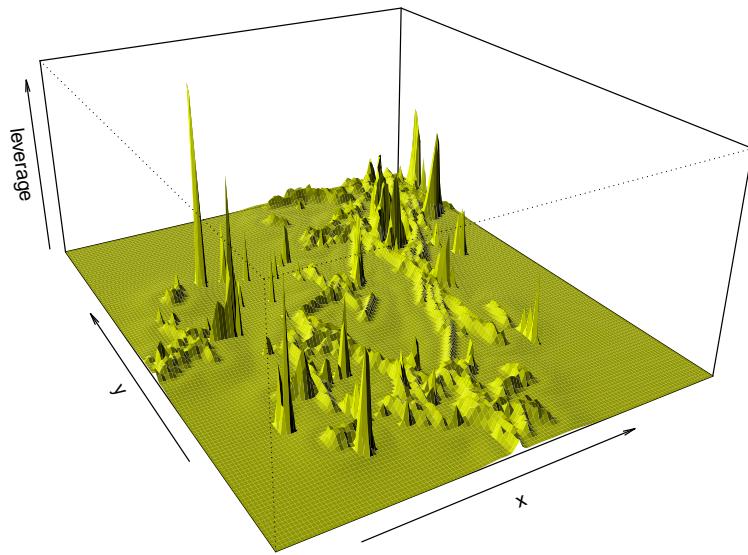


Figure 11.13: Perspective plot of leverage function for full loglinear model of Murchison data.

they lie a long way from the nearest recorded fault, and are therefore anomalous with respect to the rest of the survey data. The influence measure has automatically identified these anomalies.

For automatic identification of the most influential points, the influence values s_i can be extracted as `marks(as.ppp(murinf1x))`.

The *direction* in which each data point affects the fitted model can be read from the parameter influence measure `murdffb1x <- dfbetas(murfit1x)`. Figure 11.17 shows a detail of this measure in the same area as Figure 11.16. The influential point causes the coefficient `greenTRUE` to decrease substantially, and the coefficient `greenTRUE:dfault` to increase substantially.

What should be done with highly influential points? This depends on the context. For the strikingly large value of influence at the top right of Figure 11.14, shown in detail in the left panel of Figure 11.16, there is an issue of *edge effects*. The gold deposit lies on the boundary of the survey region. The true distance from this gold deposit to the nearest geological fault may be less than the observed distance, since only faults intersecting the survey region are recorded.

11.7.6 Chorley-Ribble example

The Chorley-Ribble data were introduced on page 323. Data analysis in [126, 129, 34] (See page 324) concluded there is evidence of an increased disease risk associated with proximity to the incinerator. An important caveat [126, 129] is that there is a cluster of four cases of laryngeal cancer very close to the incinerator. There is concern that these cases may have a strong influence on the fitted model and hence on the evidence for an incinerator effect.

The fitted model is a ‘raised incidence’ model (9.75)–(9.76) that includes an effect $b_\theta(u)$ due to proximity to the incinerator. The real parameters α, β control the magnitude and dropoff rate, respectively, of the incinerator effect. Their MLE’s are $\hat{\alpha} = 23.67$ and $\hat{\beta} = 0.91 \text{ km}^{-2}$.

For leverage and influence calculations we need the first and second partial derivatives of



Figure 11.14: Influence for Murchison gold deposits (circles) and two regions of high influence (grey shaded rectangles).

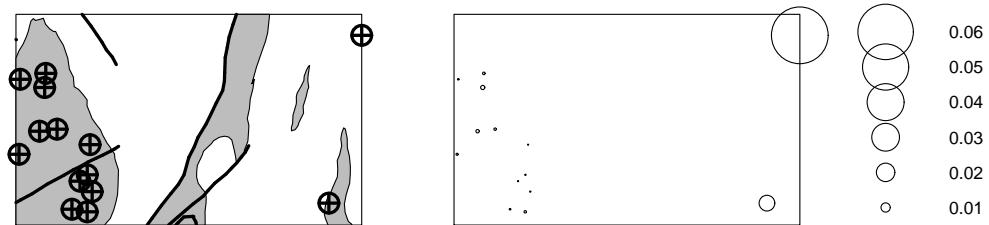


Figure 11.15: Details of Murchison survey in the vicinity of the most influential point, in the top right corner of the survey. Detail is about 50 by 30 kilometres. *Left:* data, including gold deposits (\oplus), faults (—) and greenstone outcrop (grey shading); *Right:* influence values for each gold deposit, represented by circle diameter.

$\log b_\theta(u)$ with respect to α and β . There are two ways to do this. The first is to write functions by hand which evaluate the partial derivatives, and pass them to `leverage.ppm()`, `influence.ppm()` and `dfbetas.ppm()` as the arguments `iScore` and `iHessian`. The second way is to use the symbolic calculus features of R.

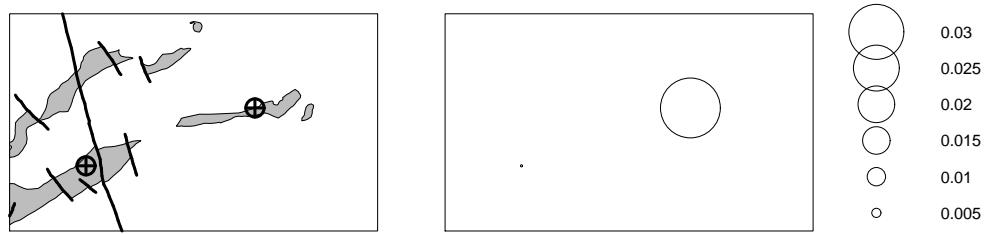


Figure 11.16: Detail of Murchison survey in the vicinity of a highly influential point at middle left of the survey. Detail is about 60 by 40 kilometres. *Left:* data; *Right:* influence values.

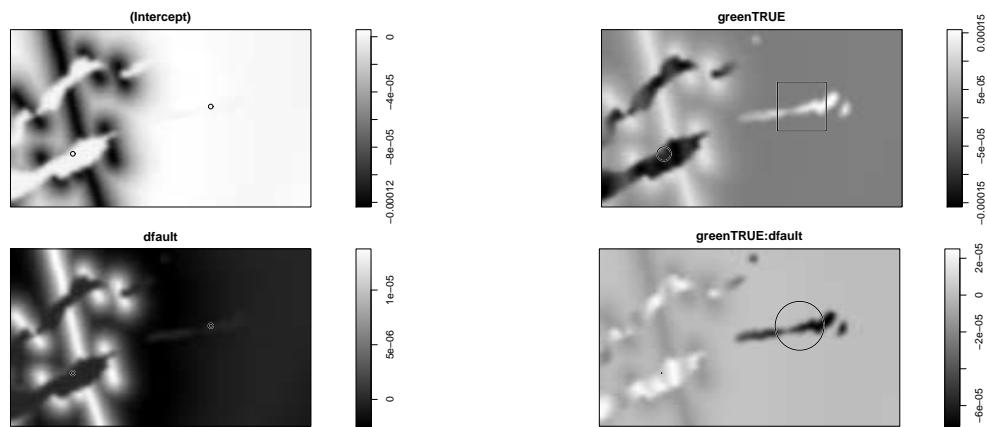


Figure 11.17: Detail of the parameter influence measure `dfbetas()` for the Murchison gold deposits survey in the same area as Figure 11.16. The lack of a symbol legend for the panels on the left indicates that the values at the data points were extremely small.

The first strategy involves differentiating $\log b_\theta(u)$ by hand and typing. The score components: (See page 324 for definition of `d2incin`)

```
> Zalpha <- function(x,y, alpha, beta) {
  expbit <- exp( - beta * d2incin(x,y))
  expbit/(1 + alpha * expbit)
}
> Zbeta <- function(x,y, alpha, beta) {
  d2 <- d2incin(x,y)
  topbit <- alpha * exp( - beta * d2)
  - d2 * topbit/(1 + topbit)
}
> Zscore <- list(alpha=Zalpha, beta=Zbeta)
```

and the Hessian:

```
> Zaa <- function(x,y, alpha, beta) {
  expbit <- exp( - beta * d2incin(x,y))
  -(expbit/(1 + alpha * expbit))^2
}
```

```

> Zab <- function(x,y, alpha, beta) {
  d2 <- d2incin(x,y)
  expbit <- exp( - beta * d2)
  - d2 * expbit/(1 + alpha * expbit)^2
}
> Zbb <- function(x,y, alpha, beta) {
  d2 <- d2incin(x,y)
  topbit <- alpha * exp( - beta * d2)
  (d2^2) * topbit/(1 + topbit)^2
}
> Zhess <- list(aa=Zaa, ab=Zab, ba=Zab, bb=Zbb)

```

Then we compute the new leverage and influence measures.

```

> chorleyDXlev <- leverage(chorleyDfit,
                           iScore=Zscore, iHessian=Zhess)
> chorleyDXinf <- influence(chorleyDfit,
                            iScore=Zscore, iHessian=Zhess)
> chorleyDXdfb <- dfbetas(chorleyDfit,
                           iScore=Zscore, iHessian=Zhess)
> chorleyDXdfbsmo <- Smooth(chorleyDXdfb, sigma=1.5)

```

The second approach uses R's facilities for symbolic calculus. First we write an expression for $\log b_\theta(u)$ using only standard mathematical functions:

```

> logbexpr <- expression(log(1 +
  alpha * exp( - beta * ((x - 354.5)^2 + (y-413.6)^2))))

```

Then we use the symbolic differentiation command `deriv()`:

```

> logB <- deriv(logbexpr, c("alpha", "beta"),
  c("x", "y", "alpha", "beta"),
  hessian=TRUE)

```

The result `logB` is a function with arguments `x,y,alpha,beta`. The function returns a vector of values of $\log b_\theta(u)$ which also has an attribute "gradient" containing the first derivatives and an attribute "hessian" containing the second derivatives. This function `logB` is passed to `ppm()` or `ippm()` as a covariate. The presence of the attributes "gradient" and "hessian" is detected, and these are used to compute the leverage and incidence diagnostics. The computation of derivatives also makes `ippm()` faster and more reliable. However the user function must be an additive term in the log intensity, so this is a bit delicate.

```

> chorleyDfitI <- ippm(Q ~offset(log(smo) + logB),
  start=list(alpha=10, beta=1))

> chorleyDfitI
Nonstationary Poisson process

Trend formula: ~offset(log(smo) + logB)

Fitted trend coefficient:
(Intercept)
-2.896741

```

```

Irregular parameters (covfunargs) fitted by 'ippm':
alpha = 22.25388
beta = 0.8889001

Estimate      S.E.    CI95.lo   CI95.hi Ztest      Zval
(Intercept) -2.896741 0.1313064 -3.154097 -2.639385 *** -22.06092

> chorleyDXlev <- leverage(chorleyDfitI)
> chorleyDXinf <- influence(chorleyDfitI)
> chorleyDXdfb <- dfbetas(chorleyDfitI)

```

Figure 11.18 shows the leverage function for Diggle's model. In order to account for the effect of estimating the irregular parameters α and β , the expression for leverage (11.28) is modified as described on page 387, with $\mathbf{Z}(u)$ replaced by (11.30) and \mathcal{J}_β replaced by (11.31). Figure 11.18 shows that the population close to the incinerator has very high leverage relative to other regions.

Figure 11.19 shows the influence masses for Diggle's model, again including the effect of estimating the irregular parameters. That is, (11.32) was modified by replacing $\mathbf{Z}(u)$ by (11.30) and \mathcal{J}_β by (11.31). The observed group of four cases very close to the incinerator has very large influence. However, the largest influence is enjoyed by a case slightly to the northwest of the incinerator, at a location where the reference population density $\rho(u)$ is small.

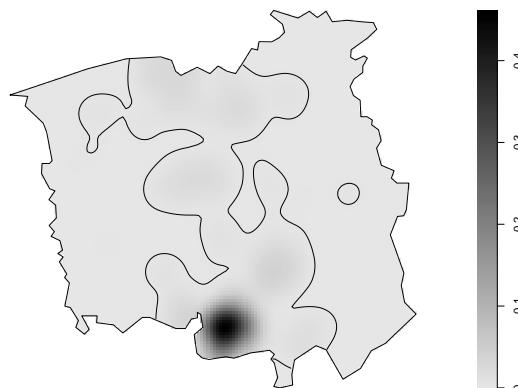


Figure 11.18: Leverage function for raised incidence model of Chorley-Ribble.

Figure 11.20 shows the components of the effect measure $\mathbb{D}\tilde{\beta}$ for the irregular parameters α and β in Diggle's model. Here (11.33) was modified by replacing $\mathbf{Z}(u)$ by (11.30) and \mathcal{J}_β by (11.31). Remember that the effect measure is the *negative* of the change in the fitted parameter value that would occur if the relevant data were deleted. Thus a positive value of effect measure means that the corresponding data is “pulling the parameter estimate upward”. Note the presence of two large squares, representing cases of laryngeal cancer with large negative effect on the parameter estimates, situated to the north-west and south-east of the incinerator. We conclude that the estimated strength α of the incinerator effect is strongly influenced by these two cases (tending to reduce the estimated effect) and by the cluster of four cases close to the incinerator (tending to increase the estimated effect). The estimated dropoff rate β of the incinerator effect is overwhelmingly influenced by the two cases with large negative effect.

Note that Diggle and Rowlingson [129] advocated a different analysis of the same data, analysing conditionally on the spatial locations. Each location has a binary response indicating whether it is a case or control. The conditional model is a binary regression (logistic regression if



Figure 11.19: Influence for raised incidence model of Chorley-Ribble. *Left:* full data. *Right:* detail close to incinerator.



Figure 11.20: Components of effect measure for parameters α (Left) and β (Right) in Diggle's model of Chorley-Ribble data; detail close to the incinerator. Density of continuous component is depicted as a grey-scale image. Masses at data points are shown as circles and squares, for positive and negative masses respectively.

α, β is fixed). The conditional analysis can be validated using the standard diagnostics for leverage and influence in logistic regression [272]. These diagnostics give leverage and influence values for each *data point* (case of cancer of the larynx or lung), in a fashion quite similar to our analysis, but without the effect of kernel smoothing.

Connect to logistic likelihood

Since clustering of disease cases is a possible alternative explanation, it would be prudent to consider point process models which incorporate positive association between points. Our diagnostics then provide information about the evidence for such clustering, as well as the evidence for covariate effects after allowing for clustering. An example is given in Chapter 13

11.8 Validating the independence assumption

11.8.1 Caveat on residual plots

The residual plots described in Sections 11.3–11.6 above are useful for detecting misspecification of the *trend* in a fitted Poisson process model. They are not very useful for checking the *independence* assumption of the Poisson process (assumption (PP3) on page ??).

An extreme example is provided by the `cells` dataset. The four-panel diagnostic plot for a uniform Poisson process fitted to the `cells` data, in Figure 11.21, shows no evidence against the model.

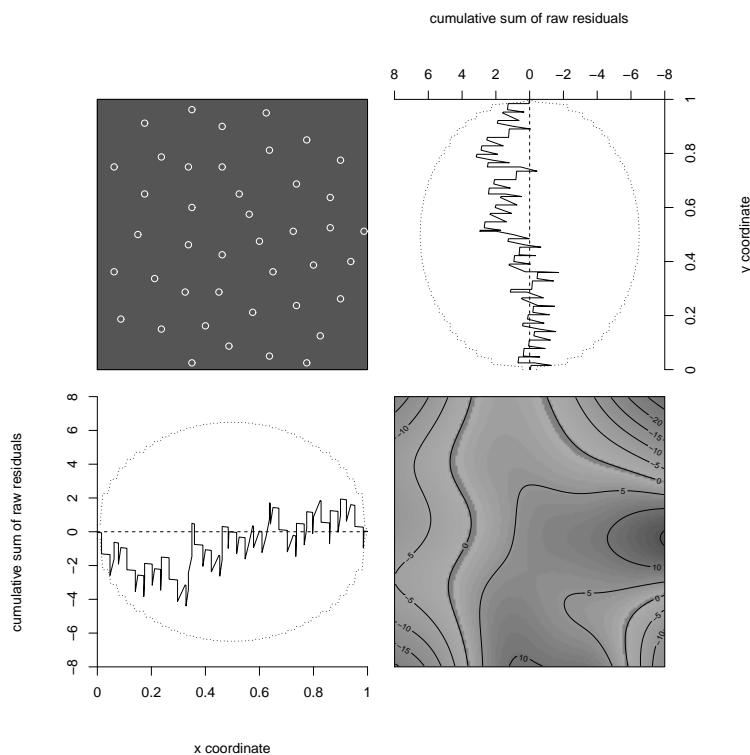


Figure 11.21: Four-panel diagnostic plot for uniform Poisson model fitted to the `cells` data.

However, the K -function (plotted in Figure 11.22) shows that the `cells` dataset is clearly not a Poisson pattern, and has strong inhibition.

11.8.2 Q-Q plot of residual field

Interaction between points in a point process affects the probability distribution of the number of points in a subregion. To detect changes in the distribution of numbers of points, we may consider an analogy with regression. To validate distributional assumptions in regression, a reliable tool is the quantile-quantile plot or *Q–Q plot* in which observed quantiles of the residuals are plotted against

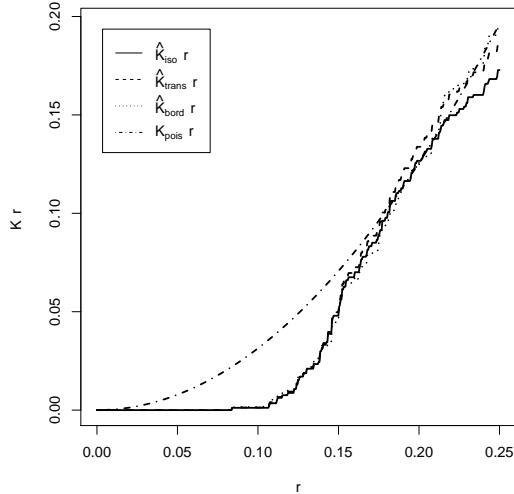


Figure 11.22: Estimated K -function for the `cells` data.

corresponding theoretical quantiles of their assumed distribution (or sometimes, against quantiles of the assumed distribution of *errors*).

By analogy, to validate the interaction terms in a point process model, we could use a Q–Q plot of the values of the *smoothed residual field* s defined in Section 11.4.2 on page 374. After computing s on a fine grid of pixels u , we effectively take the pixel values $s(u)$ as observations, and sort them into ascending order. The distribution of values of s under the model is complicated [28] so we approximate it by Monte Carlo methods. We simulate realisations from the fitted model, re-fit the model to them, compute the smoothed residual fields of these new fits, and evaluate the quantiles. Following [157] we estimate the *expected* quantiles, by averaging the j th order statistic of s over the different simulations, for each j . Details are given in [34]. The observed quantiles of s are plotted against the estimated expected quantiles of s under the fitted model.

Figure 11.23 shows the Q–Q plot of the smoothed raw residuals for the uniform Poisson model fitted to the `cells` data. Dashed lines show pointwise 5% critical envelopes from simulations of the fitted model. This indicates that the uniform Poisson model is grossly inappropriate for the `cells` data.

For particular models of interpoint interaction, the Q–Q plot is closely related to the summary functions F , G and K . See [34].

11.8.3 Summary functions

Dependence between points can be measured using summary functions such as the K -function (section 7.3), the pair correlation function (section 7.6), the empty space function F (section ??) and the nearest-neighbour distance distribution function G (section ??). However, these summary functions assume the point process is **stationary**, and they can give very misleading results if applied to non-stationary processes. In the context of model validation, these summary functions should only be used if the fitted model is the homogeneous Poisson process (Complete Spatial Randomness).

Summary functions can be modified to ‘adjust’ for inhomogeneity. One strategy is to introduce weights into the summary function estimator, which compensate for the spatially-varying intensity. Examples are the inhomogeneous K -function and inhomogeneous pair correlation function (section 7.10.2) and inhomogeneous F and G functions (REFS). In model validation, these can be used

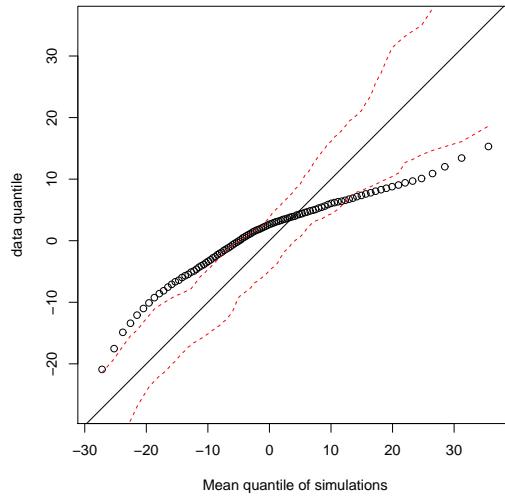


Figure 11.23: Q–Q plot of smoothed residuals for uniform Poisson model fitted to cells data.

to discriminate between a fitted inhomogeneous Poisson model and an inhomogeneous process exhibiting interpoint dependence.

Give example of Kinhom as diagnostic

An often-overlooked problem with the inhomogeneous K -function is that it also imposes an assumption on the point process, which may or may not be satisfied in a real application. Explain about second-order intensity reweighted stationarity (SOIRS), basically the assumption that the reweighted function is not dependent on location. Thus the interpretation of deviations of the inhomogeneous K -function may also be problematic. Explain about other inhomogeneity models and what would happen if they were true instead.

11.8.4 Residual summary functions

For model validation, there is an alternative strategy for compensating for spatial inhomogeneity. We can compute a *residual summary function* which would have expected value zero if the fitted model was true. This approach was developed in [30].

We start with a summary statistic or summary function T that is a sum of contributions from each point in \mathbf{x} :

$$T(\mathbf{x}) = \sum_i t(x_i, \mathbf{x} \setminus x_i) \quad (11.34)$$

for any function t , where $\mathbf{x} \setminus x_i$ is the set \mathbf{x} excluding the point x_i . Examples include the empirical K -function

$$\widehat{K}_{\mathbf{x}}(r) = \frac{1}{\widehat{\lambda}^2(\mathbf{x})|W|} \sum_i \sum_{j \neq i} e_K(x_i, x_j) \mathbf{1}\{\|x_i - x_j\| \leq r\} \quad (11.35)$$

and the empirical nearest-neighbour distance distribution function G ,

$$\widehat{G}_{\mathbf{x}}(r) = \frac{1}{n(\mathbf{x})} \sum_i e_G(x_i, \mathbf{x} \setminus x_i, r) \mathbf{1}\{d(x_i, \mathbf{x} \setminus x_i) \leq r\} \quad (11.36)$$

where $e_K(u, v)$, $e_G(u, v, r)$ are edge correction weights, and $\widehat{\lambda}^2(\mathbf{x}) = n(\mathbf{x})(n(\mathbf{x}) - 1)/|W|^2$. These are

of the general form (11.34) where the function t is respectively

$$t_K(u, \mathbf{y}) = \frac{1}{\lambda^2(\mathbf{y} \cup \{u\})|W|} \sum_j e_K(u, y_j) \mathbf{1}\{\|u - y_j\| \leq r\} \quad (11.37)$$

$$t_G(u, \mathbf{y}) = \frac{1}{n(\mathbf{y} \cup \{u\})} e_G(u, \mathbf{y}, r) \mathbf{1}\{d(x_i, \mathbf{y}) \leq r\} \quad (11.38)$$

for any given value of r .

For a Poisson point process \mathbf{X} with intensity function $\lambda(u)$,

$$\mathbb{E} \sum_i t(x_i, \mathbf{X} \setminus x_i) = \mathbb{E} \int_W t(u, \mathbf{X}) \lambda(u) du \quad (11.39)$$

under minimal conditions on the function t . Therefore the difference

$$RT(\mathbf{x}) = \sum_i t(x_i, \mathbf{x} \setminus x_i) - \int_W t(u, \mathbf{x}) \lambda(u) du \quad (11.40)$$

satisfies $\mathbb{E} RT(\mathbf{X}) = 0$ for the Poisson process with intensity function $\lambda(u)$. Notice that the integral on the right hand side of (11.40) is data-dependent: it is not equal to the expected value of $T(\mathbf{X})$, but has the same expected value as $T(\mathbf{X})$. In applications we usually estimate the intensity $\lambda(u)$ and calculate the *compensator*

$$CT(\mathbf{x}) = \int_W t(u, \mathbf{x}) \lambda(u) du \quad (11.41)$$

and the *residual* $RT(\mathbf{x}) = T(\mathbf{x}) - CT(\mathbf{x})$. [30]

Then, for example, the *residual K-function* is

$$RK_{\mathbf{x}}(r) = \hat{K}_{\mathbf{x}}(r) - \int_W t_K(u, \mathbf{x}) \hat{\lambda}(u) du. \quad (11.42)$$

The residual *K*-function depends on the model, and both terms on the right side of (11.42) are data-dependent. If the model is a good fit, then $RK_{\mathbf{x}}(r)$ should have approximately zero mean, for all values of r .

It is instructive to consider the special case where the fitted model is CSR (uniform Poisson process). The compensator can then be calculated, at least ignoring edge effects. For the *K*-function we find [30]

$$RK_{\mathbf{x}}(r) \approx \frac{n(\mathbf{x})^2}{2|W|} \left[\hat{K}_{\mathbf{x}}(r) - \pi r^2 \right]. \quad (11.43)$$

The term in brackets is a commonly-used measure of departure from CSR, and is a sensible diagnostic because $K(r) = \pi r^2$ under CSR. Similarly for the residual *G*-function, when the fitted model is CSR,

$$RG_{\mathbf{x}}(r) \approx n(\mathbf{x})(\hat{G}_{\mathbf{x}}(r) - \hat{F}_{\mathbf{x}}(r)) \quad (11.44)$$

where $\hat{F}_{\mathbf{x}}(r)$ is the empirical estimate of the empty space function F . The right hand side of (11.44) is a reasonable diagnostic for departure from CSR, since $F \equiv G$ under CSR. This argument lends support to Diggle's [121, eq. (5.7)] proposal to judge departure from CSR using the quantity $\max_r |\hat{G}(r) - \hat{F}(r)|$.

Implementation in spatstat

The residual *K*-function is computed in **spatstat** by the command `Kres()`. If `fit` is a point process model fitted by `ppm()`, then `Kres(fit)` is the residual *K*-function of the fitted model. Further arguments to `Kres()` specify the edge correction to be used, and other options for the

calculation. For a point pattern X , typing `Kres(X)` gives the residual K -function of CSR fitted to X . By specifying additional arguments, another point process model can be fitted to X instead.

Similarly `Kcom()` computes the compensator of the fitted model or the compensator of CSR. For the G -function, `Gres()` computes the residual and `Gcom()` the compensator.

Figure 11.24 shows the residual K and G functions for CSR fitted to the `cells` data, computed by

```
> cellKr <- Kres(cells, correction="best")
> cellGr <- Gres(cells, correction="best")
```

Dotted lines in Figure 11.24 show the two-standard-deviation limits based on the “Poincaré variance” [30], a rough approximation to the variance of the residual, which is easy to compute in this context. The residual K and G functions wander substantially outside these limits, strongly suggesting that CSR is not a satisfactory model for the `cells` data, and suggesting that the `cells` data show greater regularity than a Poisson process.

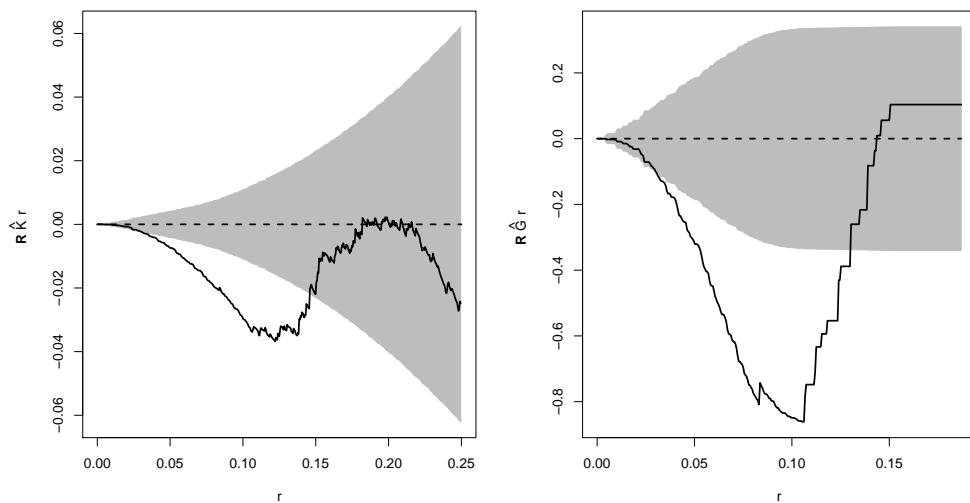


Figure 11.24: Residual K -function (Left) and G -function (Right) for the uniform Poisson point process fitted to the `cells` data. Solid lines: residual function. Grey shading: pointwise two-standard-deviation limits. Dashed lines: expected residual (zero) under fitted model.

Figure 11.25 shows the residual K function for a Poisson model with intensity a log-cubic function of the coordinates, fitted to the full Japanese Pines data, computed by

```
> jfit <- ppm(residualspaper$Fig1 ~ polynom(x,y,3))
> jKr <- Kres(jfit, correction="best")
```

The conclusion here is marginal: there may be slight evidence against the Poisson assumption, after accounting for spatial inhomogeneity.

```
> fvnames(jKr, ".s") <- c("ihi", "ilo")
```

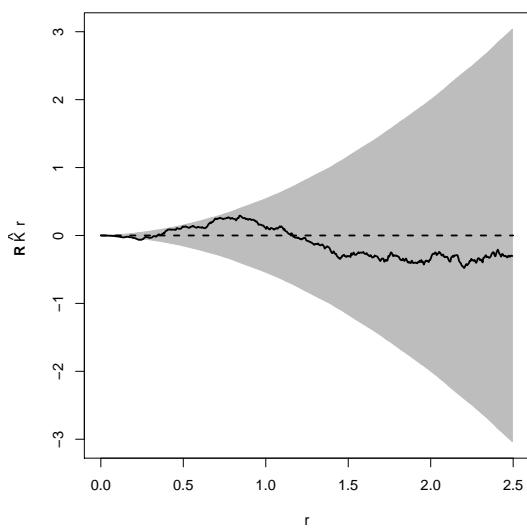


Figure 11.25: Residual K -function for the Poisson point process with intensity a log-cubic function of the coordinates, fitted to the full Japanese Pines data.



12

Model-fitting using second moments

This chapter is not part of the review draft.

12.1 Not included in review



13

Gibbs models

This chapter is not part of the review draft.

13.1 Not included in review



14

Inference for multitype patterns

This chapter is not part of the review draft.

14.1 Not included in review



Part IV

ADDITIONAL STRUCTURE



15

Inference for multivariate marks

This chapter is not part of the review draft.

15.1 Not included in review



16

Replicated point patterns

This chapter is not part of the review draft.

16.1 Not included in review

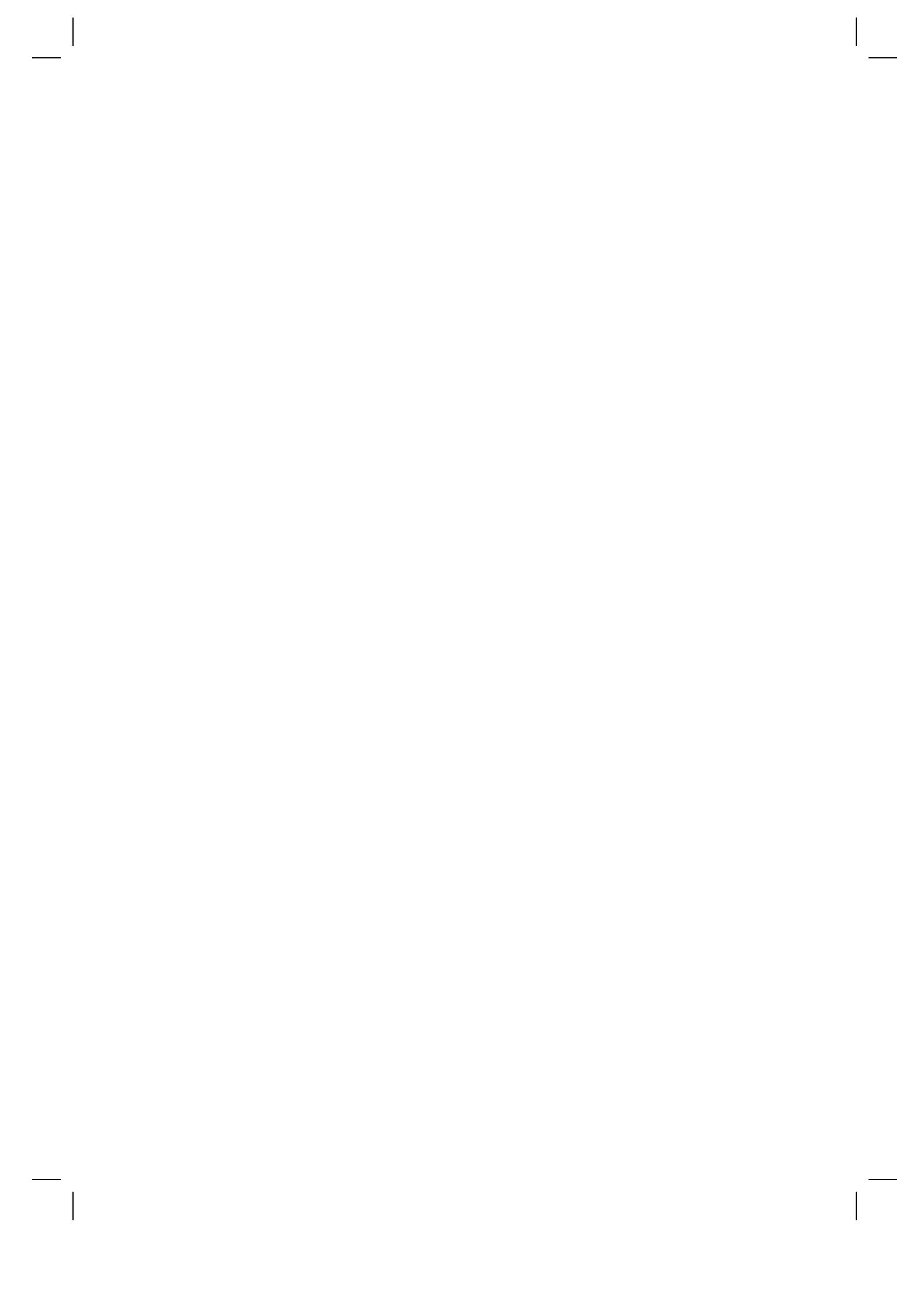


17

Point patterns on a linear network

This chapter is not part of the review draft.

17.1 Not included in review



Bibliography

- [1] F.P. Agterberg. Automatic contouring of geological maps to detect target areas for mineral exploration. *Journal of the International Association for Mathematical Geology*, 6:373–395, 1974.
- [2] N. Ahuja. Dot pattern processing using Voronoi neighbourhoods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 4:336–343, 1982.
- [3] M. Aitkin, D.A. Anderson, B.J. Francis, and J.P. Hinde. *Statistical Modelling in GLIM*. Oxford University Press, 1989.
- [4] M. Aitkin and D. Clayton. The fitting of exponential, Weibull and extreme value distributions to complex censored survival data using GLIM. *Applied Statistics*, 29:156–163, 1980.
- [5] H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974.
- [6] A. Albert and J.A. Anderson. On the existence of maximum likelihood estimates in logistic regression models. *Biometrika*, 71:1–10, 1984.
- [7] D. Allard and C. Fraley. Nonparametric maximum likelihood estimation of features in spatial point processes using Voronoi tessellation. *Journal of the American Statistical Association*, 92:1485–1493, 1997.
- [8] S.E. Alm. Approximation and simulation of the distributions of scan statistics for Poisson processes in higher dimensions. *Extremes*, 1(1):111–126, 1998.
- [9] J.E. Alt, G. King, and C.S. Signorino. Aggregation among binary, count and duration models: estimating the same quantities from different levels of data. *Political Analysis*, 9:21–44, 2001.
- [10] T.W. Anderson and D.A. Darling. Asymptotic theory of certain ‘goodness-of-fit’ criteria based on stochastic processes. *Annals of Mathematical Statistics*, 23:193–212, 1952.
- [11] T.W. Anderson and D.A. Darling. A test of goodness of fit. *Journal of the American Statistical Association*, 49:765–769, 1954.
- [12] L. Anselin. Local indicators of spatial association – LISA. *Geographical Analysis*, 27:93–115, 1995.
- [13] R. Assunção. *Robust estimation in point processes*. PhD thesis, University of Washington, Seattle, 1994.
- [14] R. Assunção and P. Guttorp. Robustness for inhomogeneous Poisson point processes. *Annals of the Institute of Statistical Mathematics*, 51:657–678, 1999.
- [15] A.C. Atkinson. Diagnostics, regression analysis and shifted power transformations. *Technometrics*, 25:23–34, 1983.

- [16] A.C. Atkinson. *Plots, Transformations and Regression*. Number 1 in Oxford Statistical Science Series. Oxford University Press/ Clarendon, 1985.
- [17] A. Baddeley. Interpreting results of spatial logistic regression. In preparation.
- [18] A. Baddeley. Analysing spatial point patterns in R. Technical report, CSIRO, 2010. Version 4. Available at www.csiro.au/resources/pf16h.html.
- [19] A. Baddeley. Modelling strategies. In A.E. Gelfand, P.J. Diggle, M. Fuentes, and P. Guttorp, editors, *Handbook of Spatial Statistics*, chapter 20, pages 339–369. CRC Press, Boca Raton, 2010.
- [20] A. Baddeley, M. Berman, N.I. Fisher, A. Hardegen, R.K. Milne, D. Schuhmacher, and R. Turner. Spatial logistic regression and change-of-support for Poisson point processes. *Electronic Journal of Statistics*, 4:1151–1201, 2010. doi: 10.1214/10-EJS581.
- [21] A. Baddeley, Y.-M. Chang, Y. Song, and R. Turner. Residual diagnostics for covariate effects in spatial point process models. *Journal of Computational and Graphical Statistics*, 22:886–905, 2013.
- [22] A. Baddeley, Y.M. Chang, and Y. Song. Leverage and influence diagnostics for spatial point processes. *Scandinavian Journal of Statistics*, 40:86–104, 2013. doi: 10.1111/j.1467-9469.2011.00786.x.
- [23] A. Baddeley, Y.M. Chang, Y. Song, and R. Turner. Nonparametric estimation of the dependence of a spatial point process on a spatial covariate. *Statistics and its Interface*, 5:221–236, 2012.
- [24] A. Baddeley, J.-F. Coeurjolly, E. Rubak, and R. Waagepetersen. Logistic regression for spatial Gibbs point processes. *Biometrika*, 101(2):377–392, 2014.
- [25] A. Baddeley, P.J. Diggle, A. Hardegen, T. Lawrence, R.K. Milne, and G. Nair. On tests of spatial pattern based on simulation envelopes. *Ecological Monographs*, 84(3):477–489, 2014.
- [26] A. Baddeley, A. Hardegen, T. Lawrence, R. Milne, and G.M. Nair. Pushing the envelope: extensions of graphical Monte Carlo tests. In preparation.
- [27] A. Baddeley, M. Kerscher, K. Schladitz, and B.T. Scott. Estimating the J function without edge correction. *Statistica Neerlandica*, 54(3):315–328, November 2000.
- [28] A. Baddeley, J. Møller, and A.G. Pakes. Properties of residuals for spatial point processes. *Annals of the Institute of Statistical Mathematics*, 60:627–649, 2008.
- [29] A. Baddeley, J. Møller, and R. Waagepetersen. Non- and semiparametric estimation of interaction in inhomogeneous point patterns. *Statistica Neerlandica*, 54(3):329–350, 2000.
- [30] A. Baddeley, E. Rubak, and J. Møller. Score, pseudo-score and residual diagnostics for spatial point process models. *Statistical Science*, 26:613–646, 2011.
- [31] 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.
- [32] 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.

- [33] 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.
- [34] 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.
- [35] A.J. Baddeley. A limit theorem for statistics of spatial data. *Advances in Applied Probability*, 12:447–461, 1980.
- [36] A.J. Baddeley. Stereology and survey sampling theory. *Bulletin of the International Statistical Institute*, 50, book 2:435–449, 1993.
- [37] 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, 1999.
- [38] A.J. Baddeley and L.M. Cruz-Orive. The Rao-Blackwell theorem in stereology and some counterexamples. *Advances in Applied Probability*, 27:2–19, 1995.
- [39] A.J. Baddeley and R.D. Gill. Kaplan-Meier estimators for interpoint distance distributions of spatial point processes. Research Report BS-R9315, Centrum voor Wiskunde en Informatica, july 1993.
- [40] A.J. Baddeley and R.D. Gill. The empty space hazard of a spatial pattern. Preprint 845, Department of Mathematics, University of Utrecht, 1994.
- [41] A.J. Baddeley and R.D. Gill. Kaplan-Meier estimators of interpoint distance distributions for spatial point processes. *Annals of Statistics*, 25:263–292, 1997.
- [42] A.J. Baddeley and H.J.A.M. Heijmans. Incidence and lattice calculus with applications to stochastic geometry and image analysis. *Applicable Algebra in Engineering, Communication, and Computing*, 6(3):129–146, 1995.
- [43] 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.
- [44] 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.
- [45] D. Bakan. The test of significance in psychological research. *Psychol. Bull.*, 66:423–437, 1966.
- [46] S. Banerjee and A.E. Gelfand. Prediction, interpolation and regression for spatially misaligned data. *Sankhya A*, 64:227–245, 2002.
- [47] J.D. Banfield and A.E. Raftery. Model-based Gaussian and non-Gaussian clustering. *Biometrics*, 49:803–821, 1993.
- [48] G. Barnard. Contribution to discussion of “The spectral analysis of point processes” by M.S. Bartlett. *Journal of the Royal Statistical Society, series B*, 25:294, 1963.
- [49] O.E. Barndorff-Nielsen. *Information and Exponential families in Statistical Theory*. Wiley, Chichester, New York, Brisbane, Toronto, 1978.

- [50] M. S. Bartlett. Processus stochastiques ponctuels. *Ann. Inst. Henri Poincaré*, 14:35–60, 1954.
- [51] M.S. Bartlett. A note on spatial pattern. *Biometrics*, 20:891–892, 1964.
- [52] M. J. Bayarri and J.O. Berger. P values for composite null models. *Journal of the American Statistical Association*, 95(452):1127–1142, December 2000.
- [53] T. Bedford and J. van den Berg. A remark on the Van Lieshout and Baddeley J-function for point processes. *Advances in Applied Probability*, 29:19–25, 1997.
- [54] M. Berman. Testing for spatial association between a point process and another stochastic process. *Applied Statistics*, 35:54–62, 1986.
- [55] M. Berman and P. Diggle. Estimating weighted integrals of the second-order intensity of a spatial point process. *Journal of the Royal Statistical Society, series B*, 51:81–92, 1989.
- [56] M. Berman and T.R. Turner. Approximating point process likelihoods with GLIM. *Applied Statistics*, 41:31–38, 1992.
- [57] R. Bernhardt, F. Meyer-Olbersleben, and B. Kieback. Fundamental investigation on the preparation of gradient structures by sedimentation of different powder fractions under gravity. In David Hui, editor, *Proceedings of the 4th International Conference on Composite Engineering, ICCE/4, July 6–12 1997*, pages 147–148, Hawaii, 1997.
- [58] J. Besag and P.J. Diggle. Simple Monte Carlo tests for spatial pattern. *Applied Statistics*, 26:327–333, 1977.
- [59] J. Besag, R. Milne, and S. Zachary. Point process limits of lattice processes. *Journal of Applied Probability*, 19:210–216, 1982.
- [60] J. Besag and J. Newell. The detection of clusters in rare diseases. *Journal of the Royal Statistical Society, series A*, 154:143–155, 1991.
- [61] J. E. Besag. Contribution to the discussion of the paper by Ripley (1977). *Journal of the Royal Statistical Society, series B*, 39:193–195, 1977.
- [62] J.E. Besag and P. Clifford. Generalized Monte Carlo significance tests. *Biometrika*, 76:633–642, 1989.
- [63] J.F. Bithell. An application of density estimation to geographical epidemiology. *Statistics in Medicine*, 9:691–701, 1990.
- [64] R. Bivand, E.J. Pebesma, and V. Gómez-Rubio. *Applied spatial data analysis with R*. Springer, 2008.
- [65] G. Bonham-Carter. *Geographic Information Systems for geoscientists: modelling with GIS*. Number 13 in Computer Methods in the Geosciences. Pergamon Press/ Elsevier, Kidlington, Oxford, UK, 1995.
- [66] G. Borgefors. Distance transformations in arbitrary dimensions. *Computer Vision, Graphics and Image Processing*, 27:321–345, 1984.
- [67] G. Borgefors. Distance transformations in digital images. *Computer Vision, Graphics and Image Processing*, 34:344–371, 1986.
- [68] D. Borland and R.M. Taylor. Rainbow color map (still) considered harmful. *IEEE Computer Graphics and Applications*, 27(2):14–17, 2007.

- [69] A. W. Bowman and A. Azzalini. *Applied smoothing techniques for data analysis: the kernel approach with S-Plus illustrations*. Oxford University Press, Oxford, 1997.
- [70] 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, New York, London, 1978.
- [71] D.R. Brillinger and H.K. Preisler. Two examples of quantal data analysis: a) multivariate point process, b) pure death process in an experimental design. In *Proceedings, XIII International Biometric Conference, Seattle*, pages 94–113. International Biometric Society, 1986.
- [72] D.R. Brillinger and J.P. Segundo. Empirical examination of the threshold model of neuron firing. *Biological Cybernetics*, 35:213–220, 1979.
- [73] S.P. Brooks, B.J.T. Morgan, M.S. Ridout, and S.E. Pack. Finite mixture models for proportions. *Biometrics*, 53:1097–1115, 1997.
- [74] W.M. Brown, T.D. Gedeon, A.J. Baddeley, and D.I. Groves. Bivariate J-function and other graphical statistical methods help select the best predictor variables as inputs for a neural network method of mineral prospectivity mapping. In U. Bayer, H. Burger, and W. Skala, editors, *IAMG 2002: 8th Annual Conference of the International Association for Mathematical Geology*, volume 1, pages 257–268. International Association of Mathematical Geology, 2002.
- [75] S. Byers and A.E. Raftery. Nearest-neighbour clutter removal for estimating features in spatial point processes. *Journal of the American Statistical Association*, 93:577–584, 1998.
- [76] S. Chainey and J. Ratcliffe. *GIS and Crime Mapping*. Wiley, Chichester, 2005.
- [77] R.M. Chambers and T. Hastie, editors. *Statistical models in S*. Wadsworth and Brooks/Cole, Monterey, 1992.
- [78] H.P. Chan. Detection of spatial clustering with average likelihood ratio test statistics. *American Statistician*, 37:3985–4010, 2009.
- [79] R.E. Chandler and E.M. Scott. *Statistical Methods for Trend Detection and Analysis in the Environmental Sciences*. Wiley, Chichester, 2011.
- [80] B.J. Chen, G.P. Lesser, D. Jackson, and R.A. Lamb. The influenza virus M2 protein cytoplasmic tail interacts with the M1 protein and influences virus assembly at the site of virus budding. *Journal of Virology*, 82:10059–10070, 2008.
- [81] S.N. Chiu. Spatial point pattern analysis by using Voronoi diagrams and Dirichlet tessellations — a comparative study. *Biometrical Journal*, 45:367–376, 2003.
- [82] S.N. Chiu and D. Stoyan. Estimators of distance distributions for spatial patterns. *Statistica Neerlandica*, 52:239–246, 1998.
- [83] E. Choi and P. Hall. Nonparametric analysis of earthquake point-process data. In M. de Gunst, C. Klaassen, and A. van der Vaart, editors, *State of the art in probability and statistics: Festschrift for Willem R. van Zwet*, pages 324–344. Institute of Mathematical Statistics, Beachwood, Ohio, 2001.
- [84] C.F. Chung and F.P. Agterberg. Regression models for estimating mineral resources from geological map data. *Mathematical Geology*, 12:473–488, 1980.

- [85] C.F. Chung and A.G. Fabbri. Probabilistic prediction models for landslide hazard mapping. *Photogrammetric engineering and remote sensing*, 62(12):1389–1399, 1999.
- [86] P.J. Clark and F.C. Evans. Distance to nearest neighbor as a measure of spatial relationships in populations. *Ecology*, 35:445–453, 1954.
- [87] M. Clyde and D. Strauss. Logistic regression for spatial pair-potential models. In A. Pos-solo, editor, *Spatial Statistics and Imaging*, volume 20 of *Lecture Notes - Monograph series*, chapter II, pages 14–30. Institute of Mathematical Statistics, Hayward, Calif., 1991. ISBN 0-940600-27-7.
- [88] J. Cohen. The earth is round ($p < .05$). *Am. Psychol.*, 49:997–1003, 1994.
- [89] D. Collett. *Modelling Binary Data*. Chapman and Hall, London, 1991.
- [90] R. Condit. *Tropical Forest Census Plots*. Springer Verlag, 1998.
- [91] R. Condit, S.P. Hubbell, and R.B. Foster. Changes in tree species abundance in a neotropical forest: impact of climate change. *Journal of Tropical Ecology*, 12:231–256, 1996.
- [92] R.D. Cook. Added-variable plots and curvature in linear regression. *Technometrics*, 38:275–278, 1996.
- [93] R.D. Cook and S. Weisberg. *Residuals and influence in regression*. Chapman and Hall, London, 1982.
- [94] R.D. Cook and S. Weisberg. *Applied regression, including computing and graphics*. John Wiley and Sons, 1999.
- [95] J.W. Cooley and J.W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19:297–301, 1965.
- [96] J.B. Copas. Plotting p against x . *Applied Statistics*, 32:25–31, 1983.
- [97] C.B. Cordy. An extension of the Horvitz-Thompson theorem to point sampling from a con-tinuous universe. *Statistics and Probability Letters*, 18:353–362, 1993.
- [98] D.R. Cox. *Planning of Experiments*. Wiley, New York, 1958.
- [99] D.R. Cox. The statistical analysis of dependencies in point processes. In P.A.W. Lewis, editor, *Stochastic Point Processes*, pages 55–66. Wiley, New York, 1972.
- [100] D.R. Cox. The role of significance tests. *Scandinavian Journal of Statistics*, 4:49–70, 1977.
- [101] D.R. Cox and C.A. Donnelly. *Principles of Applied Statistics*. Cambridge University Press, Cambridge, UK, 2011.
- [102] D.R. Cox and E.J. Snell. *Applied Statistics: principles and examples*. Chapman and Hall, 1981.
- [103] H. Cramér. On the composition of elementary errors: II, Statistical applications. *Skandinavisk Aktuarietidskrift*, 11:141–180, 1928.
- [104] N. Cressie. Change of support and the modifiable area unit problem. *Geographical Systems*, 3:159–180, 1996.
- [105] N. Cressie and L.B. Collins. Analysis of spatial point patterns using bundles of product density LISA functions. *Journal of Agricultural, Biological and Environmental Statistics*, 6:118–135, 2001.

- [106] N. Cressie and L.B. Collins. Patterns in spatial point locations: local indicators of spatial association in a minefield with clutter. *Naval Research Logistics*, 48:333–347, 2001.
- [107] N. Cressie and G.M. Laslett. Random set theory and problems of modeling. *SIAM Review*, 28:557–574, 1987.
- [108] N.A.C. Cressie. *Statistics for Spatial Data*. John Wiley and Sons, New York, 1991.
- [109] S. Csörgő and J.J. Faraway. The exact and asymptotic distributions of Cramér-von mises statistics. *Journal of the Royal Statistical Society, Series B*, 58:221–234, 1996.
- [110] J. Cuzick and R. Edwards. Spatial clustering for inhomogeneous populations (with discussion). *Journal of the Royal Statistical Society, series B*, 52:73–104, 1990.
- [111] R.B. D'Agostino and M.A. Stephens. *Goodness-of-fit techniques*. Marcel Dekker, New York, 1986.
- [112] D.J. Daley and D. Vere-Jones. *An Introduction to the Theory of Point Processes*. Springer Verlag, New York, 1988.
- [113] D.J. Daley and D. Vere-Jones. *An Introduction to the Theory of Point Processes. Volume I: Elementary Theory and Methods*. Springer Verlag, New York, second edition, 2003.
- [114] N.A. Dao and M. Genton. A Monte Carlo adjusted goodness-of-fit test for parametric models describing spatial point patterns. *Journal of Graphical and Computational Statistics*, 23:497–517, 2014.
- [115] A. Dasgupta and A. E. Raftery. Detecting features in spatial point processes with clutter via model-based clustering. *Journal of the American Statistical Association*, 93:294–302, 1998.
- [116] C.B. Dean and R. Balshaw. Efficiency lost by analyzing counts rather than exact times in Poisson and overdispersed Poisson regression. *Journal of the American Statistical Association*, 92:1387–1398, 1997.
- [117] C.B. Dean and J.F. Lawless. Tests for detecting overdispersion in Poisson regression models. *Journal of the American Statistical Association*, 84(406):467–472, 1989.
- [118] C. Dematteï, Molinari N., and J.P. Daurès. Arbitrarily shaped multiple spatial cluster detection for case event data. *Computational Statistics and Data Analysis*, 51:3931–3945, 2007.
- [119] D. Dereudre and F. Lavancier. Practical simulation and estimation for Gibbs Delaunay-Voronoi tessellations with geometric hardcore interaction. *Computational Statistics and Data Analysis*, 55:498–519, 2011.
- [120] P. J. Diggle, J. Besag, and J. T. Gleaves. Statistical analysis of spatial point patterns by means of distance methods. *Biometrics*, 32:659–667, 1976.
- [121] P.J. Diggle. On parameter estimation and goodness-of-fit testing for spatial point patterns. *Biometrika*, 35:87–101, 1979.
- [122] P.J. Diggle. Binary mosaics and the spatial pattern of heather. *Biometrics*, 37:531–539, 1981.
- [123] P.J. Diggle. *Statistical Analysis of Spatial Point Patterns*. Academic Press, London, 1983.
- [124] P.J. Diggle. A kernel method for smoothing point process data. *Journal of the Royal Statistical Society, series C (Applied Statistics)*, 34:138–147, 1985.

- [125] P.J. Diggle. Displaced amacrine cells in the retina of a rabbit: analysis of a bivariate spatial point pattern. *Journal of Neuroscience Methods*, 18:115–125, 1986.
- [126] 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.
- [127] P.J. Diggle. *Statistical Analysis of Spatial Point Patterns*. Hodder Arnold, London, second edition, 2003.
- [128] P.J. Diggle and B. Matérn. On sampling designs for the estimation of point-event nearest neighbour distributions. *Scandinavian Journal of Statistics*, 7:80–84, 1981.
- [129] 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.
- [130] A.J. Dobson and A.G. Barnett. *An introduction to generalized linear models*. CRC Press, third edition, 2008.
- [131] S.I. Doguwa. A comparative study of the edge-corrected kernel-based nearest neighbour density estimators for point processes. *Journal of Statistical Computation and Simulation*, 33:83–100, 1989.
- [132] S.I. Doguwa. On edge-corrected kernel-based pair correlation function estimators for point processes. *Biometrical Journal*, 32:95–106, 1990.
- [133] S.I. Doguwa and D.N. Choji. On edge-corrected probability density function estimators for point processes. *Biometrical Journal*, 33:623–637, 1991.
- [134] K. Donnelly. Simulations to determine the variance and edge-effect of total nearest neighbour distance. In I. Hodder, editor, *Simulation studies in archaeology*, pages 91–95. Cambridge University Press, Cambridge; New York, 1978.
- [135] M. Dudík, S. J. Phillips, and R.E. Schapire. Maximum entropy density estimation with generalized regularization and an application to species distribution modeling. *Journal of Machine Learning Research*, 8:1217–1260, 2007.
- [136] M. Dutta. On maximum (information-theoretic) entropy estimation. *Sankhya: The Indian Journal of Statistics, Series A*, 28:319–328, 1966.
- [137] M. Dwass. Modified randomization tests for nonparametric hypotheses. *Annals of Mathematical Statistics*, 28:181–187, 1957.
- [138] B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap*, volume 57 of *Monographs on Statistics and Applied Probability*. Chapman and Hall, London, 1993.
- [139] H. El Barmi and J.S. Simonoff. Transformation based density estimation for weighted distributions. *Journal of Nonparametric Statistics*, 12:861–878, 2000.
- [140] J. Elith, Steven J. Phillips, T. Hastie, M. Dudík, Yung E. Chee, and Colin J. Yates. A statistical explanation of MaxEnt for ecologists. *Diversity and Distributions*, 17:43–57, 2011.
- [141] P. Elliott, J. Wakefield, N. Best, and D. Briggs, editors. *Spatial Epidemiology: Methods and Applications*. Oxford University Press, Oxford, 2000.

- [142] ESRI. Esri shapefile technical description. Esri white paper, Environmental Systems Research Institute, Inc, July 1998. URL: www.esri.com/library/whitepapers/pdfs/shapefile.pdf.
- [143] M. Ezekiel. A method for handling curvilinear correlation for any number of variables. *Journal of the American Statistical Association*, 19:431–453, 1924.
- [144] T. Fiksel. Edge-corrected density estimators for point processes. *Statistics*, 19:67–75, 1988.
- [145] R.A. Fisher. Frequency distribution of the values of the correlation coefficient in samples of an indefinitely large population. *Biometrika*, 10:507–521, 1915.
- [146] R.A. Fisher. *Statistical Methods and Scientific Inference*. Oliver and Boyd, Edinburgh, second edition, 1959.
- [147] J. Fox. *Applied regression, linear models and related methods*. Sage, 1997.
- [148] R. Foxall and A. Baddeley. Nonparametric measures of association between a spatial point process and a random set, with geological applications. *Applied Statistics*, 51(2):165–182, 2002.
- [149] E.L. Frome. The analysis of rates using Poisson regression models. *Biometrics*, 39:665–674, 1983.
- [150] N. Fry. Random point distributions and strain measurement in rocks. *Tectonophysics*, 60:89–105, 1979.
- [151] N. Funwi-Gabga. A pastoralist survey and fire impact assessment in the kagwene gorilla sanctuary, cameroon. Master's thesis, Geology and Environmental Science, University of Buea, Cameroon, 2008.
- [152] N. Funwi-Gabga and J. Mateu. Understanding the nesting spatial behaviour of gorillas in the kagwene sanctuary, cameroon. *Stochastic Environmental Research and Risk Assessment*, 26(6):793–811, 2012.
- [153] R. Gangnon and M. Clayton. A weighted average likelihood ratio test for spatial clustering of disease. *Statistics in Medicine*, 20:2977–2987, 2001.
- [154] A.E. Gelfand, P.J. Diggle, M. Fuentes, and P. Guttorp, editors. *Handbook of Spatial Statistics*. CRC Press, Boca Raton, Fl., 2010.
- [155] C.J. Geyer. Likelihood inference for spatial point processes. In O.E. Barndorff-Nielsen, W.S. Kendall, and M.N.M. van Lieshout, editors, *Stochastic Geometry: Likelihood and Computation*, number 80 in Monographs on Statistics and Applied Probability, chapter 3, pages 79–140. Chapman and Hall / CRC, Boca Raton, Florida, 1999.
- [156] R.D. Gill. Lectures on survival analysis. In P. Bernard, editor, *22e Ecole d'Eté de Probabilités de Saint-Flour 1992*, number 1581 in Lecture Notes in Mathematics. Springer, 1994.
- [157] R. Gnanadesikan and M.B. Wilk. A probability plotting procedure for general analysis of variance. *Journal of the Royal Statistical Society, series B*, 32:88–101, 1970.
- [158] P.V. Gorsevski, P.E. Gessler, R.B. Folz, and W.J. Elliott. Spatial prediction of landslide hazard using logistic regression and ROC analysis. *Transactions in GIS*, 10:395–415, 2006.
- [159] C.A. Gotway and L.J. Young. Combining incompatible spatial data. *Journal of the American Statistical Association*, 97:632–648, 2002.

- [160] Gnu general public license, version 2. URL <http://www.gnu.org/licenses/old-licenses/gpl-2.0.html>, 2007. Last retrieved 2014-02-09.
- [161] P. Grabarnik, M. Myllymäki, and D. Stoyan. Correct testing of mark independence for marked point patterns. *Ecological Modelling*, 222:3888–3894, 2011.
- [162] P. Greig-Smith. Pattern in vegetation. *Journal of Ecology*, 67:755–779, 1979.
- [163] D.I. Groves, R.J. Goldfarb, C.M. Knox-Robinson, J. Ojala, S. Gardoll, G.Y. Yun, and P. Holyland. Late-kinematic timing of orogenic gold deposits and significance for computer-based exploration techniques with emphasis on the Yilgarn Block, Western Australia. *Ore Geology Reviews*, 17:1–38, 2000.
- [164] Y. Guan. On consistent nonparametric intensity estimation for inhomogeneous spatial point processes. *Journal of the American Statistical Association*, 103:1238–1247, 2008.
- [165] D. Guo. Regionalization with dynamically constrained agglomerative clustering and partitioning (REDCAP). *International Journal of Geographical Information Science*, 22(7):801–823, 2008.
- [166] D. Guo and H. Wang. Automatic region building for spatial analysis. *Transactions in GIS*, 15:29–45, 2011.
- [167] U. Hahn. *Global and Local Scaling in the Statistics of Spatial Point Processes*. Habilitationsschrift, Universitaet Augsburg, 2007.
- [168] U. Hahn, E.B.V. Jensen, M.-C. van Lieshout, and L.S. Nielsen. Inhomogeneous spatial point processes by location-dependent scaling. *Advances in Applied Probability (SGSA)*, 35:319–336, 2003.
- [169] P. Hall. Resampling a coverage pattern. *Stochastic Processes and their Applications*, 20:231–246, 1985.
- [170] P. Hall. *An Introduction to the Theory of Coverage Processes*. John Wiley and Sons, New York, 1988.
- [171] M.S. Handcock and M. Morris. *Relative distribution methods in the social sciences*. Springer-Verlag, New York, 1999.
- [172] K.-H. Hanisch. On Palm and second-order quantities of point processes and germ-grain models. *Wissenschaftliche Sitzung WSS-01/84*, Akademie der Wissenschaften der DDR, Berlin, 1984.
- [173] K.-H. Hanisch. Some remarks on estimators of the distribution function of nearest neighbour distance in stationary spatial point patterns. *Mathematische Operationsforschung und Statistik, series Statistics*, 15:409–412, 1984.
- [174] S.S. Hanna and N. Fry. A comparison of methods of strain determination in rocks from southwest Dyfed (Pembrokeshire) and adjacent areas. *Journal of Structural Geology*, 1:155–162, 1979.
- [175] M.B. Hansen, A.J. Baddeley, and R.D. Gill. First contact distributions for spatial patterns: regularity and estimation. *Advances in Applied Probability*, 31:15–33, 1999.
- [176] M.B. Hansen, R.D. Gill, and A.J. Baddeley. Kaplan-Meier type estimators for linear contact distributions. *Scandinavian Journal of Statistics*, 23:129–155, 1996.

- [177] R.D. Harkness and V. Isham. A bivariate spatial point pattern of ants' nests. *Applied Statistics*, 32:293–303, 1983.
- [178] F. Harrell. *Regression Modeling Strategies*. Springer, New York, 2001.
- [179] R.J. Hasenstab. A preliminary cultural resource sensitivity analysis for the proposed flood control facilities construction in the Passaic River basin of New Jersey. Technical report, Soil Systems, Inc, New York, 1983. Submitted to the Passaic River Basin Special Studies Branch, Department of the Army, USA. New York District Army Corps of Engineers.
- [180] T.J. Hastie and R.J. Tibshirani. *Generalized Additive Models*. Chapman and Hall/CRC, 1990.
- [181] W.W. Hauck, Jr. and A. Donner. Wald's test as applied to hypotheses in logit analysis. *J. Amer. Statist. Assoc.*, 72(360, part 1):851–853, 1977.
- [182] Y. Hochberg and A. Tamhane. *Multiple Comparison Procedures*. Wiley, New York, 1987.
- [183] A.C.A. Hope. A simplified Monte Carlo significance test procedure. *Journal of the Royal Statistical Society, series B*, 30:582–598, 1968.
- [184] B. Hopkins. A new method of determining the type of distribution of plant individuals. *Annals of Botany*, 18:213–227, 1954. With an appendix by J.G. Skellam.
- [185] J.L. Horowitz. The bootstrap. In J.J. Heckman and E. Leamer, editors, *Handbook of Econometrics*, volume 5, pages 3159–3228. North-Holland, Amsterdam, 2001.
- [186] D.G. Horvitz and D.J. Thompson. A generalization of sampling without replacement from a finite universe. *Journal of the American Statistical Association*, 47:663–685, 1952.
- [187] D.W. Hosmer and S. Lemeshow. *Applied logistic regression*. Wiley, second edition, 2000.
- [188] J.C. Hsu. *Multiple comparisons: Theory and Methods*. Chapman and Hall, 1996.
- [189] S.P. Hubbell and R.B. Foster. Diversity of canopy trees in a neotropical forest and implications for conservation. In S.L. Sutton, T.C. Whitmore, and A.C. Chadwick, editors, *Tropical Rain Forest: Ecology and Management*, pages 25–41. Blackwell Scientific Publications, Oxford, 1983.
- [190] D. Hug and G. Last. On support measures in Minkowski spaces and contact distributions in stochastic geometry. *Annals of Probability*, pages 796–850, 2000.
- [191] D. Hug, G. Last, and W. Weil. A local Steiner-type formula for general closed sets and applications. *Mathematische Zeitschrift*, 246(1-2):237–272, 2004.
- [192] Ross Ihaka and Robert Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3):299–314, 1996.
- [193] J. Illian, A. Penttinen, H. Stoyan, and D. Stoyan. *Statistical Analysis and Modelling of Spatial Point Patterns*. John Wiley and Sons, Chichester, 2008.
- [194] E. Jolivet. Central limit theorem and convergence of empirical processes for stationary point processes. In P. Basteia and J. Tomko, editors, *Point processes and queueing problems*, pages 117–161. North-Holland, Amsterdam, 1980.
- [195] M.C. Jones. Kernel density estimation for length-biased data. *Biometrika*, 78:511–519, 1991.
- [196] F.P. Kelly and B.D. Ripley. A note on Strauss's model for clustering. *Biometrika*, 63:357–360, 1976.

- [197] N.C. Kenkel. Pattern of self-thinning in jack pine: testing the random mortality hypothesis. *Ecology*, 69:1017–1024, 1988.
- [198] M. Kerscher. Regularity in the distribution of superclusters? *Astronomy and Astrophysics*, 336:29–34, 1998.
- [199] M. Kerscher, M. J. Pons-Borderia, J. Schmalzing, R. Trasarti-Battistoni, T. Buchert, V. J. Martinez, and R. Valdarnini. A global descriptor of spatial pattern interaction in the galaxy distribution. *Astrophysical Journal*, 513:543–548, 1999.
- [200] M. Kerscher, J. Schmalzing, T. Buchert, and H. Wagner. Fluctuations in the IRAS 1.2 Jy catalogue. *Astronomy and Astrophysics*, 333:1–12, 1998.
- [201] A. Ya. Khinchin. *Mathematical Methods in the Theory of Queueing*. Griffin, London, 1960. English translation. Originally published in Russian in *Trudy Mat. Inst. Steklov*, 1955.
- [202] S.S. Kind. *The scientific investigation of crime*. Forensic Science Services, Harrogate, UK, 1987.
- [203] C.M. Knox-Robinson and D.I. Groves. Gold prospectivity mapping using a geographic information system (GIS), with examples from the Yilgarn Block of Western Australia. *Chronique de la Recherche Minière*, 529:127–138, 1997.
- [204] A. Kolmogorov. Sulla determinazione empirica di una legge di distribuzione. *Giornale dell'Istituto Italiano degli Attuari*, 4:83–91, 1933.
- [205] J. Kornak, M.E. Irwin, and N. Cressie. Spatial point process models of defensive strategies: detecting changes. *Statistical Inference for Stochastic Processes*, 9:31–46, 2006.
- [206] K. Krickeberg. Processus ponctuels en statistique. In P.L. Hennequin, editor, *Ecole d'Eté de Probabilités de Saint-Flour X*, volume 929 of *Lecture Notes in Mathematics*, pages 205–313. Springer, 1982.
- [207] M. Kulldorff. Spatial scan statistics: models, calculations, and applications. In J. Glaz and N. Balakrishnan, editors, *Recent Advances on Scan Statistics*, pages 303–322. Birkhauser, Boston, 1999.
- [208] M. Kulldorff and N. Nagarwalla. Spatial disease clusters: detection and inference. *Statistics in Medicine*, 14:799–810, 1995.
- [209] Y.A. Kutoyants. *Statistical Inference for Spatial Poisson Processes*. Number 134 in Lecture Notes in Statistics. Springer, New York, 1998.
- [210] K.L. Kvamme. Computer processing techniques for regional modeling of archaeological site locations. *Advances in Computer Archaeology*, 1:26–52, 1983.
- [211] K.L. Kvamme. A view from across the water: the North American experience in archeological GIS. In G.R. Lock and Z. Stančič, editors, *Archaeology and Geographical Information Systems: a European Perspective*, pages 1–14. CRC Press, 1995.
- [212] K.L. Kvamme. There and back again: revisiting archeological locational modeling. In M.W. Mehrer and K.L. Wescott, editors, *GIS and archaeological site modelling*, pages 3–40. CRC Press, 2006.
- [213] J.M. Landwehr, D. Pregibon, and A.C. Shoemaker. Graphical methods for assessing logistic regression models. *Journal of the American Statistical Association*, 79:61–83, 1984.

- [214] W.A. Larsen and S.J. McCleary. The use of partial residual plots in regression analysis. *Technometrics*, 14:781–790, 1972.
- [215] N.A. Laskurain. *Dinámica espacio-temporal de un bosque secundario en el Parque Natural de Urkiola (Bizkaia)*. PhD thesis, Universidad del País Vasco /Euskal Herriko Unibertsitatea, 2008.
- [216] G. Last and M. Holtmann. On the empty space function of some germ-grain models. *Pattern Recognition*, 32(9):1587–1600, 1999.
- [217] G. Last and R Schassberger. On the distribution of the spherical contact vector of stationary germ-grain models. *Advances in Applied Probability*, pages 36–52, 1998.
- [218] G. Last and R. Szekli. Comparisons and asymptotics for empty space hazard functions of germ-grain models. *Advances in Applied Probability*, 43(4):943–962, 2011.
- [219] R. Law, J. Illian, D.F.R.P. Burslem, G. Gratzer, C.V.S. Gunatilleke, and I.A.U.N. Gunatilleke. Ecological information from spatial patterns of plants: insights from point process theory. *Journal of Ecology*, 97:616–628, 2009.
- [220] A.B. Lawson. On the analysis of mortality events around a prespecified fixed point. *Journal of the Royal Statistical Society, series A*, 156(3):363–377, 1993.
- [221] A.B. Lawson and D.G.T. Denison, editors. *Spatial cluster modelling 4*. Chapman and Hall/CRC Press, Boca Raton, 2002. ISBN 1-58488-266-2.
- [222] E. L. Lehmann and Joseph Romano. *Testing Statistical Hypotheses*. Springer, New York, 3rd edition, 2005.
- [223] E.L. Lehmann. *Theory of point estimation*. John Wiley and Sons, New York, 1983.
- [224] 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.
- [225] S. Liang, S. Banerjee, and B. Carlin. Bayesian wombling for spatial point processes. *Biometrics*, 65:1243–1253, 2009.
- [226] J.K. Lindsey. *The analysis of stochastic processes using GLIM*. Springer, Berlin, 1992.
- [227] J.K. Lindsey. *Modelling frequency and count data*. Oxford University Press, 1995.
- [228] J.K. Lindsey. *Applying generalized linear models*. Springer, 1997.
- [229] J.K. Lindsey and G. Mersch. Fitting and comparing probability distributions with log linear models. *Computational Statistics and Data Analysis*, 13:373–384, 1992.
- [230] C. Loader. *Local Regression and Likelihood*. Springer, New York, 1999.
- [231] C.R. Loader. Large-deviation approximations to the distribution of scan statistics. *Advances in Applied Probability*, 23:751–771, 1991.
- [232] J.M. Loh. A fast and valid spatial bootstrap for correlation functions. *Astrophysical Journal*, 681:726–734, 2008.
- [233] N.B. Loosmore and E.D. Ford. Statistical inference using the G or K point pattern spatial statistics. *Ecology*, 87:1925–1931, 2006.

- [234] H. W. Lotwick and B. W. Silverman. Methods for analysing spatial processes of several types of points. *Journal of the Royal Statistical Society, series B*, 44:406–413, 1982.
- [235] B.J.F. Manly, L.L. McDonald, and D.L. Thomas. *Resource selection by animals: statistical design and analysis for field studies*. Chapman and Hall, London, 1993.
- [236] A.F. Mark and A.E. Esler. An assessment of the point-centred quarter method of plotless sampling in some New Zealand forests. *Proceedings of the New Zealand Ecological Society*, 17:106–110, 1970.
- [237] F.H.C Marriott. Barnard's Monte Carlo tests: how many simulations? *Applied Statistics*, 28:75–77, 1979.
- [238] G. Marsaglia and J. Marsaglia. Evaluating the Anderson-Darling distribution. *Journal of Statistical Software*, 9(2):1–5, 2004. URL: <http://www.jstatsoft.org/v09/i02>.
- [239] V. Martinez and E. Saar. *Statistics of the galaxy distribution*. Chapman and Hall/CRC, 2002.
- [240] B. Matérn. Spatial Variation: stochastic models and their application to some problems in forest surveys and other sampling investigations. *Meddelanden från Statens Skogsforskningsinstitut*, 49(5):1–144, 1960.
- [241] B. Matérn. Doubly stochastic Poisson processes in the plane. In G.P. Patil, E.C. Pielou, and W. E. Waters, editors, *Statistical ecology, Volume 1: Spatial patterns and statistical distributions*, pages 195–213. Pennsylvania State University Press, University Park, 1971.
- [242] G. Matheron. *Random Sets and Integral Geometry*. John Wiley and Sons, New York, 1975.
- [243] P. McCullagh and J.A. Nelder. *Generalized Linear Models*. Chapman and Hall, second edition, 1989.
- [244] G. McSwiggan. Personal communication, 2013.
- [245] J. Mecke, R.G. Schneider, D. Stoyan, and W.R.R. Weil. *Stochastische Geometrie*. DMV Seminar Band 16. Birkhäuser, Basel, 1990.
- [246] R.E. Miles. On the homogeneous planar Poisson point process. *Mathematical Biosciences*, 6:85–127, 1970.
- [247] R.E. Miles. The fundamental formula of Blaschke in integral geometry and its iteration, for domains with fixed orientation. *Australian Journal of Statistics*, 16(2):111–118, 1974.
- [248] R.E. Miles. A synopsis of ‘Poisson flats in Euclidean spaces’. In E.F. Harding and D.G. Kendall, editors, *Stochastic Geometry*, pages 202–227. John Wiley and Sons, 1974.
- [249] Alan J. Miller. *Subset Selection in Regression*. Chapman and Hall, London, 1990.
- [250] J. Møller and R.P. Waagepetersen. *Statistical Inference and Simulation for Spatial Point Processes*. Chapman and Hall/CRC, Boca Raton, 2004.
- [251] Jesper Møller and Sergei Zuyev. Gamma-type results and other related properties of Poisson processes. *Adv. in Appl. Probab.*, 28(3):662–673, 1996.
- [252] M. Morisita. Measuring of the dispersion of individuals and analysis of the distributional patterns. Memoirs of the faculty of science, E 2, Kyushu Univ. Ser., Kyushu University, 1959.

- [253] E.A. Nadaraya. On estimating regression. *Theory of Probability and its Applications*, 9:141–142, 1964.
- [254] E.A. Nadaraya. *Nonparametric estimation of probability densities and regression curves*. Kluwer, Dordrecht, 1989.
- [255] Jerzy Neyman and Egon S. Pearson. On the problem of the most efficient tests of statistical hypotheses. *Philos. Trans. R. Soc. Lond. Ser. A. Math. Phys. Eng. Sci.*, 231:289 – 337, 1933.
- [256] M. Numata. Forest vegetation, particularly pine stands in the vicinity of Choshi — flora and vegetation in Choshi, Chiba prefecture, VI (in Japanese). *Bulletin of the Choshi Marine Laboratory*, (6):27–37, 1964. Chiba University.
- [257] Y. Ogata. Statistical models for earthquake occurrences and residual analysis for point processes. *Journal of the American Statistical Association*, 83:9–27, 1988.
- [258] Y. Ogata and M. Tanemura. Estimation of interaction potentials of spatial point patterns through the maximum likelihood procedure. *Annals of the Institute of Statistical Mathematics*, B 33:315–338, 1981.
- [259] Y. Ogata and M. Tanemura. Likelihood estimation of interaction potentials and external fields of inhomogeneous spatial point patterns. In I.S. Francis, B.J.F. Manly, and F.C. Lam, editors, *Pacific Statistical Congress*, pages 150–154, Amsterdam, 1986. Elsevier.
- [260] J. Ohser. On estimators for the reduced second moment measure of point processes. *Mathematische Operationsforschung und Statistik, series Statistics*, 14:63–71, 1983.
- [261] J. Ohser and D. Stoyan. On the second-order and orientation analysis of planar stationary point processes. *Biometrical Journal*, 23:523–533, 1981.
- [262] A. Okabe, B. Boots, and K. Sugihara. *Spatial tessellations: concepts and applications of Voronoi diagrams*. Wiley series in probability and mathematical statistics. John Wiley and Sons, Chichester, England; New York, 1992.
- [263] S. Openshaw. *The modifiable area unit problem*. Geo Books, Norwich, 1984.
- [264] L.S. Ornstein and F. Zernike. Accidental deviations of density and opalescence at the critical point of a single substance. *Proceedings of the Section of Sciences, Royal Academy of Sciences/ Koninklijke Akademie van Wetenschappen, Amsterdam*, 17:793–806, 1914.
- [265] C. Palm. Intensitätsschwankungen im Fernsprechverkehr. *Ericsson Technics*, 44:1–189, 1943.
- [266] A.L. Patterson. A Fourier series method for the determination of the component of interatomic distances in crystals. *Physics Reviews*, 46:372–376, 1934.
- [267] A.L. Patterson. A direct method for the determination of the components of inter-atomic distances in crystals. *Zeitschrift fuer Krystallographie*, 90:517–554, 1935.
- [268] A. Penttinen. *Modelling Interaction in Spatial Point Patterns: Parameter Estimation by the Maximum Likelihood Method*. Number 7 in Jyväskylä Studies in Computer Science, Economics and Statistics. University of Jyväskylä, 1984.
- [269] H. Persson. Root dynamics in a young Scots pine stand in Central Sweden. *Oikos*, 30:508–519, 1978.

- [270] S. J. Phillips, R. P. Anderson, and R. E. Schapire. Maximum entropy modeling of species geographic distributions. *Ecological Modelling*, 190:231–259, 2006.
- [271] W. J. Platt, G. W. Evans, and S. L. Rathbun. The population dynamics of a long-lived Conifer (*Pinus palustris*). *The American Naturalist*, 131:491–525, 1988.
- [272] D. Pregibon. Logistic regression diagnostics. *Annals of Statistics*, 9:705–724, 1981.
- [273] M. Prokešová, U. Hahn, and E.B. Vedel Jensen. Statistics for locally scaled point processes. In A. Baddeley, P. Gregori, J. Mateu, R. Stoica, and D. Stoyan, editors, *Case studies in spatial point process modeling*, number 185 in Lecture Notes in Statistics, pages 99–123. Springer, New York, 2005.
- [274] M. A. Proschan and B. Presnell. Expect the unexpected from conditional expectation. *The American Statistician*, 52(3):248–252, 1998.
- [275] Somik Raha. A critique of statistical hypothesis testing in clinical research. *J. Ayurveda Integr. Med.*, 2(3):105–114, 2011.
- [276] C.R. Rao. Large sample tests of statistical hypotheses concerning several parameters with applications to problems of estimation. *Proceedings of the Cambridge Philosophical Society*, 44:50–57, 1948.
- [277] S. L. Rathbun and N. Cressie. A space-time survival point process for a longleaf pine forest in southern Georgia. *Journal of the American Statistical Association*, 89:1164–1173, 1994.
- [278] S.L. Rathbun and N. Cressie. Asymptotic properties of estimators of the parameters of spatial inhomogeneous Poisson point processes. *Advances in Applied Probability*, 26:122–154, 1994.
- [279] T.R.C. Read and N.A.C. Cressie. *Goodness-of-fit statistics for multivariate data*. Springer-Verlag, New York, 1988.
- [280] I.W. Renner and D.I. Warton. Equivalence of MAXENT and Poisson point process models for species distribution modeling in ecology. *Biometrics*, 69:274–281, 2013.
- [281] B.D. Ripley. Modelling spatial patterns (with discussion). *Journal of the Royal Statistical Society, series B*, 39:172–212, 1977.
- [282] B.D. Ripley. Tests of ‘randomness’ for spatial point patterns. *Journal of the Royal Statistical Society, series B*, 41:368–374, 1979.
- [283] B.D. Ripley. *Spatial Statistics*. John Wiley and Sons, New York, 1981.
- [284] B.D. Ripley. *Statistical Inference for Spatial Processes*. Cambridge University Press, 1988.
- [285] B.D. Ripley and J.-P. Rasson. Finding the edge of a Poisson forest. *Journal of Applied Probability*, 14:483–491, 1977.
- [286] J.M. Robins, A. van der Vaart, and V. Ventura. Asymptotic distribution of p values in composite null models. *Journal of the American Statistical Association*, 95(452):1143–1156, December 2000.
- [287] W.S. Robinson. Ecological correlations and the behavior of individuals. *American Sociological Review*, 15:351–357, 1950.
- [288] R.T. Rockafellar. *Convex Analysis*. Princeton University Press, Princeton, NJ, 1972.

- [289] A. Rosenfeld and J. L. Pfalz. Sequential operations in digital picture processing. *Journal of the Association for Computing Machinery*, 13:471, 1966.
- [290] A. Rosenfeld and J. L. Pfalz. Distance functions on digital pictures. *Pattern Recognition*, 1:33–61, 1968.
- [291] 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.
- [292] K. Schladitz. Surprising optimal estimators for the area fraction. *Advances in Applied Probability*, 31:995–1001, 1999.
- [293] M. Schlather, P. Riberio, and P.J. Diggle. Detecting dependence between marks and locations of marked point processes. *Journal of Royal Statistical Society Series B*, 66:79–93, 2004.
- [294] S.C. Scholtz. Location choice models in Sparta. In R. Lafferty III, J.L. Ottinger, S.C. Scholtz, W.F. Limp, B. Watkins, and R. D. Jones, editors, *Settlement Predictions in Sparta: A Locational Analysis and Cultural Resource Assessment on the Uplands of Calhoun County, Arkansas*, number 14 in Arkansas Archaeological Survey Research Series, pages 207–222. Arkansas Archaeological Survey, Fayetteville, Arkansas, USA, 1981.
- [295] A.J. Scott and M.J. Symons. Clustering methods based on likelihood ratio criteria. *Biometrics*, 27:387–397, 1971.
- [296] D.W. Scott. *Multivariate Density Estimation. Theory, Practice and Visualization*. Wiley, New York, 1992.
- [297] J. Serra. *Image analysis and mathematical morphology*. Academic Press, London, 1982.
- [298] J. P. Shaffer. Multiple hypothesis testing. *Annual Review of Psychology*, 46:561 – 584, 1995.
- [299] M. Silvapulle. On the existence and uniqueness of the maximum likelihood estimates for the binomial response models. *Journal of the Royal Statistical Society, series B*, 43:310–313, 1981.
- [300] B.W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, London, 1986.
- [301] J.G. Skellam. Appendix to article by hopkins (1954). *Annals of Botany*, 18:226–227, 1954.
- [302] N. Smirnov. Table for estimating the goodness of fit of empirical distributions. *Annals of Mathematical Statistics*, 19:279–281, 1948.
- [303] G.K. Smyth. *Pearson's goodness of fit statistic as a score test statistic*, volume Volume 40 of *Lecture Notes–Monograph Series*, pages 115–126. Institute of Mathematical Statistics, Beachwood, Ohio, USA, 2003.
- [304] J. Snow. *On the Mode of Communication of Cholera*. John Churchill, New Burlington Street, London, England, 1855.
- [305] P. Soille. *Morphological image analysis: principles and applications*. Springer, New York, 2003.
- [306] A. Stein, M. N. M. van Lieshout, and H. W. G. Bootink. Spatial interaction of methylene blue stained soil pores. *Geoderma*, 102:101–121, 2001.

- [307] M.L. Stein. A new class of estimators for the reduced second moment measure of point processes. *Biometrika*, 78:281–286, 1991.
- [308] M.A. Stephens. Tests based on EDF statistics. In R.B. D'Agostino and M.A. Stephens, editors, *Goodness-of-Fit Techniques*, volume 68 of *Statistics: textbooks and monographs*, pages 97–193. Marcel Dekker, New York, 1986.
- [309] D. Stoyan. Thinnings of point processes and their use in the statistical analysis of a settlement pattern with deserted villages. *Statistics*, pages 45–56, 1988.
- [310] D. Stoyan and P. Grabarnik. Second-order characteristics for stochastic structures connected with Gibbs point processes. *Mathematische Nachrichten*, 151:95–100, 1991.
- [311] D. Stoyan, W.S. Kendall, and J. Mecke. *Stochastic Geometry and its Applications*. John Wiley and Sons, Chichester, 1987.
- [312] D. Stoyan, W.S. Kendall, and J. Mecke. *Stochastic Geometry and its Applications*. John Wiley and Sons, Chichester, second edition, 1995.
- [313] D. Stoyan and H.-D. Schnabel. Description of relations between spatial variability of microstructure and mechanical strength of alumina ceramics. *Ceramics International*, 16:11–18, 1990.
- [314] D. Stoyan and H. Stoyan. *Fractals, Random Shapes and Point Fields*. John Wiley and Sons, Chichester, 1995.
- [315] L. Strand. A model for stand growth. In *IUFRO Third Conference Advisory Group of Forest Statisticians*, pages 207–216, Paris, 1972. INRA, Institut National de la Recherche Agronomique.
- [316] D.J. Strauss. A model for clustering. *Biometrika*, 63:467–475, 1975.
- [317] D.R. Strong, Jr. Null hypotheses in ecology. *Synthese*, 43:271–285, 1980.
- [318] M. Thomas. A generalisation of Poisson's binomial limit for use in ecology. *Biometrika*, 36:18–25, 1949.
- [319] J.W. Tukey. Discussion of paper by F.P. Agterberg and S.C. Robinson. *Bulletin of the International Statistical Institute*, 44(1):596, 1972. Proceedings, 38th Congress, International Statistical Institute.
- [320] M.N.M. van Lieshout. *Markov Point Processes and their Applications*. Imperial College Press, London, 2000.
- [321] M.N.M. van Lieshout. A J-function for inhomogeneous point patterns. *Statistica Neerlandica*, 65:183–201, 2011.
- [322] M.N.M. van Lieshout and A.J. Baddeley. A nonparametric measure of spatial interaction in point patterns. *Statistica Neerlandica*, 50:344–361, 1996.
- [323] Ashlee Vance. Data analysts are mesmerised by the power of program R. *The New York Times*, page B6, 7 January 2009. New York Edition.
- [324] W.N. Venables and B.D. Ripley. *Modern Applied Statistics with S-Plus*. Springer, 1994.
- [325] W.N. Venables and B.D. Ripley. *Modern Applied Statistics with S-Plus*. Springer, fourth edition, 2002.

- [326] R. von Mises. *Wahrscheinlichkeitsrechnung und Ihre Anwendung in der Statistik und Theoretischen Physik*. Deuticke, Leipzig, 1931.
- [327] C.G. Wager, B.A. Coull, and N. Lange. Modelling spatial intensity for replicated inhomogeneous point patterns in brain imaging. *Journal of the Royal Statistical Society, Series B*, 66:429–446, 2004.
- [328] J. Wakefield. A critique of statistical aspects of ecological studies in spatial epidemiology. *Environmental and Ecological Statistics*, 11:31–54, 2004.
- [329] J. Wakefield. Disease mapping and spatial regression with count data. *Biostatistics*, 8:158–183, 2007.
- [330] L. Waller, B. Turnbull, L.C. Clark, and P. Nasca. Chronic disease surveillance and testing of clustering of disease and exposure: Application to leukaemia incidence and TCE-contaminated dumpsites in upstate New York. *Environmetrics*, 3:281–300, 1992.
- [331] L.A. Waller and C.A. Gotway. *Applied spatial statistics for public health data*. Wiley, 2004.
- [332] P.C. Wang. Adding a variable in generalized linear models. *Technometrics*, 27:273–276, 1985.
- [333] P.C. Wang. Residual plots for detecting nonlinearity in generalized linear models. *Technometrics*, 29:435–438, 1987.
- [334] D.I. Warton and L.C. Shepherd. Poisson point process models solve the “pseudo-absence problem” for presence-only data in ecology. *Annals of Applied Statistics*, 4:1383–1402, 2010.
- [335] K.P. Watkins and A.H. Hickman. Geological evolution and mineralization of the Murchison Province, Western Australia. Bulletin 137, Geological Survey of Western Australia, 1990. Published by Department of Mines, Western Australia, 1990. Available online from Department of Industry and Resources, State Government of Western Australia, www.doir.wa.gov.au.
- [336] G.S. Watson. Smooth regression analysis. *Sankhya A*, 26:359–372, 1964.
- [337] R.W.M. Wedderburn. On the existence and uniqueness of maximum likelihood estimates for certain generalized linear models. *Biometrika*, 63:27–32, 1976.
- [338] D. Wheatley and M. Gillings. *Spatial technology and archaeology: the archaeological applications of GIS*. Taylor and Francis, 2002.
- [339] M.B. Wilk and R. Gnanadesikan. Probability plotting methods for the analysis of data. *Biometrika*, 55:1–17, 1968.
- [340] G. N. Wilkinson and C. E. Rogers. Symbolic description of factorial models for analysis of variance. *Applied Statistics*, 22:392–399, 1973.
- [341] D.A. Williams. Generalised linear model diagnostics using the deviance and single case deletions. *Applied Statistics*, 36:181–191, 1987.
- [342] H. Wold. On stationary point processes and Markov chains. *Skandinavisk Aktuarieridskrift*, 31:229–240, 1948.
- [343] H. Wold. Sur les processus stationnaires ponctuels. In *Le Calcul des Probabilités et ses Applications*, number 13 in Colloques Internationaux CNRS, pages 75–86. Centre National de la Recherche Scientifique, Paris, 1949.

- [344] J.H. Wolfe. Pattern clustering by multivariate mixture analysis. *Multivariate Behavioral Research*, 5:329–350, 1970.
- [345] F. Yates. The influence of ‘Statistical Methods for Research Workers’ on the development of the science of statistics. *Journal of the American Statistical Association*, 46:19–34, 1951.
- [346] T. Yoshida and T. Hayashi. On the robust estimation in Poisson processes with periodic intensities. *Annals of the Institute of Statistical Mathematics*, 42:489–507, 1990.
- [347] S. Zuyev. Strong Markov property of Poisson processes and Slivnyak formula. In A. Baddeley, P. Gregori, J. Mateu, R. Stoica, and D. Stoyan, editors, *Case studies in spatial point process modeling*, number 185 in Lecture Notes in Statistics, pages 78–84. Springer, New York, 2005.

Index

- adaptive smoothing
 - estimator of intensity, 137
- Amacrine cells data, 5
- analysis of deviance, 300, 330
- Anderson-Darling test, 338
 - of CSR, 146
- anisotropy, 202
 - anisotropic pair correlation function, 206
 - Fry plot, 203
 - pair orientation distribution, 204
 - sector K function, 203
- Ants' nests data, 6
- bandwidth selection
 - for intensity estimation, 134
 - default rule, 133
 - Diggle-Berman, 134
 - likelihood cross-validation, 134
 - Scott's rule of thumb, 134
 - weaknesses, 135
 - matching different kernels, 150
- Beilschmiedia* dataset, 272, 273, 284, 288, 369, 374, 377, 382, 384
- Beilschmiedia* dataset, 9, 138, 139
- Berman tests, 148
- bias-variance tradeoff
 - in kernel estimation, 132
 - in quadrat size, 128
- binary mask, 62
- binary mask windows, 41
- birth-death process, *see* spatial birth-death process
- border correction
 - K function, 180
- `bw.relrisk`, 159
- Campbell's formula, 132
- caveats
 - confidence intervals, 195
 - correlation, 173
 - K -function, 173–177
 - pointwise envelopes, 199
- `cdf.test`, 146
- change of coordinates, 138
 - offset term, 287
- Chicago street crimes dataset, 10
- Chorley-Ribble dataset, 9, 287, 323, 390
- circular windows, 62
- Clark-Evans test, 224
- classes, 27
 - in R, 27
 - in `spatstat`, 40
- cluster
 - disease cluster, 149
- coincident points, 55
- compensator, 399
- complete spatial randomness
 - Anderson-Darling test, 146
 - Berman tests, 148
 - Cramér-Von Mises test, 146
 - formal definition, 266
 - Kolmogorov-Smirnov test, 146
 - quadrat counting test, 129, 145
- confidence intervals
 - caveats, 195
- contrasts, 282
- correlation, 163–218
 - and causation, 174
 - caveats, 164, 173
 - pair correlation, *see* pair correlation function
- covariate
 - determines quadrats, 139
 - spatial, 139
- covariates, 8, 273
 - in ppm, 273
- Cramér-Von Mises test
 - of CSR, 146
- Cramér-von Mises test, 338
- data entry
 - checking, 54
 - GIS formats, 64
 - marks, 51
- data sharpening, 155
- data types, *see* classes

- datasets
 - amacrine*, 5
 - ants*, 6
 - bei*, 9, 138, 139, 272, 273, 284, 288, 369, 374, 377, 382, 384
 - Beilschmiedia*, 9, 138, 139
 - chicago*, 10
 - chorley*, 9, 287, **323**, 390
 - copper*, 9
 - flu*, 5, 10
 - gorillas*, **281**, 290
 - Japanese Pines*
 - 10 metre, 4
 - japanesepines*, 286
 - lansing*, **159**
 - Lansing Woods, **159**
 - longleaf*, 7
 - mucosa*, 3
 - murchison*, 276, **276**, 285, 289, 291, 389
 - nztrees*, 321
 - osteo*, 11
 - redwood*, 149
 - shapley*, 3, 155
 - swedishpines*, 124
 - urkiola*, 4
 - waterstriders*, 4
- DCLF test, 353
- density
 - points per unit area, *see* intensity
- density, 133
- density.ppp, 133
- discretisation of coordinates, 56
- disease cluster, 149
- dispatching, 27
- distance function, 144
- distance map, 144
- distances
 - empty space, 221
 - locally scaled, 213
 - nearest neighbour, 221
 - pairwise, 167, 221
- distfun, 144
- distmap(), 222
- distmap, 144
- duplication, 55
- edge corrections
 - border correction, 180
 - isotropic correction, 181
 - K* function, 177–184
 - rigid motion correction, 184
- translation correction, 183
- edge effects
 - K* function, 178
- effect, 274
- elephant's foot, 235
- elliptical windows, 62
- empty space distances, 221
- empty space function, **228**, 227–234
 - for Poisson process, 229
 - hazard rate, 238–240
 - in spatstat, 230
 - interpretation, 232
 - plots and inference, 234–237
 - P–P plot, 236
 - Q–Q plot, 236
 - variance stabilisation, 234
- envelope
 - global, 350
 - pointwise, 348
- envelope command, 354–360
- envelopes, 197, 347
 - and Monte Carlo tests, 197, 347
 - arguments to summary function, 357
 - based on variance, 359
 - caveats, 199
 - edge correction, 357
 - for any data, 356
 - for any fitted model, 355
 - for any null hypothesis, 200
 - for any simulation procedure, 356
 - for testing CSR, 354
 - global, 200, 358
 - in spatstat, 354
 - of inhomogeneous *K* function, 212
 - of summary functions, 197, 347
 - one-sided, 359
 - pointwise, 197, 358
 - pooling, 360
 - re-computing, 359
- exchangeable, 344
- Fast Fourier Transform, 136
- Fest(), 230
- fitted model
 - goodness-of-fit test, 368
 - intensity, 297
 - interpretation of coefficients, 282
 - methods for, 293
 - prediction, 297
 - residuals, 372
 - simulation of, 302

- Fry plot, **165**
and K function, 167
and pair correlation, 206
for detecting anisotropy, 203
- fv**, 42
fv class, 184–189
convert to function, 188
function values, 188
manipulating, 188
objects, 184
plotting, 185
- gastric mucosa dataset, 3
- Gcom**, 399
- generic, 27
- geometrical transformations, 94
- Gest()**, 230
- GIS formats, 64
- global envelopes, 200
- goftest** package, 338
- goodness-of-fit test, 337
for Poisson models, 368
- Gorillas dataset, **281**, 290
- Gres**, 399
- hazard rate
of empty space function, 238–240
- Hopkins-Skellam test, 225
- hot spot, 149
- hyperframe**, 42, 64
- hypothesis test, *see* test
- im**, 41
- images
exploratory inspection of, 104
returned by a function, 111
- Influenza proteins dataset, **5**, 10
- inhomogeneity, 208
adjusting for, 208
- inhomogeneous K function, 209
- intensity, **121**, 121–162
definition
general, 125–126
homogeneous intensity, 123
intensity function, 125
depending on covariate, 139–145
hypothesis tests, 145–149
- fitted, 297
standard error, 297
variance, 297
- function, 126
- adaptive estimation, 137
- definition, 125
- estimation at data points, 135
- kernel estimator, 131
- homogeneous, 121
definition, 123
unbiased estimation, 124
- hot spots, 149
- inhomogeneous, 122, 125–139
- interpretation, 121
- measure, *see* mean measure
- of multitype point process, 158
- units, 124, 126
- intensity, 124
- intensity.ppp, 124
- intensity.quadratcount, 128
- interaction
exploratory methods, 164
Q–Q plot, 396
- isotropic correction
 K function, 181
- J** function, 241
- Jacobian, **138**, 287
- Japanese Pines dataset, 286
10 metre, 4
- K** function, 167–177
and modelling, 172
application, 170
border correction, 180
edge corrections, 177–184
edge effects, 178
empirical, **168**
in **spatstat**, 173
inference, 171
inhomogeneous, 209
isotropic correction, 181
locally scaled, 213
modifications, 203
of point process, 169
ratio of expectations, 178
rigid motion correction, 184
sector version, 203
transformation, 171
translation correction, 183
- K**-function
assumes homogeneity, 174
caveats, 173–177
not complete characterisation, 174
scale of interaction, 175

- Kcom, 399
- kernel estimation
 - algorithms, 136
 - bandwidth, 133
 - bandwidth selection, 134
 - default rule, 133
 - weaknesses, 135
 - bias, 132
 - chocolate analogy, 131
 - `density.ppp`, 133
 - leave-one-out, 135
 - of intensity, 131
 - statistical critique, 137
- kernel smoothing
 - estimate of relative distribution, 141
 - estimate of ρ , 141
- Kest, 173
- Kolmogorov-Smirnov test, 338
 - of CSR, 146
 - of inhomogeneous Poisson, 370
- kppm, 42
- Kres, 399
- L* function, 173
 - locally scaled, 213
- lansing dataset, 159
- Lansing Woods dataset, 159
- leave-one-out estimator, 135
- Lest, 173
- likelihood ratio test, 330
- linear predictor, 274
- LISA, 155
- listof, 42, 63
- locally scaled K function, 213
- locally scaled L function, 213
- locally scaled point process, 213
- Longleaf Pines data, 7
- look-elsewhere effect, 199
- lurking variable plot, 377
- MAD test, 350
- maptools package, 64
- marks, 7
 - categorical, 51
 - data entry, 51
 - operations on, 92
- maximum likelihood, 302
- mean measure, 126
 - diffuse, 126
 - singular, 126
- methods, 27
- dispatch, 27
- missing values, 54
- model validation, 367
- Monte Carlo test, 197, 340–347
 - basic principle, 340
 - DCLF, 353
 - details, 341
 - exchangeability, 344
 - MAD (maximum absolute deviation), 350
 - pointwise, 197, 348
 - simultaneous, 200
 - using summary function, 347–353
- multiple hypothesis testing, 199
- multitype point pattern, 51
- Murchison dataset, 276, 276, 285, 289, 291, 389
- NA, 54
- nearest neighbour cleaning, 152
- nearest neighbour distances, 221
 - minimum, 235
- nearest neighbour function, 228, 227–234
 - elephant's foot, 235
 - for Poisson process, 230
 - for small distances, 235
 - in spatstat, 230
 - interpretation, 232
 - plots and inference, 234–237
 - P–P plot, 236
 - Q–Q plot, 236
 - variance stabilisation, 234
- New Zealand Trees dataset, 321
- nndist(), 222
- offset, 287
 - baseline model, 287
 - due to change of coordinates, 287
 - in model formula, 287
- offset, 287
- Osteocyte Lacunae dataset, 11
- owin, 41, 59
- p*-value, 333
 - in anova.ppm, 330
- pair correlation function, 190
 - estimation of, 192
 - general, 208
- pair orientation distribution, 204
- pairdist(), 222
- pairwise distances, 167, 221
- Patterson plot, *see* Fry plot
- plot.fv, 185

- point pattern
 - marks, 7
 - multitype, 4
 - needs window, 46
- point pattern correlation
 - inference, 196
- pointwise envelopes, 197
- Poisson models
 - goodness-of-fit test, 368
 - inhomogeneous, 267
 - maximum likelihood, 302
 - residuals, 372
- Poisson point process
 - fine-pixel limit, 268
 - formal definition, 266–268
 - homogeneous, 266
 - inhomogeneous
 - definition, 267
 - modelling scenarios, 269
- polygonal windows, 41, 60
- P–P plot, 236
 - for empty space function, 236
 - for nearest neighbour function, 236
 - variance stabilised, 236
- pp3, 41
- ppm, 42, 293
 - methods for, 293
- ppp, 41
 - basic operations, 87
 - combining several, 98
 - extracting subsets, 90
 - geometrical transformations, 94
 - manipulating, 87
 - needs window, 46
 - random perturbations, 94
- ppx, 42
- predict, 297
- projection
 - changes intensity, 137
 - offset term, 287
- psp, 41
- Q–Q plot, 236
 - for empty space function, 236
 - for nearest neighbour function, 236
- quadrat counting, 127–131, 139–140, 145
- quadrat counting test
 - of CSR, 129, 145
- quadrat test
 - of inhomogeneous Poisson, 368
- quadrat.test
- class, 129
- function, 129
- plotting, 129
- quadratcount
 - class, 127
 - function, 127
- quadrats, 127
 - determined by covariate, 139
 - size
 - bias-variance tradeoff, 128
 - unequal size and shape, 128
- Queensland copper dataset, 9
- R, 19
 - command completion, 22
 - commands
 - online documentation, 22
 - contributed packages, 34
 - for spatial data formats, 64
 - for spatial statistics, 35
 - environment, 19
 - help, 22
 - search engine, 22
 - web browser interface, 22
 - information sources, 22
 - language, 19, 21
 - licence, 19
 - session, 20
 - history, 20
 - interactive, 20
 - non-interactive, 20
 - where to get, 19
 - workspace, 20
- R-help, 22
- random displacement, 269
- random perturbations, 94
- random strewing, 269
- random thinning, 269
- rectangular windows, 41, 59
- reduced second moment measure, 206
- redwood dataset, 149
- relative risk, 158
- relrisk, 159
- replicated point patterns, 10
- replication
 - in statistics, 10
- residual
 - summary functions, 398–400
- residuals, 372
 - for Poisson models, 372
 - lurking variable plot, 377

- Q–Q plot, 396
- smoothed residual field, 374
- return value, 28
- rhohat**, 142
- rigid motion correction
 - K* function, 184
- rounding of coordinates, 56
- scan test, 150
- score test, 332
- second moment measure
 - reduced, 206
- sector *K* function, 203
- shapefiles, 64
- shapefiles** package, 64
- shapley** dataset, 155
- Shapley Supercluster dataset, 3
- significance test, *see* test
- simple point process, 55
- simulation
 - of fitted Poisson model, 302
- smoothed residual field, 374
- sp** package, 64
- spatial birth-death process, 269
 - Poisson limit, 269
- spatstat**, 36
 - citing, 37
 - installing, 36
 - package, 36
- standard error
 - of fitted intensity, 297
- summary functions
 - and Monte Carlo tests, 197, 347
 - envelopes, 197, 347
 - inference using, 347
 - inhomogeneous *K*, 209
 - J*, 241
 - L*, 173
 - pair correlation, *see* pair correlation function
- Swedish Pines dataset, 124
- tess**, 41
- test, 327–365
 - Anderson-Darling, 338
 - caveats, 333
 - conclusion of, 333
 - Cramér-von Mises, 338
 - for covariate effect, 329
 - goodness-of-fit, 337
 - Kolmogorov-Smirnov, 338
 - likelihood ratio test, 330
 - logic, 333
 - score test, 332
 - Wald test, 332
- tests
 - Anderson-Darling, 146
 - Berman, 148
 - Clark-Evans, 224
 - Cramér-Von Mises, 146
 - Hopkins-Skellam, 225
 - Kolmogorov-Smirnov, 146, 370
 - Monte Carlo, 197
 - see* Monte Carlo test, 340
 - χ^2 quadrat counting, 129, 145
- thinning, *see* random thinning
- tips, 27, 29, 52, 91, 98, 230, 293
- transformation
 - changes intensity, 137
- translation correction
 - K* function, 183
- treatment contrasts, 282, 284
- unitname**, 52
- units of length, 52
- Urkiola Woods data, 4
- validation, 367
 - of independence assumption, 396–400
- variance stabilisation
 - P–P plot, 236
- Wald test, 332
- waterstriders, 4
- window, 41
 - binary mask, 41
 - polygonal, 41
 - rectangular, 41
- windows, 59
 - binary mask, 62
 - circular, 62
 - elliptical, 62
 - needed in any point pattern, 46
 - polygonal, 60
 - rectangular, 59
- χ^2 quadrat counting test, 129, 145