
Introduction to & Analysing spatial point patterns in R

2010. 12.

How to install R

Windows users:

1. Download R installer:

- *If you're not connected to the Internet:*

ask one of the workshop assistants for the USB stick and copy the file `R-2.12.0-win.exe` onto your Desktop.

- *If you are connected to the Internet:*

visit `cran.r-project.org`. In the box labelled *Download and Install R*, click on **Windows**, then **base**, then **Download R 2.12.0 for Windows**. Download and save the file `R-2.12.0-win.exe` to your Desktop. (Depending on your browser, you may need to press the control key and click on this link, or you may have to right-click and select **Save target as..**)

2. Install: double-click on the file `R-2.12.0-win.exe` to run the installer. (Agree to all the default options in the installation).

Mac users:

1. Download R installer:

- *If you're not connected to the Internet:*

ask one of the workshop assistants for the USB stick and copy the file `R-2.12.0.pkg` onto your Desktop.

- *If you are connected to the Internet:*

visit `cran.r-project.org`. In the box labelled *Download and Install R*, click on **Mac**, then **R-2.12.0.pkg**. Save the file `R-2.12.0.pkg` to your Desktop. (Depending on your browser, you may need to press the control key and click on this link, or you may have to right-click and select **Save target as..**)

2. Install: double-click on the file `R-2.12.0-win.exe` to run the installer. (Agree to all the default options in the installation).

How to install `spatstat`

Windows users:

- *If you're not connected to the Internet:*
 - ask one of the workshop assistants for the USB stick and copy the files `spatstat_1.21-2.zip`, `deldir_0.0-13.zip` and `gpclib_1.5-1.zip` onto your Desktop.
 - Start R.
 - Pull down the Packages menu to select `Install package(s) from local zip files...` In the pop-up dialogue box, use the browser to select `gpclib_1.5-1.zip`, then click `Open` to install the package. Repeat this for `deldir_0.0-13.zip` and finally for `spatstat_1.21-2.zip`.
- *If you are connected to the Internet:*
 - Start R.
 - Pull down the Packages menu and select `Set CRAN mirror...` In the pop-up dialogue box, select `Australia` (not `Austria!!`)
 - Pull down the Packages menu and select `Install package(s)...` In the pop-up dialogue box, select `spatstat`
- *If you are connected to the Internet but the instructions above did not work:*
 - Visit `cran.r-project.org`
 - In the box headed `Source code for all platforms`, click on `Contributed extension packages`.
 - Under `Available packages` click on the letter `S` then scroll down to `spatstat` and click on it.
 - In the page for `spatstat`, scroll down to the Windows file `spatstat_1.21-2.zip` and download this file to your Desktop.
 - Repeat this for the packages `deldir` and `gpclib`.
 - Pull down the Packages menu to select `Install package(s) from local zip files...` In the pop-up dialogue box, use the browser to select `gpclib_1.5-1.zip`, then click `Open` to install the package. Repeat this for `deldir_0.0-13.zip` and finally for `spatstat_1.21-2.zip`.

Linux users:

- *If you're not connected to the Internet:*
 - ask one of the workshop assistants for the USB stick and copy the files `spatstat_1.21-2.tar.gz`, `deldir_0.0-13.tar.gz` and `gpplib_1.5-1.tar.gz` into your home area.
 - Make yourself superuser, and type `R CMD INSTALL gpplib_1.5-1.tar.gz` to install the `gpplib` package. Similarly install `deldir` and finally `spatstat`.
- *If you are connected to the Internet:*
 - Start R.
 - Type `chooseCRANmirror()`. In the pop-up dialogue box, select `Australia` (not `Austria`!!)
 - Type `install.packages("spatstat")`.
- *If you are connected to the Internet but the instructions above did not work:*
 - Visit `cran.r-project.org`
 - In the box headed `Source code for all platforms`, click on `Contributed extension packages`.
 - Under `Available packages` click on the letter `S` then scroll down to `spatstat` and click on it.
 - In the page for `spatstat`, scroll down to the Linux source file `spatstat_1.21-2.tar.gz` and download this file to your home area.
 - Repeat this for the packages `deldir` and `gpplib`.
 - Make yourself superuser, and type `R CMD INSTALL gpplib_1.5-1.tar.gz` to install the `gpplib` package. Similarly install `deldir` and finally `spatstat`.

Introduction to R

Martin Hazelton
M.Hazelton@massey.ac.nz

Berwin Turlach
berwin@maths.uwa.edu.au

Lecture 1: An R Primer

In this lecture we will look at:

- What is R?
- Basic R syntax

What is R?

- R is a statistical software system.
- R is a programming language that has many “inbuilt” statistical commands (e.g. to fit a linear regression).
- R started life as a quasi-clone of commercial package S-Plus (now TIBCO Spotfire S+), but development of R and S-Plus now slowly diverging.
- Advantages over other statistics packages include flexibility, power, and quality of graphical display.
- R is open-source software, and part of GNU project. It can be downloaded from <http://cran.r-project.org/> and used for free.
- There are versions of R for all common operating systems – Windows, Linux, and MacOS.

Typographical Conventions

- With computer projection, R code (both input and output) will appear as `red text in typewriter font`.
- In your course materials, which are reproduced in black and white, this will appear as grey text in typewriter font.
- With computer projection, computer folder/file paths and http addresses will appear as `blue text in typewriter font`
- Menu names and items will appear in typewriter font (but with standard colour).
- A line of four or five full stops, `.....`, in R input/output indicates that material has been omitted or truncated.

Starting and Quitting R

- You can **start R for the first time** via the shortcut on your laptop (if such exists).
- To **quit R**, either:
 - type `q()` at the R command line; or
 - select `Exit` from the R `File` menu.

Setting the Working Directory

- When you quit R you will have the option of saving your work.
- If you choose to save your work, R will retain:
 - The R workspace that you have created;
 - An **R history file**, listing of all the R commands that you issued,
- These files are saved in your **working directory**.
- By default the working directory is something like `C:/Documents and Settings/username/My Documents` (Windows XP) or `C:/Users/username/Documents` (Windows Vista/Windows 7).
- The **working directory can be changed** using the `Change dir...` command from the R `File` menu.

Working on an Existing R Workspace

- The computer should recognise R as the program associated with an R workspace.
- Double clicking an R workspace will start up R with that workspace loaded, and with the working directory set appropriately.

A first dip into R

Elementary commands are either **expressions** or **assignments**.

- An **expression** simply displays result of a calculation; not retained in the computer's memory.
- An **assignment** passes the result of a calculation to a variable name which is stored; result not displayed.

```
> 5 + 2.6
[1] 7.6
> x <- 5 + 2.6
> x
[1] 7.6
```

Syntax for Assignment

- Assignment from right to left can be done by `<-`. This format is used throughout these notes.
- A (more confusing) alternative is to use `=`.
- Assignment from left to right can be done by `->`.

```
> x <- 1
> x
[1] 1
> x = 2
> x
[1] 2
> 3 -> x
> x
[1] 3
```

Stored objects

- All assigned variables (or any other R **objects**) are stored until overwritten or explicitly **removed** (deleted) by the command `rm()`.
- To **list stored objects** type `ls()` or `objects()`.

```
> x <- 8
> y <- 3.1415
> ls()
[1] "x" "y"
> rm(x)
> objects()
[1] "y"
```

Syntax for R commands

- R commands, e.g. `ls()`, `rm()`, are followed by parentheses which may contain additional information for the function.
- Writing a command name without parentheses returns the R source code for the function.

```
> rm(y)
> rm
function (..., list = character(0), pos = -1, envir .....
  inherits = FALSE)
{
  names <- sapply(match.call(expand.dots = FALSE)$.....
  if (length(names) == 0)
    names <- character(0)
  .....
}
```

Creating vectors in R

- The command `c()` (for *concatenate*) creates R vectors.

```
> x <- c(2.3,1.2,2.4)
> x
[1] 2.3 1.2 2.4
> c(x,9.0,x)
[1] 2.3 1.2 2.4 9.0 2.3 1.2 2.4
```

- Expression `1:n` denotes the **sequence** $1, 2, \dots, n$.
- `seq(i, j, by=k)` is a sequence from i to j in steps of k .

```
> 1:5
[1] 1 2 3 4 5
> seq(3,10,by=2)
[1] 3 5 7 9
```

R vector arithmetic

- R uses `+`, `-`, `*` and `/` for the **basic arithmetic** operations, and `^` for exponentiation (raising to a power).
- **Vector operations are done element by element**, with recycling of short vectors if required.

```
> x <- c(2,3)
> y <- c(1,4,5,6)
> 2*x
[1] 4 6
> 2 + x
[1] 4 5
> y^2
[1] 1 16 25 36
> x + y
[1] 3 7 7 9
```

Types of vector

- All the vectors we have seen so far have been *numeric*.
- R also understands vectors of:
 - characters
 - logical values **TRUE** and **FALSE**; abbreviations **T**, **F** OK.
 - factors (i.e. categorical variables)

```
> mywords <- c("This","is","a","character")
> mywords
[1] "This"      "is"        "a"         "character"
> c(F,T,F,F)
[1] FALSE TRUE FALSE FALSE
> factor(c("Low","Low","Medium","High","High"))
[1] Low   Low   Medium High  High
Levels: High Low Medium
```

Logic Operators

- R understands logical comparisons `<`, `>`, `<=`, `>=` which are applied elementwise.
- Logical equality is `==` and inequality is `!=`, while `&` is 'logical and', and `|` is 'logical or'.

```
> (1:5) == (5:1)
[1] FALSE FALSE TRUE FALSE FALSE
> (1:5) > (5:1)
[1] FALSE FALSE FALSE TRUE TRUE
> ( (1:5)==(5:1) ) | ( (1:5)>(5:1) )
[1] FALSE FALSE TRUE TRUE TRUE
> ( (1:5)==(5:1) ) & ( (1:5)>(5:1) )
[1] FALSE FALSE FALSE FALSE FALSE
```

Indexing vector components

- To **index components of a vector** `x`, use the form `x[...]`.
- Square brackets can contain:
 - numeric vector specifying elements;
 - logical vector (only **TRUE** elements required).

```
> x <- c(1,3,4,7)
> x[c(2,4)]
[1] 3 7
> x[-2]
[1] 1 4 7
> x[x > 3.5]
[1] 4 7
> which(x > 3.5)
[1] 3 4
```


Data frames

- A **data frame** is a collection of column vectors each of the same length.
- The vectors may be numeric, factor, or whatever.
- Each particular column and row of a data frame is given a name which can be chosen by the user, or assigned a default by R.

A Simple Data Frame

```
> cartoon <- c("Dilbert", "Wally", "Catbert", "TheBoss")
> role <- factor(c("Engineer", "Engineer", "Manager", "Manager"))
> x <- c(8, 1, NA, -2)
> dilbert <- data.frame(cartoon, role, competence=x)
> dilbert
```

	cartoon	role	competence
1	Dilbert	Engineer	8
2	Wally	Engineer	1
3	Catbert	Manager	NA
4	TheBoss	Manager	-2



More About Data Handling

- The variables (columns) of a data frame cannot be accessed by name until the data frame has been **attached** with `attach()`.

```
> rm(cartoon,role,x)
> dilbert$competence
[1] 8 1 NA -2
> role
Error: Object "role" not found
> attach(dilbert)
> role
[1] Engineer Engineer Manager Manager
Levels: Engineer Manager
> detach(dilbert)
```

Importing Data

- Data can be read in from text files with `scan()` for a single vector of data, and `read.table()` for a data frame.

```
> ibm <- scan(file="C:/Documents and Settings/...../ibm.txt")
Read 250 items
> ibm
 [1] 64.37 62.50 63.50 63.37 63.12 67.37 65.37 .....
.....
[239] 68.12 66.50 67.87 68.00 68.37 67.50 66.37 .....
> life <- read.table(file="C:/...../life.txt", header=TRUE)
> life
      Country LifeExp People.per.TV People.per.Dr .....
1   Argentina   70.5           4.0           370 .....
.....
40      Zaire    54.0           NA           23193 .....
```

Importing Data (cont.)

- On Windows, you can also read data in from the clipboard

```
> test <- read.table(file="clipboard", header=TRUE)
```

- For data sets with missing values, it is best to use the Comma Separated Value (CSV) file format

```
> Oz1 <- read.csv("data/Oz1.txt")
```

NOTE: `read.csv()` has `header=TRUE` by default.

Information on the CSV file format can be found at:

http://en.wikipedia.org/wiki/Comma-separated_values

and

<http://www.creativyst.com/Doc/Articles/CSV/CSV01.htm>

Editing Data

- Vectors of data or data frames can be edited by reassigning values to individual elements.
- On some platforms either of the commands `edit()`, `data.entry()` or `de()` starts up a (basic) spreadsheet for data manipulation.

```
> ibm[5]
[1] 63.12
> ibm[5] <- 65.12
> life[2,]
      Country LifeExp People.per.TV People.per.Dr .....
2 Bangladesh   53.5           315           6166 .....
> life[2,4]
[1] 6166
> life[2,4] <- 7166
> life <- edit(life)
```

Some R Functions to Get You Started

- `?` accesses R's help system; e.g. `?ls`.
- `mean()`, `sd()`, `min()`, `max()` and `range()` give mean, standard deviation, minimum, maximum and range respectively for a vector argument.
- `var()` returns variance of a vector argument, or the covariance (dispersion) matrix for a matrix argument.
- `summary()` returns summary information dependent on argument type.
- `plot()` produces a plot on the current graphics tool. The type of plot depends on the type of argument. Simplest is `plot(x,y)` which produces a scatter-plot of vectors `x` and `y`.

Getting Started:

Some tips and hints when typing in R code:

- R is case sensitive (so `LM()` and `lm()` are not equivalent).
- R is tolerant to the use of spaces, so `x <- 1` and `x<-1` are equivalent.
- You can use the arrow keys to speed things up. The ‘up’ arrow gives you the previous command that you typed.
- The usual prompt sign for R is `>`. If you get a `+` prompt sign instead, it means that R is awaiting the completion of the previous command that you typed in. This can happen because you have forgotten to close parentheses, for instance. Just type in the remainder of the command.

Exercise 1: R as a Calculator

Use R to compute the following:

1. $98765 - 43210$
2. 12.3456789^2
3. $\sqrt{(1.23 + 4.56) \times (7.8 + 9.0)}$

Type the following expressions into R, and check that you understand the results.

1. `3+4*5`
2. `3/4/5`
3. `1:20/2`
4. `(1:20)/2`

Exercise 2: Exploratory Analysis of IBM Stock Price Data

This exercise is concerned with the closing price (in U.S. dollars) of IBM stock over a sequence of 250 consecutive trading days in 1980. These data can be read in and assigned to the vector `ibm` by

```
ibm <- scan(file=choose.files())
```

and navigate to where the file `ibm.txt` is stored on your computer.

1. Take a look at the data by typing the name of the vector: `ibm`.
2. Find the mean, standard deviation, maximum and minimum of the IBM stock price data.
3. You should have found in part 1. that the maximum price is the (rather unlikely) value 6575. This is, in fact, an outlier generated by a typographical error. Find out which element of `ibm` corresponds to this outlier, and replace that element by the correct value which is 65.75.

4. Recalculate the summary statistics from part 1. using the corrected data set.
5. The *return* on any given day is defined to be the stock price on that day divided by the stock price on the previous day. Create a vector containing the returns.
6. It is often assumed in financial modelling that the log-returns are (approximately) independent and normally distributed. Create a vector containing the logarithms of the daily returns; call it `lreturn`.
7. Investigate whether the normality assumption seems reasonable by producing a histogram of the data with the command `hist(lreturn)`. Try `hist(lreturn,breaks=15)` to get a finer level of discretization. (You will learn how to beautify this type of plot of Lecture 3.)
8. Type `t.test(lreturn,mu=0)` and interpret the results. Find a 99% confidence interval for the (population) mean log-return by `t.test(lreturn,mu=0,conf.level=0.99)`.

Exercise 3: Analysis of Clinical Trial Data

This exercise is concerned with data from a clinical trial conducted in the late 1940s. The purpose was to compare the effects of three possible treatments for pulmonary tuberculosis:

- treatment by para-amino-salicylic acid (coded P)
- treatment by streptomycin (coded S)
- treatment by a combination of the above two (coded SP).

The outcomes for sputum samples obtained for each of 273 subjects in the trial were classified as

- positive smear (coded `sm`)
- negative smear but positive culture (coded `cul`)
- negative smear and negative culture (coded `neg`).

These data can be read in and assigned to the data frame `tuber` by

```
tuber <- read.table(file=choose.files(), header=TRUE)
```

and navigate to where the file `tuber.txt` is stored on your computer.

1. Take a look at the data by typing `tuber`. You should see that the data are listed by subject. Type `names(tuber)` to get a list of the column headings in the data frame.
2. Attach the data frame by `attach(tuber)`. Obtain a frequency tables by treatment and outcome respectively using `table(trt)` and `table(sputum)`.
3. Obtain a two-way table of frequencies classified by treatment and outcome by `table(trt,sputum)`. Display the results in a bar plot by `barplot(table(trt,sputum))`.
4. Investigate whether there is an association between the outcome and the treatment using a Chi-square test. Type `chisq.test(table(trt,sputum))` and interpret the results.
5. You can store the results of your Chi-square test in an object named `tuber.test` (or whatever name you find memorable) by `tuber.test <- chisq.test(table(trt,sputum))` You will find that this object has a number of components that you may like to examine. For example, `tuber.test$expected` will produce a table of expected counts, while `tuber.test$residuals` will output a table of the Pearson residuals. Based on this table of residuals, which treatment would you recommend?

Finishing Off:

When you've finished, close down R by typing `q()`. Choose 'Save' when prompted as to whether you want to retain your workspace.

Introduction to R

Martin Hazelton
M.Hazelton@massey.ac.nz

Berwin Turlach
berwin@maths.uwa.edu.au

Lecture 2: Linear Modelling in R

- In this lecture we will introduce linear modelling in R.
- Much of the material covered via example:
 - Models and notation
 - Linear regression modelling of bird species data;
 - ANOVA modelling of cow milk butterfat content;
 - Modelling pollen counts with nested factors;

Models and Notation

Linear regression:

$$Y_i = \beta_0 + \beta_1 x_{i1} + \dots + \beta_p x_{ip} + \varepsilon_i \quad (i = 1, \dots, n)$$

- Y_i is response, x_{ij} is j th predictor for individual i ;
- $\varepsilon_1, \dots, \varepsilon_n$ random sample of $N(0, \sigma^2)$ errors;
- β_0, \dots, β_p are regression parameters.

Models and Notation (cont.)

ANOVA type models: e.g. the one way model

$$Y_{ij} = \mu + \alpha_i + \varepsilon_{ij} \quad (i = 1, \dots, I, j = 1, \dots, n_i)$$

- Y_{ij} is j th individual at level i of the factor;
- $\{\varepsilon_{ij}\}$ random sample of $N(0, \sigma^2)$ errors;
- $\mu, \alpha_1, \dots, \alpha_I$ are parameters.

Models and Notation (cont.)

General Linear Model

- Linear regression, ANOVA type models and hybrids are all types of general linear model.
- Described in matrix format by

$$\mathbf{y} = X\boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

- \mathbf{y} is vector of responses,
- X the design matrix,
- $\boldsymbol{\beta}$ the vector of parameters,
- $\boldsymbol{\varepsilon}$ the vector of normal random sample of error terms.

Models and Notation (cont.)

- Vector of parameters $\boldsymbol{\beta}$ is typically estimated by method of least squares
- Least squares estimate given by

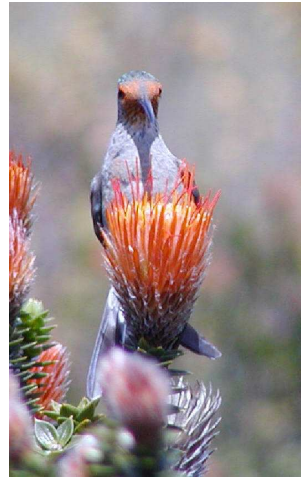
$$\hat{\boldsymbol{\beta}} = (X^T X)^{-1} X^T \mathbf{y}$$

Fitted values and residuals defined respectively by

$$\hat{\mathbf{y}} = X\hat{\boldsymbol{\beta}} \quad \text{and} \quad \mathbf{r} = \mathbf{y} - \hat{\mathbf{y}}$$

Example 1: Regression for Paramo Birds Data

- A paramo is an exposed, high plateau in the tropical parts of South America.
- In the northern Andes, there is a pattern of 'islands' of vegetation within the otherwise bare paramo.
- A study conducted to investigate the bird life in this region.
- One question of interest – what characteristics of these islands (if any) affect the diversity of bird species?



Example 1 continued

For each of 14 islands of vegetation the following variables were recorded:

- number of species of bird present, N
- area of the island in square kilometres (AR),
- elevation in thousands of meters (EL),
- the distance from Ecuador in kilometres (DEc)
- distance to the nearest other island in kilometres (DNI).

Data source: Vuilleumier (1970), 'Insular biogeography in continental regions. I. The northern Andes of South America', *American Naturaliste*, **104**, 373-388.

A Multiple Linear Regression Model

- Can try to model N as a function of other variables using multiple linear regression:

$$\mathbb{E}[N] = \beta_0 + \beta_1 AR + \beta_2 EL + \beta_3 DEc + \beta_4 DNI$$

- Linear regression models can be fitted using the `lm()` command in R.
- The syntax is of the form `lm(formula, data, ...)` where `formula` is a R formula of the form

$$y \sim 1 + x_1 + \dots + x_p$$

and `data` is an optional argument specifying the data frame containing the variables.

Fitting the Regression in R

- The following code:
 - Reads in the data and saves it as a data frame `paramo`
 - Displays the contents of `paramo`
 - Fits the linear regression described previously and assigns the result to the **fitted linear model object** `paramo.lm`.

```
> paramo <- read.table(file="data/paramo.txt",header=T)
> paramo
      Island N   AR   EL  DEc DNI
1      Chiles 36 0.33 1.26   36  14
.....
14     Cende  15 0.07 0.55 1380  35
> paramo.lm <- lm(N ~ AR + EL + DEc + DNI, data=paramo)
```

Notes of this R Code

- R includes intercept by default, so $N \sim AR+EL+DEc+DNI$ and $N \sim 1+AR+EL+DEc+DNI$ are equivalent.
- Short cut on RHS of formula is `.` which stands for 'all other variables in the data frame'. E.g. `paramo.lm <- lm(N ~ . - Island, data=paramo)`.
- Nothing special about fitted model name `paramo.lm`; could have chosen e.g. `vjc4njb`.
- Fitted model object has many attributes; e.g. parameter estimates and standard errors, fitted values, residuals etc.
- The `summary()` command applied to linear model object generates listing of parameter estimates, standard errors etc.; `anova()` produces an ANOVA table.

Fitted Model for Paramo Birds Data

```
> summary(paramo.lm)
....
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) 27.889386   6.181843   4.511  0.00146 **
AR           5.153864   3.098074   1.664  0.13056
EL           3.075136   4.000326   0.769  0.46175
DEc         -0.017216   0.005243  -3.284  0.00947 **
DNI          0.016591   0.077573   0.214  0.83541
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 '...'

Residual standard error: 6.705 on 9 degrees of freedom
Multiple R-Squared:  0.7301,    Adjusted R-squared:  0.6101
F-statistic: 6.085 on 4 and 9 DF,  p-value: 0.01182
```

ANOVA Table for Fitted Model

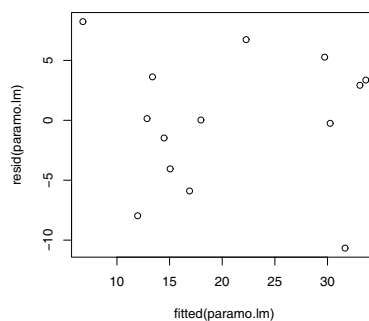
```
> anova(paramo.lm)
Analysis of Variance Table

Response: N
      Df Sum Sq Mean Sq F value    Pr(>F)
AR      1  508.92   508.92  11.3208 0.008328 **
EL      1   45.90    45.90   1.0211 0.338661
DEc     1  537.39   537.39  11.9541 0.007189 **
DNI     1    2.06    2.06   0.0457 0.835412
Residuals 9 404.59   44.95
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 '...' 1
```

Remember, ANOVA table P-values are dependent on ordering of terms.

Residuals Versus Fitted Values

- Syntax to view properties of fitted model is typically natural.
- `coef()`, `resid()`, `fitted()` returns estimated regression **coefficients**, **residuals** and **fitted values** respectively.
- E.g. `plot(fitted(paramo.lm), resid(paramo.lm))`:



Updating Models

- A number of predictors in the fitted regression are not significant.
- We can conveniently update with
`update(original.lm, . ~ . - del.x + add.x)`
- For `update()`, the dots in the formula represent all the terms in the corresponding positions in the original model formula.
- The variables `del.x` and `add.x` are just example names of terms which we wish to remove and add to the model respectively.

Updating the Paramo Birds Regression

```
> paramo.lm.2 <- update(paramo.lm, . ~ . - DNI)
> summary(paramo.lm.2)
....
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) 28.10415     5.80141   4.844 0.000677 ***
AR           5.26428     2.90535   1.812 0.100087
EL           3.04394     3.80214   0.801 0.441977
DEc         -0.01679     0.00462  -3.635 0.004572 **
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 '...'

Residual standard error: 6.377 on 10 degrees of freedom
Multiple R-Squared: 0.7287,    Adjusted R-squared: 0.6473
F-statistic: 8.953 on 3 and 10 DF,  p-value: 0.003499
```

Updating the Paramo Birds Regression (cont.)

```
> paramo.lm.3 <- update(paramo.lm.2, . ~ . - EL)
> summary(paramo.lm.3)
....
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) 30.797969   4.648155   6.626 3.73e-05 ***
AR           6.683038   2.264403   2.951 0.01318 *
DEc        -0.017057   0.004532  -3.764 0.00313 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 '....'

Residual standard error: 6.272 on 11 degrees of freedom
Multiple R-Squared:  0.7113,    Adjusted R-squared:  0.6588
F-statistic: 13.55 on 2 and 11 DF,  p-value: 0.001077
```

Example 2: ANOVA for Dairy Cattle Data

- Experiment performed to investigate butterfat content of milk (the response variable, measured as a percentage).
- Factors:
 - Cow breed with five levels: Ayrshire, Canadian, Guernsey, Holstein-Fresian, Jersey.
 - Cow age with two levels: mature and 2 years old.



Example 2 (cont.)

- 10 replicates (cows) observed at each treatment.
- The design is complete and balanced, so therefore orthogonal.

Data source: Sokal, R. and Rohlf, F. (1995) *Biometry: the principles and practice of statistics in biological research (3rd edition)*, WH Freeman and Co.

Summary of Dairy Cattle Data

```
> cows <- read.table(file="data/cows.txt",header=T)
> cows
  Butterfat      Breed   Age
1      3.74    Ayrshire Mature
2      4.01    Ayrshire  2year
....
100     5.72      Jersey  2year
> summary(cows)
  Butterfat      Breed      Age
Min.   :3.300    Ayrshire   :20  2year :50
1st Qu.:3.938    Canadian   :20  Mature:50
Median :4.405    Guernsey   :20
Mean   :4.482    Holstein-Fresian:20
3rd Qu.:4.987    Jersey     :20
Max.   :6.550
```


Specification of ANOVA Type Models in R

- ANOVA type models fitted using `lm()` function with same kind of formula as for regression models.
- R detects difference between quantitative predictors and qualitative factors as explanatory variables, and constructs the appropriate design matrix and model.
- If A and B are factors, then in an **R model formula**:
 - A on RHS indicates inclusion of **main effect** for A ;
 - $A:B$ on RHS indicates inclusion of **interaction** between A and B ;
 - shorthand syntax is $A*B = A + B + A:B$
 - brackets expanded in natural fashion; e.g.
 $(A+B)*C = A*C + B*C = A + B + C + A:C + B:C$

ANOVA Model for Dairy Cattle Data

```
> cows.lm.1 <- lm(Butterfat ~ Breed*Age, data=cows)
> anova(cows.lm.1)
Analysis of Variance Table

Response: Butterfat
          Df Sum Sq Mean Sq F value Pr(>F)
Breed      4 34.321   8.580 49.5651 <2e-16 ***
Age        1  0.274   0.274  1.5801 0.2120
Breed:Age  4  0.514   0.128  0.7421 0.5658
Residuals 90 15.580   0.173
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' ...
```

- Breed but not age (or breed:age interaction) affect butterfat content.

Parametrisation for ANOVA Models in R

- The effects of factors in linear models can be parametrised in any number of ways.
- E.g. the one-way ANOVA model is typically written as

$$Y_{ij} = \mu + \alpha_i + \varepsilon_{ij} \quad (i = 1, \dots, I, j = 1, \dots, n_i)$$

where a **constraint** must be placed on $\alpha_1, \dots, \alpha_I$.

- Common constraints include:

- Sum constraint:** $\sum_{i=1}^I n_i \alpha_i = 0$.
- Treatment constraint:** $\alpha_1 = 0$.

- R employs the treatment constraint by default: the first level of each factor is regarded as a 'baseline'.

Parameter Estimates for Dairy Cattle Data

```
> summary(cows.lm.1)
....
              Estimate Std. Error t value Pr(>|t|)
(Intercept)      3.9660     0.1316  30.143 < 2e-16
BreedCanadian    0.5220     0.1861   2.805 0.00616
BreedGuernsey     0.9330     0.1861   5.014 2.65e-06
BreedHolstein-Fresian -0.3030     0.1861  -1.628 0.10693
BreedJersey       1.1670     0.1861   6.272 1.22e-08
AgeMature         0.1880     0.1861   1.010 0.31503
BreedCanadian:AgeMa.... -0.2870     0.2631  -1.091 0.27834
BreedGuernsey:AgeMa.... -0.0860     0.2631  -0.327 0.74457
BreedHolstein-Fresi.... -0.1750     0.2631  -0.665 0.50773
BreedJersey:AgeMature  0.1310     0.2631   0.498 0.61982
....
```

Interpretation of Parameter Estimates

- Model is

$$Y_{ijk} = \mu + \alpha_i + \beta_j + (\alpha\beta)_{ij} + \varepsilon_{ijk}$$

- i indexes breed (with 5 levels)
- j indexes age (with 2 levels)
- k indexes animal in each group ($k = 1, \dots, 10$).

- By default, R sets factor levels in alphabetical order. Hence:

- Ayrshire is level one of the breed factor.
- 2year is level one of the age factor.

- Treatment constraint implies $\alpha_1 = \beta_1 = 0$, so 2 year old Ayrshire is reference level.

Interpretation of Parameter Estimates: Examples

- Intercept estimate of $\hat{\mu} = 3.97$ is fitted value for 2 year old Ayrshire.

- $\hat{\alpha}_3 = 0.93$, labelled **BreedGuernsey** in output, is **contrast** between 2 year old Guernsey and 2 year old Ayrshire.

- Fitted value for mature Guernsey is

$$\hat{\mu} + \hat{\alpha}_3 + \hat{\beta}_2 + (\alpha\beta)_{3,2} = 3.97 + 0.93 + 0.19 - 0.09 = 5.00$$

- Check:

```
> attach(cows)
> fitted(cows.lm.1)[Breed=="Guernsey" & Age=="Mature"]
  41  43  45  47  49  51  53  55  ....
5.001 5.001 5.001 5.001 5.001 5.001 5.001 5.001  ....
```

Example 3: Nested Factors and Pollen Count Data

- Interest in abundance of pine pollen in cores taken from bogs in northern Alberta, Canada.
- Pollen sampled at three depths: *shallow*, *medium* and *deep*.
- Two samples of peat at each depth.
- Two slides prepared from each sample.
- Number of pollen grains on a slide is the response.



Model for Pollen Count Data

- Factors are *Sample* and *Depth*, with *Sample* **nested** within *Depth*.
- Model can be written
$$Y_{ijk} = \mu + \alpha_i + \beta_{j(i)} + \varepsilon_{ijk}.$$
 - Depth indexed by i (3 levels)
 - Sample indexed by j (2 levels per depth)
 - Slide indexed by k ($k = 1, 2$)
- R formula uses A/B to represent B nested within A .
- Formally, $A/B = A + A:B$.

Analysis of Pollen Count Data

```
> pollen <- read.table(file="data/pollen.txt",header=T)
> pollen
  Depth Sample Count
1 shallow     A    12
2 shallow     A    14
3 shallow     B    10
4 shallow     B     7
5 medium     A    16
6 medium     A    12
7 medium     B    10
8 medium     B    19
9 deep       A    21
10 deep      A    29
11 deep      B    33
12 deep      B    30
```

Model Fitting

```
> pollen.lm <- lm(Count ~ Depth/Sample, data=pollen)
> anova(pollen.lm)
Analysis of Variance Table

Response: Count
          Df Sum Sq Mean Sq F value Pr(>F)
Depth      2  686.00   343.00  22.4918 0.00163 **
Depth:Sample 3   62.75    20.92   1.3716 0.33847
Residuals  6   91.50    15.25
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 '....'
```

- No evidence clustering within *Sample*.

Summary

- Linear models fitted by R function `lm()`.
- Models specified by R model formulae.
- In a formula, R identifies predictors and factors.
- Interaction specified by `A:B`; nesting by `A/B`.
- Fitted model object can be scrutinised by commands like `summary()` (table of estimates) and `anova()` (ANOVA table).

INTRODUCTION TO R
FREMANTLE, 10 DEC. 2010

Computer Laboratory 2

Exercise 1: Regression Modelling for Swiss Fertility Data

This exercise is concerned with data on fertility rates in Switzerland in 1888. At that time Switzerland was going through a period of “demographic transition”, with fertility rates falling from the previous high levels to those more typical of modern industrialized European countries. The aim is to relate standardized fertility rate to five socioeconomic indicators using data from 47 French speaking provinces in Switzerland. The variables in the data set are (for each province):

Fertility	A common standardized fertility measure
Agriculture	Percentage of males involved in agriculture as occupation
Examination	Percentage of ‘draftees’ receiving highest mark on army examination
Education	Percentage of education beyond primary school for ‘draftees’
Catholic	Percentage of Catholics in the population
Infant.Mortality	Percentage of live births who live less than one year

Data source: Mosteller, F. and Tukey, J.W. (1977) *Data Analysis and Regression: A Second Course in Statistics*. Addison-Wesley, Reading Mass.

You can read these data into R, and store them as a data frame `swiss`, with the command

```
swiss <- read.table(file=choose.files(), header=TRUE, row.names=1)
```

and navigate to where the file `swiss.txt` is stored on your computer. Note that the argument `row.names=1` tells R that the first column in the text file `swiss.txt` should be read in as row names.

Fit a multiple linear regression model to the data by

```
swiss.lm.1 <- lm(Fertility ~ ., data=swiss)
```

Notice the use of the dot in the model formulae to represent ‘all other variables in the data frame’.

Use the `summary()` command to scrutinize the fitted model. Is `Fertility` related to at least one of the predictors? (Check the ‘omnibus’ F test statistic at the bottom of the summary output.)

Consider whether we can remove some of the predictors by using backwards variable selection implemented by F testing. You should have found from the summary table for `swiss.lm.1` that `Examination` is not statistically significant having adjusted for the other variables ($p = 0.315$). Remove this predictor from the model and refit, by

```
swiss.lm.2 <- update(swiss.lm.1, . ~ . - Examination)
```

Look at the summary table for `swiss.lm.2`. Should any further terms be dropped? Update the model if necessary.

Have a look at the coefficients of the predictors in your preferred model. Can you explain the signs of these coefficients in the context of the data?

Exercise 2: Linear Modelling for Ion Absorption Data

This Exercise concerns a study on the absorption over time of rubidium and bromide ions in potato slices. Three variables are measured on each slice:

Variable	Description
Absorption	amount of ion absorbed in the tissue
Duration	time of immersion (in hours) in ion solution
Ions	Rubidium, R, or Bromide, B

Data source: Hand, D.J., Daly F., Lunn A.D., McConway K.J., Ostrowski E. (1994). *A Handbook of Small Data Sets*, Chapman & Hall, London.

You can read these data into R, and store them as a data frame `ion`, with

```
ion <- read.table(file=choose.files(), header=TRUE)
```

and navigate to where the file `ion.txt` is stored on your computer. Take a look at the data frame (by simply typing its name).

We will seek to construct a suitable linear model `Absorption` as a response and `Duration` and `Ions` as explanatory variables. There are three candidate models:

M1: Simple linear regression of Absorption on Duration (ignoring Ions).

M2: Parallel linear regression of Absorption (the response) on Duration for each type of ion.

M3: Separate linear regressions of Absorption (the response) on Duration for each type of ion.

These models can be fitted by the following commands respectively:

```
ion.lm.1 <- lm(Absorption ~ Duration, data=ion)
ion.lm.2 <- lm(Absorption ~ Duration + Ions, data=ion)
ion.lm.3 <- lm(Absorption ~ Ions/Duration, data=ion)
```

The first of these commands should need no explanation. The second command indicates a constant slope (specified by the inclusion of the `Duration` term) but different intercepts (specified by the inclusion of the `Ions` term). The formula in the third command can be expanded to

```
Absorption ~ Ions + Ions:Duration
```

which indicates different intercepts (specified by the inclusion of the `Ions` ‘main effect’) and different slopes (specified by the `Ions:Duration` term).

Take a look at the fitted models. You should be able to figure out the fitted model equations from the R summary output. The correct equations are as follows:

$$\begin{aligned} M1: & \quad \mathbb{E}[\text{Absorption}] = 0.186 + 0.180 \text{Duration} \\ M2: & \quad \mathbb{E}[\text{Absorption}] = \begin{cases} -2.444 + 0.181 \text{Duration} & \text{for Bromide ions} \\ 2.816 + 0.181 \text{Duration} & \text{for Rubidium ions} \end{cases} \\ M3: & \quad \mathbb{E}[\text{Absorption}] = \begin{cases} -2.922 + 0.188 \text{Duration} & \text{for Bromide ions} \\ 3.295 + 0.173 \text{Duration} & \text{for Rubidium ions} \end{cases} \end{aligned}$$

We can compare nested linear models by F tests. This can be achieved using the `anova()` command in R. Specifically,

```
anova(ion.lm.1, ion.lm.2)
```

will compare *M1* and *M2* by an F test (i.e. a null hypothesis of equal intercepts for Bromide and Rubidium ions will be tested). Compare *M2* and *M3* using the same methodology. What is your preferred model?

Finishing Off:

When you’ve finished, close down R by typing `q()`. Choose ‘Save’ when prompted as to whether you want to retain your workspace.

Introduction to R

Martin Hazelton
M.Hazelton@massey.ac.nz

Berwin Turlach
berwin@maths.uwa.edu.au

Lecture 3: Graphics

In this lecture we shall discuss

- graphics devices
- R base graphics
- graphics parameters for fine tuning graphics

Graphics Devices

R can produce graphics on a variety of graphical devices.

```
> help("Devices")
```

will tell you which devices are available on your platform; some of which may be:

<code>windows()</code>	graphics driver for Windows
<code>X11()</code>	graphics driver for the X11 Window system
<code>postscript()</code>	writes PostScript graphics commands to a file
<code>pdf()</code>	writes PDF graphics commands to a file

You may have several graphics devices open at the same time.

But only one device is the *active* device.

Handling Multiple Graphics Devices

The following commands are useful for handling multiple graphics devices:

<code>dev.cur()</code>	returns the number and name of the active device
<code>dev.list()</code>	returns the numbers of all open devices
<code>dev.next()</code>	returns the number of the next device
<code>dev.prev()</code>	returns the number of the previous device
<code>dev.set()</code>	makes the specified device the active device
<code>dev.off()</code>	shuts down the specified device
<code>graphics.off()</code>	shuts down all open graphics devices
<code>dev.copy()</code>	copies the graphics contents of the current device into a new device, e.g. <code>dev.copy(windows)</code>

Plotting Commands

We shall discuss the commands in the package `graphics`.

This package contains functions for base graphics, the traditional S graphics.

A list of all commands available in this package can be obtained either via the command

```
library(help=graphics)
```

or

```
help("graphics-package")
```

`demo(graphics)` illustrates some of R's graphics capabilities.

A rewrite of the graphics capabilities is given by the (base) package `grid` and the (recommended) package `lattice`. The latter implements Trellis Graphics (Cleveland, W.S. (1993). *Visualizing Data*, Hobart Press, Summit, N.J.).

Plotting Commands (cont.)

Plotting commands can be divided into three basic groups:

- *High-level* plotting commands:
These commands typically create new plots on the active graphics device.
- *Low-level* plotting commands:
These commands add more information to an existing plot.
- *Interactive* graphics commands:
Commands for adding information to, or extracting information from, an existing plot in an interactive manner.

Some High-Level Plotting Commands

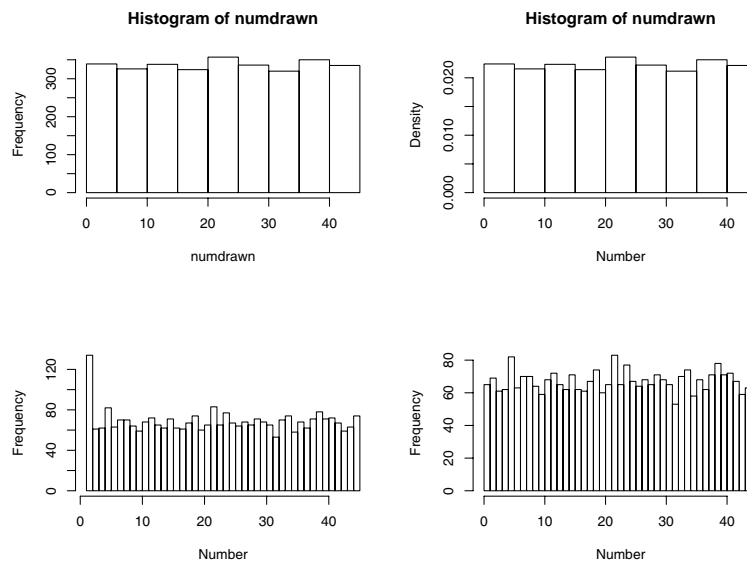
<code>plot()</code>	Generic X-Y Plotting
<code>barplot()</code>	Bar Plots
<code>boxplot()</code>	Box Plots
<code>curve()</code>	Draw Function Plots
<code>contour()</code>	Display Contours
<code>coplot()</code>	Conditioning Plots
<code>hist()</code>	Histograms
<code>image()</code>	Display a Color Image
<code>matplot()</code>	Plot Columns of Matrices
<code>pairs()</code>	Scatterplot Matrices
<code>persp()</code>	Perspective Plots
<code>stripchart()</code>	1-D Scatter Plots

Powerball data

```
> dat <- read.csv("data/Powerball.csv")
> head(dat)
  NumTDA WeeksSLDA NumTDB WeeksSLDB
1     65         12    14         6
2     69          3    14         3
3     61          1    14         2
4     62          9    19        38
5     82          7    11        24
6     63          9    11       170

> numdrawn <- rep(1:45, dat[,1])
> hist(numdrawn)
> hist(numdrawn, probability=TRUE, xlab="Number")
> hist(numdrawn, breaks=1:45, xlab="Number", main="")
> hist(numdrawn, breaks=0:45, xlab="Number", main="")
```

Powerball data (cont.)



Powerball data (cont.)

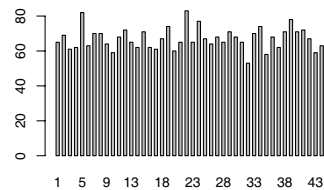
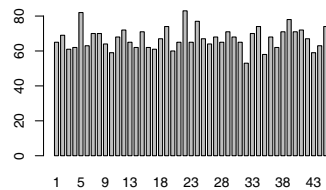
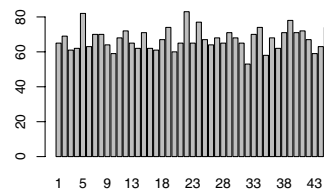
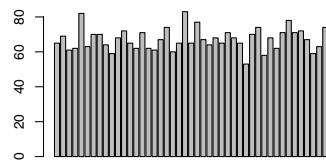
```
> barplot(dat[,1])
> barplot(dat$NumTDA, names.arg=1:45)
> barplot(dat[, "NumTDA"], names.arg=1:45, space=0.5)
> barplot(dat[,1], names.arg=1:45, space=0.9)

> chisq.test(dat[,1])
```

Chi-squared test for given probabilities

```
data: dat[, 1]
X-squared = 25.2264, df = 44, p-value = 0.9897
```

Powerball data (cont.)



Arguments to High-Level Plotting Commands

Most high-level plotting commands have the following arguments:

- **add=TRUE**: forces the function to act as a low-level graphics function, superimposing the plot on the current plot
- **axes=FALSE**: suppresses generation of axes—useful for adding customised axes with the `axis()` command.
- **log="x", log="y" or log="xy"**: Causes the x axis, y axis or both axes to be logarithmic
- **xlab=string, ylab=string**: labels for the x and y axes, respectively
- **xlim=c(a,b), ylim=c(a,b)**: specifies the ranges for the x and y axes, respectively

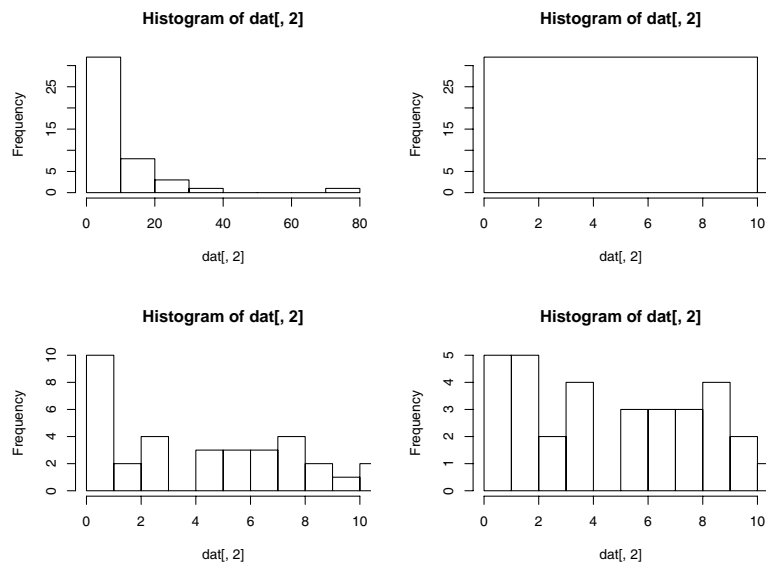
Arguments to High-Level Plotting Commands (cont.)

- **main=string:**
figure title, placed at the top of the plot in a large font
- **sub=string:**
sub-title, placed just below the x -axis in a smaller font
- **type=string:** the type of plot to be produced.
 - "p" plot individual points
 - "l" plot lines
 - "b" plot points connected by lines (*both*)
 - "o" plot points overlaid by lines
 - "h" plot vertical lines from points to zero axis (*high density*)
 - "s" or "S" step-function plot (top or bottom of vertical defines point, respectively)
 - "n" plot nothing at all, however the axes are still drawn and the coordinate system is set up according to the data

Powerball data (cont.)

```
> hist(dat[,2])
> hist(dat[,2], xlim=c(0,10))
> hist(dat[,2], xlim=c(0,10), breaks=0:80)
> hist(dat[,2], xlim=c(0,10), breaks=0:80, right=FALSE)
```

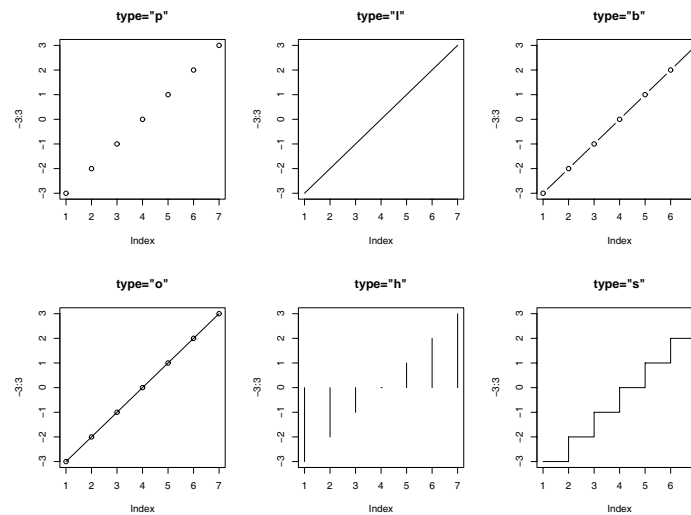
Powerball data (cont.)



Example 4: Illustrating the `type` Argument

```
> postscript("graph1.eps", paper="special",
+           width=8, height=6,
+           bg="white", horizontal=FALSE)
> par(mfrow=c(2,3))
> plot(-3:3, type="p", main="type=\"p\"")
> plot(-3:3, type="l", main="type=\"l\"")
> plot(-3:3, type="b", main="type=\"b\"")
> plot(-3:3, type="o", main="type=\"o\"")
> plot(-3:3, type="h", main="type=\"h\"")
> plot(-3:3, type="s", main="type=\"s\"")
> dev.off()
```


Illustrating the `type` Argument (cont.)



Some Low-Level Plotting Commands

<code>abline()</code>	Add a Straight Line to a Plot
<code>axis()</code>	Add an Axis to a Plot
<code>legend()</code>	Add Legends to Plots
<code>lines()</code>	Add Connected Line Segments to a Plot
<code>mtext()</code>	Write Text into the Margins of a Plot
<code>points()</code>	Add Points to a Plot
<code>polygon()</code>	Polygon Drawing
<code>rect()</code>	Draw One or More Rectangles
<code>rug()</code>	Add a Rug to a Plot
<code>segments()</code>	Add Line Segments to a Plot
<code>text()</code>	Add Text to a Plot
<code>title()</code>	Plot Annotation

Graphics Parameters

In addition to low-level plotting commands, the presentation of graphics is influenced by *graphics parameter*.

Graphics parameters can be changed *permanently* using the function `par()`.

If called without argument, `par()` returns the current values of all graphics parameters.

Currently, there are 70 parameters, 65 of which can be changed by the user.

A call like `par("col", "lty")` or `par(c("col", "lty"))` returns only the values of the specified graphics parameters.

A call with named arguments, e.g. `par(col=4, lty=2)` sets the values of the named graphics parameters and returns a list with the original values of the parameters.

Graphics Parameters (cont.)

The complete list of graphics parameters, and the description of each one, can be obtained via `help(par)`.

Some of the more useful ones are:

- `ask`: ask before a new figure is drawn
- `adj`: controls text justification
- `bg`: specifies the background colour
- `fg`: specifies the foreground colour
- `cex`: character expansion, the desired size of text characters *relative* to the default text size
- `cex.axis`, `cex.lab`, `cex.main`, `cex.sub`: magnification to be used for axis annotation, *x* and *y* labels, main titles and sub-titles, respectively, *relative* to the current setting of `cex`

Graphics Parameters (cont.)

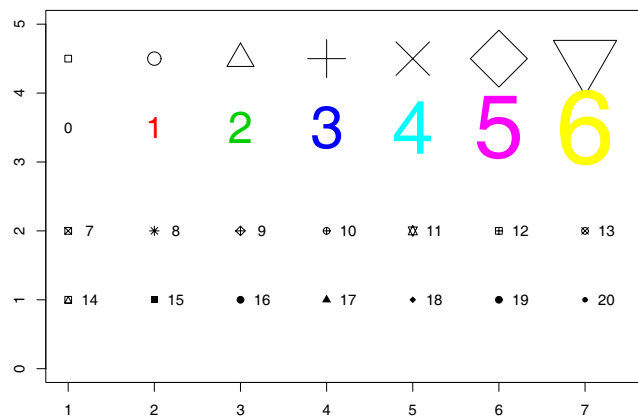
- `col`: default plotting colour
- `col.axis`, `col.lab`, `col.main`, `col.sub`: colour to be used for axis annotation, x and y labels, main titles and sub-titles, respectively
- `family`: the name of a font family
- `las`: orientation of axis labels
- `lty`: the line style
- `lwd`: the line width
- `pch`: integer specifying a symbol or a single character to be used for plotting points

Graphics Parameters (cont.)

Graphics parameters may also be passed to (most) graphics function as named arguments to have only a temporary effect.

```
> plot(1, 1, xlim=c(1, 7.5), ylim=c(0,5),
+      type="n", xlab="", ylab="")
> points(1:7, rep(4.5, 7), pch=0:6, cex=1:7)
> text(1:7, rep(3.5, 7),
+      labels=paste(0:6), cex=1:7, col=1:7)
> points(1:7, rep(2, 7), pch=(0:6)+7)
> text(1:7+0.25, rep(2, 7), paste((0:6)+7))
> points(1:7, rep(1, 7), pch=(0:6)+14)
> text(1:7+0.25, rep(1, 7), paste((0:6)+14))
```

Graphics Parameters (cont.)



Graphics Parameters: Multiple Figures

The graphics parameters `mfc` and `mfrow` can be used to create multiple figures on a graphics device.

These graphics parameters take a numerical vector of the form `c(nr, nc)` and set up an `nr` by `nc` array of figures.

Subsequent plotting commands will fill this array either by columns (`mfc`) or by rows (`mfrow`).

The graphics parameter `mfg` can be used to either enquire or set which figure in an array of figures is drawn next.

The commands `layout()` and `screen()` provide alternative ways of creating multiple figures on a graphics device. These commands are somewhat more flexible.

NOTE: These methods of creating multiple figures are mutually incompatible.

Powerball data (cont.)

```
> par(mfrow=c(2,2))
> hist(numdrawn, breaks=0:45, xlab="Number", main="")

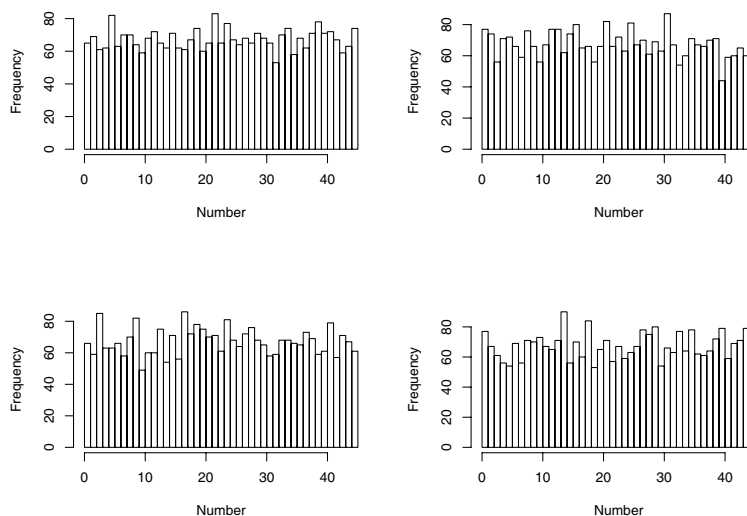
> nd1 <- 0*numdrawn

> for(i in 1:605) nd1[(5*(i-1)+1):(5*i)] <- sample(1:45,5)
> hist(nd1, breaks=0:45, xlab="Number", main="")

> for(i in 1:605) nd1[(5*(i-1)+1):(5*i)] <- sample(1:45,5)
> hist(nd1, breaks=0:45, xlab="Number", main="")

> for(i in 1:605) nd1[(5*(i-1)+1):(5*i)] <- sample(1:45,5)
> hist(nd1, breaks=0:45, xlab="Number", main="")
```

Powerball data (cont.)



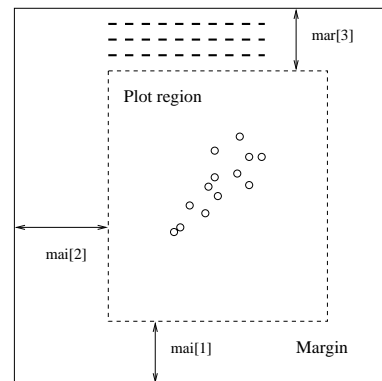
Graphics Parameters: Figure Margins

The graphics parameters `mai` and `mar` control the margins of a figure.

These are numerical vectors of the form `c(bottom, left, top, right)` giving the margin sizes of each side.

`mai` specifies the margin sizes in inches.

`mar` specifies the margin in lines of text.



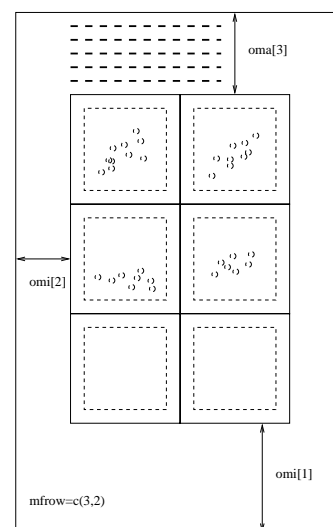
Graphics Parameters: Figure Margins

Analogously, in multiple figure plots the graphics parameters `omi` and `oma` control the *outer* margins of the plot.

These are numerical vectors of the form `c(bottom, left, top, right)` giving the margin sizes of each side.

`omi` specifies the margin sizes in inches.

`oma` specifies the margin in lines of text.



Interacting with Graphics

There are essentially two functions available for interacting with graphics

- `locator(n, type="n")`:

returns upon completion the locations of points selected by the user as a list with components `x` and `y`.

Typical usage is the interactive positioning of legends

```
legend(locator(1),....)
```

- `identify(x, y, labels)`:

allows the highlighting of the points defined by `x` and `y`. The indices of the selected points are returned upon completion of the command.

Where to From Here?

More examples can be found at the R Graph Gallery:

<http://addictedtor.free.fr/graphiques/>

Study “From Data to Graphics” and “Customizing graphics” at Statistics with R:

http://zoonek2.free.fr/UNIX/48_R/all.html

“Graphing” at Rtips:

<http://pj.freefaculty.org/R/Rtips.html>

Murrell, P. (2006). *R Graphics*, Computer Science and Data Analysis Series, Chapman & Hall/CRC.

<http://www.stat.auckland.ac.nz/~paul/RGraphics/rgraphics.html>

Where to From Here? (cont.)

Also of interest:

Cook, D. and Swayne, D.F. (2007). *Interactive and Dynamic Graphics for Data Analysis*, Use R!, Springer-Verlag, New York.

Sarkar, D. (2008). *Lattice: Multivariate Data Visualization with R*, Use R!, Springer-Verlag, New York.

<http://lmdvr.r-forge.r-project.org/figures/figures.html>

Wickham, H. (2009). *ggplot2: Elegant Graphics for Data Analysis*, Use R!, Springer-Verlag, New York.

<http://had.co.nz/ggplot2/book/>

The next lab.

Exercise 1: Plotting Functions of One Variable

The aim of this exercise is to help you learn how to plot functions of one variable in R. Consider, for example, the function

$$f(x) = x^2$$

and suppose that we wish to plot the graph of $f(x)$ for x in the range $(-5, 5)$. The way to do this in R is as follows:

- (i) Create a fine grid (sequence) of x values on the interval $(-5, 5)$.
- (ii) Evaluate the function $f(x)$ at each grid point.
- (iii) Join up the resultant pairs $(x, f(x))$ with tiny straight line segments.

This is perhaps easiest understood by actually doing it!

Start by creating the grid of x values using the command

```
x <- seq(-5, 5, length=201)
```

Take a look at the object `x` (just type its name) to see what you have made. Now evaluate the function on the grid:

```
fx <- x^2
```

Finally, plot `fx` against `x` with

```
plot(x, fx, type="l")
```

The argument `type="l"` of `plot` tells R to connect the points with line segments (hence "l"). By default R would simply plot the unconnected points – see what happens if you omit the (optional) `type="l"` argument. You can beautify your graph by adding labels to the axes, by adding a title, and by changing the colour scheme. Try

```
plot(x, fx, type="l", ylab="f(x)", col="red", main="Graph of f(x)=x^2")
```

and then try some variations of your own devising.

Now try plotting the following functions:

1. $f(x) = 7 \log(x) - x$ for x in the interval $(0.1, 20)$.
2. $f(x) = 5 \log(x) + 10 \log(1 - x)$ for x in the interval $(0.1, 0.9)$.

Exercise 2: Plotting Functions of Two Variables

It is much more difficult to visualise functions of many variables than functions of one variable. Nonetheless, we can get useful displays of functions of two variables using perspective ('wire frame'), contour and image ('heat') plots. This exercise will help you learn how to generate such plots in R.

Consider the function

$$f(x, y) = e^{-(x^2+y^2)}$$

and suppose that we wish to plot its graph over the range $x \in (-3, 3)$ and $y \in (-3, 3)$. To do this we will:

- (i) Create a fine grid of points on the x - y plane over the ranges indicated above.
- (ii) Evaluate the function $f(x, y)$ at each grid point.
- (iii) Supply grid and function evaluations to R's `persp`, `contour` or `image` commands.

Start by creating the locations of the grid lines on the x and y axes:

```
x <- seq(-3, 3, length=51)
y <- seq(-3, 3, length=51)
```

All the R plotting commands for functions of two variables expect the evaluations of $f(x, y)$ to be presented as a *matrix*, with elements corresponding to the appropriate grid locations. We start by creating a matrix full of zeros:

```
fxxy <- matrix(0, ncol=51, nrow=51)
```

Note that the number of columns (`ncol`) and rows (`nrow`) matches the number of grid lines on each axis. We now insert the function values into this matrix using loops¹:

```
for (i in 1:51){
  for (j in 1:51){
    fxy[i,j] <- exp(-(x[i]^2 + y[j]^2))
  }
}
```

Finally, we can get perspective, contour and image with the following syntax:

```
persp(x, y, fxy)
contour(x, y, fxy)
image(x, y, fxy)
```

If you have time, try beautifying these plots; e.g.

```
persp(x, y, fxy, col="cyan", theta=45)
contour(x, y, fxy, nlevels=25)
image(x, y, fxy, col=terrain.colors(50))
```

Use R's help system to guide you.

Exercise 3: Studying Examples

R, as a programming environment, has excellent quality control tools. When installing R itself (or any R package) from source these tools try to ensure that the installation will be successful. Part of the installation process is that the examples in every help page are checked. This process produces a large set of examples of graphical figures which is readily accessible.

In the material provided electronically, you will find a folder called **GraphicsEx** with graphical output produced by some of the base and recommended packages during installation. You may peruse these files at your leisure.

Note the file **GraphicsEx.R**. After starting R, change your working directory (**Change dir...** from the **File** menu) to an easily accessible place, e.g. the desktop. Then select **Source R code...** from the **File** menu, locate and select the file **GraphicsEx.R** to execute the commands in this file. After R has run the commands in this file, you will find a file called **GraphicsEx.pdf** in your working directory. This file will contain the graphics from the example section of selected commands. By editing **GraphicsEx.R**, you can easily create your own gallery of favourite R figures. (NOTE: some packages may first have to be loaded (via `library()`) before you can run the examples of commands provided by the package.)

Finishing Off:

When you've finished, close down R by typing `q()`. Choose 'Save' when prompted as to whether you want to retain your workspace.

¹NOTE: Alternatively, to avoid loops we could 'vectorise' these calculations. A simple command like

```
fxxy <- exp(-outer(x^2, y^2, "+"))
```

would do the same calculations. In this case, we also would not have to issue the previous command that created `fxxy` and filled it with zeros.

Introduction to R

Martin Hazelton
M.Hazelton@massey.ac.nz

Berwin Turlach
berwin@maths.uwa.edu.au

Lecture 4: Generalized Linear Models in R

- In this lecture we will introduce GLMs in R.
- Much of the material covered via example:
 - Models and notation
 - Logistic regression for Titanic survivors data
 - Gamma regression for hormone assay data

Basic Elements of a Generalized Linear Model

- Response Y_i for i th individual, observed independently of other responses.
- Corresponding explanatory variables are $x_{i1}, x_{i2}, \dots, x_{ip}$.
- **Linear predictor** is

$$\eta_i = \beta_0 + \beta_1 x_{i1} + \dots + \beta_p x_{ip}.$$

- Expected response $\mathbb{E}[Y_i] = \mu_i$ and linear predictor related by

$$\eta_i = g(\mu_i).$$

where g is smooth invertible **link function**.

- Response Y_i follows exponential family distribution.

GLM Families

- Combination of response distribution and link function is called the **family** of the GLM.
- **Canonical links** pair naturally with response distribution.

Response distribution	Canonical link
Normal	identity $g(\mu) = \mu$
Binomial	logit: $g(\mu) = \log(\mu/(1 - \mu))$
Poisson	log: $g(\mu) = \log(\mu)$
Gamma	inverse: $g(\mu) = \mu^{-1}$

- Gamma response with identity link, and binomial response with probit link are examples of common non-canonical links.

Inference for GLMs

- Parameters β_0, \dots, β_p in GLMs are (typically) estimated by maximum likelihood.
- Goodness of fit of a GLM can be measured by (residual) **deviance**, which is analogous to the (residual) sum of squares in a linear model.
- The **fitted values** for a GLM are defined by

$$\hat{\mu}_i = g^{-1}(\hat{\eta}_i)$$

where

$$\hat{\eta}_i = \hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \dots + \hat{\beta}_p x_{ip}$$

- **Raw residuals** are $r_i = y_i - \hat{\mu}_i$, but alternatives like **deviance residuals** are available.

Fitting GLMs in R

- GLMs are fitted (by maximum likelihood estimation) in R using the `glm()` function.
- The function `glm()` essentially works in the same way as `lm()`, but with an additional argument to specify the GLM family.
- Options for the family argument `binomial`, `Gamma` and `poisson`. E.g.
`glm(y ~ x1 + x2, family=Gamma)`
- Canonical link is assumed by default. Alternative link specified in parentheses, e.g.
`glm(y ~ x1 + x2, family=Gamma(link="identity"))`.

Example 5: Logistic Regression for Titanic Data

- Titanic data contains observations from passengers of the Titanic.
- For each individual four variables are recorded:
 - Survived:** Binary response, 1 = survived, 0 = died.
 - PClass:** Travelling class; 1st, 2nd or 3rd.
 - Age:** Age in years.
 - Sex:** Factor with levels 'female' and 'male'.
- Shall use logistic regression – i.e. GLM with binomial family and canonical link – to model survival in terms of age and sex.

Data Summary

```
> titanic <- read.table(file="data/titanic.txt",header=T)
> summary(titanic)
      Name      PClass      Age      Sex      ....
Carlsson, ....., 2 1st:322  Min.   : 0.17  female:462
Connolly, ....., 2 2nd:280 1st Qu.: 21.00  male  :851
Kelly, ....., 2 3rd:711  Median : 28.00
Abbing, ....., 1           Mean   : 30.40
Abbott, ....., 1           3rd Qu.: 39.00
Abbott, ....., 1           Max.   : 71.00
(Other)      :1304      NA's   :557.00
```

Data Summary: Comments

- Three pairs of passengers shared the same names.
- The majority of passengers were travelling third class.
- Age is missing (**NA**) for about half the passengers.
- We will assume that the age data is missing at random.
- For `lm()` or `glm()`, default is to omit entire record for each individual with a pertinent missing value.
- Alternatives available by specifying `na.action` argument in `lm()` or `glm()`.

Fitting Logistic Regression to Titanic Data

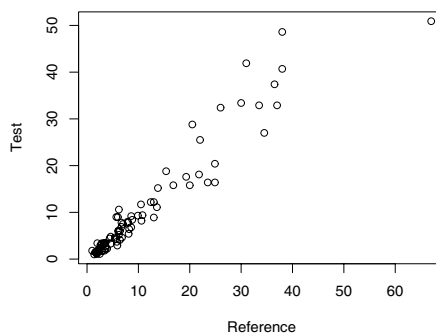
```
> titanic.glm.1 <- glm(Survived ~ . - Name, family=binomial,
+ data=titanic)
> summary(titanic.glm.1)
....
Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept)  3.759662   0.397567   9.457 < 2e-16 ***
PClass2nd   -1.291962   0.260076  -4.968 6.78e-07 ***
PClass3rd   -2.521419   0.276657  -9.114 < 2e-16 ***
Age         -0.039177   0.007616  -5.144 2.69e-07 ***
Sexmale     -2.631357   0.201505 -13.058 < 2e-16 ***
---
....
Null deviance: 1025.57 on 755 degrees of freedom
Residual deviance: 695.14 on 751 degrees of freedom
```

Summary of Results for Titanic Data

- By default, first level of **Sex** (i.e. female) and of **PClass** (i.e. 1st class) are set as reference.
- The better the travelling class, the better the chances of survival.
- Women more likely to survive than men.
- Young more likely to survive than old.
- Analysis supports idea of 'women and children first'.
- Could refine analysis by checking for interactions and varying slope for **Age**.

Example 6: Gamma Regression for Hormone Assay

- Hormone assays conducted on 85 samples.
- In the experiment, the old (or reference) method is compared to a new (or test) method.



Modelling the Hormone Assay Data

- Seek to model test method (response) in terms of reference method (predictor).
- Could use standard simple linear regression model, but scatter-plot provides clear evidence of heteroscedasticity.
- Possible solutions:
 - Model on the log-scale for both response and predictor.
 - Use gamma regression with identity link, since mean proportional to std. dev. for gamma distribution.
- Second solution allows us to maintain additive model on original scale of data.

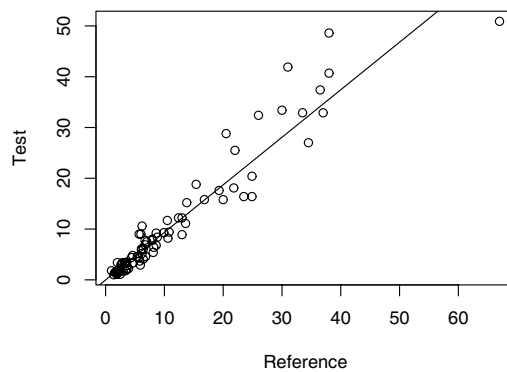
Model Fitting for Hormone Data

```
> hormone <- read.table(file="data/hormone.txt",header=T)
> hormone.glm <- glm(Test ~ Reference, family=
+ Gamma(link="identity"), data=hormone)
> summary(hormone.glm)
....
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.01802    0.15212  -0.118   0.906
Reference    0.93629    0.04497  20.819 <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 '....'

(Dispersion parameter for Gamma family .... 0.08433066)
....
```

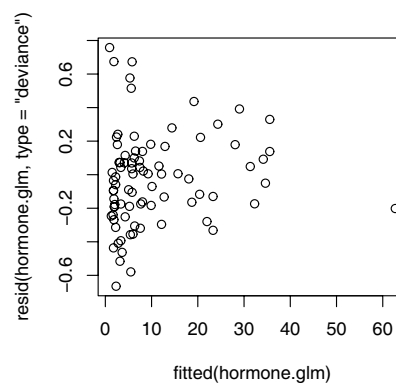
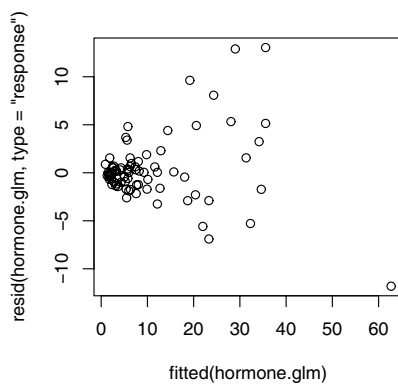
Plot of Fitted Model

```
> attach(hormone)
> plot(Reference, Test)
> abline(coef(hormone.glm))
```



Model Residuals

```
> par(mfrow=c(1,2))
> plot(fitted(hormone.glm), resid(hormone.glm, type="response"))
> plot(fitted(hormone.glm), resid(hormone.glm, type="deviance"))
```



Comments: Model Residuals

- Recall that residuals not uniquely defined for GLMs.
- Response (raw) residuals not appropriately scaled; display fanning.
- Deviance residuals appropriate; do not suggest any major problems with model.
- Can use `plot(hormone.glm)` to get standard set of diagnostic plots for GLM.

Comments on the Fitted Model

- Data do not completely discount the possibility that (true) regression line goes through origin and has slope 1.
- The dispersion parameter estimate in the output indicates that $\sigma_Y = \sqrt{0.0843} \mu_Y$ for responses.
- Could try fitting regression through origin. Use formula
`Test ~ -1 + Reference` to force R to remove intercept in model.

Exercise 1: Survival Times for Leukemia Patients

This exercise concerns data collected on survival times for Leukemia patients in the 1960s. (Data source: Feigl, P. and Zelen, M., (1965), 'Estimation of exponential survival probabilities with concomitant information,' *Biometrics*, **21**, 826–838.) For 33 subjects the following variables are recorded:

WBC:	White blood cell count
AG:	Subject is positive or negative for aminiothuteglucide (AG)
Time:	Time to death, in weeks

We will seek to model survival time as a function of WBC and AG using Gamma regression with the canonical inverse link.

Read in the data by

```
leuk <- read.table(file=choose.files(), header=TRUE)
```

and navigate to where the file `leukemia.txt` is stored on your computer. Take a look at the data frame `leuk`.

Fit the following Gamma regression:

```
leuk.glm <- glm(Time ~ AG + WBC, family=Gamma, data=leuk)
```

Perform an analysis of deviance by

```
anova(leuk.glm, test="Chisq")
```

You should see that there is evidence that both AG and WBC affect survival time. Look at the estimated coefficients (regression parameters) by

```
coef(leuk.glm)
```

Does survival time increase or decrease with WBC?

Exercise 2: Hearing Impairment and Maternal Attachment

This exercise concerns a study to investigate a possible connection between hearing-impairment in toddlers and the development of a secure attachment to their mothers. (Source: A.R. Lederberg and C.E. Mobley, "The effect of hearing impairment on the quality of attachment and mother-toddler interaction," *Child Development*, **61** (1990), pp. 1596-1604.) A group of 41 hearing-impaired toddlers, and another group of 41 toddlers without hearing impairment, were included in the study. For each toddler an assessment was made on whether he/she had a secure, or insecure, attachment to his/her mother. The resulting data are displayed below.

Impaired	Attachment	
	Secure	Insecure
Yes	23	18
No	25	16

A two way contingency table such as this can be modelled using logistic regression. The response in this case is a binomial count of the number of secure attachments from the sample of 41 toddlers. The single factor is the hearing impairment status.

We can create the necessary objects in R as follows.

```
Secure <- c(23, 25)
Insecure <- c(18, 16)
Impaired <- factor(c("yes", "no"))
toddlers <- data.frame(Secure, Insecure, Impaired)
```

Take a look at the data frame `toddlers` that you have just created to ensure that you understand its structure.

For binomial count responses as opposed to simply binary responses, the LHS of the R formula must be a matrix with two columns. The first column contains the number of ‘successes’ (i.e. secure attachments in our case) and the second column contains the corresponding number of ‘failures’ (i.e. insecure attachments in our case). Hence to fit a logistic regression to these data the code is

```
toddlers.glm <- glm( cbind(Secure, Insecure) ~ Impaired, family=binomial, data=toddlers)
```

In this formula the `cbind()` command has been used to bind together two vectors as columns of a matrix. Generate a summary table of parameter estimates and an analysis of deviance table. Is there any evidence that the level of maternal attachment is affected by hearing impairment?

The analysis that you have just done used the canonical logit link for the binomial family GLM. Just for practice, repeat the analysis using a probit link.

Additional Exercise: Analysis of Copenhagen Housing Survey Data

This exercise is intended only for those familiar with log-linear modelling for contingency tables (and who have completed earlier exercises *very* quickly!). It is concerned with data from a housing survey conducted in Copenhagen (in Denmark). A total of 1681 respondents in rental accommodation were classified according to the following variables:

Sat:	Satisfaction of householders with housing circumstances, (High, Medium or Low).
Infl:	Perceived influence householders have on management of property (High, Medium, Low).
Type:	Type of rental accommodation, (Tower, Atrium, Apartment, Terrace).
Cont:	Contact residents are afforded with other residents, (Low, High).

Load that data by

```
housing <- read.table(file=choose.files(), header=TRUE)
```

and navigate to where the file `housing.txt` is stored on your computer.

In this data frame the numbers (frequencies) in each class are listed under the variable `Freq`.

We can analyse the associations between the factors in these contingency table data using a Poisson log-linear model. Fit the saturated model by

```
housing.glm.1 <- glm(Freq ~ Sat*Infl*Type*Cont, family=poisson, data=housing)
```

Confirm that the four way interaction `Sat:Infl:Type:Cont` can be dropped by inspecting the result of `drop1(housing.glm.1, test="Chisq")`

Update the model with the

```
housing.glm.2 <- update(housing.glm.1, . ~ . - Sat:Infl:Type:Cont)
```

Now check if any more terms can be removed by

```
drop1(housing.glm.2, test="Chisq")
```

Repeat this process, dropping one term at a time until no more terms need to be removed from the model.

What do you conclude about the associations between the variables? In particular, what factors affect a householder's level of satisfaction with his or her housing circumstances?

Finishing Off:

When you've finished, close down R by typing `q()`. Choose 'Save' when prompted as to whether you want to retain your workspace.

R Reference Card

by Tom Short, EPRI PEAC, tshort@epri-peac.com 2004-11-07
Granted to the public domain. See www.Rpad.org for the source and latest version. Includes material from *R for Beginners* by Emmanuel Paradis (with permission).

Getting help

Most R functions have online documentation.
help(topic) documentation on `topic`
?topic id.

help.search("topic") search the help system
apropos("topic") the names of all objects in the search list matching the regular expression "topic"

help.start() start the HTML version of help
str(a) display the internal "structure" of an R object
summary(a) gives a "summary" of `a`, usually a statistical summary but it is generic meaning it has different operations for different classes of a
ls() show objects in the search path; specify `pat="pat"` to search on a pattern

ls.str() `str()` for each variable in the search path
dir() show files in the current directory
methods(a) shows S3 methods of `a`
methods(class=class(a)) lists all the methods to handle objects of class `a`

Input and output

load() load the datasets written with `save`
data(x) loads specified data sets
library(x) load add-on packages
read.table(file) reads a file in table format and creates a data frame from it; the default separator `sep=""` is any whitespace; use `header=TRUE` to read the first line as a header of column names; use `as.is=TRUE` to prevent character vectors from being converted to factors; use `comment.char=""` to prevent "#" from being interpreted as a comment; use `skip=n` to skip `n` lines before reading data; see the help for options on row naming, NA treatment, and others
read.csv("filename",header=TRUE) id. but with defaults set for reading comma-delimited files
read.delim("filename",header=TRUE) id. but with defaults set for reading tab-delimited files

read.fwf(file,widths,header=FALSE,sep=" ",as.is=FALSE) read a table of fixed width/formatted data into a "data.frame"; `widths` is an integer vector, giving the widths of the fixed-width fields
save(file,...) saves the specified objects (...) in the XDR platform-independent binary format

save.image(file) saves all objects
cat(...,file="",sep=" ") prints the arguments after coercing to character; `sep` is the character separator between arguments
print(a,...) prints its arguments; generic, meaning it can have different methods for different objects
format(x,...) format an R object for pretty printing
write.table(x,file="",row.names=TRUE,col.names=TRUE,sep=" ") prints `x` after converting to a data frame; if `quote` is `TRUE`,

character or factor columns are surrounded by quotes (""); `sep` is the field separator; `col` is the end-of-line separator; `na` is the string for missing values; use `col.names=NA` to add a blank column header to get the column headers aligned correctly for spreadsheet input
sink(file) output to file, until `sink()`

Most of the I/O functions have a `file` argument. This can often be a character string naming a file or a connection. `file=""` means the standard input or output. Connections can include files, pipes, zipped files, and R variables. On windows, the file connection can also be used with `description = "clipboard"`. To read a table copied from Excel, use `x <- read.delim("clipboard")`

To write a table to the clipboard for Excel, use `write.table(x,"clipboard",sep="\t",col.names=NA)`
For database interaction, see packages RODBC, DBI, RMySQL, RPostgreSQL, ROracle. See packages XML, hdf5, netCDF for reading other file formats.

Data creation

c(...) generic function to combine arguments with the default forming a vector; with `recursive=TRUE` descends through lists combining all elements into one vector

from: to generates a sequence; ":" has operator priority; `1:4 + 1` is "2,3,4,5"
seq(from,to) generates a sequence `by=` specifies increment; `length=` specifies desired length

seq(along=x) generates `1, 2, ..., length(along)`; useful for for loops
rep(x,times) replicate `x` times; use `each=` to repeat "each" element of `x` each times; `rep(c(1,2,3),2)` is `1 2 3 1 2 3`;
`rep(c(1,2,3),each=2)` is `1 1 2 2 3 3`

data.frame(...) create a data frame of the named or unnamed arguments; `data.frame(v=1:4,cb=c("a","B","c","d"),n=10)`; shorter vectors are recycled to the length of the longest

list(...) create a list of the named or unnamed arguments; `list(a=c(1,2),b="hi",c=3i)`;

array(x,dim=) array with data `x`; specify dimensions like `dim=c(3,4,2)`; elements of `x` recycle if `x` is not long enough

matrix(x,nrow=,ncol=) matrix; elements of `x` recycle
factor(x,levels=) encodes a vector `x` as a factor
gl(n,k,length=n*k,labels=1:n) generate levels (factors) by specifying the pattern of their levels; `k` is the number of levels, and `n` is the number of replications

expand.grid() a data frame from all combinations of the supplied vectors or factors
rbind(...) combine arguments by rows for matrices, data frames, and others

cbind(...) id. by columns
Slicing and extracting data

Indexing vectors
`x[n]` n^{th} element
`x[-n]` all *but* the n^{th} element
`x[1:n]` first `n` elements
`x[-(1:n)]` elements from `n+1` to the end
`x[c(1,4,2)]` specific elements
`x["name"]` element named "name"
`x[x > 3]` all elements greater than 3
`x[x > 3 & x < 5]` all elements between 3 and 5
`x[x %in% c("a"," " and", "che")]` elements in the given set

Indexing lists
`x[n]` list with elements `n`
`x[[n]]` n^{th} element of the list
`x[["name"]]` element of the list named "name"
`x$name` id.

Indexing matrices
`x[i,j]` element at row `i`, column `j`
`x[,j]` column `j`
`x[i,]` row `i`
`x[,c(1,3)]` columns 1 and 3
`x["name",]` row named "name"
`x[["name"]]` column named "name"
`x$name` id.

Indexing data frames (matrix indexing plus the following)
`x[["name"]]` column named "name"

Variable conversion

as.array(x), as.data.frame(x), as.numeric(x), as.logical(x), as.complex(x), as.character(x), ... convert type; for a complete list, use `methods(as)`

Variable information

is.na(x), is.null(x), is.array(x), is.data.frame(x), is.numeric(x), is.complex(x), is.character(x), ... test for type; for a complete list, use `methods(is)`
length(x) number of elements in `x`
dim(x) Retrieve or set the dimension of an object; `dim(x) <- c(3,2)`

dimnames(x) Retrieve or set the dimension names of an object
nrow(x) number of rows; `NROW(x)` is the same but treats a vector as a one-row matrix

ncol(x) and `NCOL(x)` id. for columns
class(x) get or set the class of `x`; `class(x) <- "myClass"`

unclass(x) remove the class attribute of `x`
attr(x,which) get or set the attribute which of `x`
attributes(obj) get or set the list of attributes of `obj`

Data selection and manipulation

which.max(x) returns the index of the greatest element of `x`
which.min(x) returns the index of the smallest element of `x`
rev(x) reverses the elements of `x`
sort(x) sorts the elements of `x` in increasing order; to sort in decreasing order: `rev(sort(x))`

cut(x,breaks) divides `x` into intervals (factors); `breaks` is the number of cut intervals or a vector of cut points
match(x,y) returns a vector of the same length than `x` with the elements of `x` which are in `y` (NA otherwise)

which(x == a) returns a vector of the indices of `x` if the comparison operation is true (TRUE), in this example the values of `i` for which `x[i] == a` (the argument of this function must be a variable of mode logical)

choose(n,k) computes the combinations of `k` events among `n` repetitions
`= n!/((n-k)!k!)`

na.omit(x) suppresses the observations with missing data (NA) (suppresses the corresponding line if `x` is a matrix or a data frame)

na.fail(x) returns an error message if `x` contains at least one NA

unique(x) if *x* is a vector or a data frame, returns a similar object but with the duplicate elements suppressed

table(x) returns a table with the numbers of the differents values of *x* (typically for integers or factors)

subset(x, ...) returns a selection of *x* with respect to criteria (...), typically comparisons: `x$V1 < 10`; if *x* is a data frame, the option `select` gives the variables to be kept or dropped using a minus sign

sample(x, size) resample randomly and without replacement `size` elements in the vector *x*, the option `replace = TRUE` allows to resample with replacement

prop.table(x, margin=) table entries as fraction of marginal table

Math

sin, cos, tan, asin, acos, atan, atan2, log, log10, exp

max(x) maximum of the elements of *x*

min(x) minimum of the elements of *x*

range(x) id. then `c(min(x), max(x))`

sum(x) sum of the elements of *x*

diff(x) lagged and iterated differences of vector *x*

prod(x) product of the elements of *x*

mean(x) mean of the elements of *x*

median(x) median of the elements of *x*

quantile(x, probs=) sample quantiles corresponding to the given probabilities (defaults to 0.25, 0.5, 0.75, 1)

weighted.mean(x, w) mean of *x* with weights *w*

rank(x) ranks of the elements of *x*

var(x) or `cov(x)` variance of the elements of *x* (calculated on *n* - 1); if *x* is a matrix or a data frame, the variance-covariance matrix is calculated

sd(x) standard deviation of *x*

cor(x) correlation matrix of *x* if it is a matrix or a data frame (1 if *x* is a vector)

var(x, y) or `cov(x, y)` covariance between *x* and *y*, or between the columns of *x* and those of *y* if they are matrices or data frames

cor(x, y) linear correlation between *x* and *y*, or correlation matrix if they are matrices or data frames

round(x, n) rounds the elements of *x* to *n* decimals

log(x, base) computes the logarithm of *x* with base `base`

scale(x) if *x* is a matrix, centers and reduces the data; to center only use the option `center=FALSE`, to reduce only `scale=FALSE` (by default `center=TRUE, scale=TRUE`)

pmin(x, y, ...) a vector which *i*th element is the minimum of `x[i], y[i], ...`

pmax(x, y, ...) id. for the maximum

cumsum(x) a vector which *i*th element is the sum from `x[1]` to `x[i]`

cumprod(x) id. for the product

cummin(x) id. for the minimum

cummax(x) id. for the maximum

union(x, y), intersect(x, y), setdiff(x, y), setequal(x, y), is.element(e1, set) "set" functions

Re(x) real part of a complex number

Im(x) imaginary part

Mod(x) modulus; `abs(x)` is the same

Arg(x) angle in radians of the complex number

Conj(x) complex conjugate

convolve(x, y) compute the several kinds of convolutions of two sequences

fft(x) Fast Fourier Transform of an array

mvfft(x) FFT of each column of a matrix

filter(x, filter) applies linear filtering to a univariate time series or to each series separately of a multivariate time series

Many math functions have a logical parameter `na.rm=FALSE` to specify missing data (NA) removal.

Matrices

t(x) transpose

diag(x) diagonal

%% matrix multiplication

solve(a, b) solves a `%%` *x* = *b* for *x*

solve(a) matrix inverse of *a*

rowsum(x) sum of rows for a matrix-like object, **rowSums(x)** is a faster version

colsum(x), colSums(x) id. for columns

rowMeans(x) fast version of row means

colMeans(x) id. for columns

Advanced data processing

apply(X, INDEX, FUN=) a vector or array or list of values obtained by applying a function `FUN` to margins (INDEX) of *X*

lapply(X, FUN) apply `FUN` to each element of the list *X*

tapply(X, INDEX, FUN=) apply `FUN` to each cell of a ragged array given by *X* with indexes INDEX

by(data, INDEX, FUN) apply `FUN` to data frame `data` subsetted by INDEX

merge(a, b) merge two data frames by common columns or row names

xtabs(a, b, data=x) a contingency table from cross-classifying factors

aggregate(x, by, FUN) splits the data frame *x* into subsets, computes summary statistics for each, and returns the result in a convenient form; `by` is a list of grouping elements, each as long as the variables in *x*

stack(x, ...) transform data available as separate columns in a data frame or list into a single column

unstack(x, ...) inverse of `stack()`

reshape(x, ...) reshapes a data frame between 'wide' format with repeated measurements in separate columns of the same record and 'long' format with the repeated measurements in separate records; use `(direction="wide")` or `(direction="long")`

Strings

paste(...) concatenate vectors after converting to character; `sep=` is the string to separate terms (a single space is the default); `collapse=` is an optional string to separate "collapsed" results

substr(x, start, stop) substrings in a character vector; can also as sign, as `substr(x, start, stop) <- value`

strsplit(x, split) split *x* according to the substring `split`

grep(pattern, x) searches for matches to `pattern` within *x*; see `?regex`

gsub(pattern, replacement, x) replacement of matches determined by regular expression matching `sub()` is the same but only replaces the first occurrence.

tolower(x) convert to lowercase

toupper(x) convert to uppercase

match(x, table) a vector of the positions of first matches for the elements of *x* among `table`

x %in% table id. but returns a logical vector

pmatch(x, table) partial matches for the elements of *x* among `table`

nchar(x) number of characters

Dates and Times

The class `Date` has dates without times, `POSIXct` has dates and times, including time zones. Comparisons (e.g. `>`), `seq()`, and `difftime()` are useful. `Date` also allows `+` and `-`. `?DateTimeClasses` gives more information. See also package `chron`.

as.Date(s) and **as.POSIXct(s)** convert to the respective class; `format(dt)` converts to a string representation. The default string format is "2001-02-21". These accept a second argument to specify a format for conversion. Some common formats are:

`%a, %A` Abbreviated and full weekday name.

`%b, %B` Abbreviated and full month name.

`%d` Day of the month (01-31).

`%H` Hours (00-23).

`%I` Hours (01-12).

`%j` Day of year (001-366).

`%m` Month (01-12).

`%M` Minute (00-59).

`%p` AM/PM indicator.

`%S` Second as decimal number (00-61).

`%U` Week (00-53); the first Sunday as day 1 of week 1.

`%W` Week (0-6, Sunday is 0).

`%x` Weekday (0-6, Sunday as day 1 of week 1).

`%Y` Year without century (00-99). Don't use.

`%y` Year with century.

`%z` (output only.) Offset from Greenwich; -0800 is 8 hours west of.

`%Z` (output only.) Time zone as a character string (empty if not available).

Where leading zeros are shown they will be used on output but are optional on input. See `?strftime`.

Plotting

plot(x) plot of the values of *x* (on the y-axis) ordered on the *x*-axis

plot(x, y) bivariate plot of *x* (on the *x*-axis) and *y* (on the *y*-axis)

hist(x) histogram of the frequencies of *x*

barplot(x) histogram of the values of *x*; use `horiz=FALSE` for horizontal bars

dotchart(x) if *x* is a data frame, plots a Cleveland dot plot (stacked plots line-by-line and column-by-column)

pie(x) circular pie-chart

boxplot(x) "box-and-whiskers" plot

sunflowerplot(x, y) id. than `plot()` but the points with similar coordinates are drawn as flowers which petal number represents the number of points

stripplot(x) plot of the values of *x* on a line (an alternative to `boxplot()` for small sample sizes)

coplot(x~y | z) bivariate plot of *x* and *y* for each value or interval of values of *z*

interaction.plot(f1, f2, y) if `f1` and `f2` are factors, plots the means of *y* (on the *y*-axis) with respect to the values of `f1` (on the *x*-axis) and of `f2` (different curves); the option `fun=mean` allows to choose the summary statistic of *y* (by default `fun=mean`)

matplot(x, y) bivariate plot of the first column of *x* vs. the first one of *y*, the second one of *x* vs. the second one of *y*, etc.

fourfoldplot(x) visualizes, with quarters of circles, the association between two dichotomous variables for different populations (*x* must be an array with `dim=c(2, 2, k)`), or a matrix with `dim=c(2, 2)` if *k*=1)

assocplot(x) Cohen-Friendly graph showing the deviations from independence of rows and columns in a two dimensional contingency table

mosaicplot(x) 'mosaic' graph of the residuals from a log-linear regression of a contingency table

pairs(x) if *x* is a matrix or a data frame, draws all possible bivariate plots between the columns of *x*

plot.ts(x) if *x* is an object of class "ts", plot of *x* with respect to time, *x* may be multivariate but the series must have the same frequency and dates

ts.plot(x) id. but if *x* is multivariate the series may have different dates and must have the same frequency

qnorm(x) quantiles of *x* with respect to the values expected under a normal law

qqplot(x, y) quantiles of *y* with respect to the quantiles of *x*

contour(x, y, z) contour plot (data are interpolated to draw the curves), *x* and *y* must be vectors and *z* must be a matrix so that `dim(z)=c(length(x), length(y))` (*x* and *y* may be omitted)

filled.contour(x, y, z) id. but the areas between the contours are coloured, and a legend of the colours is drawn as well

image(x, y, z) id. but with colours (actual data are plotted)

persp(x, y, z) id. but in perspective (actual data are plotted)

stars(x) if *x* is a matrix or a data frame, draws a graph with segments or a star where each row of *x* is represented by a star and the columns are the lengths of the segments

symbols(x, y, ...) draws, at the coordinates given by *x* and *y*, symbols (circles, squares, rectangles, stars, thermometres or "boxplots") which sizes, colours ... are specified by supplementary arguments

termpLOT(mod, obj) plot of the (partial) effects of a regression model (mod, obj)

The following parameters are common to many plotting functions:
add=FALSE if TRUE superposes the plot on the previous one (if it exists)
axes=TRUE if FALSE does not draw the axes and the box

type="p" specifies the type of plot, "p": points, "l": lines, "b": points connected by lines, "o": id. but the lines are over the points, "h": vertical lines, "s": steps, the data are represented by the top of the vertical lines, "S": id. but the data are represented by the bottom of the vertical lines

xlim, ylim specifies the lower and upper limits of the axes, for example with `xlim=c(1, 10)` or `xlim=range(x)`
xlab, ylab annotates the axes, must be variables of mode character
main= main title, must be a variable of mode character
sub= sub-title (written in a smaller font)

Low-level plotting commands

points(x, y) adds points (the option `type=` can be used)
lines(x, y) id. but with lines
text(x, y, labels, ...) adds text given by `labels` at coordinates (*x*,*y*): a typical use is: `plot(x, y, type="n"); text(x, y, names)`

mtext(text, side=3, line=0, ...) adds text given by `text` in the margin specified by `side` (see `axis()` below); `line` specifies the line from the plotting area

segments(x0, y0, x1, y1) draws lines from points (*x0*,*y0*) to points (*x1*,*y1*)

arrows(x0, y0, x1, y1, angle=30, code=2) id. with arrows at points (*x0*,*y0*) if `code=2`, at points (*x1*,*y1*) if `code=1`, or both if `code=3`; `angle` controls the angle from the shaft of the arrow to the edge of the arrow head

abline(a, b) draws a line of slope *b* and intercept *a*

abline(h=y) draws a horizontal line at ordinate *y*

abline(v=x) draws a vertical line at abscissa *x*

abline(lm.obj) draws the regression line given by `lm.obj`

rect(x1, y1, x2, y2) draws a rectangle which left, right, bottom, and top limits are *x1*, *x2*, *y1*, and *y2*, respectively

polygon(x, y) draws a polygon linking the points with coordinates given by *x* and *y*

legend(x, y, legend) adds the legend at the point (*x*,*y*) with the symbols given by `legend`

title() adds a title and optionally a sub-title

axis(side, vect) adds an axis at the bottom (`side=1`), on the left (`2`), at the top (`3`), or on the right (`4`); `vect` (optional) gives the abscissa (or ordinate) where tick-marks are drawn

rug(x) draws the data *x* on the *x*-axis as small vertical lines

locator(n, type="n", ...) returns the coordinates (*x*,*y*) after the user has clicked *n* times on the plot with the mouse; also draws symbols (`type="p"`) or lines (`type="l"`) with respect to optional graphic parameters (...); by default nothing is drawn (`type="n"`)

Graphical parameters

These can be set globally with `par(...)`: many can be passed as parameters to plotting commands.

adj controls text justification (0 left-justified, 0.5 centred, 1 right-justified)

bg specifies the colour of the background (ex.: `bg="red"`, `bg="blue"`, ... the list of the 657 available colours is displayed with `colors()`)

bty controls the type of box drawn around the plot, allowed values are: "o", "l", "7", "c", "n", "u", "j" (the box looks like the corresponding character); if `bty="n"` the box is not drawn

cex a value controlling the size of texts and symbols with respect to the default; the following parameters have the same control for numbers on the axes, `cex.axis`, the axis labels, `cex.lab`, the title, `cex.main`, and the sub-title, `cex.sub`

col controls the color of symbols and lines; use color names: "red", "blue" see `colors()` or as "#RRGGBB"; see `rgb()`, `hsv()`, `gray()`, and `rainbow()`; as for `cex` there are: `col.axis`, `col.lab`, `col.main`, `col.sub`

font an integer which controls the style of text (1: normal, 2: italics, 3: bold, 4: bold italics); as for `cex` there are: `font.axis`, `font.lab`, `font.main`, `font.sub`

las an integer which controls the orientation of the axis labels (0: parallel to the axes, 1: horizontal, 2: perpendicular to the axes, 3: vertical)

lty controls the type of lines, can be an integer or string (1: "solid", 2: "dashed", 3: "dotted", 4: "dotted", 5: "longdash", 6: "twodash", or a string of up to eight characters (between "0" and "9") which specifies alternatively the length, in points or pixels, of the drawn elements and the blanks, for example `lty="44"` will have the same effect than `lty=2`

lwd a numeric which controls the width of lines, default 1

mar a vector of 4 numeric values which control the space between the axes and the border of the graph of the form `c(bottom, left, top, right)`, the default values are `c(5.1, 4.1, 4.1, 2.1)`

mfcol a vector of the form `c(nr, nc)` which partitions the graphic window as a matrix of *nr* lines and *nc* columns, the plots are then drawn in columns

mflow id. but the plots are drawn by row

pch controls the type of symbol, either an integer between 1 and 25, or any single character within "

10 2Δ 3+ 4 X 5◇ 6▽ 7⊠ 8* 9⊕ 10⊗ 11⊠ 12⊠ 13⊠ 14⊠ 15■
16● 17▲ 18◆ 19● 20● 21○ 22◇ 23◇ 24Δ 25▽ * . . X X a a ? ?

ps an integer which controls the size in points of texts and symbols

pty a character which specifies the type of the plotting region, "s": square, "m": maximal

tick a value which specifies the length of tick-marks on the axes as a fraction of the smallest of the width or height of the plot; if `tick=1` a grid is drawn

tcl a value which specifies the length of tick-marks on the axes as a fraction of the height of a line of text (by default `tcl=-0.5`)

xaxt if `xaxt="n"` the *x*-axis is set but not drawn (useful in conjunction with `axis(side=1, ...)`)

yaxt if `yaxt="n"` the *y*-axis is set but not drawn (useful in conjunction with `axis(side=2, ...)`)

Lattice (Trellis) graphics

xplot(y~x) bivariate plots (with many functionalities)

barchart(y~x) histogram of the values of *y*, with respect to those of *x*

dotplot(y~x) Cleveland dot plot (stacked plots line-by-line and column-by-column)

densityplot(~x) density functions plot

histogram(~x) histogram of the frequencies of *x*

bwplot(y~x) "box-and-whiskers" plot

qqmath(~x) quantiles of *x* with respect to the values expected under a theoretical distribution

stripplot(y~x) single dimension plot, *x* must be numeric, *y* may be a factor

qq(y~x) quantiles to compare two distributions, *x* must be numeric, *y* may be numeric, character, or factor but must have two 'levels'

sploM(~x) matrix of bivariate plots

parallel(~x) parallel coordinates plot

levelplot(z~x*y | g1+g2) coloured plot of the values of *z* at the coordinates given by *x* and *y* (*x*,*y* and *z* are all of the same length)

wireframe(z~x*y | g1+g2) 3d surface plot

cloud(z~x*y | g1+g2) 3d scatter plot

In the normal Lattice formula, $y \sim |q1 * g2$ has combinations of optional conditioning variables `g1` and `g2` plotted on separate panels. Lattice functions take many of the same arguments as base graphics plus also `data`= the data frame for the formula variables and `subset`= for subsetting. Use `panel=` to define a custom panel function (see `apropos("panel")` and `?lattice`). Lattice functions return an object of class `trellis` and have to be printed to produce the graph. Use `print(xyplot(...))` inside functions where automatic printing doesn't work. Use `lattice.theme` and `lset` to change Lattice defaults.

Optimization and model fitting

`optim(par, fn, method = c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN"))` general-purpose optimization; `par` is initial values, `fn` is function to optimize (normally minimize) `nlm(f, p)` minimize function `f` using a Newton-type algorithm with starting values `p`

`lm(formula)` fit linear models; `formula` is typically of the form `response ~ termA + termB + ...`; use `I(x*y)` + `I(x^2)` for terms made of nonlinear components

`glm(formula, family=)` fit generalized linear models, specified by giving a symbolic description of the linear predictor and a description of the error distribution; `family` is a description of the error distribution and link function to be used in the model, see `?family`

`nls(formula)` nonlinear least-squares estimates of the nonlinear model parameters

`approx(x, y=)` linearly interpolate given data points; `x` can be an `xyplot`-ing structure

`spline(x, y=)` cubic spline interpolation

`loess(formula)` fit a polynomial surface using local fitting
Many of the formula-based modeling functions have several common arguments: `data`= the data frame for the formula variables, `subset`= a subset of variables used in the fit, `na.action`= action for missing values: "na.fail", "na.omit", or a function. The following generics often apply to model fitting functions:

`predict(fit, ...)` predictions from `fit` based on input data
`df.residual(fit)` returns the number of residual degrees of freedom
`coef(fit)` returns the estimated coefficients (sometimes with their standard-errors)

`residuals(fit)` returns the residuals

`deviance(fit)` returns the deviance

`fitted(fit)` returns the fitted values

`logLik(fit)` computes the logarithm of the likelihood and the number of parameters

`AIC(fit)` computes the Akaike information criterion or AIC

Statistics

`aov(formula)` analysis of variance model

`anova(fit, ...)` analysis of variance (or deviance) tables for one or more fitted model objects

`density(x)` kernel density estimates of `x`

`binom.test()`, `pairwise.t.test()`, `power.t.test()`,

`prop.test()`, `t.test()`, ... use `help.search("test")`

Distributions

`rnorm(n, mean=0, sd=1)` Gaussian (normal)

`rexp(n, rate=1)` exponential

`rgamma(n, shape, scale=1)` gamma

`rpois(n, lambda)` Poisson
`rweibull(n, shape, scale=1)` Weibull
`rcauchy(n, location=0, scale=1)` Cauchy
`rbeta(n, shape1, shape2)` beta
`rt(n, df)` 'Student' (t)
`rf(n, df1, df2)` Fisher-Snedecor (F) (χ^2)
`rchisq(n, df)` Pearson
`rbinom(n, size, prob)` binomial
`rgeom(n, prob)` geometric
`rhyper(nn, m, n, k)` hypergeometric
`rlogis(n, location=0, scale=1)` logistic
`rlnorm(n, meanlog=0, sdlog=1)` lognormal
`rlnbinom(n, size, prob)` negative binomial
`runif(n, min=0, max=1)` uniform
`rwilcox(nn, m, n)`, `rsignrank(nn, n)` Wilcoxon's statistics
All these functions can be used by replacing the letter `r` with `d`, `p` or `q` to get, respectively, the probability density (`dfunc(x, ...)`), the cumulative probability density (`pfunc(x, ...)`), and the value of quantile (`qfunc(p, ...)`), with $0 < p < 1$.

Programming

`function(arglist)` `expr` function definition

`return(value)`

`if(cond) expr`

`if(cond) cons.expr else alt.expr`

`for(var in seq) expr`

`while(cond) expr`

`repeat expr`

`break`

`next`

Use braces `{}` around statements

`ifelse(test, yes, no)` a value with the same shape as `test` filled

with elements from either `yes` or `no`

`do.call(funname, args)` executes a function call from the name of the function and a list of arguments to be passed to it

R reference card, by Jonathan Baron

Parentheses are for functions, brackets are for indicating the position of items in a vector or matrix. (Here, items with numbers like x1 are user-supplied variables.)

Miscellaneous

q(): quit
<-: assign
INSTALL package1: install package1
m1[,2]: column 2 of matrix m1
m1[,2:5] or m1[,c(2,3,4,5)]: columns 2-5
m1\$a1: variable a1 in data frame m1
NA: missing data
is.na: true if data missing
library(mva): load (e.g.) the mva package

Help

help(command1): get help with command1 (NOTE: USE THIS FOR MORE DETAIL THAN THIS CARD CAN PROVIDE.)
help.start(): start browser help
help(package=mva): help with (e.g.) package mva
apropos("topic1"): commands relevant to topic1
example(command1): examples of command1

Input and output

source("file1"): run the commands in file1.
read.table("file1"): read in data from file1
data.entry(): spreadsheet
scan(x1): read a vector x1
download.file(url1): from internet
url.show(url1), read.table.url(url1): remote input
sink("file1"): output to file1, until sink()
write(object, "file1"): writes an object to file1
write.table(dataframe1, "file1"): writes a table

Managing variables and objects

attach(x1): put variables in x1 in search path
detach(x1): remove from search path
ls(): lists all the active objects.
rm(object1): removes object1
dim(matrix1): dimensions of matrix1
dimnames(x1): names of dimensions of x1
length(vector1): length of vector1
1:3: the vector 1,2,3
c(1,2,3): creates the same vector
rep(x1,n1): repeats the vector x1 n1 times
cbind(a1,b1,c1), rbind(a1,b1,c1): binds columns or rows into a matrix
merge(df1,df2): merge data frames
matrix(vector1,r1,c1): make vector1 into a matrix with r1 rows and c1 columns
data.frame(v1,v2): make a data frame from vectors v1 and v2

as.factor(), as.matrix(), as.vector(): conversion
is.factor(), is.matrix(), is.vector(): what it is
t(): switch rows and columns
which(x1==a1): returns indices of x1 where x1==a1

Control flow

for (i1 in vector1): repeat what follows
if (condition1) ...else ...: conditional

Arithmetic

%*%: matrix multiplication
%/%, ^, %%, sqrt(): integer division, power, modulus, square root

Statistics

max(), min(), mean(), median(), sum(), var(): as named
summary(data.frame): prints statistics
rank(), sort() rank and sort
ave(x1,y1): averages of x1 grouped by factor y1
by(): apply function to data frame by factor
apply(x1,n1,function1): apply function1 (e.g. mean) to x by rows (n1=1) or columns (n2=2)
tapply(x1,list1,function1): apply function to x1 by list1
table(): make a table
tabulate(): tabulate a vector

basic statistical analysis

aov(), anova(), lm(), glm(): linear and nonlinear models, anova
t.test(): t test
prop.test(), binom.test(): sign test
chisq.test(x1): chi-square test on matrix x1
fisher.test(): Fisher exact test
cor(a): show correlations
cor.test(a,b): test correlation
friedman.test(): Friedman test

some statistics in mva package

prcomp(): principal components
kmeans(): kmeans cluster analysis
factanal(): factor analysis
cancor(): canonical correlation

Graphics

plot(), barplot(), boxplot(), stem(), hist(): basic plots
matplot(): matrix plot
pairs(matrix): scatterplots
coplot(): conditional plot
stripplot(): strip plot
qqplot(): quantile-quantile plot
qqnorm(), qqline(): fit normal distribution

Analysing spatial point patterns in R

Adrian Baddeley
CSIRO and University of Western Australia

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

Workshop Notes
Version 4.1
December 2010

Copyright ©CSIRO 2010

Abstract

This is a detailed set of notes for a workshop on *Analysing spatial point patterns in R*, presented by the author in Australia and New Zealand since 2006.

The goal of the workshop is to equip researchers with a range of practical techniques for the statistical analysis of spatial point patterns. Some of the techniques are well established in the applications literature, while some are very recent developments. The workshop is based on `spatstat`, a contributed library for the statistical package R, which is free open source software.

Topics covered include: statistical formulation and methodological issues; data input and handling; R concepts such as classes and methods; exploratory data analysis; nonparametric intensity and risk estimates; goodness-of-fit testing for Complete Spatial Randomness; maximum likelihood inference for Poisson processes; spatial logistic regression; model validation for Poisson processes; exploratory analysis of dependence; distance methods and summary functions such as Ripley's K function; simulation techniques; non-Poisson point process models; fitting models using summary statistics; LISA and local analysis; inhomogeneous K -functions; Gibbs point process models; fitting Gibbs models; simulating Gibbs models; validating Gibbs models; multitype and marked point patterns; exploratory analysis of multitype and marked point patterns; multitype Poisson process models and maximum likelihood inference; multitype Gibbs process models and maximum pseudolikelihood; line segment patterns, 3-dimensional point patterns, multidimensional space-time point patterns, replicated point patterns, and stochastic geometry methods.

These notes require R **version 2.10.0** or later, and `spatstat` **version 1.21-2** or later.

Acknowledgements

The author gratefully acknowledges countless comments and suggestions from workshop participants and colleagues, and the support of CSIRO MATHEMATICS INFORMATICS AND STATISTICS, THE UNIVERSITY OF WESTERN AUSTRALIA, THE STATISTICAL SOCIETY OF AUSTRALIA, THE NEW ZEALAND STATISTICAL ASSOCIATION, and THE UNIVERSITY OF WAIKATO.

Copyright ©CSIRO Australia 2010

All rights are reserved. Permission to reproduce individual copies of this document for personal use is granted. Redistribution in any other form is prohibited.

The information contained in this document is based on a number of technical, circumstantial or otherwise specified assumptions and parameters. The user must make its own analysis and assessment of the suitability of the information or material contained in or generated from this document. To the extent permitted by law, CSIRO excludes all liability to any party for any expenses, losses, damages and costs arising directly or indirectly from using this document.

Contents

PART I. OVERVIEW	5
1 Introduction	6
2 Statistical formulation	13
3 The R system	18
4 Introduction to spatstat	20
PART II. DATA TYPES & DATA ENTRY	31
5 Objects, classes and methods in R	32
6 Entering point pattern data into spatstat	38
7 Converting from GIS formats	45
8 Windows in spatstat	46
9 Manipulating point patterns	53
10 Pixel images in spatstat	63
11 Tessellations	71
PART III. INTENSITY	77
12 Exploring intensity	78
13 Dependence of intensity on a covariate	82
PART IV. POISSON MODELS	87
14 Tests of Complete Spatial Randomness	88
15 Maximum likelihood for Poisson processes	95
16 Checking a fitted Poisson model	106
17 Spatial logistic regression	112
PART V. INTERACTION	113
18 Exploring dependence between points	114
19 Distance methods for point patterns	115
20 Simulation envelopes and goodness-of-fit tests	132
21 Spatial bootstrap methods	139

22 Simple models of non-Poisson patterns	139
23 Model-fitting using summary statistics	144
24 Exploring local features	148
25 Adjusting for inhomogeneity	149
PART VI. GIBBS MODELS	155
26 Gibbs models	156
27 Fitting Gibbs models	162
28 Validation of fitted Gibbs models	171
PART VII. MARKED POINT PATTERNS	177
29 Marked point patterns	178
30 Handling marked point pattern data	181
31 Exploratory tools for multitype point patterns	187
32 Exploratory tools for marked point patterns	200
33 Multitype Poisson models	204
34 Gibbs models for multitype point patterns	210
PART VIII. HIGHER DIMENSIONS AND OTHER SPATIAL DATA	215
35 Line segment data	216
36 Point patterns in 3D	218
37 Point patterns in multi-dimensional space-time	219
38 Replicated data and hyperframes	221
39 Stochastic geometry	222
40 Further information on spatstat	224
Bibliography	225
Index	228

PART I. OVERVIEW

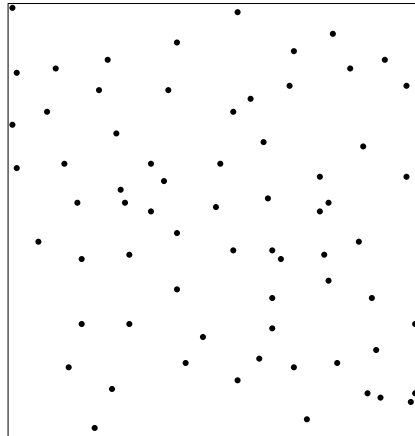
The first part of the workshop is a quick overview of spatial statistics for point patterns, and a very quick introduction to the software.

1 Introduction

1.1 Types of data

1.1.1 Points

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



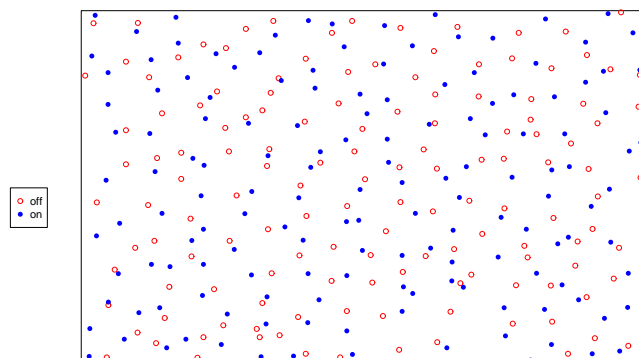
The points could represent trees, animal nests, earthquake epicentres, petty crimes, domiciles of new cases of influenza, galaxies, etc.

The points might be situated in a region of the two-dimensional (2D) plane, or on the Earth's surface, or a 3D volume, etc. They could be points in space-time (e.g. earthquake epicentre location and time).

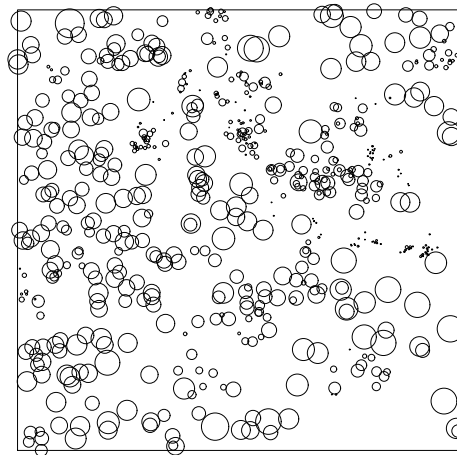
The **spatstat** package was originally implemented for 2D point patterns. However it is being extended progressively to 3D, space-time, and multi-dimensional space-time point patterns (see Sections 36–37).

1.1.2 Marks

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



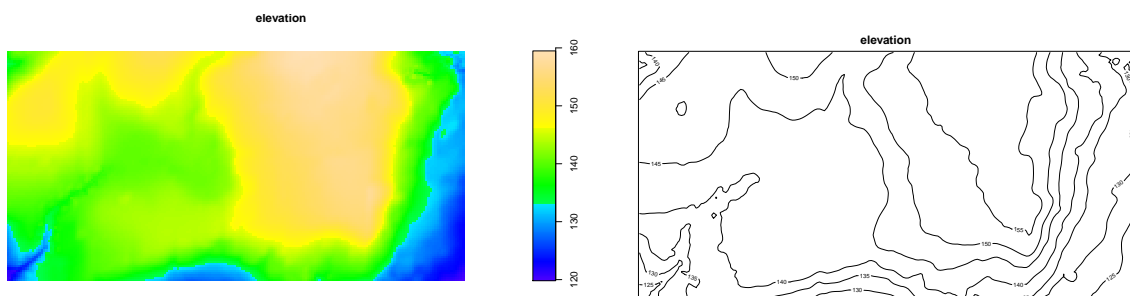
The mark variable could be *continuous*, e.g. tree diameter:



The mark could be multivariate (for example, a tree could be marked by its species and its diameter) or even more complicated.

1.1.3 Covariates

Our dataset may also include **covariates** — any data that we treat as explanatory, rather than as part of the ‘response’. Covariate data may be of any kind. One type of covariate is a *spatial function* $Z(u)$ defined at all spatial locations u , e.g. terrain altitude. Such functions can be displayed as a pixel image or a contour plot:



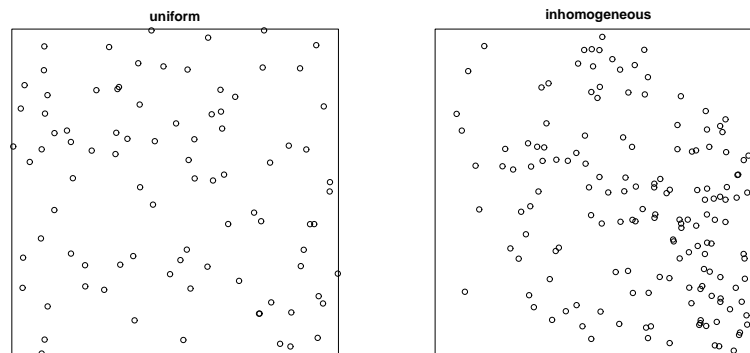
Another common type of covariate data is a *spatial pattern* such as another point pattern, or a line segment pattern, e.g. a map of geological faults:



1.2 Typical scientific questions

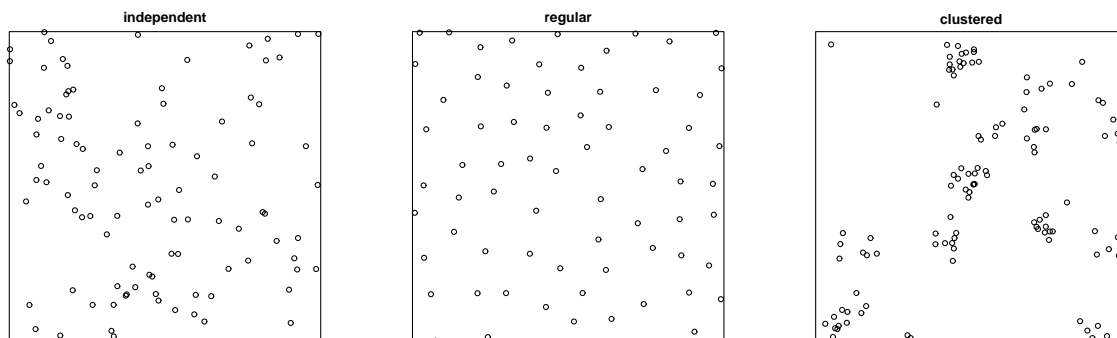
1.2.1 Intensity

‘Intensity’ is the average density of points (expected number of points per unit area). It measures the ‘abundance’ or ‘frequency’ of the events recorded by the points. Intensity may be constant (‘uniform’ or ‘homogeneous’) or may vary from location to location (‘non-uniform’ or ‘inhomogeneous’).



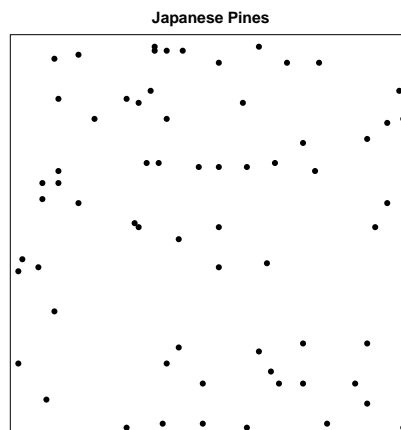
1.2.2 Interaction

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



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

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



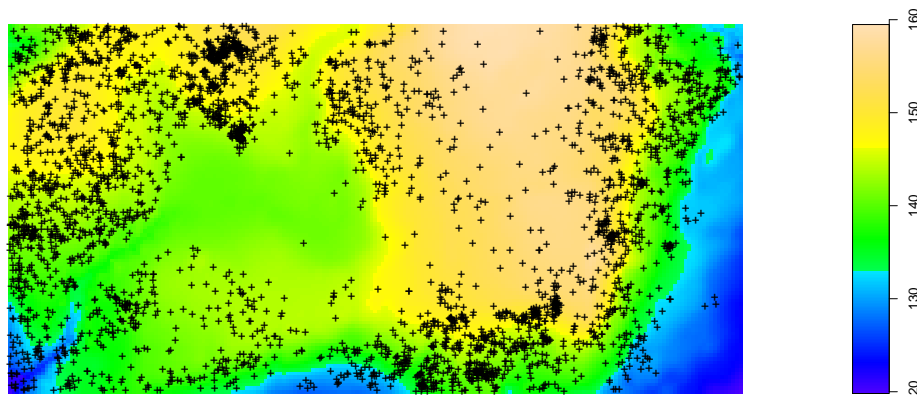
1.2.3 Covariate effects

For a point pattern dataset with covariate data, we typically want to

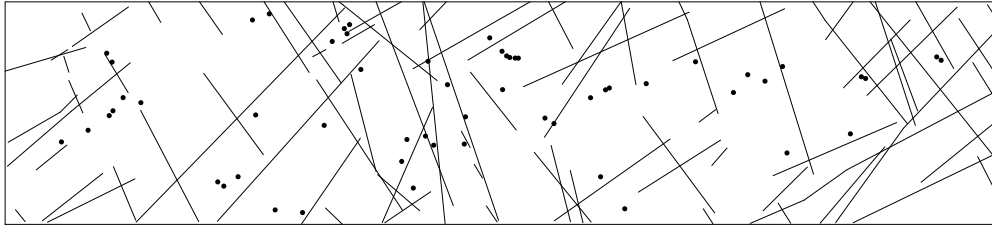
- investigate whether the intensity depends on the covariates
- allow for covariate effects on intensity before studying interaction between points

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

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

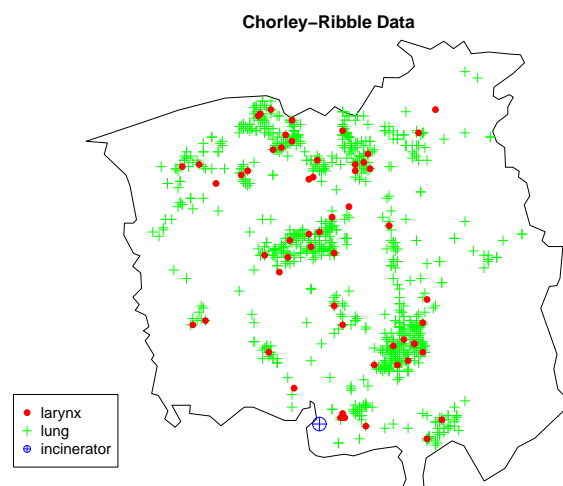


Example 3 (Queensland copper data) *A intensive mineralogical survey yields a map of copper deposits (essentially pointlike at this scale) and geological faults (straight lines). The faults can easily be observed from satellites, but the copper deposits are hard to find. The main question is whether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



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

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

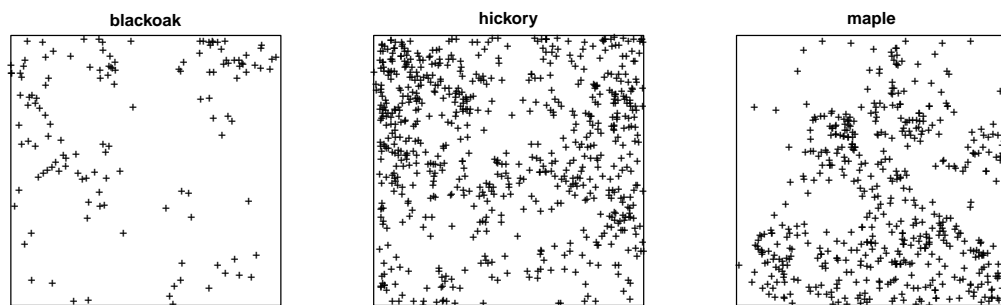


1.2.4 Segregation of points with different marks

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

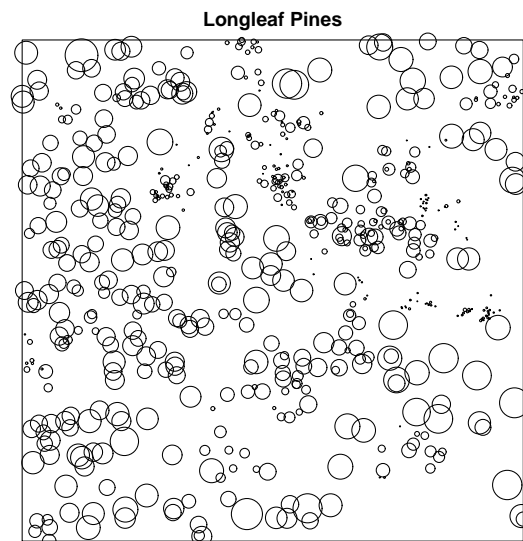
Example 5 (Lansing Woods) *In a 20-acre study region in Lansing Woods, Michigan, the locations of 2251 trees and the botanical classification of each tree were recorded.*

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



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

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



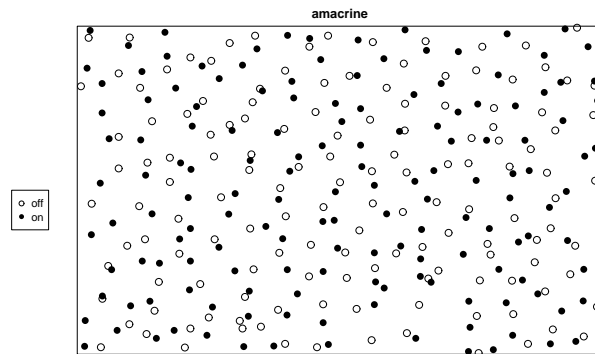
1.2.5 Dependence between points of different types

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

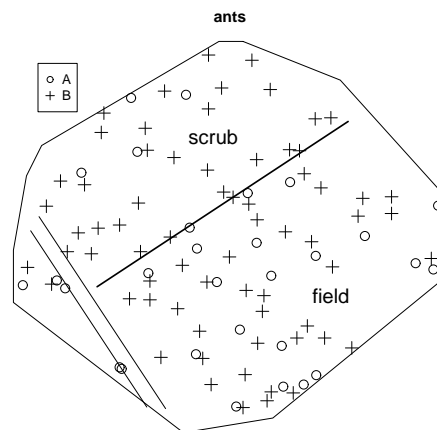
- interaction between the sub-pattern of points of type i and the sub-pattern of points of type j ; or
- dependence between the mark values of points at two specified locations.

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

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



Example 8 (Ants' nests) *The nests of two species of ants in a plot in Greece were mapped. Auxiliary information records a field/scrub boundary, and the position of a walking track. Main question: does species A intentionally place its nests close to species B?*



1.3 Overview of statistical methods

Statistical methods for spatial point patterns have a quirky history. Although there is a highly-developed branch of probability theory for *point processes*, the corresponding statistical methodology is relatively underdeveloped. Until recently, practical techniques for analysing spatial point patterns were often developed in application areas (notably forestry, ecology, geology, geography and astronomy) rather than in statistical science. Techniques include:

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

We'll cover both classical and modern methods. Useful textbooks include [24, 30, 35, 44, 61, 51]. An important recent survey is [50].

2 Statistical formulation

2.1 Point processes

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

2.2 Should I treat the data as a point process?

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

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

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

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

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

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

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

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

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

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

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

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

This is a designed experiment in which the response is a point pattern. Methods for this problem are in their infancy. There are some methods for *replicated* spatial point patterns [15, 19, 36, 37, 42] that apply when each experimental group contains several point patterns.

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

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

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

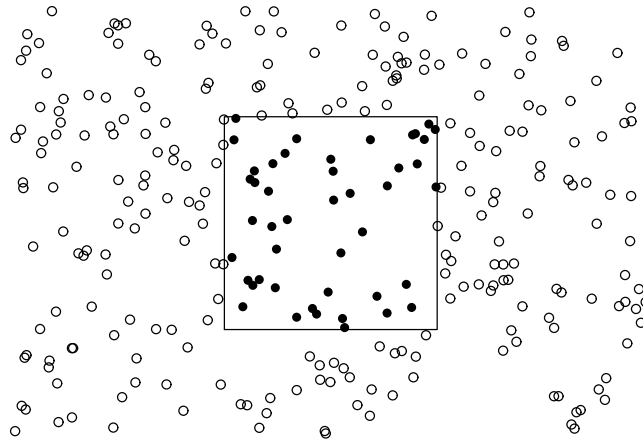
For more discussion about these topics, see [3].

2.3 Assumptions about the data

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

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

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



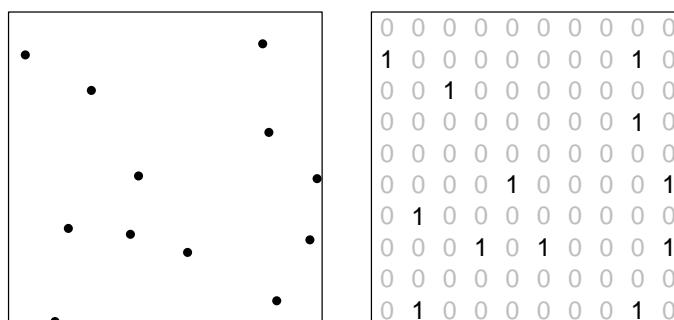
Data are often supplied without information about the sampling window W . **It is important to know the window W** , since we need to know where points were *not* observed. Even something as simple as estimating the density of points depends on the window. It would be wrong, or at least different, to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

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

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

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

When thinking about methodological issues it’s often useful to think about the discretised version of a point process. Suppose the window W is chopped into a large number of tiny ‘pixels’. Each pixel is assigned the value $I = 1$ if it contains a point of \mathbf{X} , and $I = 0$ otherwise. This array of 0’s and 1’s constitutes the data that must be modelled. Thus we need to know where points did *not* occur, as well as where they *did* occur.



To investigate the dependence of these indicators on a covariate, we need to observe the covariate value at some locations where $I = 0$, and not only at locations where $I = 1$.

2.4 Marks and covariates

The main differences between marks and covariates are that

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

2.4.1 Marks

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

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

A marked point pattern is an unordered set

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

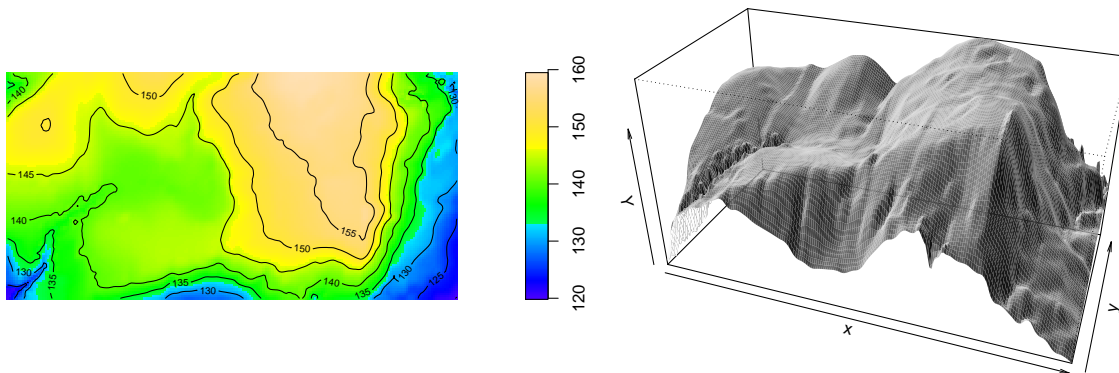
where x_i are the locations and m_i are the corresponding marks.

Marked point patterns are discussed in detail in section 29.

2.4.2 Covariates

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

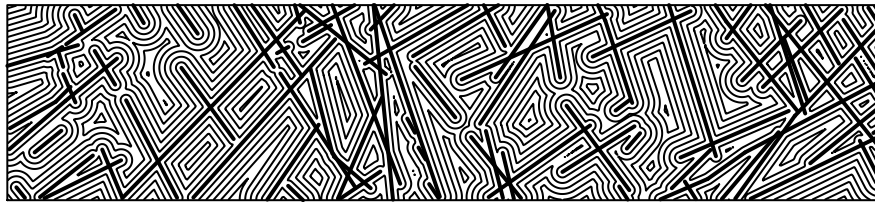
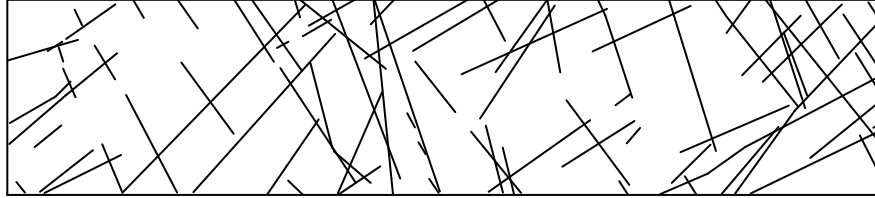
A ‘spatial function’, ‘spatial covariate’ or ‘geostatistical covariate’ is a function $Z(u)$ observable (potentially) at every spatial location $u \in W$. Values of $Z(u)$ may be available for a fine grid of locations u , for example, a terrain elevation map:



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

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

example, $Z(u)$ may be the distance from u to the nearest line segment. Here is a line segment dataset representing the locations of geological faults, and its distance function Z :



3 The R system

We will be using the statistical package R.

3.1 How to obtain R

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

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

3.2 How commands are printed in the notes

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

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

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

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

```
> pi/2

[1] 1.570796

> sin(pi/2)

[1] 1

> x <- sqrt(2)
> x

[1] 1.414214
```

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

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

[1] -0.09132148
```

3.3 Contributed libraries for R

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

Packages that may be useful for analysing spatial data are listed under the Spatial Task View (follow the links to *Task Views — Spatial* on `cran.r-project.org`). For spatial point pattern data, the useful packages include:

<code>adehabitat</code>	habitat selection analysis
<code>ads</code>	spatial point pattern analysis
<code>ash</code>	(includes functions for hexagonal binning)
<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>fields</code>	curve and function fitting
<code>geoR</code>	model-based geostatistical methods
<code>geoRglm</code>	model-based geostatistical methods
<code>GeoXp</code>	interactive spatial exploratory data analysis
<code>maptools</code>	geographical information systems
<code>MarkedPointProcess</code>	nonparametric analysis of marked spatial point processes
<code>RArcInfo</code>	interface to ArcInfo system and data format
<code>rgdal</code>	interface to GDAL geographical data analysis
<code>SGCS</code>	spatial graph techniques for detecting clusters
<code>sp</code>	base library for some spatial data analysis packages
<code>sparr</code>	analysis of spatially varying relative risk
<code>spatgraphs</code>	graphs constructed from spatial point patterns
<code>spatialCovariance</code>	spatial covariance for data on grids
<code>spatialkernel</code>	interpolation and segregation of point patterns
<code>spatialsegregation</code>	segregation of multitype point patterns
<code>spatstat</code>	Spatial point pattern analysis and modelling
<code>spBayes</code>	Gaussian spatial process MCMC (grid data)
<code>spdep</code>	spatial statistics for variables observed at fixed sites
<code>spgwr</code>	geographically weighted regression
<code>splancs</code>	spatial and space-time point pattern analysis
<code>spsurvey</code>	spatial survey methods
<code>trip</code>	spatial trip data formats
<code>tripEstimation</code>	analysis of spatial trip data

To make use of a package, you need to:

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

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

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

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

4 Introduction to spatstat

4.1 The spatstat package

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

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

Spatstat is one of the largest contributed packages available for R, containing over 1000 user-level functions and a 750-page manual. It has its own web domain, `www.spatstat.org`, offering information about the package.

Spatstat can be downloaded from `cran.r-project.org` (under ‘Contributed packages — **spatstat**’). To install **spatstat** you will also need to download the package `deldir` (some other packages are also recommended but not compulsory).

4.2 Please acknowledge spatstat

If you use **spatstat** for research that leads to publications, it would be much appreciated if you could acknowledge **spatstat** in your publications, preferably citing [10]. Citations help us to justify the expenditure of time and effort on maintaining and developing the package.

4.3 Getting started

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

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

```
> library(spatstat)
```

The response will be something like this:

```
deldir 0.0-12
```

```
    Please note: The process for determining duplicated points
    has changed from that used in version 0.0-9 (and previously).
```

```
spatstat 1.21-2
```

```
Type help(spatstat) for an overview of spatstat
latest.news() for news on latest version
licence.polygons() for licence information on polygon calculations
```

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

For a list of the commands available in `spatstat`, type

```
> help(spatstat)
```

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

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

4.4 Licence

The `spatstat` package is free open source software, under the GNU Public Licence.

However, some of the facilities in `spatstat` depend on a polygon geometry package called `gpplib`, and this has a restricted licence, that forbids commercial use. For details, type `licence.polygons()`.

By default, the `gpplib` package is disabled when you start `spatstat`. If you are doing non-commercial work, please enable the polygon clipping library by typing

```
> spatstat.options(gpplib = TRUE)
```

4.5 Inspecting data

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

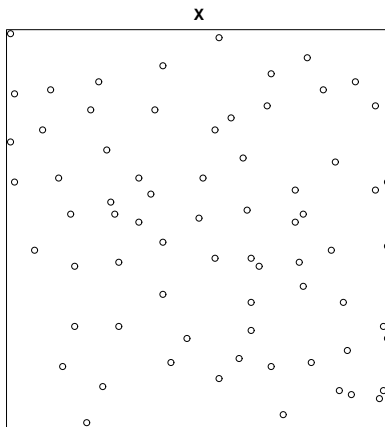
```
> data(swedishpines)
```

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

```
> X <- swedishpines
```

You can immediately plot the point pattern by typing

```
> plot(X)
```

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

```
> X
```

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

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

```
> summary(X)
```

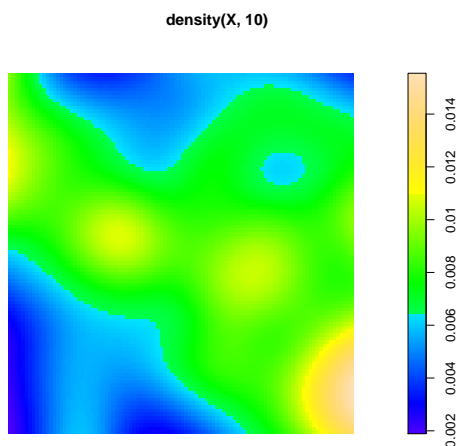
```
Planar point pattern: 71 points
Average intensity 0.0074 points per square unit (one unit = 0.1 metres)
```

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

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

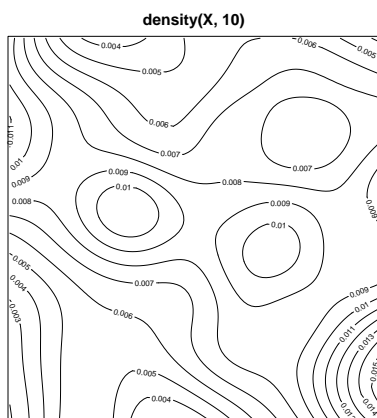
To get an impression of local spatial variations in intensity, we can plot a kernel estimate of intensity:

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



where 10 is my chosen value for the standard deviation of the Gaussian smoothing kernel: it is 10 decimetres, i.e. one metre. If you prefer a contour plot,

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



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

4.6 Exploratory data analysis

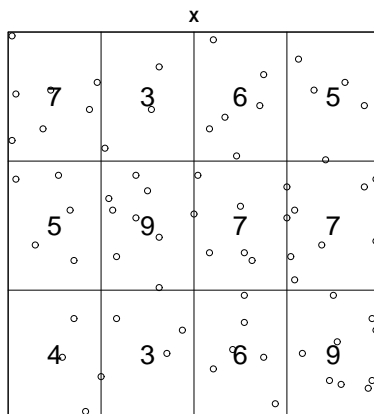
Spatstat is designed to support all the standard types of exploratory data analysis for point patterns.

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

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

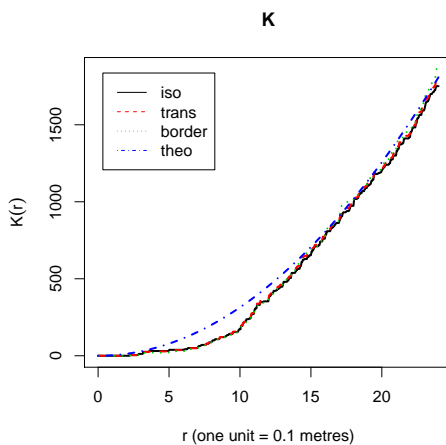
	x			
y	[0,24]	(24,48]	(48,72]	(72,96]
(66.7,100]	7	3	6	5
(33.3,66.7]	5	9	7	7
[0,33.3]	4	3	6	9

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



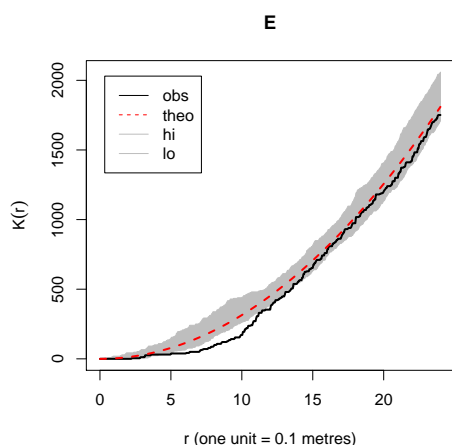
Another common example is *Ripley's K function*. I'll explain more about the K function later. For now, we'll just demonstrate how easy it is to compute and plot it. To compute the K function for a point pattern \mathbf{X} , type `Kest(X)`. This returns an object which can be plotted.

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



In this plot, the empirical K function (solid lines) deviates from the theoretical expected value assuming the points are completely random (dashed lines). To test whether this deviation is statistically significant, the standard approach is to use a Monte Carlo test based on envelopes of the K function obtained from simulated point patterns. In `spatstat` this is done with the `envelope` function:

```
> E <- envelope(X, Kest, nsim = 39)
> plot(E)
```



4.7 Models

The main strength of `spatstat` is that it supports **statistical models** of point patterns. Models can be fitted to point pattern data; the fitted models can be used to summarise the data or make predictions; the fitted models can be simulated (i.e. a random pattern can be generated according to the model); and there are facilities for model selection, for testing whether a term in the model is required (like analysis of variance), and for model criticism (like residuals, regression diagnostics, and goodness-of-fit tests).

Participants in this workshop 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. We can’t say something is ‘statistically significant’ unless we assume a model, because the p -value is the 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 it to be wrong. 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.

A statistical model for a point pattern is technically termed a *point process* model. Think of a point process as a black box that generates a random spatial point pattern according to some rules. To fit a point process model to a point pattern dataset in `spatstat`, use the function `ppm` (point process model). This is analogous to the standard functions in R for fitting linear models (`lm`), generalized linear models (`glm`) and so on.

```
> data(swedishpines)
> X <- swedishpines
> fit <- ppm(X, ~1, Strauss(9))
> fit
```

Stationary Strauss process

First order term:

```
beta
0.04378316
```

```

Interaction: Strauss process
interaction distance:      9
Fitted interaction parameter gamma:    0.2904

```

Relevant coefficients:

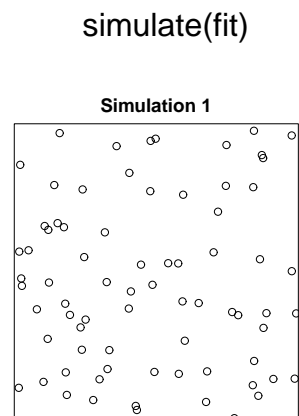
```

Interaction
-1.236324

```

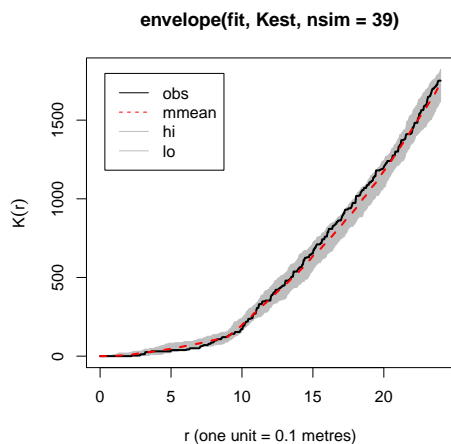
We have fitted a model called the “Strauss point process” to these data. We can generate a simulated realisation of this model:

```
> plot(simulate(fit))
```



We can perform a goodness-of-fit test for this fitted model:

```
> plot(envelope(fit, Kest, nsim = 39))
```



This plot suggests good agreement between the model and the data.

There are many, many other facilities for point process models in `spatstat`, described throughout these notes (mainly in Sections 15–16, 23.1, 27–28 and 34).

4.8 Multitype point patterns

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

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

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

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

```
> summary(lansing)
```

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

```
*Pattern contains duplicated points*
```

```
Multitype:
```

	frequency	proportion	intensity
blackoak	135	0.0600	135
hickory	703	0.3120	703
maple	514	0.2280	514
misc	105	0.0466	105
redoak	346	0.1540	346
whiteoak	448	0.1990	448

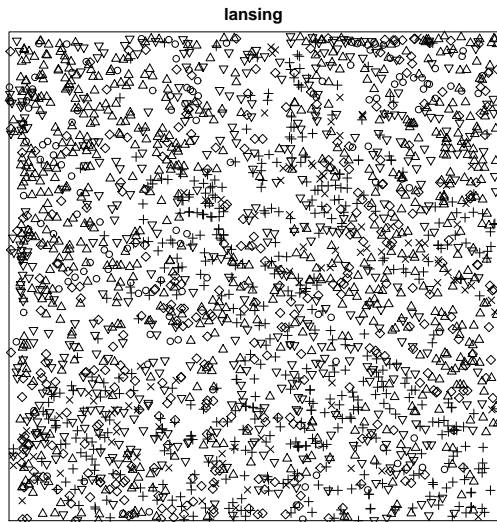
```
Window: rectangle = [0, 1]x[0, 1]units
```

```
Window area = 1 square unit
```

```
Unit of length: 924 feet
```

```
> plot(lansing)
```

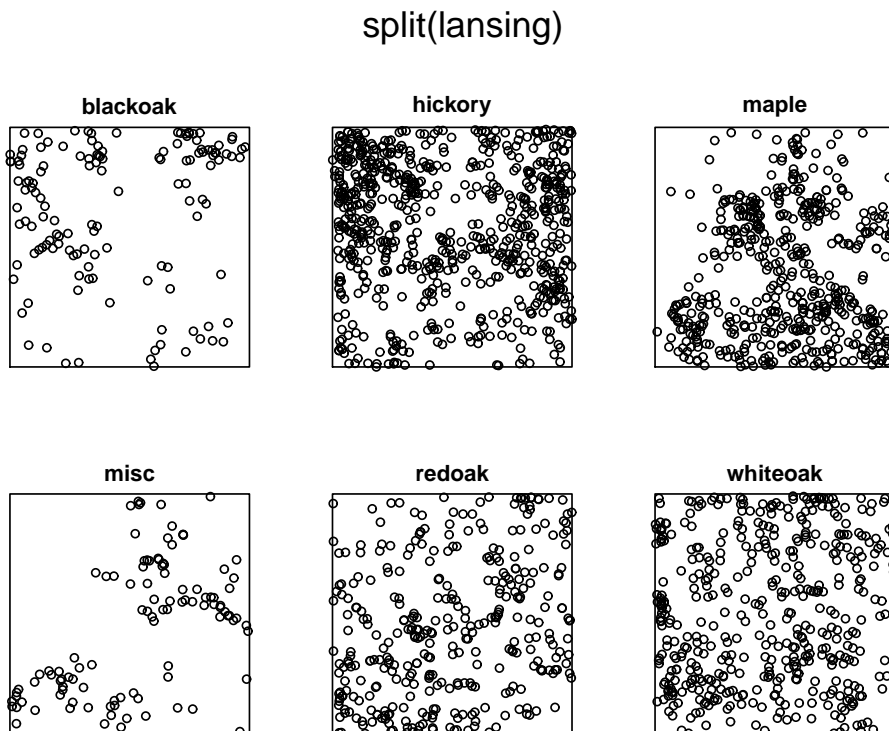
```
blackoak  hickory  maple  misc  redoak  whiteoak
         1         2         3         4         5         6
```



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

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

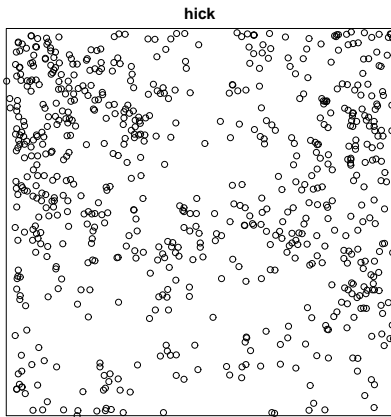
```
> plot(split(lansing))
```



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

e.g. the hickories,





























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






It's also possible to do exploratory analysis and model-fitting for multitype point patterns.

4.9 Installed datasets

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

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

The shape of the window containing the point pattern is indicated by the symbols  (rectangle),  (convex polygon) and  (irregular polygon).

There are also the following datasets which are not 2D point patterns:

name	description	format
<code>heather</code>	Diggle's heather data	binary image (three versions)
<code>osteo</code>	osteocyte lacunae	replicated 3D point patterns with covariates
<code>residualspaper</code>	data from [12]	it's complicated

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

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

PART II. DATA TYPES & DATA ENTRY

In Part II of the workshop, we look at the different types of spatial data in `spatstat` (point patterns, windows, pixel images, etc). We explain how to read data into the package and manipulate these data types.

5 Objects, classes and methods in R

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

5.1 Classes in R

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

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

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

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

```
> class(sunspots)

[1] "ts"
```

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

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

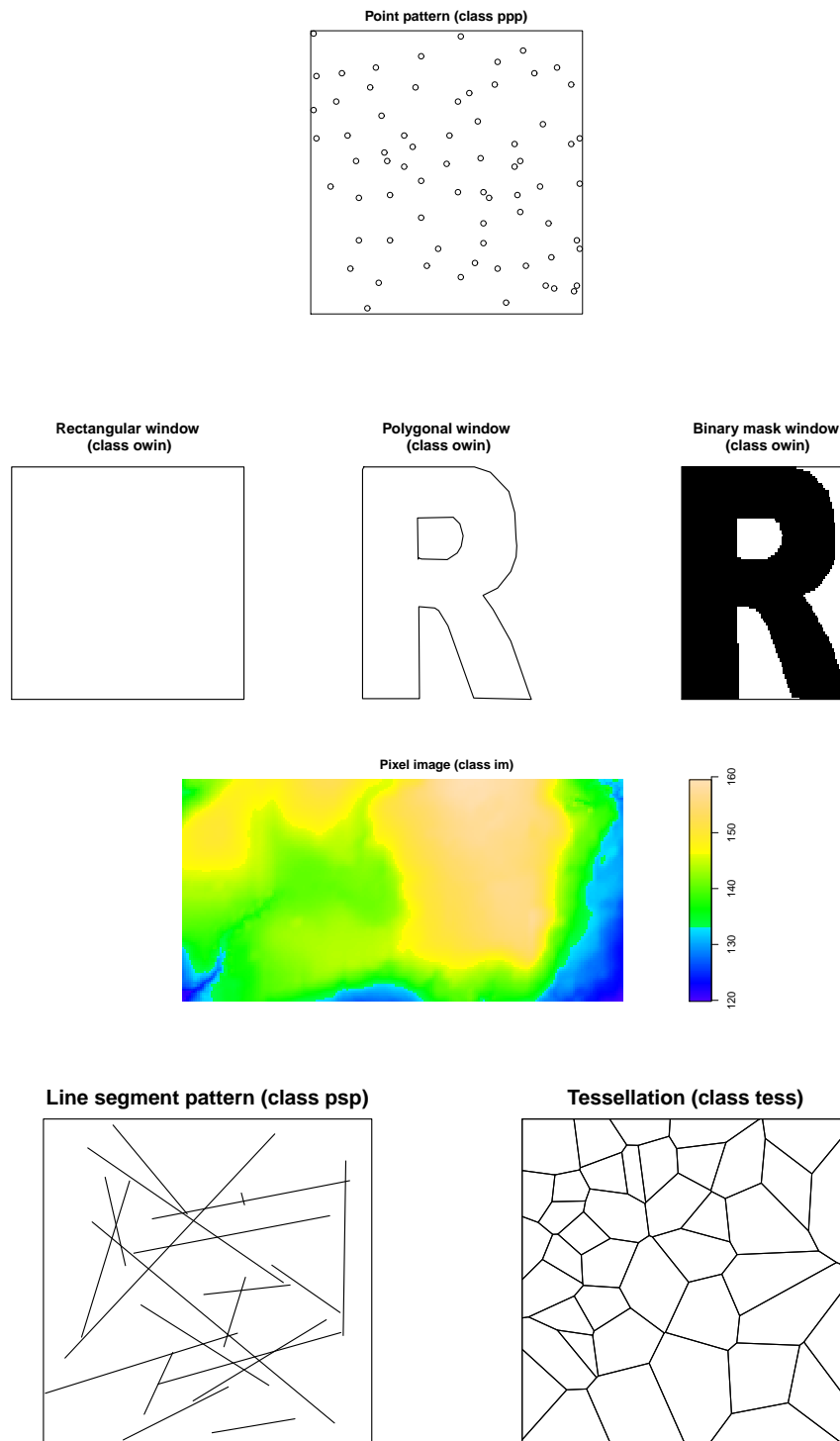
Tip: to find out how to modify the plot for an object of class `"foo"`, consult `help(plot.foo)` rather than `help(plot)`.

5.2 Classes in spatstat

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

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

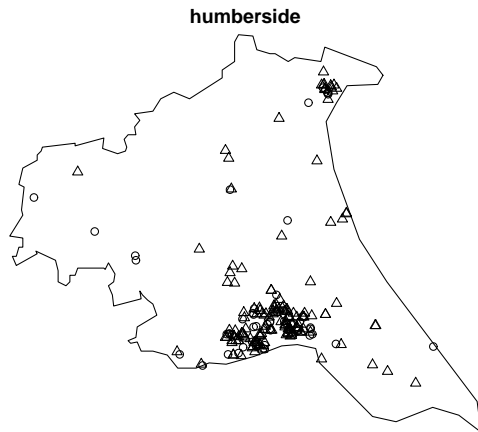
(there are also other classes for specialised use, such as `pp3` for three-dimensional point patterns, `ppx` for multidimensional space-time point patterns, and `hyperframe`).



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

In particular `spatstat` has methods for `plot`, `print` and `summary` for each of these classes. For example, the `plot` method for point patterns, `plot.ppp`, ensures that the x and y scales are equal, and does various other things that are sensible when plotting a spatial point pattern rather than just a list of (x, y) pairs.

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



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

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

```
> swedishpines
```

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

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

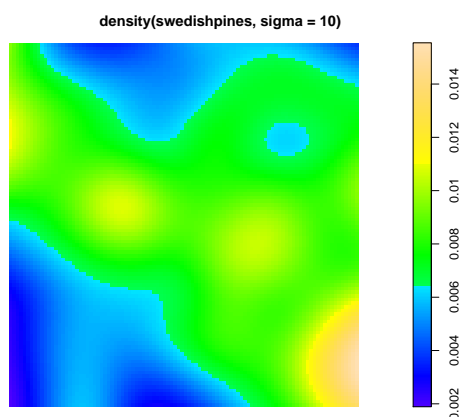
```
> summary(swedishpines)
```

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

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

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

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



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

```
> methods(plot)
```

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

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

```
[1] affine.ppp          as.data.frame.ppp as.im.ppp          as.owin.ppp
[5] as.ppp.ppp         bermantest.ppp    by.ppp             closing.ppp
[9] coords<-.ppp      coords.ppp        crossdist.ppp     cut.ppp
[13] density.ppp        dilation.ppp      distfun.ppp       distmap.ppp
[17] duplicated.ppp     envelope.ppp      erosion.ppp        identify.ppp
[21] is.empty.ppp       is.marked.ppp    is.multitype.ppp  kstest.ppp
[25] markformat.ppp    marks<-.ppp      marks.ppp          nnclean.ppp
[29] nndist.ppp        nnwhich.ppp      npoints.ppp       opening.ppp
[33] pairedist.ppp     pcf.ppp          periodify.ppp     pixellate.ppp
[37] plot.ppp           [<-.ppp          [.ppp             print.ppp
[41] quadratcount.ppp  quadrat.test.ppp rebound.ppp        rescale.ppp
[45] rotate.ppp        rshift.ppp       sharpen.ppp        shift.ppp
[49] split<-.ppp       split.ppp         summary.ppp        unique.ppp
[53] unitname<-.ppp   unitname.ppp     unmark.ppp
```

5.3 Return values

5.3.1 The return value of a function

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

```
> 1 + 1
```

```
[1] 2
```

```
> sin(pi/3)
```

```
[1] 0.8660254
```

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

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

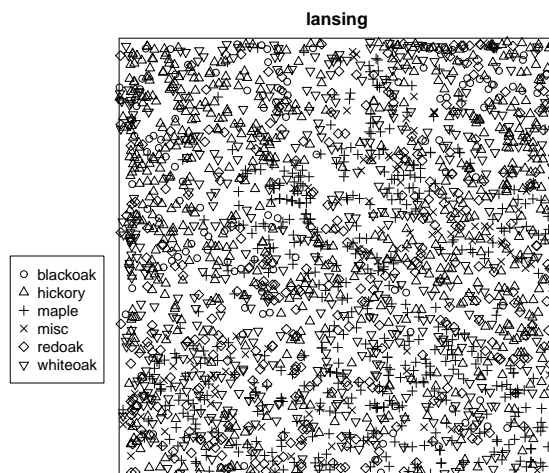
There's still a return value from the function, which can be captured by assigning the result to a variable:

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

NULL

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

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



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

5.3.2 Returning an object

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

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

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

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

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

```
> summary(Z)
```

```
real-valued pixel image
100 x 100 pixel array (ny, nx)
enclosing rectangle: [0, 96] x [0, 100] units
dimensions of each pixel: 0.96 x 1 units
(one unit = 0.1 metres)
Image is defined on the full rectangular grid
Frame area = 9600 square units
Pixel values :
  range = [0.00188947243195949,0.0155470858797917]
  integral = 71.3036909843861
  mean = 0.00742746781087355
```

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

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

```
Function value object (class fv)
for the function r -> K(r)
Entries:
id      label      description
--      -
r       r           distance argument r
theo    K[pois](r)    theoretical Poisson K(r)
border  K[bord](r)    border-corrected estimate of K(r)
trans   K[trans](r)   translation-corrected estimate of K(r)
iso     K[iso](r)     Ripley isotropic correction estimate of K(r)
-----
```

```
Default plot formula:
```

```
. ~ r
```



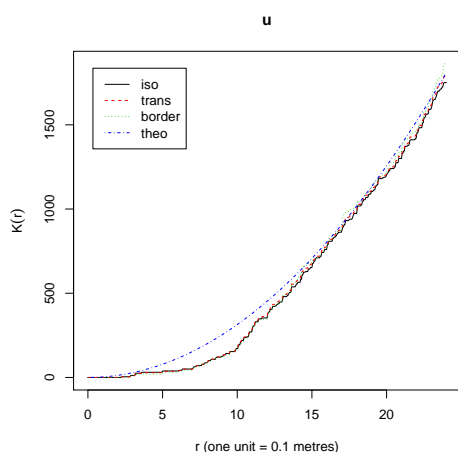
```
<environment: 0x5f821a0>
```

```
Recommended range of argument r: [0, 24]
```

```
Available range of argument r: [0, 24]
```

```
Unit of length: 0.1 metres
```

```
> plot(u)
```



6 Entering point pattern data into spatstat

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

This section explains how to handle 'raw' data in a text file. Section 7 explains how to handle data files in other formats (such as ESRI shapefiles).

6.1 Reading raw data into R

It's good practice to keep a copy of your original data in a text file (where it is not dependent on changes to software, data formats etc). The data can then be loaded into R using standard operations.

Two common formats for the data are

- a comma-separated values (`csv`) file, generated by many spreadsheet packages. To read data from a `csv` file into R, use the command `read.csv`.
- a table format file. The data are arranged in rows and columns, one row for each spatial point, something like this:

Easting	Northing	Diameter
176.111	32.105	10.4
175.989	31.979	7.6
....

The first line of the file is an (optional) *header*. To read these data into R, use the command `read.table`.

To read these datafiles, type either

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

In either of these cases, the resulting object `mydata` is a “data frame” in R. You can print the data frame by typing its name.

Individual columns of the data frame can be extracted using `$`. For example

```
> east <- mydata$Easting
```

extracts the Easting column of data, and saves it as a vector of numbers, called `east`.

```
> east

[1] 176.111 175.989 176.786 176.394 176.501 175.480 175.041 175.909 176.955
[10] 175.232 176.842 176.752 176.166 175.778 175.176 175.124 175.853 175.866
```

You can also use `scan()` to read a stream of numbers that you type at the keyboard, or `scan(file="filename")` to read a stream of numbers from a file.

6.2 Creating a ppp object

Here is a simple recipe to create a point pattern object from raw data in R.

1. store the x and y coordinates for the points in two vectors `x` and `y`.
2. create two vectors `xrange`, `yrange` of length 2 giving the x and y dimensions of a rectangle that contains all the points.
3. create the point pattern object by

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

The value returned by the function `ppp` is an object of class “`ppp`” representing a point pattern inside a rectangle.

If the natural window for the point pattern is **not** a rectangle, then you need to use a command like

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

where `W` is a window object. See Section 8.5 for details on how to do this.

For example, the following code reads raw data from a text file in table format, and creates a point pattern:

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

A slicker way to do the same thing is:

```
> mydata <- read.table("mydatafile.txt", header = TRUE)
> attach(mydata)
> X <- ppp(Easting, Northing, c(174, 178), c(29, 33))
```

6.3 Marks

Recall 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, etc.

Suppose x and y are vectors containing the coordinates of the point locations, as before. If there are marks attached to the points, store the corresponding marks in a vector m (with one entry for each point) or in a matrix or data frame m (with one row for each point and one column for each mark variable). Recall 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, etc.

Suppose x and y are vectors containing the coordinates of the point locations, as before. If there are marks attached to the points, store the corresponding marks in a vector m (with one entry for each point) or in a matrix or data frame m (with one row for each point and one column for each mark variable). 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:

```
> mydata <- read.table("mydatafile.txt", header = TRUE)
> attach(mydata)
> X <- 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`:

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

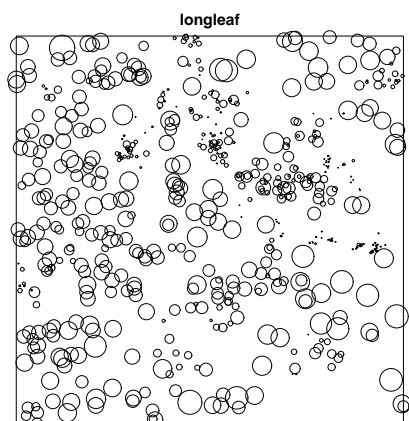
The handling of marks in `spatstat` depends on what type of data they are. Mark values may belong to any of the atomic data types: numeric, integer, character, logical, or complex. They may also be a `factor`, representing categorical values (see below).

Use the commands `is.numeric`, `is.integer`, ..., `is.factor` to test whether your data have the intended type. In a matrix, all the entries have the same atomic type, so you can ask `is.numeric(m)` where m is a matrix. A data frame is like a matrix except that different columns may have different types. You can ask `is.numeric(m[,2])` to find out if column 2 in the data frame m contains numeric data.

Here is the typical output from `plot.ppp` when the marks are numeric:

```
> data(longleaf)
> plot(longleaf)

      0      20      40      60      80
0.000000 1.722522 3.445045 5.167567 6.890090
```



The last line of output is the return value from `plot(longleaf)`, which indicates the scale used to plot the marks. The mark value 20 was plotted as a circle of radius 1.72.

6.4 Categorical marks

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

Here’s an example of an installed dataset with categorical marks:

```
> data(demopat)
> 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 A A A A B B B A A B B B B B A A B
[38] 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 B A B B A A B B
[75] 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 B B B B B A A A B A B B
[112] A
Levels: A B
```

This output (from the base R system) indicates that `marks(demopat)` is a factor with levels A and B.

```
> m <- marks(demopat)
> is.factor(m)
```

```
[1] TRUE
```

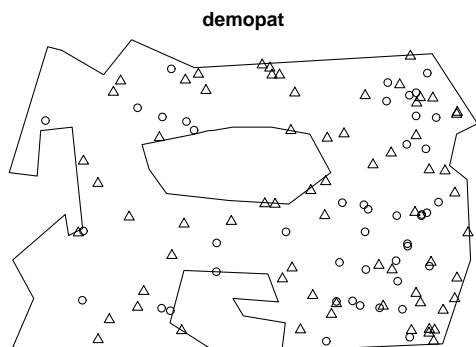
If the marks are intended to be a categorical variable, ensure that `m` is stored as a **‘factor’**.

Here is the typical output from `plot.ppp` when the marks are a factor:

```
> plot(demopat)
```

```
A B
```

```
1 2
```



The last line of output indicates how the marks were plotted: the mark **A** was plotted as symbol 1 (circle) and mark **B** was plotted as symbol 2 (triangle).

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

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

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

Other ways of adding marks to a point pattern will be described in Section 32.

6.5 Multivariate marks

The marks attached to a point pattern may be multivariate (i.e. several variables are attached to each point). For example, the `finpines` point pattern is marked by tree diameter and tree height.

```
> data(finpines)
```

```
> finpines
```

```
marked planar point pattern: 126 points
```

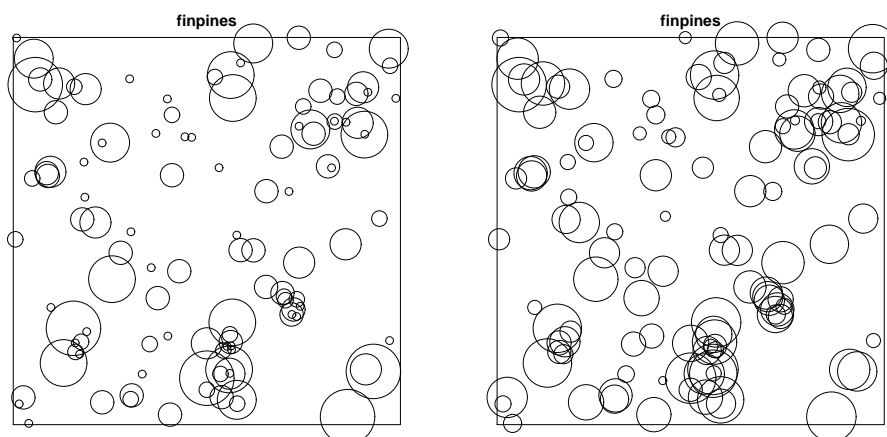
```
Mark variables: diameter, height
```

```
window: rectangle = [-5, 5] x [-8, 2] metres
```

To create such a point pattern, the mark data should be supplied as a data frame. It's important to check that each column of data has the intended type — especially for columns that are intended to be factors.

When a point pattern with multivariate marks is plotted, only one of the columns of marks will be displayed. By default, the first column is selected. You can select another column using the argument `which.marks`.

```
> par(mfrow = c(1, 2))
> plot(finpines)
> plot(finpines, which.marks = "height")
> par(mfrow = c(1, 1))
```



6.6 Checking data

It is prudent to check for quirks in the data.

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

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

6.7 Units

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

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

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

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

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

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

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

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

Altering the `unitname` in an existing dataset is usually not sensible; it simply alters the name of the unit, without changing the entries in the x and y vectors. If you want to convert to different units (e.g. from metres to kilometres or from imperial to metric units), use the command `rescale` as described in Section 9.2.4. If you want to actually change the coordinates by a linear transformation, producing a dataset that is not equivalent to the original one, use `affine`.

6.8 Other ways to make point patterns

To create a point pattern object we can either

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

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

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

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

More alternatives for using `ppp` will be covered in Section 8.5.

7 Converting from GIS formats

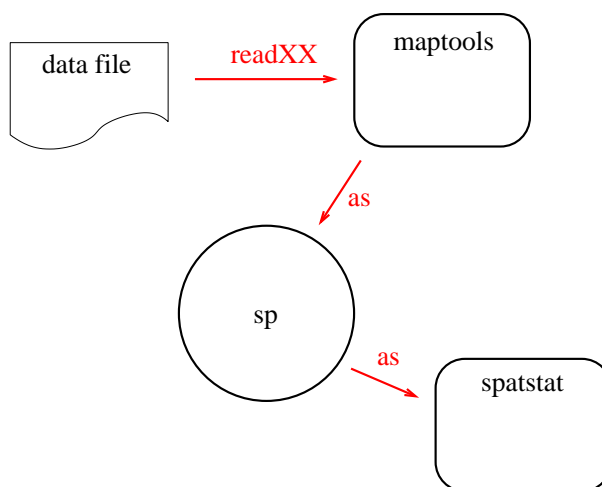
There is a wide variety of software packages for handling spatial data, especially Geographical Information Systems (GIS). These packages use many different formats to represent spatial data. Typically `spatstat` does not support these formats: this would not be good software design.

Specialised R packages exist for handling different spatial data file formats. The most useful ones are `rgdal`, `shapefiles` and `maptools`. These packages will make it possible for you to read your data from a file into an R session. The `rgdal` package has the most functionality, but can sometimes be difficult to install, as it requires installation of an external library on your system. The packages `shapefiles` and `maptools` have no such difficulty.

The package `sp` provides generic support for spatial data types in R. It enables you to convert between different representations of your data in R.

The usual procedure for converting spatial data is:

1. read your data file into R using a package designed specifically for that file format (e.g. `shapefiles` for ESRI shapefiles), converting it into an R dataset;
2. convert this R dataset into a generic format used by `sp`;
3. convert the generic `sp` format to the required `spatstat` format, using `sp`.



This procedure has to be followed separately for different types of spatial data. Point patterns, windows and pixel images are handled slightly differently. If your point pattern locations are supplied as an ESRI shapefile `mypoints.shp`, then the commands would be

```

> S <- readShapePoints("myfile.shp")
> SP <- as(S, "SpatialPoints")
> P <- as(SP, "ppp")

```

The result is a point pattern (object of class `"ppp"`) in `spatstat`, but you then need to assign the correct window to it.

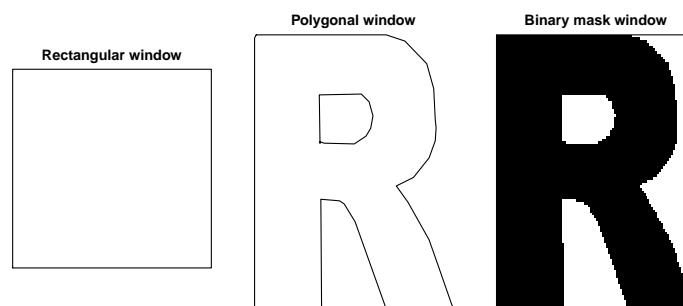
For further details, see the package vignette *Handling shapefiles in the spatstat package* which is available by typing `vignette("shapefiles")`. For further information on handling GIS formats see [24].

8 Windows in spatstat

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

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

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



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

8.1 Making windows by hand

8.1.1 Rectangular window

To create a rectangular window, type

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

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

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

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

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

```
> square(5)
```

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

8.1.2 Circular window

For a circular window use `disc`:

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

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

8.1.3 Polygonal window

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

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

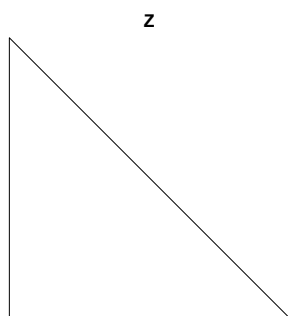
or

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

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

If the window boundary is a single polygon, then `p` should be a 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 with corners $(0,0)$, $(1,0)$ and $(0,1)$ is created by

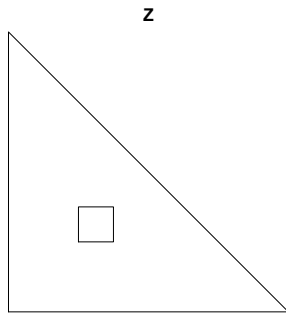
```
> Z <- owin(poly = list(x = c(0, 1, 0), y = c(0, 0, 1)))
> plot(Z)
```



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

If the window boundary consists of several separate polygons, then `p` should be a list, each of whose components `p[[i]]` is a 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 a triangle with a square hole.

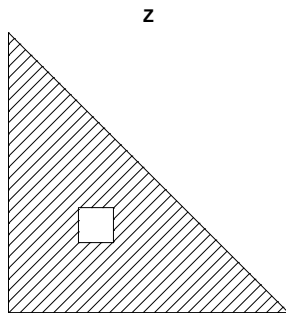
```
> Z <- owin(poly = list(list(x = c(0, 8, 0), y = c(0, 0, 8)), list(x = c(2,
+ 2, 3, 3), y = c(2, 3, 3, 2))))
> plot(Z)
```



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

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

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



8.1.4 Binary mask

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

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

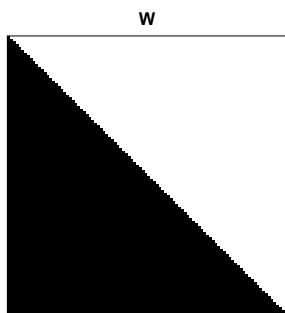
to create the window object. Here `m` should be a matrix with logical entries; it will be interpreted as a binary pixel image whose entries are `TRUE` where the corresponding pixel belongs to the window.

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

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

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

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



8.2 Converting from GIS formats

If your window (spatial region) is supplied as an “ESRI shapefile” with a name like `myfile.shp`, then type the following:

```
> library(maptools)
> S <- readShapePoly("myfile.shp")
> library(sp)
> SP <- as(S, "SpatialPolygons")
> W <- as(SP, "owin")
```

The `readShapePoly` command reads the file `myfile.shp` and returns an object `S` of class `"SpatialPolygonsDataFrame"`. The next command converts this to an object of class `"SpatialPolygons"` and the last command converts this in turn into a window (object of class `"owin"`) in `spatstat`.

For further information on handling GIS formats see [24] or `vignette("shapefiles")`.

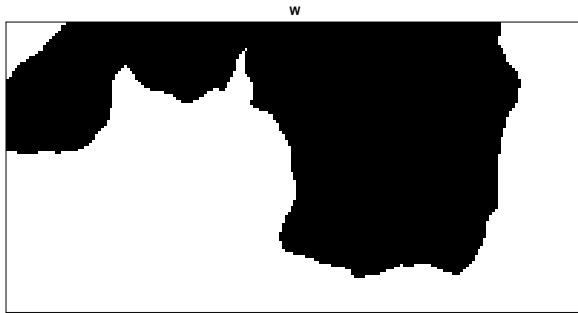
8.3 Functions that make a window

Some functions make a window object. They include

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

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

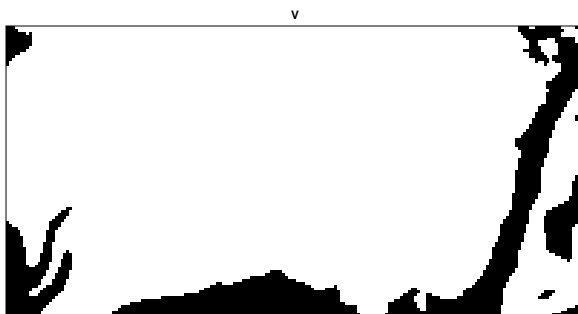
```
> elev <- bei.extra$elev
> W <- levelset(elev, 145, ">")
> plot(W)
```



The result W is a window.

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

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



8.4 Operations on windows

Basic methods for the class "owin" include

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

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

<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>area.owin</code>	compute window's area
<code>diameter</code>	compute window's diameter
<code>perimeter</code>	compute window's perimeter length
<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>is.subset.owin</code>	determine whether one window contains another
<code>complement.owin</code>	swap inside and outside
<code>bounding.box</code>	Find a tight bounding box for the window
<code>convexhull</code>	Convex hull of a window
<code>is.convex</code>	Test whether a window is convex
<code>rotate</code>	rotate window
<code>shift</code>	translate window
<code>affine</code>	apply affine transformation
<code>rescale</code>	change scale and adjust units
<code>as.mask</code>	convert window to binary image mask
<code>pixellate.owin</code>	convert window to pixel image
<code>dilate.owin</code>	morphological dilation
<code>dilation.owin</code>	morphological dilation
<code>erode.owin</code>	morphological erosion
<code>erosion.owin</code>	morphological erosion
<code>opening.owin</code>	morphological opening
<code>closing.owin</code>	morphological closing
<code>dilated.areas</code>	compute areas of dilated windows
<code>eroded.areas</code>	compute areas of eroded windows
<code>border</code>	create a border region around a window
<code>inside.owin</code>	determine whether a point is inside a window
<code>distmap.owin</code>	distance transform image
<code>distfun.owin</code>	distance transform function
<code>centroid.owin</code>	compute centroid (centre of mass) of window
<code>incircle</code>	find largest circle inside window
<code>simplify.owin</code>	Approximate window by a polygon
<code>deltametric</code>	Measure discrepancy between two windows

8.5 Creating a point pattern in any window

As we saw in Section 6.2, the function `ppp()` will create a point pattern (an object of class "ppp") from raw numerical data in R.

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

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

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

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

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

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

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

will create the point pattern.

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

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

9 Manipulating point patterns

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

9.1 Point pattern objects

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

9.1.1 Internal Format

WARNING: It is strongly advisable NOT to directly access or modify the internal components of an object.

It is a “beginner’s mistake” to modify the internal components of a structured object such as a point pattern (object of class `"ppp"`). The internal structure of objects in a package can change from one version of the package to another. It is much safer to use operators defined in the package to extract and modify information.

However, in the spirit of “open source”, here is a description of the internal format.

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

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

It is possible to extract these components individually; for example, to make a histogram of the x coordinates you could just type `hist(P$x)`. However, **do not assign values to these components directly**, or you may create inconsistencies in the data which cause `spatstat` to crash.

To extract or manipulate the data in a point pattern object, use the functions provided in the package. Important ones are:

<code>npoints(X)</code>	number of points in <code>X</code>
<code>marks(X)</code>	marks of <code>X</code>
<code>coords(X)</code>	coordinates of points in <code>X</code>
<code>as.owin(X)</code>	window of <code>X</code>
<code>as.data.frame(X)</code>	coordinates and marks of <code>X</code>
<code>marks(X) <- value</code>	change the marks of <code>X</code>
<code>coords(X) <- value</code>	change the coordinates of <code>X</code>
<code>X[W]</code>	change the window of <code>X</code>

9.1.2 A point pattern needs a window

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

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

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

9.1.3 Order of points

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.

```
> data(longleaf)
> as.data.frame(longleaf)[1:5, ]

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

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

```
> data(chorley)
> as.data.frame(chorley)[55:60, ]

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

> type <- marks(chorley)
> is.factor(type)

[1] TRUE

> levels(type)
```

```
[1] "larynx" "lung"
```

```
> table(type)
```

```
type
larynx  lung
      58   978
```

9.2 Operations on ppp objects

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

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

9.2.1 Extracting and altering data

<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.owin(X)</code>	window of X
<code>as.rectangle(X)</code>	bounding rectangle of X
<code>as.data.frame(X)</code>	coordinates and marks of X
<code>marks(X) <- value</code>	change the marks of X
<code>coords(X) <- value</code>	change the coordinates of X
<code>X[W]</code>	change the window of X

9.2.2 Extracting a subset of a point pattern

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

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

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

```
> data(bei)
```

```
> bei
```

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

```
> bei[1:10]
```

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

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

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

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

```
> bei[W]
```

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

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

The command `split.ppp` allows you to divide a point pattern into sub-patterns, and the command `by.ppp` allows you to perform an operation on each sub-pattern.

9.2.3 Fiddling with marks

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

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

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

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

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

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

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

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

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

```
> unmark(X)
```

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

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

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

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

9.2.4 Changing scales and units

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

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

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

The command `rescale` performs the same function:

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

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

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

9.2.5 Geometrical transformations

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

9.2.6 Random perturbations of a point pattern

It is sometimes useful to randomise the data, for example for hypothesis testing. The command `rshift` will apply the same random shift to each point, while `rjitter` will apply a different random shift to each point. The command `quadratresample` performs a block resampling procedure in which the window is divided into rectangles and these rectangles are randomly resampled.

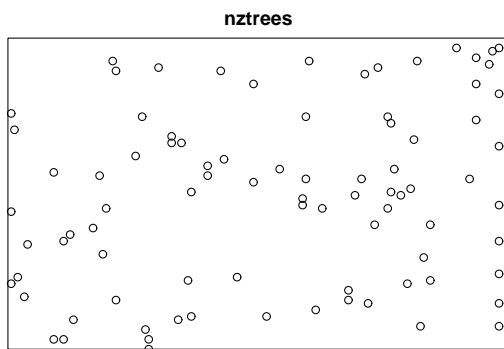
9.3 Example

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

```
> data(nztrees)
> nztrees

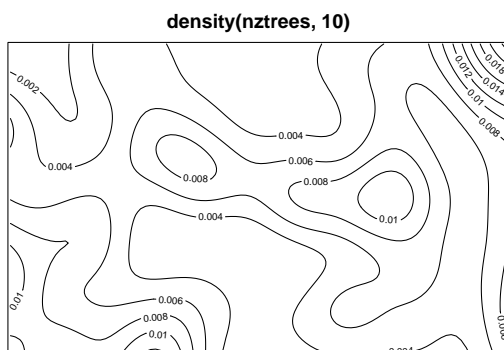
planar point pattern: 86 points
window: rectangle = [0, 153] x [0, 95] feet

> plot(nztrees)
```



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

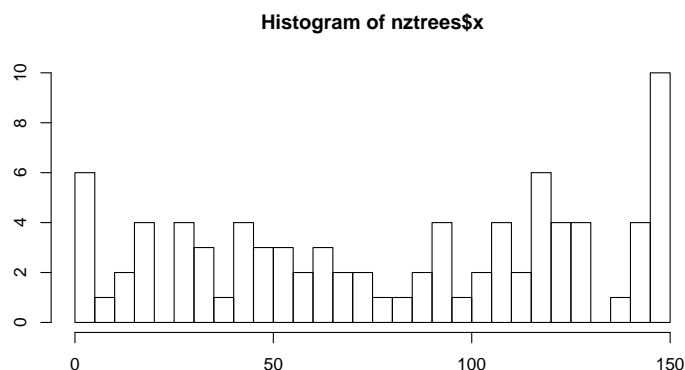
```
> contour(density(nztrees, 10), axes = FALSE)
```



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

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

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



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

First we create the new, trimmed study region:

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

or more slickly,

```
> win <- nztrees$window
```

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

```
> chopped
```

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

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

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

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

We can now study the 'chopped' point pattern:

```
> summary(nzchop)
```

```
Planar point pattern: 78 points
```

```
Average intensity 0.00555 points per square foot
```

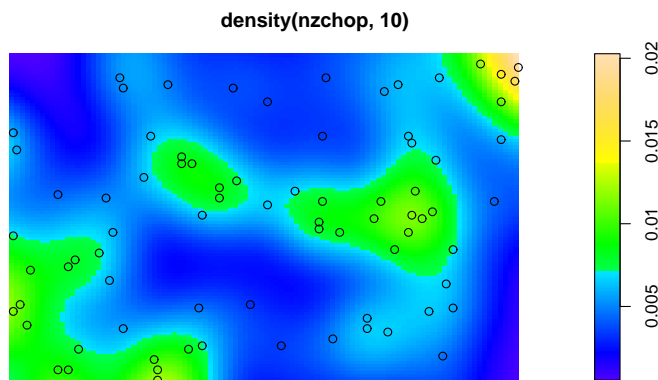
```
Window: rectangle = [0, 148]x[0, 95]feet
```

```
Window area = 14060 square feet
```

```
Unit of length: 1 foot
```

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

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



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

9.4 Splitting and combining point patterns

Sometimes it is useful to split a point pattern dataset into several sub-patterns, and perform some calculations on each sub-pattern.

9.4.1 Splitting a point pattern into sub-patterns

The powerful R command `split` has a method for point patterns. This enables the user to divide a point pattern into sub-patterns using any suitable criterion.

- If X is a marked point pattern, and the marks are a factor, then `split(X)` separates the data points into different point patterns according to their mark value.
- If Z is a pixel image with factor values, then `split(X,Z)` separates the data points into different point patterns according to the pixel value of Z at each point.
- If Z is a tessellation, then `split(X,Z)` separates the point pattern X into sub-patterns delineated by the tiles of Z .

In each case the result is a list of point patterns. You can then use the R command `lapply` to perform any desired operation on each element of the list. For example, to apply adaptive estimation of intensity to each species of tree in the Lansing Woods data,

```
> data(lansing)
> 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=Z, FUN=f)` is essentially equivalent to `lapply(split(X,Z), f)`. It splits the dataset X into sub-patterns according to Z , then applies the function f to each sub-pattern. So to apply adaptive estimation of intensity to each species of tree in the Lansing Woods data,

```
> data(lansing)
> A <- by(lansing, FUN = adaptive.density)
> plot(A)
```

9.4.2 Combining point patterns

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

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

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

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

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

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

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

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

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

9.5 List of operations on point patterns

Here's a summary of basic operations available for a point pattern `X`.

X	print basic info
print(X)	print basic info
summary(X)	print detailed summary
npoints(X)	number of points
coords(X)	extract coordinates of points
coords(X)<-value	assign new coordinates to points
marks(X)	extract marks from point pattern
marks(X)<-value	assign new marks to point pattern
unmark(X)	remove marks
marks(X)<-NULL	remove marks
as.owin(X)	extract window of point pattern
X[subset]	subset of point pattern
plot(X)	plot a point pattern
superimpose(X1, X2,...)	combine several point patterns
duplicated(X)	detect duplicated points
unique(X)	remove duplicated points
identify(X)	point-and-click to identify individual points
cut(X, ...)	classify points into groups
split(X)	divide pattern into sub-patterns
by(X, ...)	apply function to sub-patterns
discretise(X)	discretise coordinate values
pixellate(X)	approximate point pattern by pixel image
as.im(X)	approximate point pattern by pixel image
rotate(X, ...)	rotate entire point pattern and window
shift(X, ...)	shift entire point pattern and window
affine(X, ...)	affine transformation
density(X)	kernel smoothed intensity estimate
smooth.ppp(X)	spatial interpolation of mark values
convexhull(X)	convex hull of point pattern
delaunay(X)	Delaunay triangulation of point pattern
dirichlet(X)	Dirichlet-Voronoi tessellation based on point pattern
periodify(X)	make several translated copies of point pattern
rlabel(X)	random re-labelling of multitype point pattern
rshift(X)	random shifting of points

10 Pixel images in spatstat

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

A pixel image represents a spatial function $Z(u)$ in many different contexts. It may contain experimental data (such as a map of terrain elevation) or computed values (such as a kernel estimate of point process intensity) or it may be directly obtained from a camera (such as a satellite image).

10.1 Creating a pixel image

10.1.1 Creating an image from raw data

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

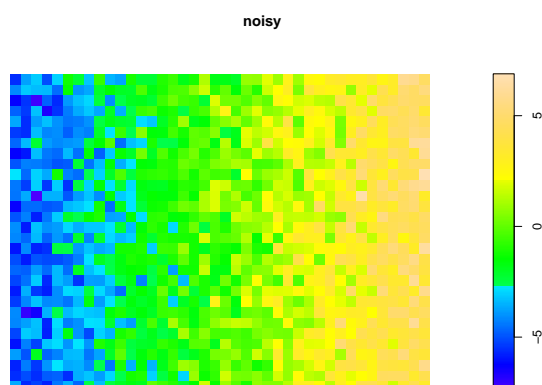
```
> im(mat, xcol, yrow)
```

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

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

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

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



10.1.2 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.

```
> cutvec <- cut(mat, 3)
> is.factor(cutvec)
```

```
[1] TRUE
```

```
> is.matrix(cutvec)
```

```
[1] FALSE
```

Although `mat` was a matrix, `cutvec` 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` reshape it:

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

```
factor-valued pixel image
```

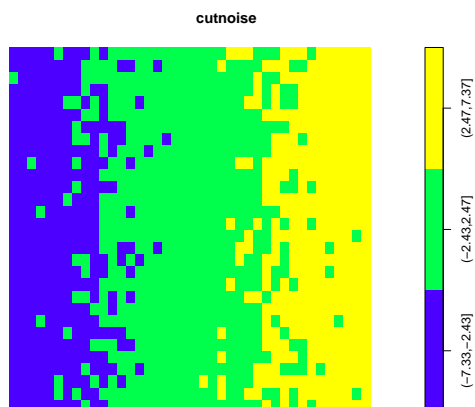
```
factor levels:
```

```
[1] "(-7.33,-2.43]" "(-2.43,2.47]" "(2.47,7.37]"
```

```
30 x 40 pixel array (ny, nx)
```

```
enclosing rectangle: [-0.012821, 1.0128] x [-0.017241, 1.0172] units
```

```
> plot(cutnoise)
```



[Another 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.]

10.1.3 Converting a function to an image

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

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

```

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

```

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

10.1.4 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

10.2 Inspecting an image

10.2.1 Basic information

For basic information about an image `Z`, use the following:

<code>Z</code>	print basic information
<code>print(Z)</code>	print basic information
<code>summary(Z)</code>	print detailed information
<code>dim(Z)</code>	pixel raster dimensions (y, x)
<code>nrow(Z)</code>	number of rows (y coordinate)
<code>ncol(Z)</code>	number of columns (x coordinate)
<code>range(Z)</code>	range of pixel values
<code>max(Z)</code>	maximum of pixel values
<code>min(Z)</code>	minimum of pixel values
<code>mean(Z)</code>	mean of pixel values
<code>median(Z)</code>	median of pixel values
<code>quantile(Z, ...)</code>	quantiles of pixel values
<code>sum(Z)</code>	sum of pixel values
<code>integral.im(Z)</code>	sum of pixel values times pixel area

To compute other numerical summaries of pixel values that are not on this list, you can extract the pixel values using `as.matrix(Z)` then apply the desired operation.

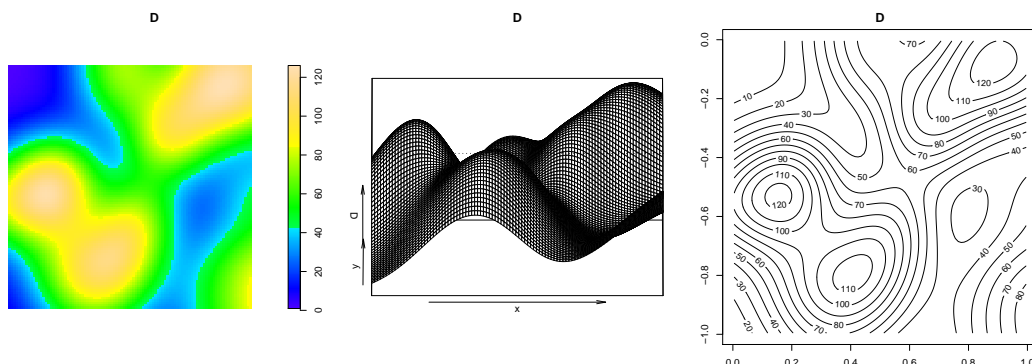
10.2.2 Plotting an image

Methods for plotting an image object include:

<code>plot.im</code>	display as colour image
<code>contour.im</code>	contour plot
<code>persp.im</code>	perspective plot of surface

These are methods for generic functions, so you would type `plot(Z)`, `contour(Z)` or `persp(Z)` to display a pixel image `Z`.

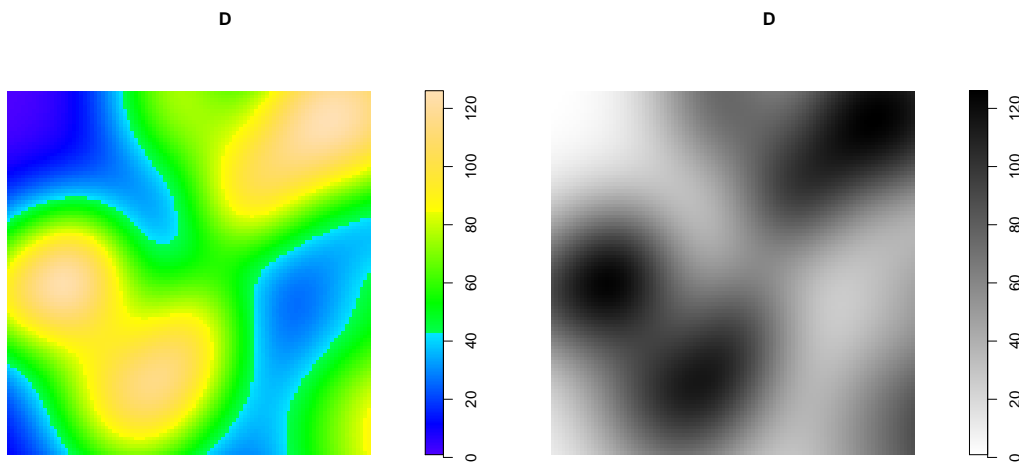
```
> opa <- par(mfrow = c(1, 3))
> data(redwood)
> D <- density(redwood)
> plot(D)
> persp(D)
> contour(D)
> par(opa)
```



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

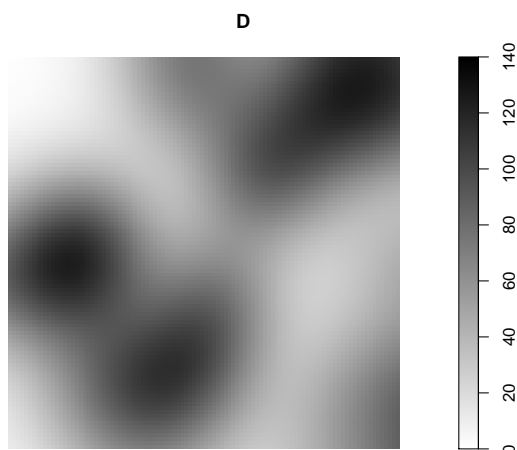
```
> opa <- par(mfrow = c(1, 2))
> plot(D)
```

```
> plot(D, col = grey(seq(1, 0, length = 512)))
> par(opa)
```



In the example above, the argument `col` was a vector of colour data. The range of pixel values in the image `Z` was mapped to these colours. Unfortunately this means that if we plot two images `Z1`, `Z2` using the same `col` vector, the interpretation of the colours will be different! To avoid this, set the argument `col` to be an object of the special class "colourmap", created by the function `colourmap`. An object of this class specifies a mapping between numerical values and colours.

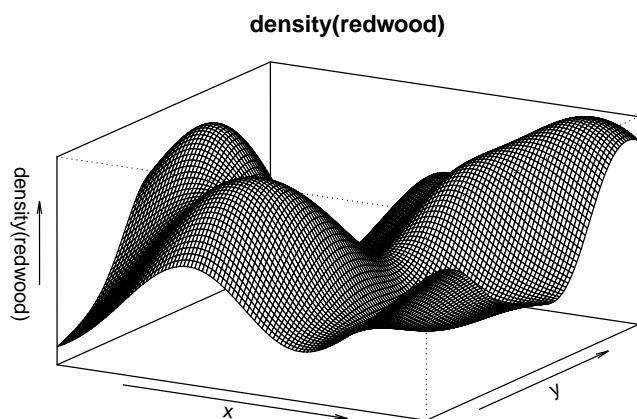
```
> mymap <- colourmap(grey(seq(1, 0, length = 512)), range = c(0,
+ 140))
> plot(D, col = mymap)
```



See `help(colourtools)` for tools that manipulate colours.

For `persp.im`, see also the help for `persp.default` for the names of various arguments to control the appearance of the plot. For example, the viewing direction is controlled by the angles `theta` and `phi`.

```
> persp(density(redwood), theta = 30)
```



Similarly for `contour.im`, consult also the help file for `contour.default` to control the appearance of the contours.

For some inspiring examples of perspective and contour plots with beautiful colour schemes and shading, see the R graphics demonstration by typing `demo(graphics)`.

10.2.3 Exploratory analysis

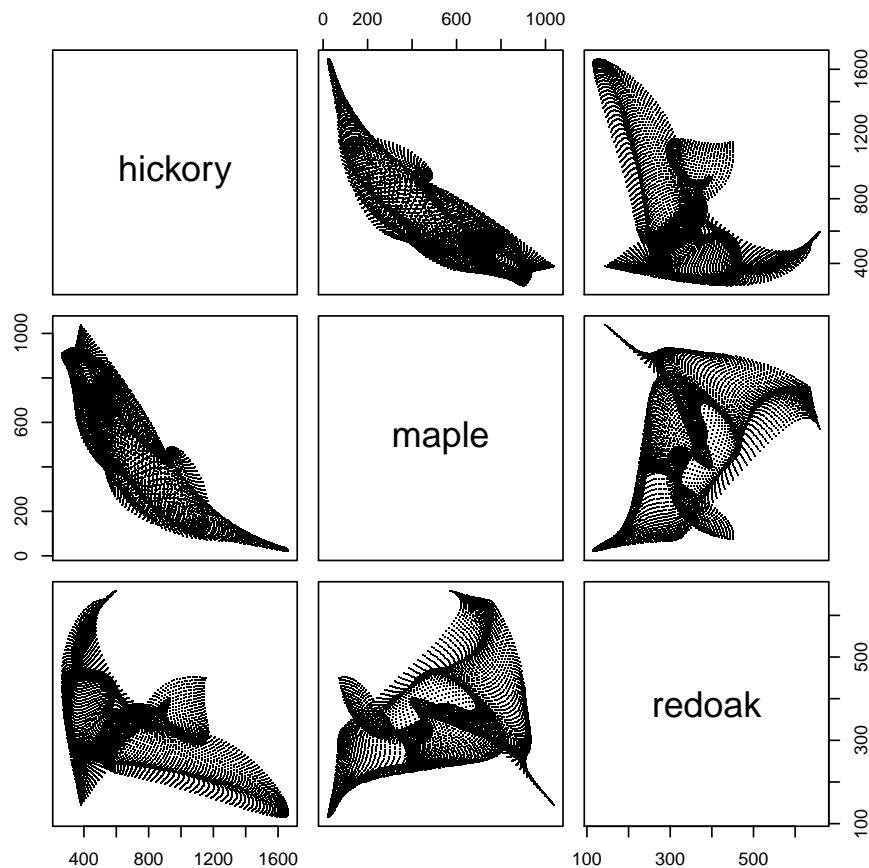
To inspect an image, the following are useful.

<code>as.matrix</code>	extract matrix of pixel values from image
<code>cut.im</code>	convert numeric image to factor image
<code>hist.im</code>	histogram of pixel values
<code>ecdf.im</code>	cumulative distribution function of pixel values

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

To study the relation between two or more images, it's useful to display the `pairs` plot, a scatterplot of the corresponding pixel values of each image. See `pairs.im`.

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



This command divided the Lansing Woods point pattern dataset into 6 sub-patterns of different tree species, extracted the 3 most common species, computed the kernel smoothed intensity estimate for each species, and then displayed scatterplots of the intensity estimates for each pair of species. 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).

10.3 Manipulating images

10.3.1 Subsets of an image

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

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

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

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

- If S is a pixel image with logical values, it is interpreted as a window (with TRUE inside the window).

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

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

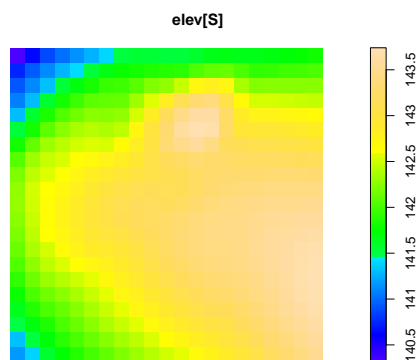
The subset operator can be used to look up the value of a pixel image at a single point:

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

```
[1] 147.08
```

or to display a subregion:

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



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

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

10.3.2 Computation with images

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

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

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

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

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

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

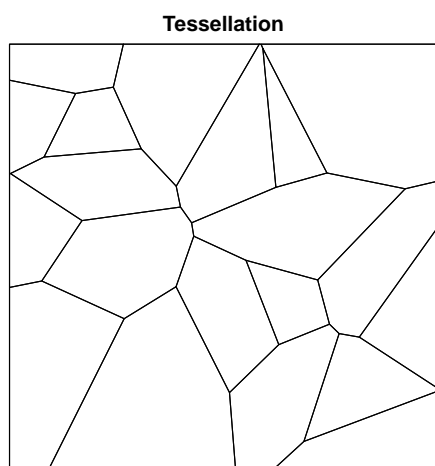
```
> W <- eval.im(sin(pi * Z))
> V <- eval.im(Z > 3)
> U <- eval.im(ifelse(Z > 3, 42, Z))
```

Other functions which manipulate images include the following:

<code>shift.im</code>	vector shift of an image
<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>	spatially interpolate an image
<code>levelset</code>	threshold an image (produces a window)
<code>solutionset</code>	find the region where a statement is true (produces a window)

11 Tessellations

A “tessellation” is a division of space into non-overlapping regions (“tiles”).

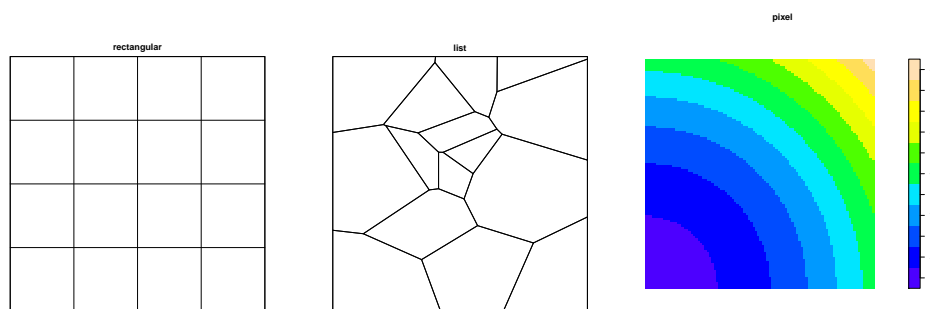


Tessellations have several uses in `spatstat`. The tessellation may be ‘real’, for example, a continent divided into states or provinces. The tessellation may be completely artificial, for example, the rectangular quadrats which we use in quadrat counting. Or the tessellation may be computed from other data, for example, the Dirichlet tessellation defined by a set of points.

11.1 Creating a tessellation

An object of class “`tess`” represents a tessellation. 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.



All three types of tessellation can be created by the command `tess`.

To create a rectangular tessellation:

```
> 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 divide a rectangular window `W` into rectangles of equal size, you can 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, postcodes, counties) and the boundaries of each sub-region are provided by GIS data files.

To create a tessellation from a pixel image,

```
> 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.

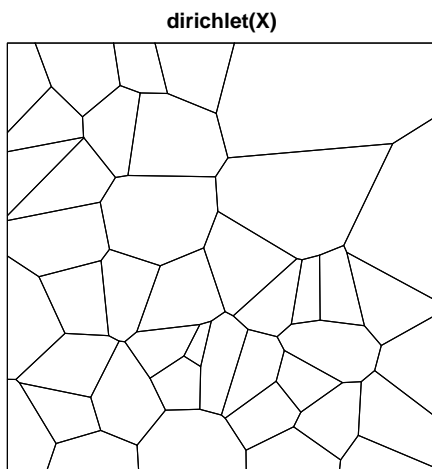
The command `as.tess` can also be used to convert other types of data to a tessellation.

11.2 Computed tessellations

There are two commands which compute a tessellation from a point pattern.

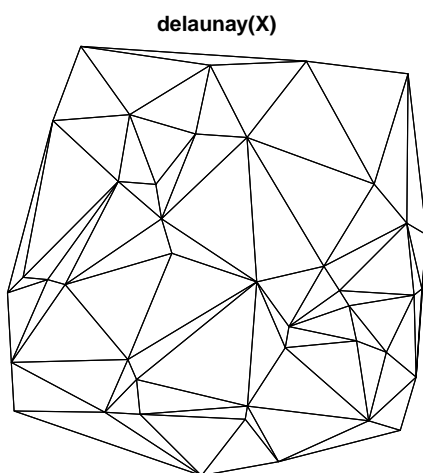
The command `dirichlet(X)` computes the *Dirichlet tessellation* or *Voronoi tessellation* of the point pattern `X`. The tile associated with a given point of the pattern `X` is the region of space which is closer to that point than to any other point of `X`. The Dirichlet tiles are polygons. The command `dirichlet(X)` computes these polygons and intersects them with the window of `X`.

```
> X <- runifpoint(42)
> plot(dirichlet(X))
```



The command `deLaunay(X)` computes the *Delaunay triangulation* of the point pattern `X`. Strictly speaking this is not a tessellation but a network or graph, 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`.

```
> plot(deLaunay(X))
```



11.3 Operations involving a tessellation

There are methods for `print`, `plot` and `[` for tessellations.

Use the command `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.owin))
```

Tile 1	Tile 2	Tile 3	Tile 4	Tile 5	Tile 6	Tile 7
0.17096279	0.09645337	0.02587679	0.18646932	0.15352600	0.07760811	0.03357145
Tile 8	Tile 9	Tile 10				
0.01904539	0.15038824	0.08609854				

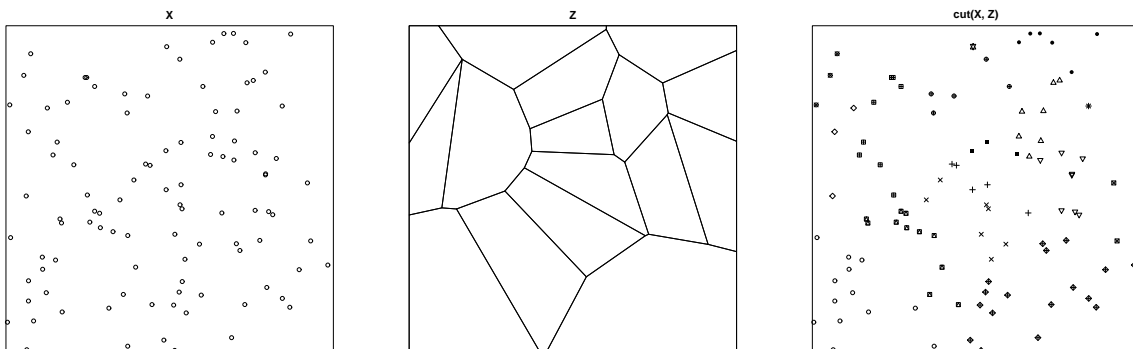
Tessellations can be used to classify the points of a point pattern, in `split.ppp`, `cut.ppp` and `by.ppp`. If `X` is a point pattern and `V` is a tessellation, then

- `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.

```
> par(mfrow = c(1, 3))
> X <- runifpoint(100)
> plot(X)
> Z <- dirichlet(runifpoint(16))
> plot(Z)
> plot(cut(X, Z))
```

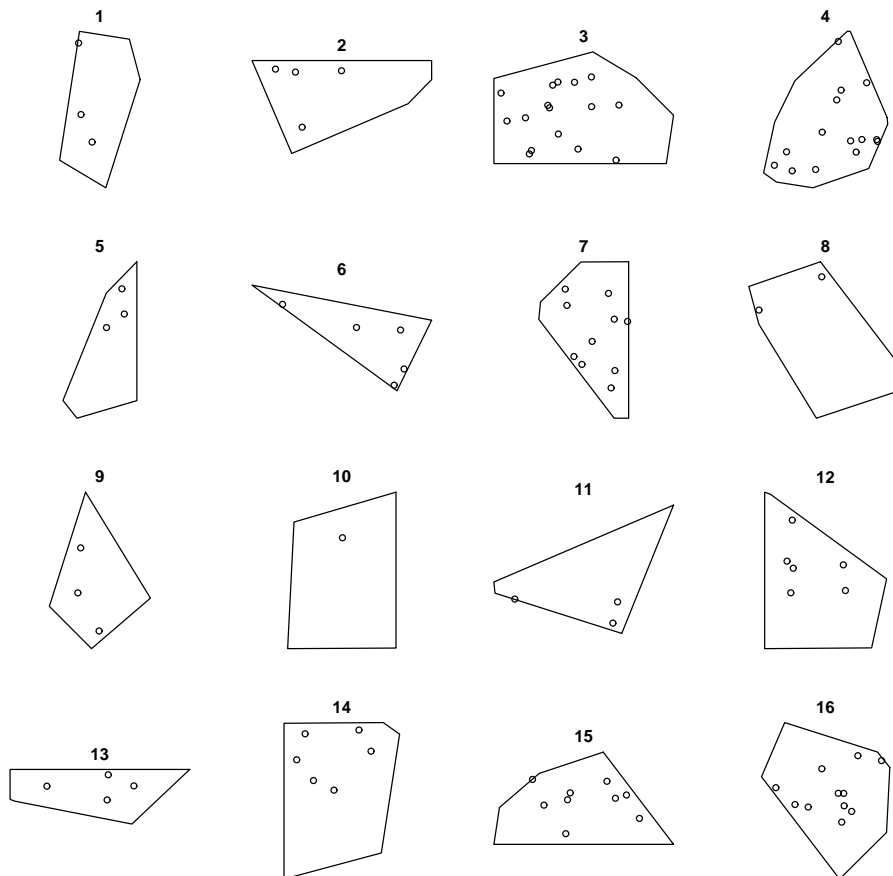
```
1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
```

```
> par(mfrow = c(1, 1))
```



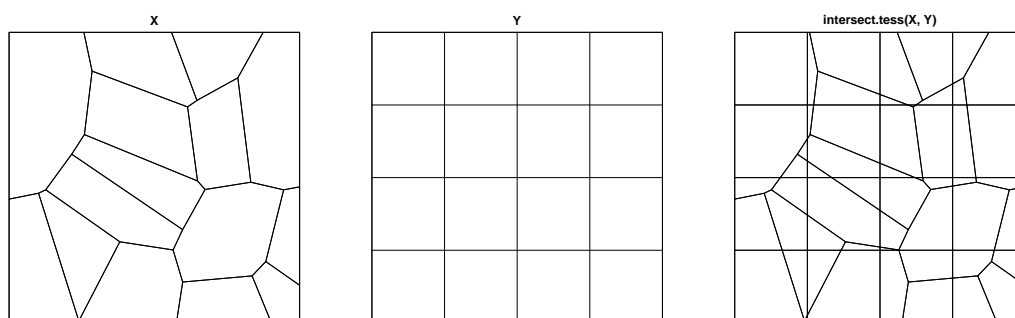
```
> plot(split(X, Z))
```

split(X, Z)



If we plot two tessellations on the same spatial domain, 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.

```
> opa <- par(mfrow = c(1, 3))
> plot(X)
> plot(Y)
> plot(intersect.tess(X, Y))
> par(opa)
```



Other operations for tessellations include:

`bdist.tess` compute distance from tile to boundary line
`chop.tess` divide tessellation along a line
`rpoislinetess` generate tessellation based on random lines

PART III. INTENSITY

Finally we can start working on statistical methods for analysing point pattern data. Part III of the workshop discusses how to investigate the intensity of a point pattern, and its dependence on covariates.

12 Exploring intensity

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

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

12.1 Uniform intensity

12.1.1 Theory

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

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

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

$$\bar{\lambda} = \frac{n(\mathbf{x})}{\text{area}(W)}$$

is an unbiased estimator of the true intensity λ .

12.1.2 Implementation in spatstat

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

```
> data(swedishpines)
> summary(swedishpines)
```

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

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

The estimated intensity is $\bar{\lambda} = 0.0074$ points per square unit. To extract this intensity value, type

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

```
[1] 0.007395833
```

The units are decimetres, so this is 0.74 points per square metre.

12.2 Inhomogeneous intensity

12.2.1 Theory

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

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

for all regions B .

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

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

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

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

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

The usual *kernel estimator* of the intensity function is

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

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

$$e(u)^{-1} = \int_W \kappa(u - v) dv \quad (2)$$

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

$$\lambda^*(u) = e(u) \int_W \kappa(u - v) \lambda(v) dv,$$

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

Intensity can also be estimated using parametric methods, as we explain in Section 15.

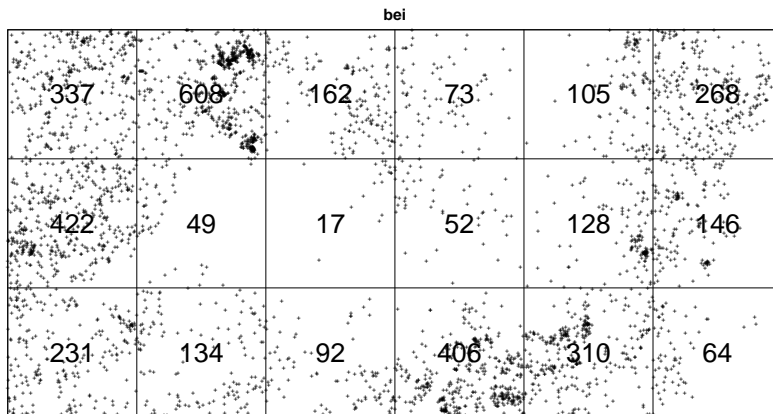
12.2.2 Implementation in spatstat

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

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

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

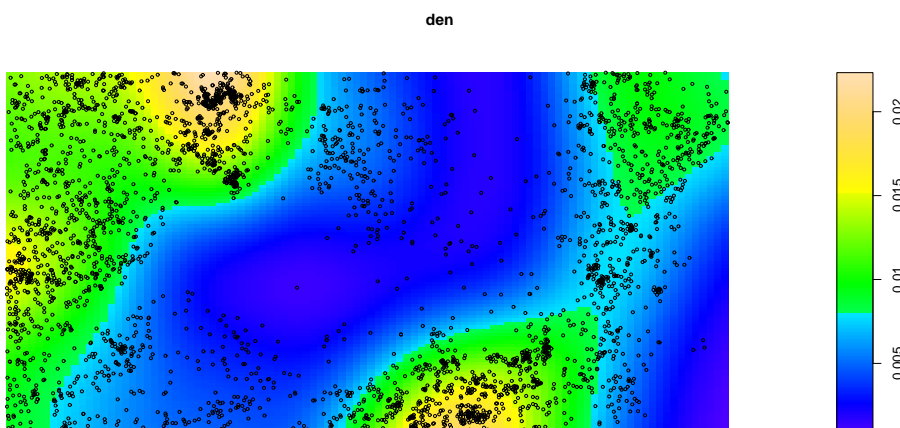
```
> Q <- quadratcount(bei, nx = 6, ny = 3)
> plot(bei, cex = 0.5, pch = "+")
> plot(Q, add = TRUE, cex = 2)
```



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

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

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



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

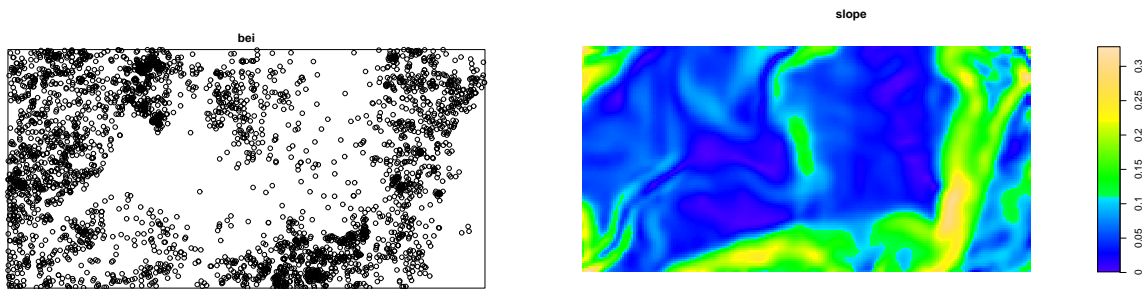
```
> persp(den)
```


13 Dependence of intensity on a covariate

13.1 Spatial covariates

Often we want to know whether 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`. It is of interest to determine whether the trees prefer steep or flat terrain, and whether they prefer a particular altitude.

```
> data(bei)
> slope <- bei.extra$grad
> par(mfrow = c(1, 2))
> plot(bei)
> plot(slope)
> par(mfrow = c(1, 1))
```



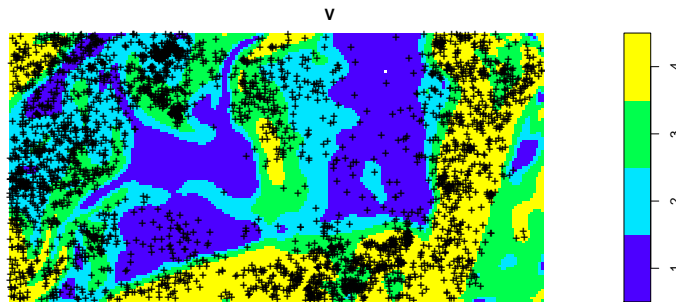
13.2 Quadrats determined by a covariate

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 slope.

```
> data(bei)
> Z <- bei.extra$grad
> b <- quantile(Z, probs = (0:4)/4)
> Zcut <- cut(Z, breaks = b, labels = 1:4)
> V <- tess(image = Zcut)
> plot(V)
> plot(bei, add = TRUE, pch = "+")
```



The call to `quantile` gave us the quartiles of the slope 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 slope.

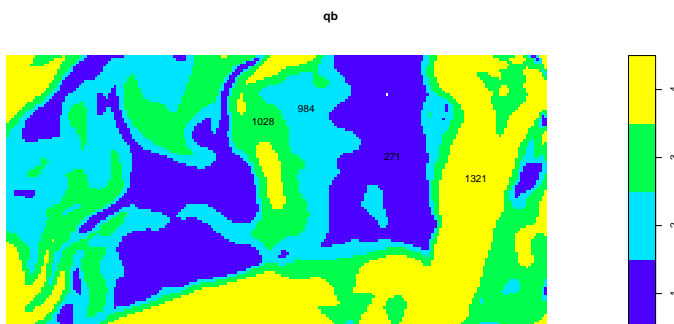
We can now use this tessellation to study the point pattern `bei`. We could invoke the commands `split`, `cut` or `by` to divide the points according to this tessellation and manipulate the sub-patterns.

The command `quadratcount` also works with tessellations:

```
> qb <- quadratcount(bei, tess = V)
> qb

tile
  1   2   3   4
271 984 1028 1321

> plot(qb)
```



The text annotations show 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 steeper slopes.

13.3 Relative distribution estimate

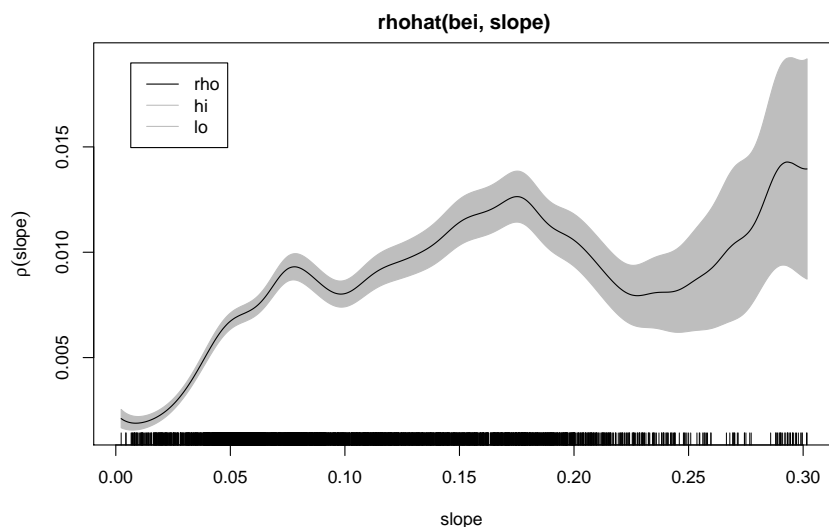
Let us assume that the intensity of the point process is a *function* of the covariate Z . At any spatial location u , let $\lambda(u)$ be the intensity of the point process, and $Z(u)$ the value of the covariate. Then we are assuming

$$\lambda(u) = \rho(Z(u))$$

where ρ is a function that we want to investigate, telling us how the intensity of points depends on the value of the covariate.

Kernel smoothing can be used to estimate the function ρ , using methods of relative distribution or relative risk, as explained in [11, 5].

```
> plot(rhohat(bei, slope))
```



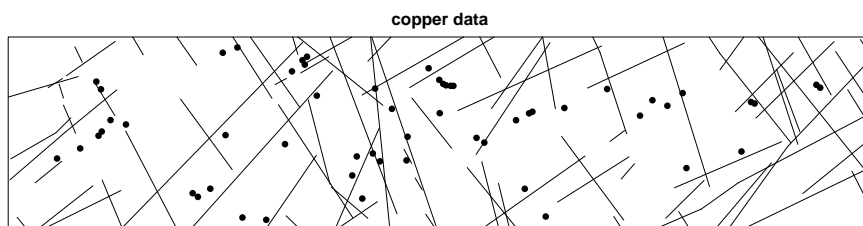
The plot is an estimate of the intensity $\rho(z)$ as a function of terrain slope z . It indicates that the *Beilschmiedia* trees are relatively unlikely to be found on flat terrain (where the slope is less than 0.05) compared to steeper slopes.

Additional capabilities will be added into `spatstat` in the near future.

13.4 Distance map

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.

```
> data(copper)
> X <- rotate(copper$SouthPoints, pi/2)
> L <- rotate(copper$SouthLines, pi/2)
> plot(X, pch = 16, main = "copper data")
> plot(L, add = TRUE)
```

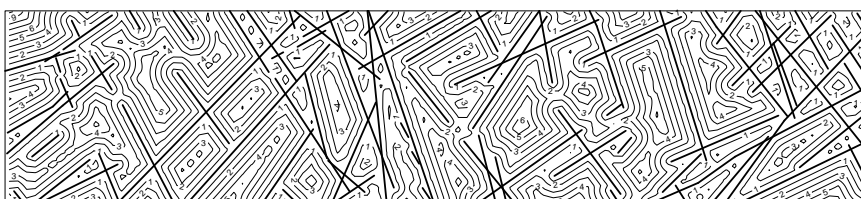


To apply the methods described above, the covariate information contained in the map of geological faults L must be converted into a covariate that is a function $Z(u)$ of spatial location u . A natural choice is the *distance function*

$$Z(u) = \text{distance from } u \text{ to } L$$

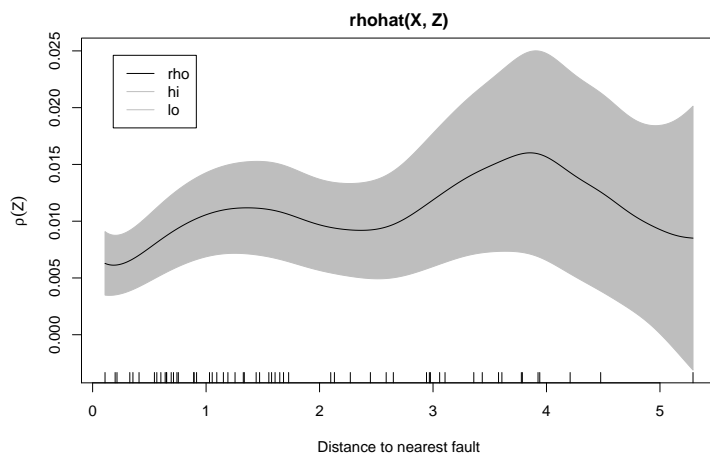
This can be computed by the command `distmap`, which returns a pixel image containing the values $Z(u)$ at a fine grid of pixels u .

```
> Z <- distmap(L)
> plot(L, lwd = 2, main = "")
> contour(Z, add = TRUE)
```



Having created this covariate image we can now apply the other techniques such as relative distributions.

```
> plot(rhohat(X, Z), xlab = "Distance to nearest fault")
```



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)
```

```
[1] 2.387029
```

In most commands in `spatstat` where a pixel image is required, a `distfun` can be used in its place. This increases the precision of many calculations. It is usually advisable to call `distfun`, unless you really need a pixel image.

PART IV. POISSON MODELS

Part IV of the workshop discusses how to assess whether a pattern is completely random, and how to model a random pattern with a spatial trend.

14 Tests of Complete Spatial Randomness

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

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

For historical reasons, many researchers are focussed on establishing that their data do *not* conform to this model. The logic is that, if a point pattern is completely random, then there is nothing “interesting” happening (because the points are completely unpredictable, and have no trend or association with anything else). Statisticians would say that the uniform Poisson process often serves as the ‘null model’ in a statistical analysis.

14.1 Definition

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

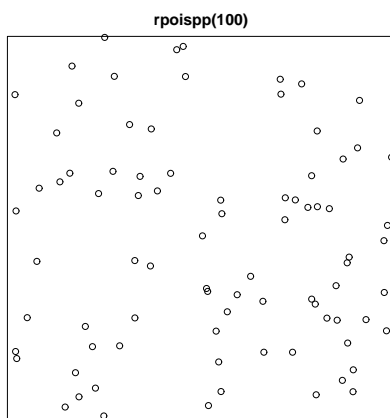
- (PP1): the number $N(\mathbf{X} \cap B)$ of points falling in any region B is a Poisson random variable;
- (PP2): the expected number of points falling in B is $\mathbb{E}[N(\mathbf{X} \cap B)] = \lambda \cdot \text{area}(B)$;
- (PP3): if B_1, B_2 are disjoint sets then $N(\mathbf{X} \cap B_1)$ and $N(\mathbf{X} \cap B_2)$ are independent random variables;
- (PP4): given that $N(\mathbf{X} \cap B) = n$, the n points are independent and uniformly distributed in B .

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

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

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

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

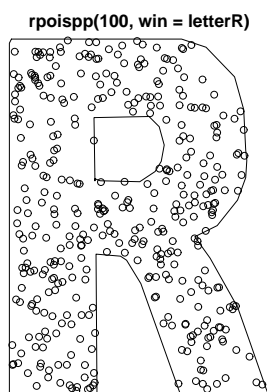


Conceptually, if we discretise a homogeneous Poisson process into infinitesimal pixels, the indicators I are independent and identically distributed, with success probability $\mathbb{P}\{I = 1\} = \lambda dA$ where dA is the infinitesimal area of a pixel.

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

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

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



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

```
> runifpoint(100, win = letterR)
```

```
planar point pattern: 100 points
window: polygonal boundary
enclosing rectangle: [2.017, 3.93] x [0.645, 3.278] units
```

14.2 Quadrat counting tests for CSR

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

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

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

Chi-squared test of CSR using quadrat counts

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

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

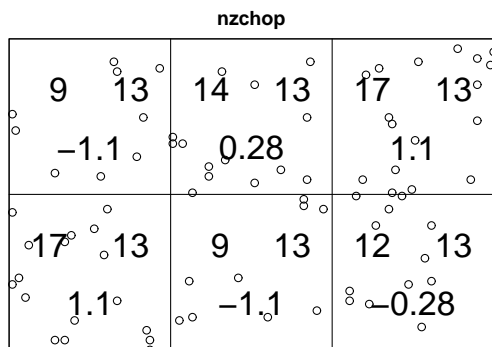
The return value `quadrat.test` also belongs to the special class "quadrat.test". Plotting the object will display the quadrats, annotated by their observed and expected counts and the Pearson residuals (observed counts n_j at top left; expected count at top right; Pearson residuals at bottom).

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

Chi-squared test of CSR using quadrat counts

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

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



The p -value can also be extracted by

```
> M$p.value
[1] 0.4065648
```

14.3 Critique

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

The main critique of the quadrat test approach is the lack of information. This is a goodness-of-fit test in which the alternative hypothesis H_1 is simply the negation of H_0 , that is, the alternative is that “the process is not a homogeneous Poisson process”. A point process may fail to satisfy properties (PP1)–(PP4) either because it violates (PP2) by having non-uniform

intensity, or because it violates (PP3)–(PP4) by exhibiting dependence between points. There are too many types of departure from H_0 .

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

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

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

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

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

14.4 Kolmogorov-Smirnov test of CSR

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

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

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

If `X` is the data point pattern, then

```
> kstest(X, covariate)
```

performs the test, 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.

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

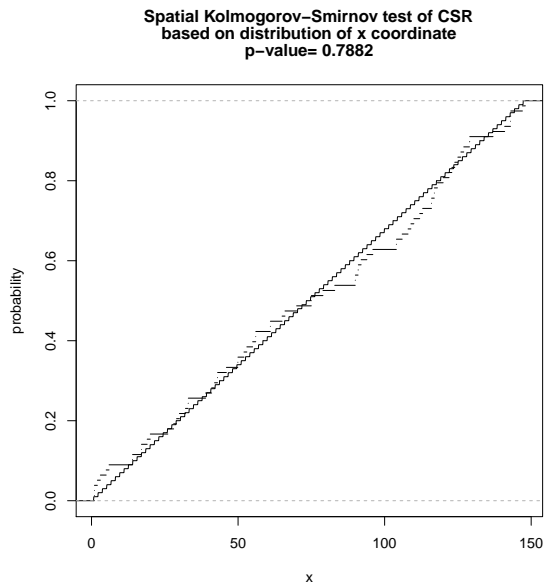
```
> kstest(nzchop, "x")
```

```
Spatial Kolmogorov-Smirnov test of CSR
```

```
data: covariate x evaluated at points of nzchop
      and transformed to uniform distribution under CSR
D = 0.0719, p-value = 0.7882
alternative hypothesis: two-sided
```

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

```
> KS <- kstest(nzchop, "x")
> plot(KS)
> pval <- KS$p.value
```



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

14.5 Using covariate data

We are often interested in testing whether the point pattern intensity depends on a covariate. For example, our preliminary analysis of the tropical rainforest pattern `bei` in Section 13.2 suggested that the density of trees depends on terrain slope. To test this formally we can divide the region into irregular quadrats according to the terrain slope, and apply the χ^2 test. The command `quadrat.test` accepts a tessellation and uses the tiles of the tessellation as the quadrats:

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

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

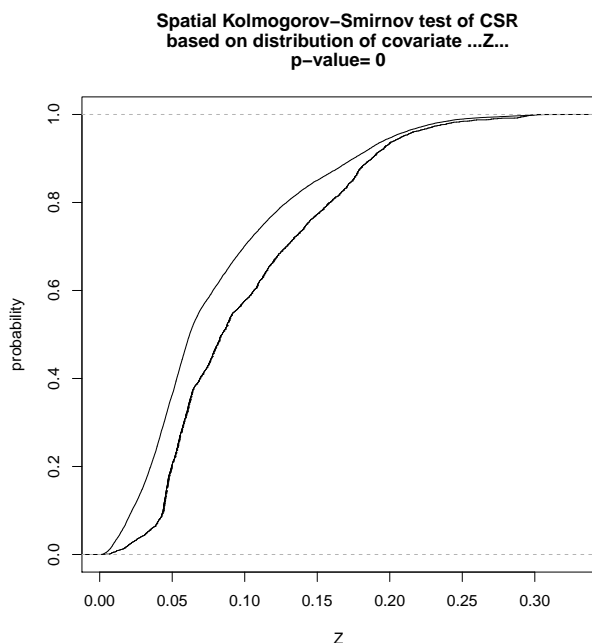
Because of the large counts in these regions, we can probably ignore concerns about independence, and conclude that the trees are not uniform in their intensity.

A more powerful test (if that were needed!) is the Kolmogorov-Smirnov test using the slope covariate:

```
> KS <- kstest(bei, Z)
> plot(KS)
> KS
```

Spatial Kolmogorov-Smirnov test of CSR

```
data: covariate Z evaluated at points of bei
      and transformed to uniform distribution under CSR
D = 0.1948, p-value < 2.2e-16
alternative hypothesis: two-sided
```



The Kolmogorov-Smirnov test would typically be preferred if the covariate Z has continuously-varying numerical values. If the covariate is a factor or discrete variable, then the Kolmogorov-Smirnov test is ineffective because of tied values, and the χ^2 test based on quadrat counts would be preferable.

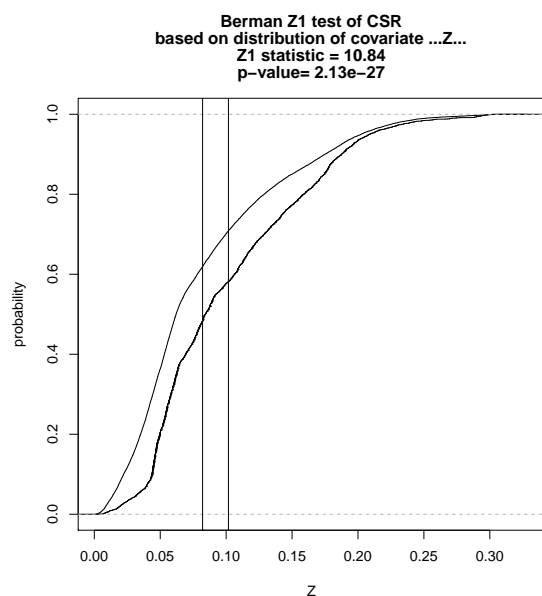
14.6 Berman's tests

Berman [20] proposed two tests for the dependence of a point process on a spatial covariate. These tests are optimal against a certain class of alternatives. They are performed by the command `bermantest` which is analogous to `kstest`.

```
> B <- berrmantest(bei, Z)
> plot(B)
> B
```

Berman Z1 test of CSR

```
data: covariate Z evaluated at points of bei
Z1 = 10.844, p-value < 2.2e-16
alternative hypothesis: two-sided
```



Two vertical lines show the mean values of these distributions. If the model is correct, the two curves should be close; the test is based on comparing the two vertical lines.

When the covariate Z is the distance to a spatial pattern, another useful diagnostic is Foxall's J -function [40], available using `Jfox`.

15 Maximum likelihood for Poisson processes

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

15.1 Inhomogeneous Poisson process

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

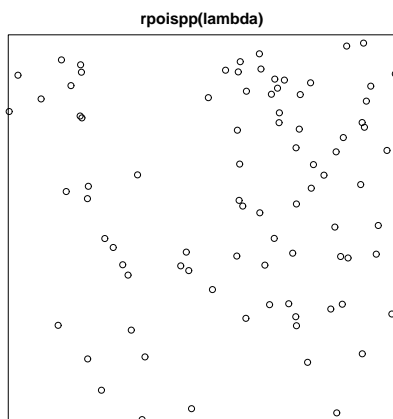
(PP2'): the number $N(\mathbf{X} \cap B)$ of points falling in a region B has expectation

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

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

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

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



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

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

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

15.2 Likelihood methods

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

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

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

$$\hat{\lambda} = \frac{n(\mathbf{x})}{\text{area}(W)}$$

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

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

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

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

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

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

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

15.3 Fitting Poisson processes in spatstat

Mark Berman and Rolf Turner [21] (see also [47, 25, 48]) developed a clever computational device for finding the MLE of θ by exploiting a formal similarity between the Poisson log-likelihood (4) and that of a loglinear Poisson regression.

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

$$\log \lambda_\theta(u) = \theta \cdot S(u) \quad (5)$$

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

15.3.1 Model-fitting function

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

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

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

To fit the homogeneous Poisson model:

```
> ppm(bei, ~1)
```

Stationary Poisson process

Uniform intensity: 0.007208

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

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

Nonstationary Poisson process

Trend formula: $\sim x + y$

Fitted coefficients for trend formula:

(Intercept)	x	y
-4.7245290274	-0.0008031288	0.0006496090

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

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

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

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

Nonstationary Poisson process

Trend formula: \sim polynom(x, y, 2)

Fitted coefficients for trend formula:

(Intercept)	polynom(x, y, 2)[x]	polynom(x, y, 2)[y]
-4.275762e+00	-1.609187e-03	-4.895166e-03
polynom(x, y, 2)[x^2]	polynom(x, y, 2)[x.y]	polynom(x, y, 2)[y^2]
1.625968e-06	-2.836387e-06	1.331331e-05

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

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

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

Nonstationary Poisson process

Trend formula: ~side(x)

Fitted coefficients for trend formula:

```
(Intercept) side(x)right
-4.8026460   -0.2792705
```

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

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

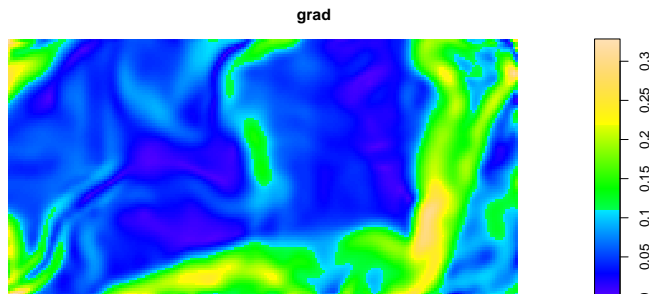
Rather than relying on such interpretations, it is prudent to use the command `predict` to compute predicted values of the model, as explained in Section 15.4 below.

15.3.2 Models involving spatial covariates

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

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

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



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

$$\lambda(u) = \exp(\beta_0 + \beta_1 Z(u)) \quad (6)$$

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

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

Nonstationary Poisson process

Trend formula: ~slope

Fitted coefficients for trend formula:

```
(Intercept)      slope
-5.390553      5.022021
```

In the call to `ppm`, the argument `covariates` should be a list of `name=value` pairs that provide the covariate data for the model. Every variable name that appears in the model formula should match one of the `names` in this list. Each value should be either a pixel image, a `function(x,y)` in the R language, a window of class `"owin"`, or a single number.

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

$$\lambda(u) = \exp(-5.390553 + 5.022021 Z(u)). \quad (7)$$

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

$$\lambda(u) = \beta Z(u) \quad (8)$$

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

$$\log \lambda(u) = \log \beta + \log Z(u). \quad (9)$$

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

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

Nonstationary Poisson process

Trend formula: ~offset(log(slope))

Fitted coefficients for trend formula:

```
(Intercept)
-2.427127
```

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

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

15.4 Fitted models

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

15.4.1 Standard operations

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

<code>print</code>	print basic information
<code>summary</code>	print detailed summary information
<code>plot</code>	plot the fitted intensity
<code>predict</code>	compute the fitted intensity
<code>fitted</code>	compute the fitted intensity at data points
<code>update</code>	re-fit the model
<code>coef</code>	extract the fitted coefficient vector $\hat{\theta}$
<code>vcov</code>	variance-covariance matrix of $\hat{\theta}$
<code>anova</code>	analysis of deviance
<code>logLik</code>	log-likelihood value
<code>formula</code>	extract the model formula
<code>terms</code>	extract the terms in the model
<code>model.matrix</code>	compute the design matrix

For information on these methods, see `print.ppm`, `summary.ppm`, `plot.ppm` etc. The following commands also work on "ppm" objects:

<code>step</code>	stepwise model selection
<code>drop1</code>	one step model deletion
<code>AIC</code>	Akaike Information Criterion

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

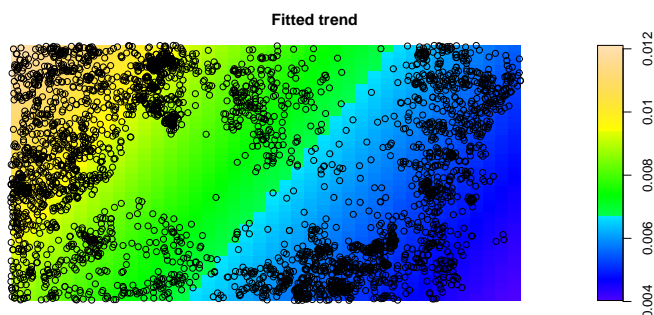
Nonstationary Poisson process

Trend formula: $\sim x + y$

Fitted coefficients for trend formula:

(Intercept)	x	y
-4.7245290274	-0.0008031288	0.0006496090

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



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

real-valued pixel image

50 x 50 pixel array (ny, nx)

enclosing rectangle: [0, 1000] x [0, 500] metres

```

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

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

> coef(fit)

      (Intercept)           x           y
-4.7245290274 -0.0008031288  0.0006496090

> vcov(fit)

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

> sqrt(diag(vcov(fit)))

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

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

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

```

This is the fitted model with intensity function

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

with the following estimates:

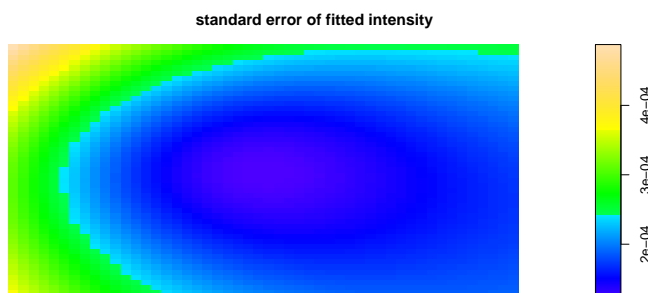
i	θ_i	$\text{var}(\hat{\theta}_i)$	standard deviation
0	-4.724529	0.001854091	0.04305915
1	-0.0008031288	3.437842e-09	5.863311e-05
2	0.000649609	1.338955e-08	0.0001157132

It is also possible to compute the standard error of the fitted intensity $\lambda_{\theta}(u)$ at each location u , as a pixel image. Use `predict(fit, type="se")` or `plot(fit, se=TRUE)`.

```

> SE <- predict(fit, type = "se")
> plot(SE, main = "standard error of fitted intensity")

```



If the model formula involves transformations of the original covariates, then `model.matrix(fit)` gives the design matrix whose columns contain these transformed covariates, and `model.images(fit)` gives a list of pixel images of these transformed covariates.

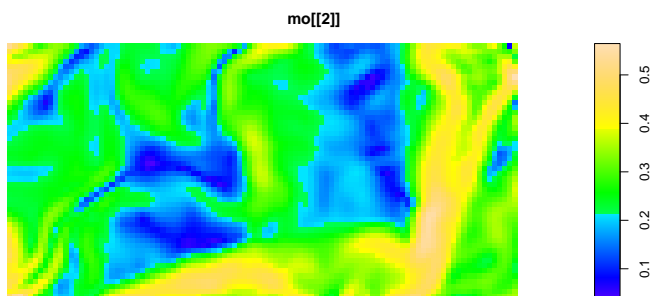
```
> fit <- ppm(bei, ~sqrt(slope) + x, covariates = list(slope = grad))
> mo <- model.images(fit)
> mo
```

```
(Intercept) :
real-valued pixel image
100 x 100 pixel array (ny, nx)
enclosing rectangle: [0, 1000] x [0, 500] metres
```

```
sqrt(slope) :
real-valued pixel image
100 x 100 pixel array (ny, nx)
enclosing rectangle: [0, 1000] x [0, 500] metres
```

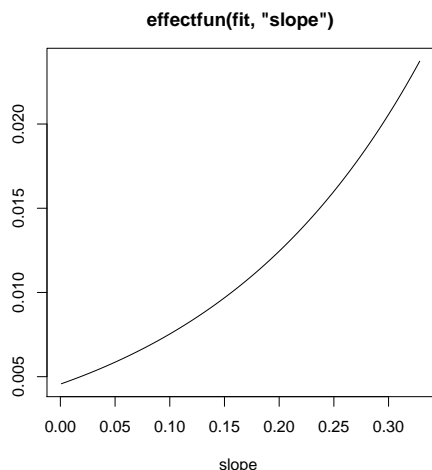
```
x :
real-valued pixel image
100 x 100 pixel array (ny, nx)
enclosing rectangle: [0, 1000] x [0, 500] metres
```

```
> plot(mo[[2]])
```



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. This is chiefly useful if the model only has one covariate.

```
> fit <- ppm(bei, ~slope, covariates = list(slope = grad))
> plot(effectfun(fit, "slope"))
```



15.4.2 Model selection

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

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

Analysis of Deviance Table

```
Model 1: .mpl.Y ~ 1
Model 2: .mpl.Y ~ slope
  Resid. Df Resid. Dev Df Deviance P(>|Chi|)
1      20507      18728
2      20506      18346  1    382.25 < 2.2e-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 (6). The p -value is extremely small, indicating rejection of CSR in favour of the alternative. (Please ignore the columns `Resid.Df` and `Resid.Dev` which are artefacts of the discretisation. Only the deviance difference and the difference in degrees of freedom are valid.)

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

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

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

```
[1] 42496.65
```

```
> AIC(fitnull)
```

```
[1] 42763.92
```

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

Automatic model selection can be performed using `step`. 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)
```

```
Start:  AIC=-580.96
```

```
~x + y
```

	Df	AIC
- x	1	-582.29
- y	1	-581.91
<none>		-580.96

```
Step:  AIC=-582.29
```

```
~y
```

	Df	AIC
- y	1	-583.25
<none>		-582.29

```
Step:  AIC=-583.25
```

```
~1
```

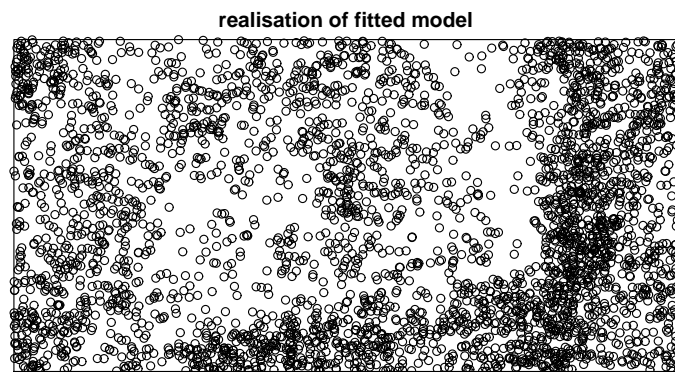
```
Stationary Poisson process
```

```
Uniform intensity:      85
```

15.5 Simulating the fitted model

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

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



It is 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.

16 Checking a fitted Poisson model

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

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

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

16.1 Goodness-of-fit

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

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

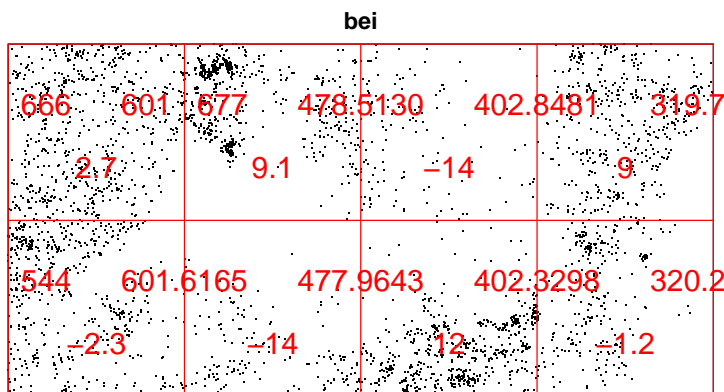
```
> data(bei)
> fit <- ppm(bei, ~x)
> M <- quadrat.test(fit, nx = 4, ny = 2)
> M
```

Chi-squared test of fitted Poisson model fit using quadrat counts

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

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

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



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

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

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

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

```
> kstest(fit, "y")
```

Spatial Kolmogorov-Smirnov test of inhomogeneous Poisson process

```
data: covariate y evaluated at points of bei
      and transformed to uniform distribution under fit
D = 0.1068, p-value < 2.2e-16
alternative hypothesis: two-sided
```

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

16.2 Validation using residuals

16.2.1 Residuals

Residuals from the fitted model are an important diagnostic tool in other areas of applied statistics, but in spatial statistics they have only recently been developed ([52, 60], [58, pp. 49–50], [12]).

For a fitted Poisson process model, with fitted intensity $\hat{\lambda}(u)$, the predicted number of points falling in any region B is $\int_B \hat{\lambda}(u) du$. Hence the **residual** in each region $B \subset \mathbb{R}^2$ is defined [12] to be the *observed minus predicted* number of points falling in B : [12]

$$R(B) = n(\mathbf{x} \cap B) - \int_B \hat{\lambda}(u) du \quad (11)$$

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

These residuals are closely related to the residuals for quadrat counts that were used above. Taking the set B to be one of our quadrats, the ‘observed’ quadrat count is $n(\mathbf{x} \cap B)$. The ‘expected’ quadrat count is $\hat{\lambda} \text{area}(B)$ 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’ is **observed** – **expected** = $n(\mathbf{x} \cap B) - \int_B \hat{\lambda}(u) du$.

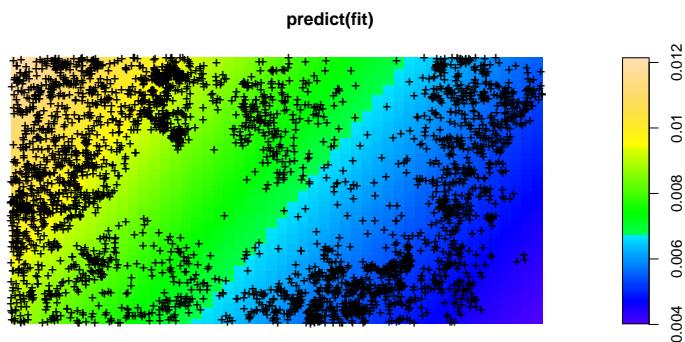
16.2.2 Residual measure

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

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

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

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

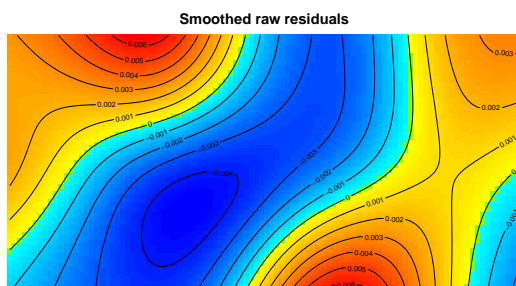


Each data point should be visualised as a charge of +1, while the colour image indicates a negative charge density. If the model is true then these positive and negative charges should even out to zero.

16.2.3 Smoothed residuals

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

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



This is an image plot of the ‘smoothed residual field’

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

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

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

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

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

Here κ is the smoothing kernel and $e(u)$ is the edge correction (2) on page 79. The difference (12) should be approximately zero if the model is true.

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

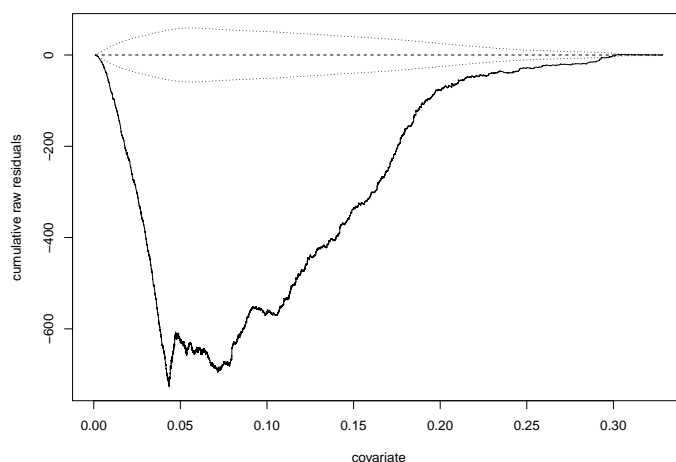
16.2.4 Lurking variable plot

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

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

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

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



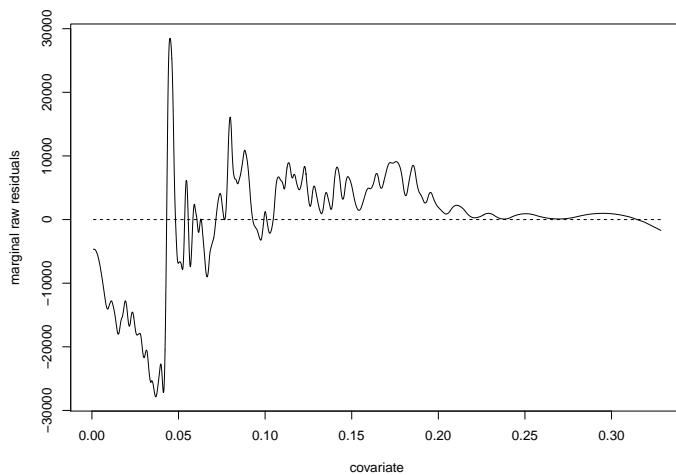
Note that the lurking variable plot typically starts and ends at the horizontal axis, since (for any model with an intercept term) the total residual for the entire window W must equal zero. This is analogous to the fact that the residuals in linear regression sum to zero.

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

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

It can be helpful to display the derivative $C'(z)$, which often indicates which values of z are associated with a lack of fit.

```
> lurking(fitx, grad, type = "raw", cumulative = FALSE)
```



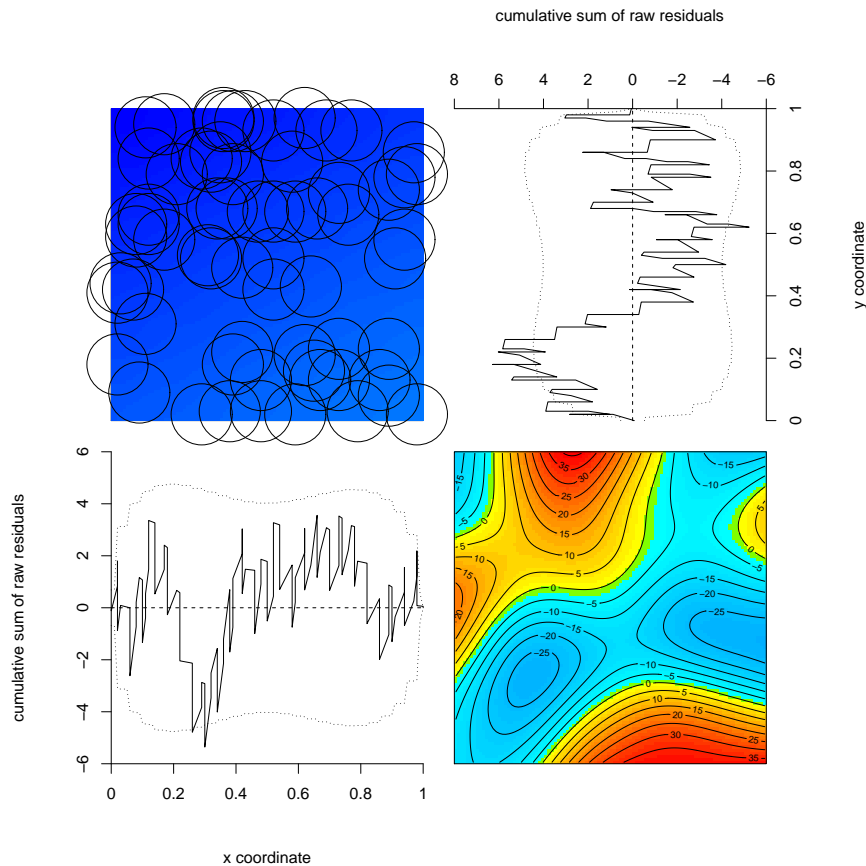
The derivative is estimated using a smoothing spline and you may need to tweak the smoothing parameters (argument `splineargs`) to get a useful plot. Also the package currently does not plot significance bands for $C'(z)$.

Additional techniques described in [5] will soon be added to `spatstat`.

16.2.5 Four-panel plot

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

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



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

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

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

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

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

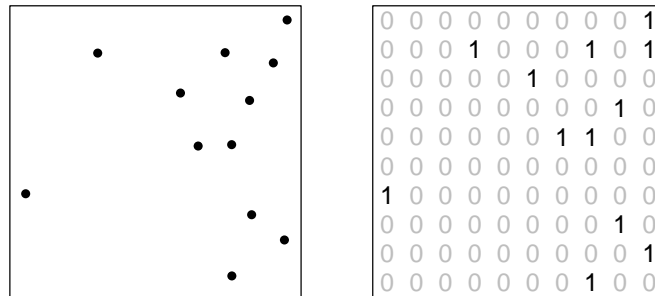
16.2.6 Caveats

The residual plots described 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, that is, for checking the properties (PP3)–(PP4) of a Poisson process listed on page 88.

Effective diagnostics of independence or dependence between points include the K -function (section 19.4) and a Q–Q plot of the residuals (section 28.2.3).

17 Spatial logistic regression

A popular technique for analysing point pattern data in Geographical Information Systems is *spatial logistic regression* [62, 1]. 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.

In fact, spatial logistic regression is very close to a Poisson point process model [4]. However, `spatstat` provides facilities for performing spatial logistic regression, for comparison purposes. Models are fitted using the command `slrm`.

```
> data(copper)
> 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
-4.69015189  0.05211539
```

`slrm` produces a “fitted spatial logistic regression model” object of class “`slrm`”. Methods for this class include `print`, `plot`, `predict`, `coef`, `fitted`, `update`, `terms`, `formula`, `anova` and `logLik`. You can also use `step` for model selection.

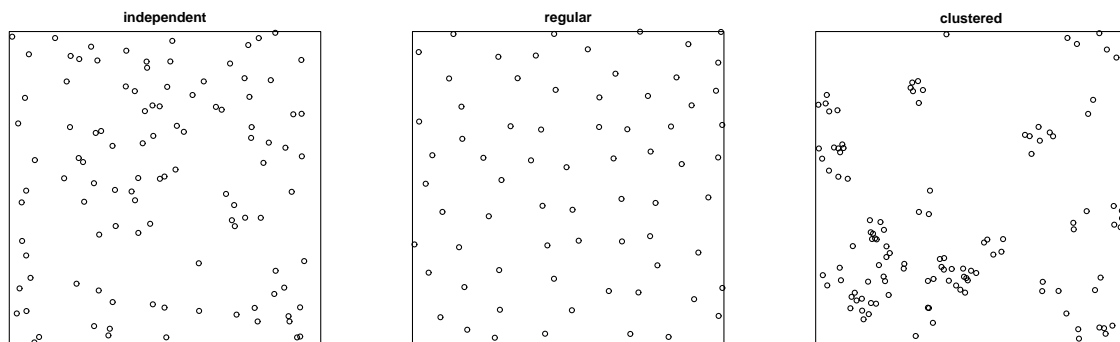
PART V. INTERACTION

Part V of the workshop explains how to investigate dependence between the points in a point pattern.

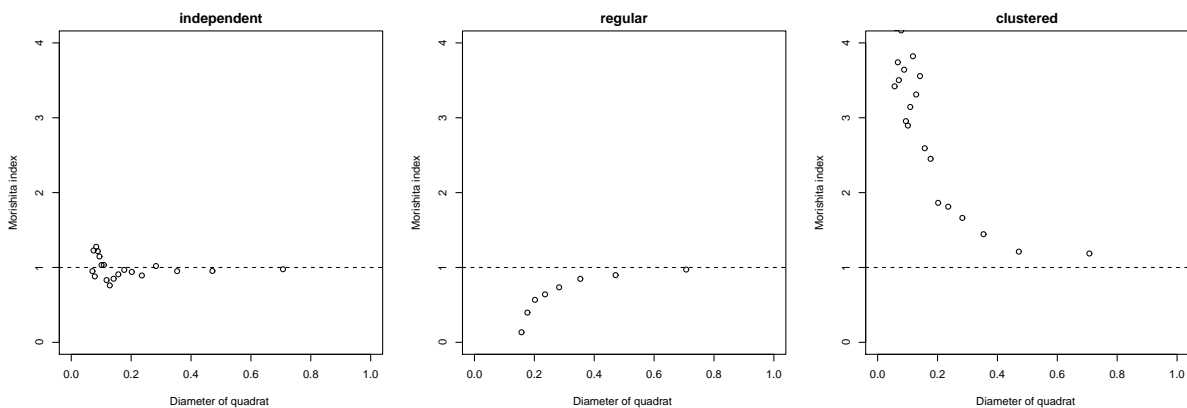
18 Exploring dependence between points

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

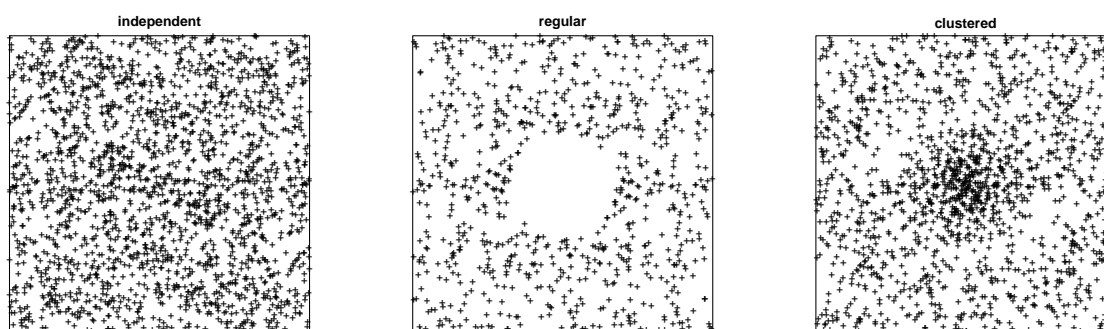
Classical writers suggested a simple trichotomy between ‘independence’ (the Poisson process), ‘regularity’ (where points tend to avoid each other), and ‘clustering’ (where points tend to be close together). [The concept of ‘clustering’ does not imply that the points are organised into identifiable ‘clusters’;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



One simple diagnostic for dependence between points is a *Morishita plot*. The spatial domain is divided into quadrats, and the χ^2 statistic based on quadrat counts is computed. The quadrats are repeatedly subdivided. The Morishita plot shows the χ^2 statistic against the linear size of the quadrats. It is computed by the command `mipLOT`.



A more sophisticated option is the *Fry plot*. This is a scatterplot of the vector differences $x_j - x_i$ between all pairs of distinct points in the pattern. Suppose you have printed the point pattern on a sheet of paper. Take a sheet of tracing paper, and mark a red dot in the middle. Now place the tracing paper over the point pattern, and move it until the red dot coincides with one of the data points. Now copy all the other data points onto the tracing paper. Repeat this for every data point, and you have the Fry plot. It is computed by the command `fryplot`.



While these graphical diagnostics can be very useful for spotting features of the point pattern, they involve subjective interpretation.

19 Distance methods for point patterns

19.1 Distances

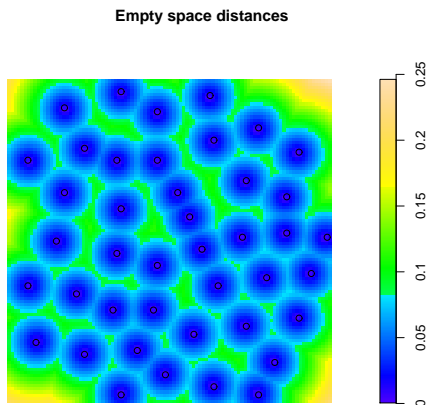
The main classical techniques for investigating interpoint interaction are *distance methods*, based on measuring the distances between points. Specifically we may consider

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

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

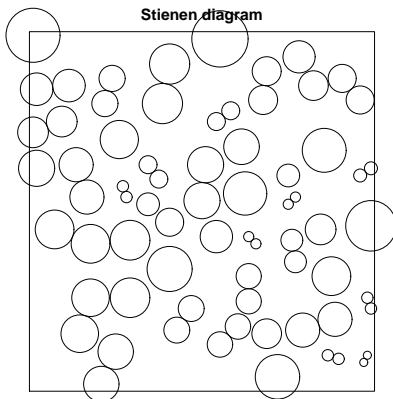
- `pairdist(X)` returns the matrix of pairwise distances.
- `ndist(X)` returns the vector of nearest neighbour distances.
- `distmap(X)` returns a pixel image whose pixel values are the empty space distances to the pattern `X` measured from every pixel.

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



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

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



In order to develop formal statistical analysis, we typically use the empirical cumulative distribution function of these distances. This is explained in the following subsections.

19.2 Empty space distances

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

The distance

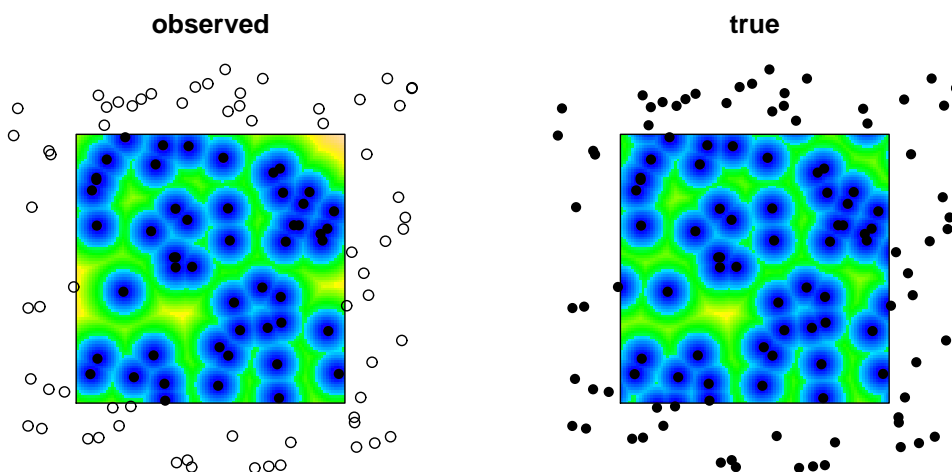
$$d(u, \mathbf{x}) = \min\{\|u - x_i\| : x_i \in \mathbf{x}\}$$

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

19.2.1 Edge effects

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

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



19.2.2 Empty space function F

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

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

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

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

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

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

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

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

$$\hat{F}(r) = \sum_j e(u_j, r) \mathbf{1} \{d(u_j, \mathbf{x}) \leq r\} \quad (15)$$

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

The edge effect problem can also be regarded as a form of censoring (analogous to right-censoring in survival data), as first pointed out by CSIRO researcher Geoff Laslett [46]. A counterpart of the Kaplan-Meier estimator is available. For further information see [13].

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

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

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

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

19.2.3 Implementation in spatstat

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

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

Function value object (class fv)

for the function $r \rightarrow F(r)$

Entries:

id	label	description
---	-----	-----
r	r	distance argument r
theo	F[pois](r)	theoretical Poisson F(r)
cs	F[cs](r)	Chiu-Stoyan estimate of F(r)
rs	F[bord](r)	border corrected estimate of F(r)
km	F[km](r)	Kaplan-Meier estimate of F(r)
hazard	hazard(r)	Kaplan-Meier estimate of hazard function lambda(r)

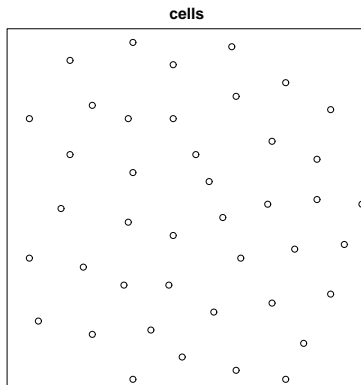
Default plot formula:

. ~ r

<environment: 0x6a251e0>

Recommended range of argument r: [0, 0.085]

Available range of argument r: [0, 0.2975]



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

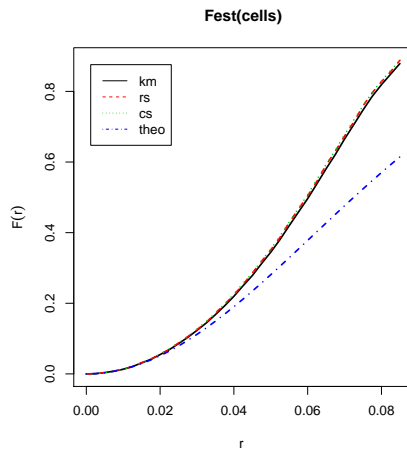
The value returned by `Fest` is an object of class `"fv"` ("function value table"). This is effectively a data frame with some extra information. The printout for `Fc` indicates that the columns in the data frame are named `r`, `theo`, `cs`, `rs`, `km` and `hazard`. The first column `r` contains a sequence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

If you don't want to compute all these estimates (for example, for the sake of efficiency), you can use the argument `correction` to specify which estimate or estimates are required.

```
> Fest(cells, correction = "km")
> Fest(cells, correction = c("km", "cs"))
```

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

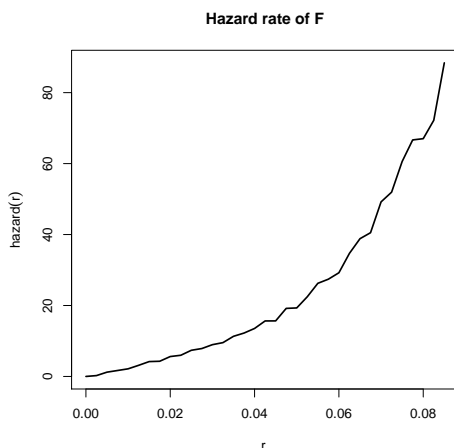
	lty	col	key	label	meaning
km	1	1	km	F[km](r)	Kaplan-Meier estimate of F(r)
rs	2	2	rs	F[bord](r)	border corrected estimate of F(r)
cs	3	3	cs	F[cs](r)	Chiu-Stoyan estimate of F(r)
theo	4	4	theo	F[pois](r)	theoretical Poisson F(r)



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

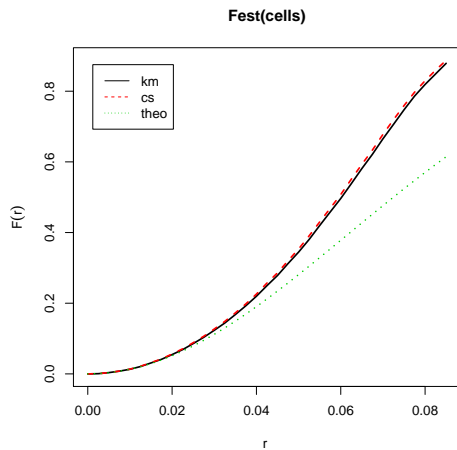
You'll notice that, by default, the hazard rate `hazard` was not plotted. The choice of estimates to be plotted, and the style in which they are plotted, are controlled by the second argument to `plot.fv`, which should be an R language formula involving the identifier names `r`, `theo`, `cs`, `rs`, `km`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

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



To plot only the Kaplan-Meier and Chiu-Stoyan estimators with the theoretical curve,

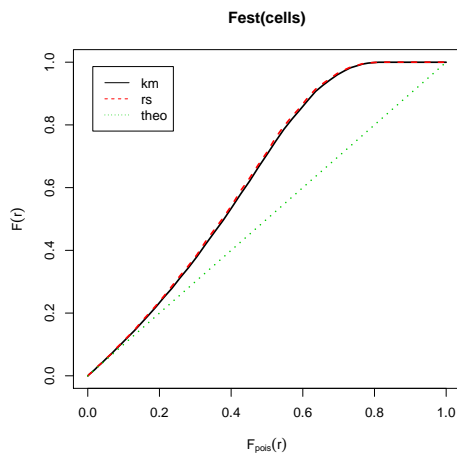
```
> plot(Fest(cells), cbind(km, cs, theo) ~ r)
```



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

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

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



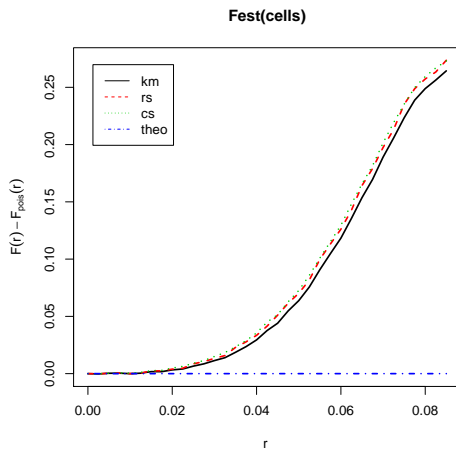
(including `theo` on the left side here gives us the diagonal line).

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

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

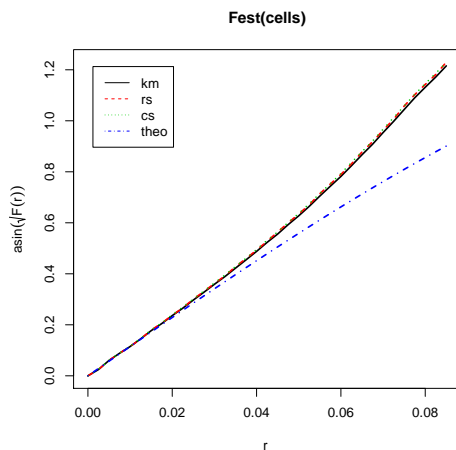
Transformations can be applied to these function values. For example, to subtract the theoretical Poisson value from the estimates,

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



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

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



19.3 Nearest neighbour distances

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

19.3.1 G function

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

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

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

The empirical distribution function of the observed nearest-neighbour distances

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

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

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

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

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

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

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

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

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

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

```
Function value object (class fv)
for the function r -> G(r)
```

```
Entries:
```

id	label	description
r	r	distance argument r
theo	G[pois](r)	theoretical Poisson G(r)
han	G[han](r)	Hanisch estimate of G(r)
rs	G[bord](r)	border corrected estimate of G(r)
km	G[km](r)	Kaplan-Meier estimate of G(r)
hazard	hazard(r)	Kaplan-Meier estimate of hazard function lambda(r)

```
Default plot formula:
```

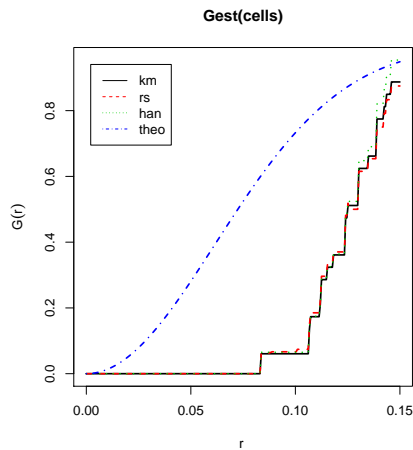
```
. ~ r
```

```
<environment: 0x6ab90d8>
```

```
Recommended range of argument r: [0, 0.15]
```

```
Available range of argument r: [0, 0.29539]
```

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



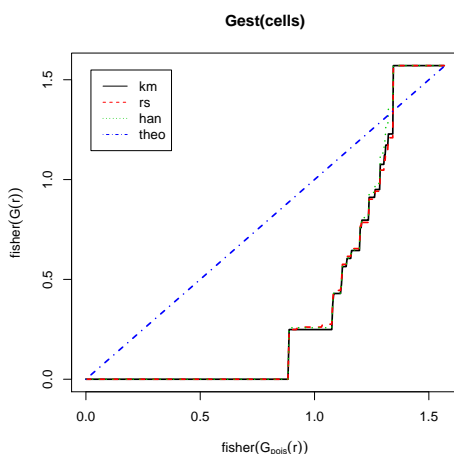
The estimate of $G(r)$ suggests strongly that the pattern is regular. Indeed, $\hat{G}(r)$ is zero for $r \leq 0.07$ which indicates that there are no nearest-neighbour distances shorter than 0.07.

Common ways of plotting \hat{G} include:

$\hat{G}(r)$ and $G_{\text{pois}}(r)$ plotted against r	<code>plot(Gest(X))</code>
$\hat{G}(r) - G_{\text{pois}}(r)$ plotted against r	<code>plot(Gest(X), . - theo ~ r)</code>
$\hat{G}(r)$ plotted against $G_{\text{pois}}(r)$ in P-P style	<code>plot(Gest(X), . ~ theo)</code>

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

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



19.4 Pairwise distances and the K function

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

Ripley [54] defined the K -function for a stationary point process so that $\lambda K(r)$ is the expected number of other points of the process within a distance r of a typical point of the process. Formally

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

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

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

regardless of the intensity.

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

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

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

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

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

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

```
Function value object (class fv)
```

```
for the function r -> K(r)
```

```
Entries:
```

id	label	description
---	-----	-----
r	r	distance argument r
theo	K[pois](r)	theoretical Poisson K(r)
border	K[bord](r)	border-corrected estimate of K(r)
trans	K[trans](r)	translation-corrected estimate of K(r)
iso	K[iso](r)	Ripley isotropic correction estimate of K(r)

```
Default plot formula:
```

```
. ~ r
```

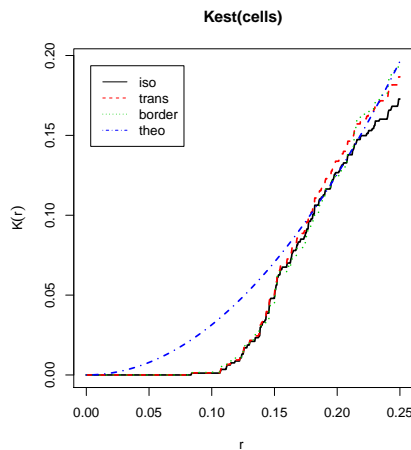
```
<environment: 0x60d97a8>
```

```
Recommended range of argument r: [0, 0.25]
```

```
Available range of argument r: [0, 0.25]
```



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



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

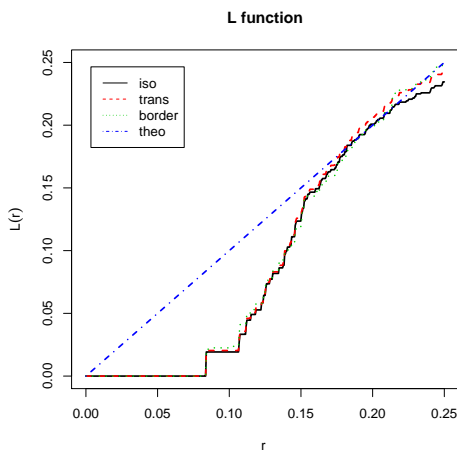
A commonly-used transformation of K is the L -function

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

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

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

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



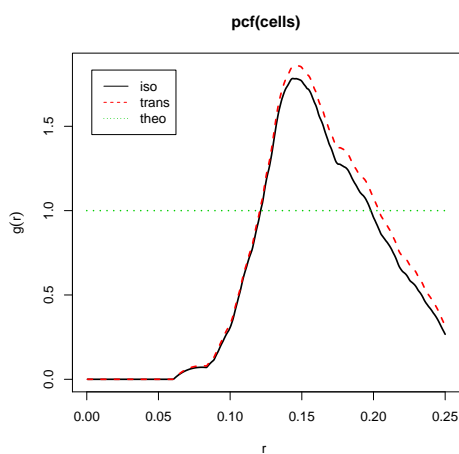
Another related summary function is the *pair correlation function*

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

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

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

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



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

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

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

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

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

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

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

19.5 J function

A useful combination of F and G is the J function [64]

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

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

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

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

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

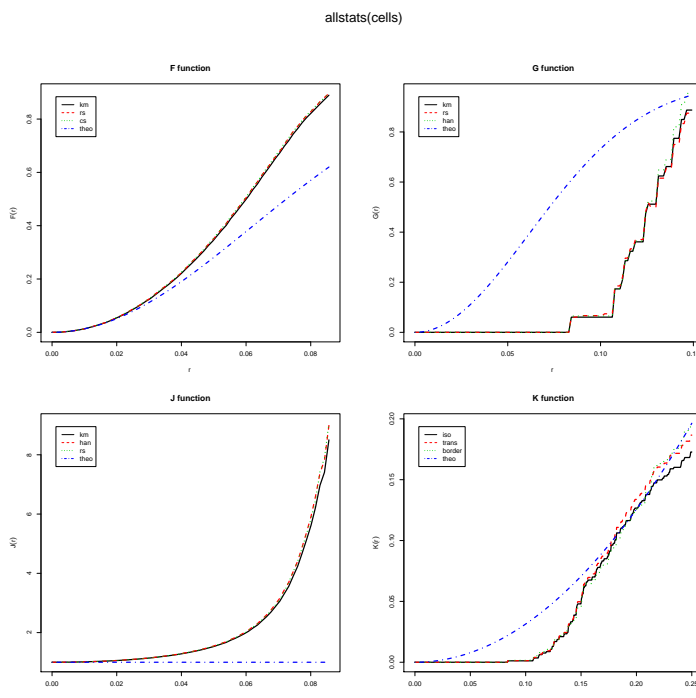
$$J(t) = \frac{\lambda_1}{\lambda_1 + \lambda_2} J_1(t) + \frac{\lambda_2}{\lambda_1 + \lambda_2} J_2(t)$$

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

The J function is computed by `Jest`.

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

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



19.6 Manipulating and plotting summary functions

As explained above, the summary function commands `Fest`, `Gest`, `Kest`, `Lest`, `pcf` etc. return a function value table (an object of class "fv"). This is a data frame (i.e. it also belongs to the class "data.frame") with some extra information. One column of the data frame contains

values of the distance argument r , while the other columns contain different estimates of the value of the function, or the theoretical value of the function under CSR.

The following operations are defined on this class:

```
print.fv      print a summary description
plot.fv      plot the function estimates
as.data.frame strip extra information (returns a data frame)
$            extract one column (returns a numeric vector)
[.fv        extract subset (returns an "fv" object)
with.fv     perform calculations with specific columns of data frame
eval.fv     perform calculation on all columns of data frame
cbind.fv    combine several "fv" objects
collapse.fv combine several redundant "fv" objects
bind.fv     combine an "fv" object and a data frame
smooth.fv   apply smoothing to function values
stieltjes   compute Stieltjes integral with respect to function
```

To make life easier, there are several options for manipulating the function values.

To manipulate or combine one or more columns of the data frame, it is typically easiest to use the command `with.fv`, which is a method for the generic command `with`. For example:

```
> data(redwood)
> K <- Kest(redwood)
> y <- with(K, iso - theo)
> x <- with(K, r)
```

In this case, 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 `K` contains columns named `r`, `iso` and `theo`.

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 symbol `.` representing "all recommended estimates of the function". Thus:

```
> L <- with(K, sqrt(./pi))
```

computes the estimates of $L(r) = \sqrt{K(r)}$ by all the available edge correction methods. In this case, the result `L` is an "fv" object. You can also get a single numeric result, for example

```
> with(Kest(redwood), max(abs(iso - theo)))
```

```
[1] 0.04945199
```

To plot a transformed function, you can also use the `plot` method. Its second argument is a formula in the R language. The left side of the formula represents what curve or curves will be plotted on the y axis, and the right side determines the x variable for the plot. Thus:

```
> plot(K, sqrt(./pi) ~ r)
```

plots the estimates of $L(r) = \sqrt{K(r)}$, by all the available edge correction methods, against r . The symbol `.` again signifies “all recommended estimates of the function”. The left hand side of the formula may use the command `cbind` to indicate that several different curves should be plotted. For example, to plot only two curves, giving the isotropic correction estimate and the theoretical value of $K(r)$:

```
> plot(K, cbind(iso, theo) ~ r)
```

The right-hand side 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.

To manipulate or combine one or more “fv” objects, use `eval.fv`. Its argument is an expression containing the names of “fv” objects. For example

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

This can be used to perform computations involving several “fv” objects provided they are compatible (they must have the same vector of r values).

```
> K1 <- Kest(redwood)
> K2 <- Kest(runifpoint(redwood$n, redwood>window))
> DK <- eval.fv(K1 - K2)
```

If these facilities are not sufficient, then direct access to the function values is also possible. A single column of the data frame can be extracted using the `$` operator in the usual way. The object can also be converted to a data frame using `as.data.frame` and the entries extracted in any desired fashion.

19.7 Caveats

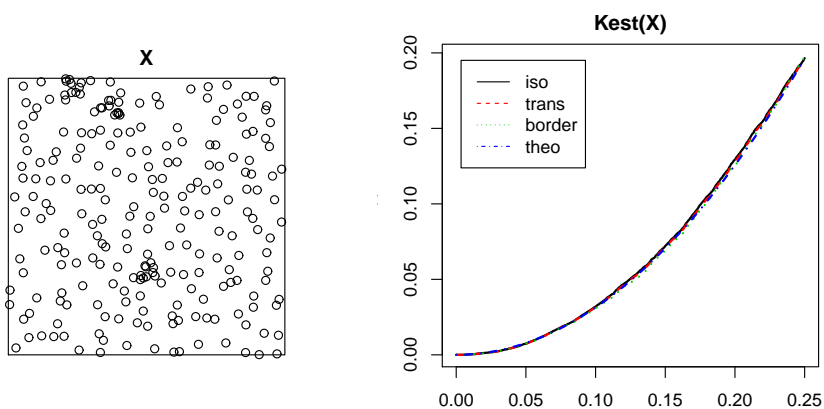
The use of summary functions for analysing point patterns has become established across wide areas of applied science, following Ripley’s influential paper [54] and many subsequent textbooks [30, 33, 35, 63, 56, 57, 61] until quite recently.

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

1. the functions F , G and K are defined and estimated under the *assumption that the point process is stationary (homogeneous)*.
2. these summary functions *do not completely characterise the process*.
3. if the process is not stationary, deviations between the empirical and theoretical functions (e.g. \hat{K} and K_{pois}) are not necessarily evidence of interpoint interaction, since they may also be attributable to variations in intensity.

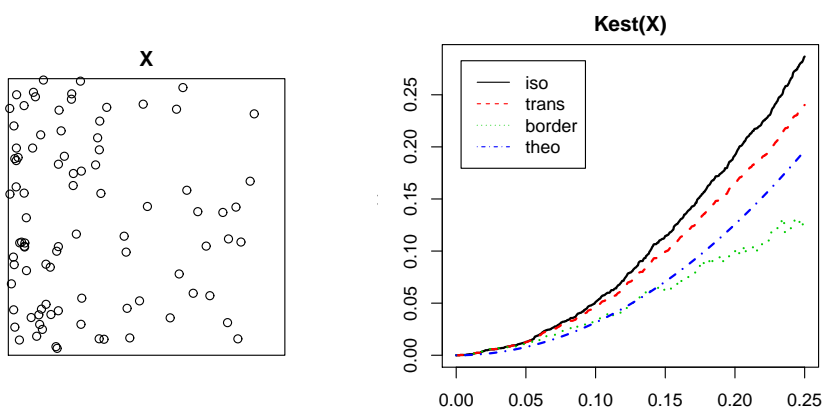
For an example of caveat 2, here is a point process constructed by Baddeley and Silverman [16] which has the same K function as the homogeneous Poisson process:

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



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

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



20 Simulation envelopes and goodness-of-fit tests

Although summary statistics such as the K -function are intended primarily for exploratory purposes, it is also possible to use them as a basis for statistical inference.

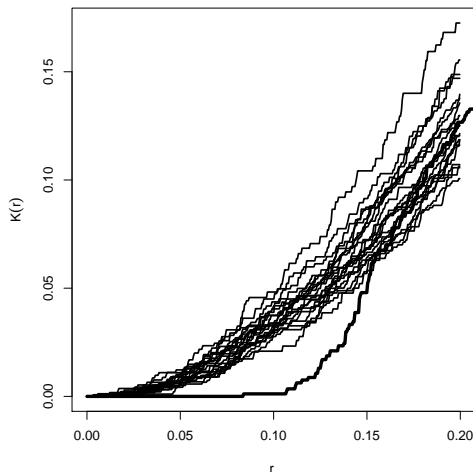
20.1 Envelopes and Monte Carlo tests

20.1.1 Motivation

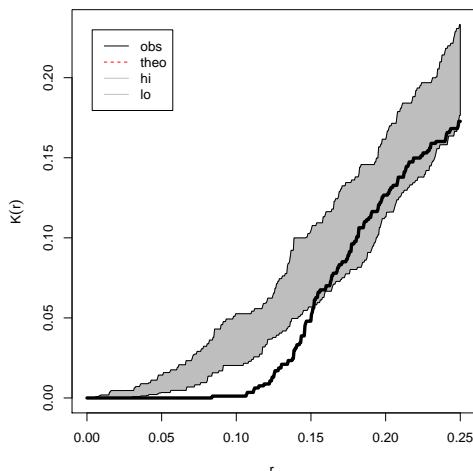
In Section 19 we examined plots of the K -function to judge whether a point pattern dataset is completely random. The K -function estimated from the point pattern data, $\hat{K}(r)$, was compared graphically with the theoretical K -function for a completely random pattern, $K_{\text{pois}}(r) = \pi r^2$. In the toy examples, large discrepancies between \hat{K} and K_{pois} were observed, indicating that the toy examples were not completely random patterns.

However, because of random variability, we will never obtain perfect agreement between \hat{K} and K_{pois} , even with a completely random pattern. Try typing `plot(Kest(rpoispp(50)))` a few times to get an idea of the inherent variability.

The following plot shows the K -function estimated from the `cells` dataset (thick line), and also the K -functions of 20 simulated realisations of CSR with the same intensity (thin lines).



The next plot shows the upper and lower *envelopes* of the simulated K -functions, that is, the maximum and minimum values of $\hat{K}(r)$ for each value of r . The region between the envelopes is shaded.



Clearly, the K -function estimated from the `cells` data lies outside the typical range of values of the K -function for a completely random pattern.

To conclude formally that there is a ‘significant’ difference between \widehat{K} and K_{pois} , we use the language of hypothesis testing. Our *null hypothesis* is that the data point pattern is a realisation of complete spatial randomness. The *alternative hypothesis* is that the data pattern is a realisation of another, unspecified point process.

20.1.2 Monte Carlo tests

A *Monte Carlo test* is a test based on simulations from the null hypothesis. The principle was originated independently by Barnard [18] and Dwass [39]. It was applied in spatial statistics by Ripley [54, 56] and Besag [22, 23]. See also [41]. Monte Carlo tests are a special case of *randomisation tests* which are commonly used in nonparametric statistics.

Suppose the reference curve is the theoretical K function for CSR. Generate M independent simulations of CSR inside the study region W . Compute the estimated K functions for each of these realisations, say $\widehat{K}^{(j)}(r)$ for $j = 1, \dots, M$. Obtain the pointwise upper and lower envelopes of these simulated curves,

$$L(r) = \min_j \widehat{K}^{(j)}(r)$$

$$U(r) = \max_j \widehat{K}^{(j)}(r).$$

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

20.1.3 Envelopes in spatstat

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

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

> E
```

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


obs	obs(r)	observed value of $K(r)$ for data pattern
theo	theo(r)	theoretical value of $K(r)$ for CSR
lo	lo(r)	lower pointwise envelope of $K(r)$ from simulations
hi	hi(r)	upper pointwise envelope of $K(r)$ from simulations

Default plot formula:

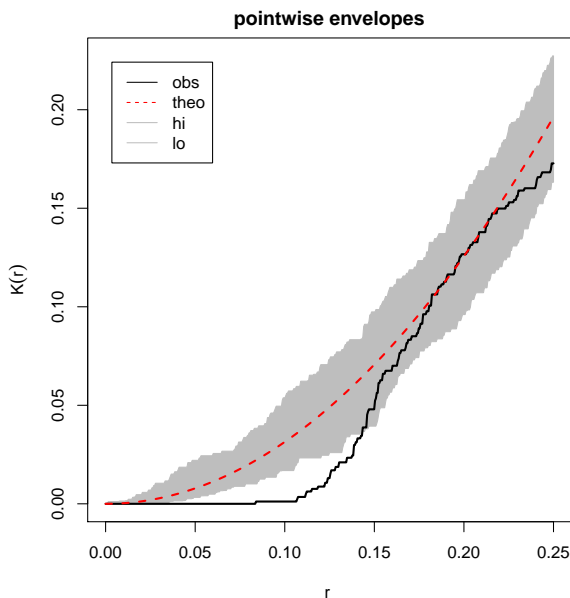
. ~ r

<environment: 0x5d76db8>

Recommended range of argument r: [0, 0.25]

Available range of argument r: [0, 0.25]

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



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

Tip: A common and dangerous mistake is to misinterpret the simulation envelopes as “confidence intervals” around \hat{K} . They cannot be interpreted as a measure of accuracy of the estimated K function! They are the critical values for a test of the hypothesis that $K(r) = \pi r^2$. **They assume that the pattern is completely random.** [See Section 21 for ways of making confidence intervals for $K(r)$.]

The value returned by `envelope` is an object of class "fv" that can be manipulated in the usual way: you can plot it, transform it, extract columns, and so on (see Section 19.6 on page 128).

20.1.4 Simultaneous Monte Carlo test

Note that the theory of the Monte Carlo test, as presented above, requires that r be fixed in advance. If we plot the envelope and check whether the empirical K function ever wanders

outside the envelope, this is equivalent to choosing the value of r in a data-dependent way, and the true significance level is higher (less ‘significant’).

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

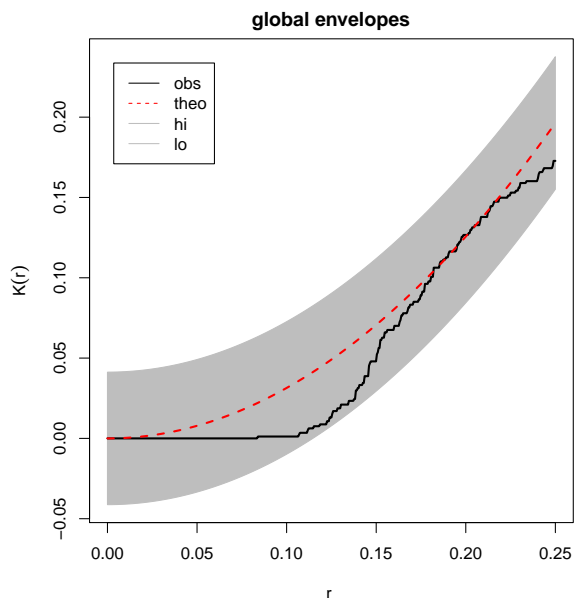
One simple way to achieve this is to compute, for each estimate $\widehat{K}(r)$, its maximum deviation from the Poisson K function, $D = \max_r |\widehat{K}(r) - K_{\text{pois}}(r)|$. This is computed for each of the M simulated datasets, and the maximum value D_{max} obtained. Then the upper and lower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

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

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

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

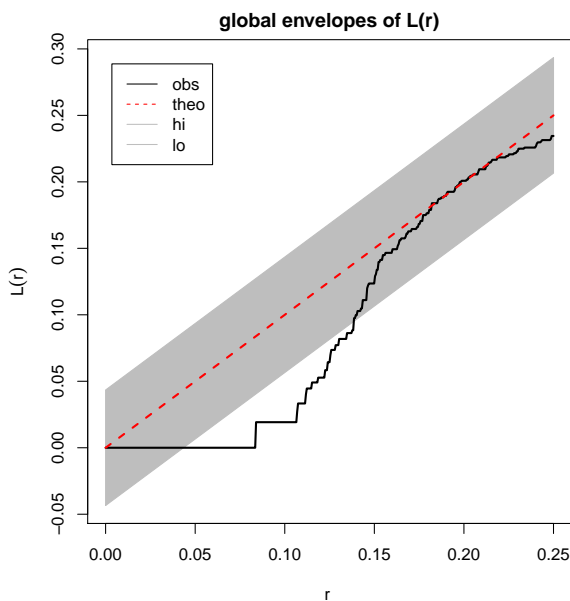
```
> plot(E, main = "global envelopes")
```



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

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

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



20.1.5 Envelopes for any fitted model

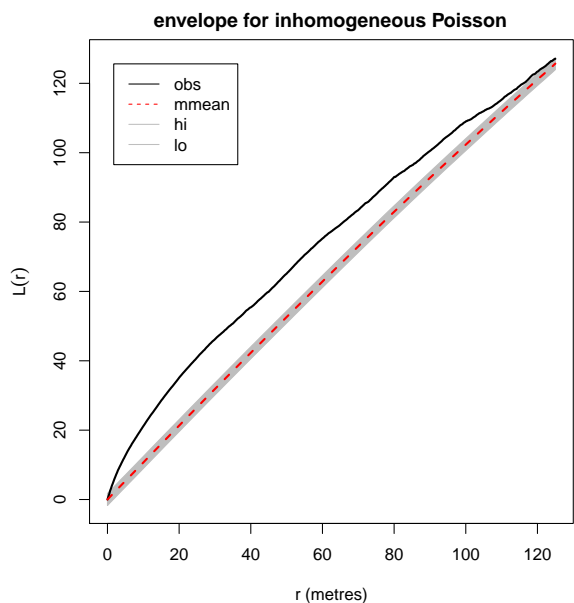
In the explanation above, we assumed that the null hypothesis was CSR (complete spatial randomness, a uniform Poisson process). In fact the Monte Carlo testing rationale can be applied to any point process model serving as a null hypothesis. We simply have to generate simulated realisations from the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

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

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

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

> plot(E, main = "envelope for inhomogeneous Poisson")
```



20.1.6 Envelopes based on any simulation procedure

Envelopes can also be computed using any user-specified procedure to generate the simulated realisations. This allows us to perform randomisation tests, for example.

The simulation procedure should be encoded as an R expression, which will be evaluated each time a simulation is required. For example if we type

```
> sim <- expression(rpoispp(100))
```

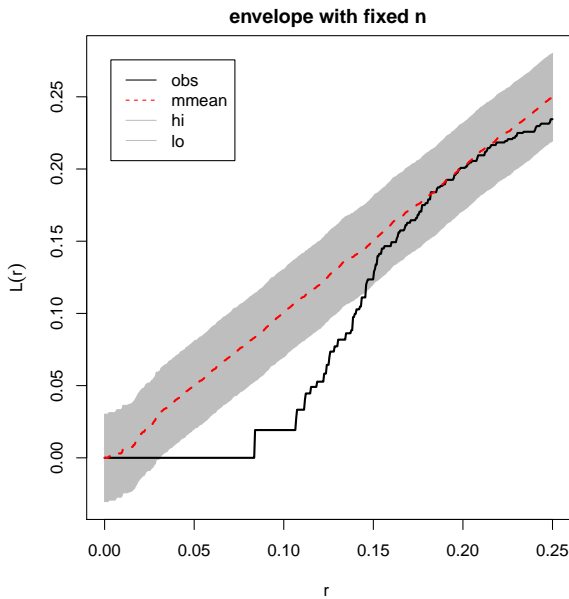
then each time the expression `sim` is evaluated, it will yield a different random outcome of the Poisson process with intensity 100 in the unit square.

This expression should be passed to the `envelope` function as the argument `simulate`.

The following code generates simulation envelopes for the L function based on simulations of CSR which have the same number of points as the data pattern.

```
> data(cells)
> n <- npoints(cells)
> W <- as.owin(cells)
> e <- expression(runifpoint(n, W))
> E <- envelope(cells, Lest, nsim = 19, global = TRUE, simulate = e)
```

```
> plot(E, main = "envelope with fixed n")
```



20.1.7 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:

```
> data(cells)
> 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.

```
> data(cells)
> 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.

20.1.8 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.

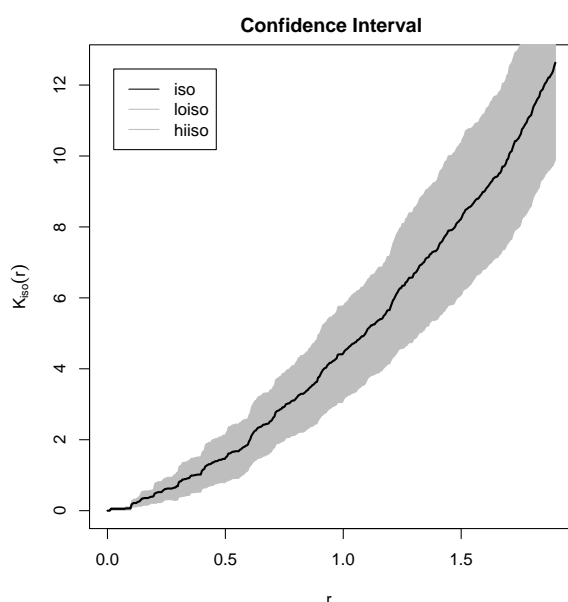
```
> envelope(cells, Kest, nsim = 100, VARIANCE = TRUE)
```

21 Spatial bootstrap methods

`Spatstat` includes some elementary bootstrap methods for estimating the variance and the distribution of a summary statistic. These can be used to construct **confidence intervals** for the true value of $K(r)$, for example.

The function `varblock` divides the point pattern's window into several quadrats. The specified summary statistic is applied to each of the corresponding sub-patterns. Then the pointwise sample mean, sample variance and sample standard deviation of these summary statistics are computed. The two-standard-deviation confidence intervals are computed. This is an elementary bootstrap estimate of the sampling variance of the summary statistic [35, eq. (4.21), p. 52].

```
> data(finpines)
> Kci <- varblock(finpines, Kest, nx = 3, ny = 3)
> plot(Kci, iso ~ r, shade = c("loiso", "hiiso"), main = "Confidence Interval")
```



The function `quadratresample` generates a randomly resampled version of the data point pattern. The spatial domain is divided into several rectangular quadrats of equal shape and size; the sub-patterns in each quadrat are extracted into a list; and the list is randomly permuted or resampled. The resampled sub-patterns are replaced in the original domains.

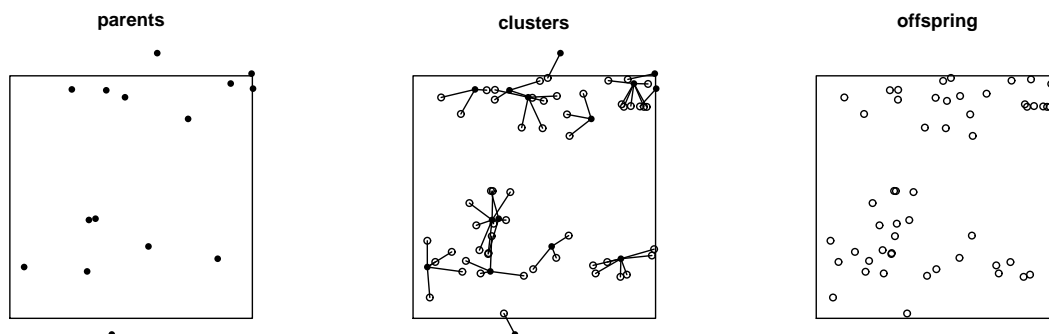
```
> X <- unmark(finpines)
> Xlist <- quadratresample(X, nx = 4, ny = 4, nsamples = 100)
> E <- envelope(X, Kest, simulate = Xlist, nsim = 100, VARIANCE = TRUE)
> plot(E)
```

22 Simple models of non-Poisson patterns

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

22.1 Poisson cluster processes

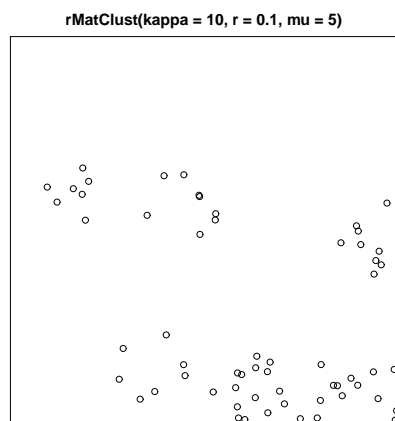
In a Poisson cluster process, we begin with a Poisson process \mathbf{Y} of ‘parent’ points. Each ‘parent’ point $y_i \in \mathbf{Y}$ then gives rise to a finite set Z_i of ‘offspring’ points according to some stochastic mechanism. The set comprising all the offspring points forms a point process \mathbf{X}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



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

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

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



Other Poisson cluster processes implemented in `spatstat` are

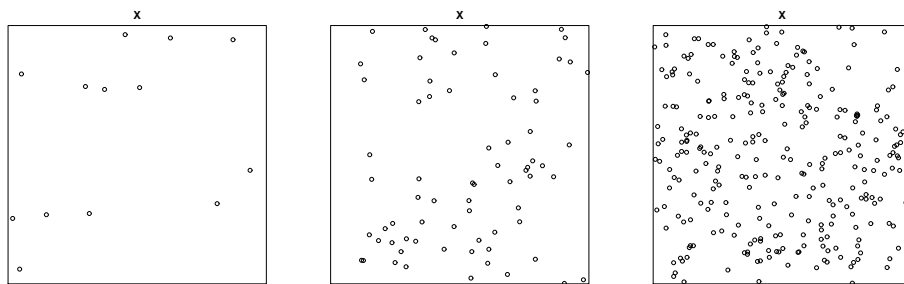
- `rThomas`: the *Thomas process*, in which each cluster consists of a Poisson(μ) number of random points, each having an isotropic Gaussian $N(0, \sigma^2 I)$ displacement from its parent.
- `rGaussPoisson`: the *Gauss-Poisson process* in which each cluster is either a single point or a pair of points.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

22.2 Cox processes

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

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

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



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

A Cox model is the analogue of a ‘random effects’ model. It is always overdispersed relative to a Poisson process (i.e. the variance of the number of points falling in a region, is greater than the mean). Cox processes are the most convenient models for clustered point patterns. A particularly interesting and useful class is that of *log-Gaussian Cox processes (LGCP)* in which $\log \Lambda(u)$ is a Gaussian random function [51, 50].

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

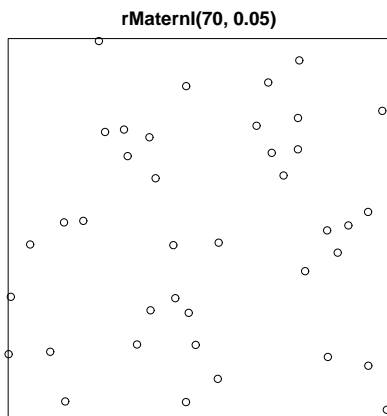
Currently there are no functions in `spatstat` for generating the general Cox process, but if you have a way of generating realisations of a random function Λ of interest, then you can use `rpoispp` to generate the Cox process. The intensity argument `lambda` of `rpoispp` can be a `function(x,y)` or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

22.3 Thinned processes

‘*Thinning*’ means deleting some of the points from a point pattern. Under ‘*independent thinning*’ the fate of each point is independent of other points. When independent thinning is applied to a Poisson process, the resulting process of retained points is Poisson. To get a non-Poisson process we need some kind of *dependent thinning* mechanism.

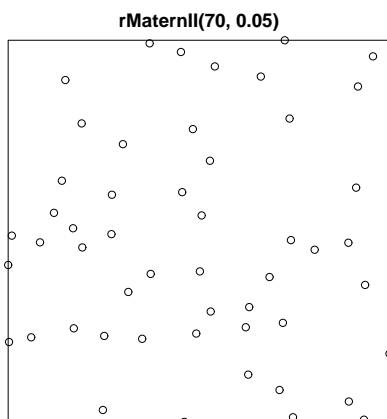
In *Matérn’s Model I*, a homogeneous Poisson process \mathbf{Y} is first generated. Any point in \mathbf{Y} that lies closer than a distance r from the nearest other point of \mathbf{Y} , is deleted. Thus, pairs of close neighbours annihilate each other.

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

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

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

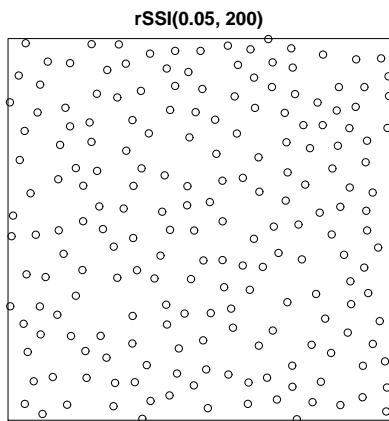


22.4 Sequential models

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

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

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



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

23 Model-fitting using summary statistics

Summary statistics can also be used to fit models to data.

In the ‘method of moments’ we estimate a parameter θ by solving

$$\mathbb{E}_\theta[S(\mathbf{X})] = S(\mathbf{x})$$

where $S(\mathbf{x})$ is the observed value of a statistic S for our data \mathbf{x} , and the left side is the theoretical mean of S for the model governed by parameter θ .

The analogue for point process models is to fit the model by matching a summary statistic such as the K function to its theoretical value under the model.

23.1 Fitting a cluster process

In a precious few cases, the K function of a point process is known exactly, as an analytic expression in terms of the model parameters. These happy cases include many Neyman-Scott cluster processes. For example, the K -function of the Thomas process (Section 22.1, page 140) with parameters $\theta = (\kappa, \mu, \sigma)$ is

$$K_\theta(r) = \pi r^2 + \frac{1}{\kappa} \left(1 - \exp\left(-\frac{r^2}{4\sigma^2}\right)\right). \quad (26)$$

We can use this to fit a Thomas model to data. We determine the values of the parameters $\theta = (\kappa, \mu, \sigma)$ to achieve the best match between $K_\theta(r)$ and the estimated K -function of the data, $\widehat{K}(r)$. The best match is determined by minimising the discrepancy between the two functions over some range $[a, b]$:

$$D(\theta) = \int_a^b \left| \widehat{K}(r)^q - K_\theta(r)^q \right|^p dr \quad (27)$$

where $0 \leq a < b$, and where $p, q > 0$ are indices. This method was originally advocated by Peter Diggle and collaborators, and is now known as the *method of minimum contrast*. See [35].

The command `kppm` fits cluster point process models by the method of minimum contrast. To fit the Thomas model to the redwood data:

```
> data(redwood)
> fit <- kppm(redwood, ~1, "Thomas")
```

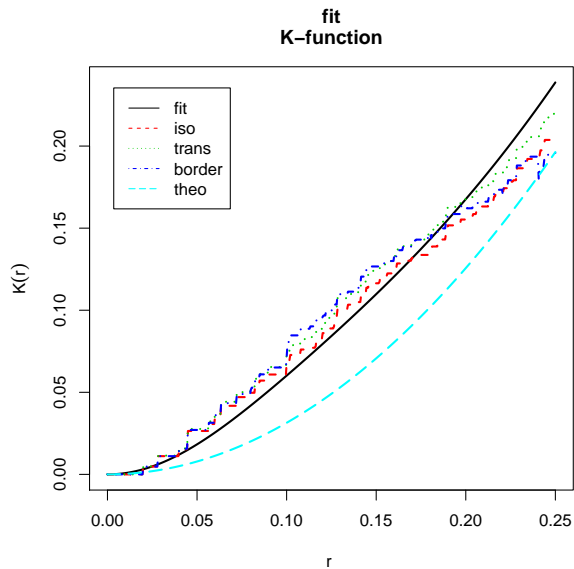
The first argument to `kppm` is a point pattern dataset. The second argument is a formula (with no left hand side) describing the log intensity of the model; the formula `~1` indicates a stationary process (see section 25.3 for nonstationary models). The third argument is the name of the cluster mechanism; currently the only options are "Thomas" and "MatClust".

The fitted model, `fit`, is an object of class `kppm`. There are methods for printing and plotting objects of this class.

```
> fit
```

```
Stationary cluster point process model
Fitted to point pattern dataset redwood
Fitted using the K-function
Cluster model: Thomas process
Fitted parameters:
      kappa      sigma      mu
23.55389789  0.04704965  2.63226071
```

```
> plot(fit)
```



The plot shows the theoretical K function of the fitted Thomas process (`fit`), three non-parametric estimates of the K function (`iso`, `trans`, `border`) and the Poisson K function (`theo`).

At present, the only cluster process models that can be fitted using `kppm` are the Thomas process and the Matérn cluster process. To fit the Matérn cluster process to the redwood data,

```
> fitM <- kppm(redwood, ~1, "MatClust")
```

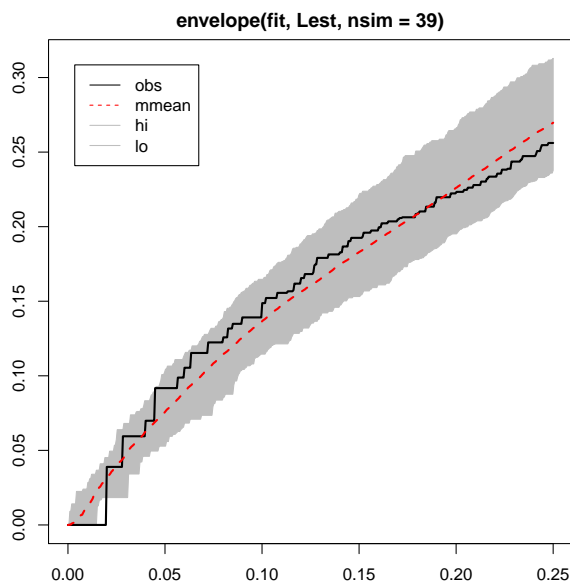
A fitted model returned by `kppm` can be simulated immediately:

```
> plot(simulate(fit, nsim = 4))
```

The command `simulate` is generic; here we have used the method `simulate.kppm`.

Simulation envelopes are also available, using the method `envelope.kppm`.

```
> plot(envelope(fit, Lest, nsim = 39))
```



23.2 Fitting cluster models using the pair correlation

Minimum contrast estimation can be applied to any summary statistic. In particular we can use the pair correlation function instead of the K -function.

```
> fitp <- kppm(redwood, ~1, "Thomas", statistic = "pcf")
> fitp
```

```
Stationary cluster point process model
Fitted to point pattern dataset redwood
Fitted using the pair correlation function
Cluster model: Thomas process
Fitted parameters:
      kappa      sigma      mu
25.30537171  0.03968295  2.45007268
```

23.3 Fitting other models with known K function

Apart from cluster processes, there are certain other point process models for which the K -function is known as a function of the model parameters. Minimum contrast methods are also available for these models.

One special case is the *log-Gaussian Cox processes* described in detail in [51]. To fit a log-Gaussian Cox process with exponential covariance function to the redwood data:

```
> fit <- lgcp.estK(redwood, c(sigma2 = 0.1, alpha = 1))
> fit
```

```
Minimum contrast fit (object of class minconfit)
Model: log-Gaussian Cox process
Fitted by matching theoretical K function to Kest(redwood)
Parameters fitted by minimum contrast ($par):
      sigma2      alpha
```

```

1.0485493 0.0997963
Derived parameters of log-Gaussian Cox process ($modelpar):
  sigma2      alpha      mu
1.0485493 0.0997963 3.6028597
Converged successfully after 145 iterations.
Domain of integration: [ 0 , 0.25 ]
Exponents: p= 2, q= 0.25

```

The second argument to `lgcp.estK` gives initial values for the model parameters σ^2 and α .

The result of `lgcp.estK` is an object of class `minconfit` (representing a ‘minimum contrast fit’). There are methods for printing and plotting the fit. Simulation of these models has not yet been implemented in `spatstat`.

Estimation can also be based on the pair correlation function:

```

> fit <- lgcp.estpcf(redwood, c(sigma2 = 0.1, alpha = 1))
> fit

```

```

Minimum contrast fit (object of class minconfit)
Model: log-Gaussian Cox process
Fitted by matching theoretical g function to pcf(redwood)
Parameters fitted by minimum contrast ($par):
  sigma2      alpha
1.30244010 0.07011464
Derived parameters of log-Gaussian Cox process ($modelpar):
  sigma2      alpha      mu
1.30244010 0.07011464 3.47591433
Converged successfully after 83 iterations.
Domain of integration: [ 0.0004883 , 0.25 ]
Exponents: p= 2, q= 0.25

```

23.4 Generic algorithm for minimum contrast

The command `mincontrast` is a generic fitting algorithm for the method of minimum contrast. It can be used in any context where the theoretical function can be computed exactly from the model parameters. A basic call to `mincontrast` is:

```

> mincontrast(observed, theoretical, starpar)

```

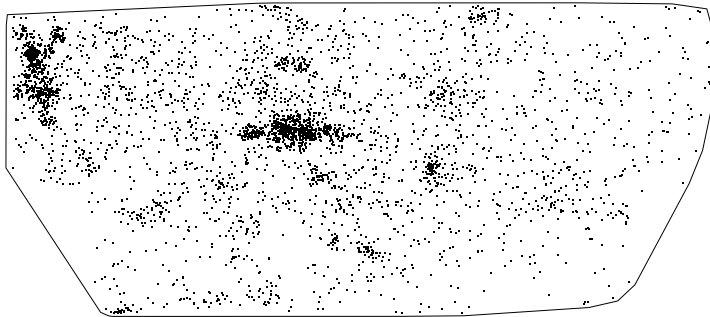
where `observed` is an object of class `"fv"` containing the summary function calculated from the data; `theoretical` is a function which returns the theoretical value of the summary function for a given parameter value; and `startpar` is a vector of initial values of the model parameters. For details, see the help file for `mincontrast`.

For the vast majority of point process models, the true K function $K_\theta(r)$ is not known analytically in terms of the parameter θ . In principle we could use Monte Carlo simulation to determine an approximation to $K_\theta(r)$, for any given θ , by generating a large number of simulated realisations of the process with parameter θ , computing the estimated K -function for each realisation, and taking the pointwise sample average. It’s possible to do this in `spatstat` using the generic algorithm `mincontrast`. Details are not given here as it is rather fiddly at present, and will change soon.

24 Exploring local features

The `shapley` dataset 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.

```
> data(shapley)
> X <- unmark(shapley)
> plot(X, pch = ".", main = "")
```



Exploratory techniques for investigating localised features in a point pattern include LISA (Local Indicators of Spatial Association), nearest-neighbour cleaning, and data sharpening.

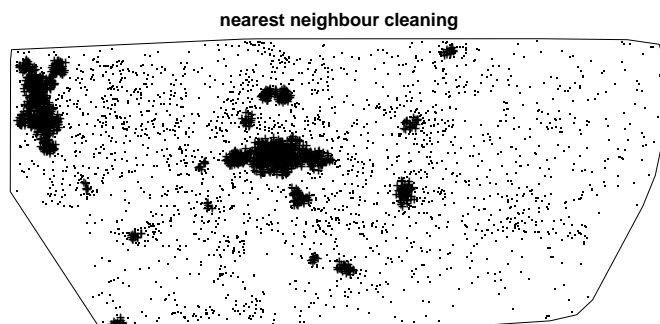
In *LISA methods*, a summary statistic is separated into contributions from each of the data points. For example the K function is expressed as a sum of the *local K functions* of each of the data points. These local functions are then compared, and classified into several groups of functions, perhaps using principal component analysis [2, 29, 28].

The `spatstat` functions `localK`, `localL`, `localpcf` compute local versions of the K -function, L -function and pair correlation function, respectively.

Nearest-neighbour cleaning [26] groups the points into two classes — ‘feature’ and ‘noise’ — on the basis of their nearest-neighbour distances. It is quick and often very useful.

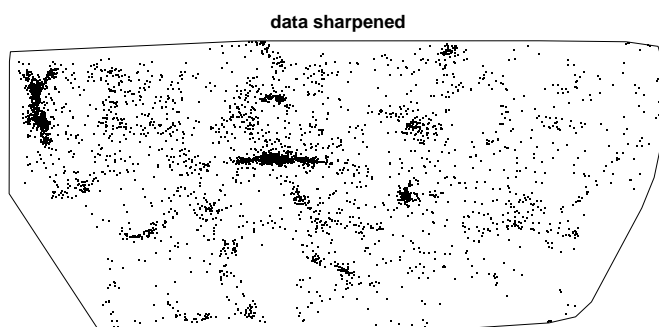
```
> Z <- nnclean(X, k = 17)
> plot(Z, chars = c(".", "+"), main = "nearest neighbour cleaning")
```

```
noise feature
"."      "+"
```



In *data sharpening* [27] 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.

```
> Y <- sharpen(X, sigma = 0.5, edgcorrect = TRUE)
> plot(Y, pch = ".", main = "data sharpened")
```



25 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.

25.1 Inhomogeneous K function

There is a modification of the K function that applies to inhomogeneous processes [7]. If $\lambda(u)$ is the true intensity function of the point process \mathbf{X} , then the idea is that each point x_i will be weighted by $w_i = 1/\lambda(x_i)$.

The *inhomogeneous K -function* is defined as

$$K_{\text{inhom}}(r) = \mathbb{E} \left[\sum_{x_j \in \mathbf{X}} \frac{1}{\lambda(x_j)} \mathbf{1}\{0 < \|u - x_j\| \leq r\} \mid u \in \mathbf{X} \right] \quad (28)$$

assuming that this does not depend on location u . Thus, $\lambda(u)K(r)$ is the expected total ‘weight’ of all random points within a distance r of the point u , where the ‘weight’ of a point x_i is $1/\lambda(x_i)$.

If the process is actually homogeneous, then $\lambda(u)$ is constant and $K_{\text{inhom}}(r)$ reduces to the usual K function (21).

It turns out that, for an inhomogeneous Poisson process with intensity function $\lambda(u)$, the inhomogeneous K function is

$$K_{\text{inhom, pois}}(r) = \pi r^2 \quad (29)$$

exactly as for the homogeneous case.

The standard estimators of K can be extended to the inhomogeneous K function:

$$\hat{K}_{\text{inhom}}(r) = \frac{1}{D} \sum_i \sum_{j \neq i} \frac{\mathbf{1}\{\|x_i - x_j\| \leq r\}}{\hat{\lambda}(x_i)\hat{\lambda}(x_j)} e(x_i, x_j; r) \quad (30)$$

where $e(u, v, r)$ is an edge correction weight as before, and $\hat{\lambda}(u)$ is an estimate of the intensity function $\lambda(u)$.

The denominator D in (30) is either the area of the window $D_1 = \text{area}(W)$, or

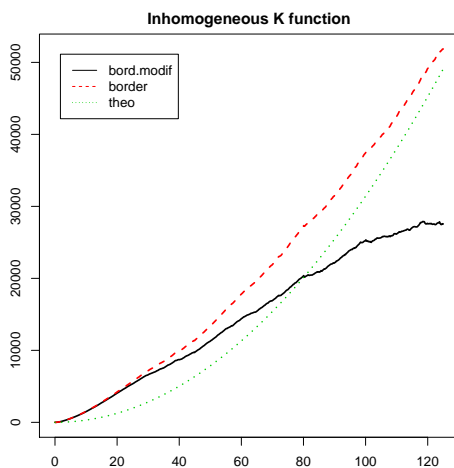
$$D_2 = \sum_i \frac{1}{\hat{\lambda}(x_i)}$$

which is an unbiased estimator of $\text{area}(W)$ if the intensity is correctly estimated. The denominator D_2 is often preferred on grounds of statistical performance, because it introduces a data-dependent normalisation.

The inhomogeneous K function is computed 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 numeric vector giving the values $\hat{\lambda}(x_i)$ 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_2 is used.

There remains the question of how to estimate the intensity function $\lambda(u)$. It is usually advisable to obtain the intensity estimate $\hat{\lambda}(u)$ by fitting a parametric model, to avoid overfitting. Here is an example for the tropical rainforest data, using the covariate data to suggest a model for the intensity.

```
> data(bei)
> fit <- ppm(bei, ~elev + grad, covariates = bei.extra)
> lam <- predict(fit, locations = bei)
> Ki <- Kinhom(bei, lam)
> plot(Ki, main = "Inhomogeneous K function")
```



The plot suggests that, even after accounting for dependence on altitude and slope, the trees still appear to be clustered.

The intensity function $\lambda(u)$ could also be estimated by kernel smoothing the point pattern data. However, notice that the estimator (30) of the inhomogeneous K function depends on the estimated intensity values at the *data points*, $\hat{\lambda}(x_i)$. These are positively biased estimates of the true values $\lambda(x_i)$. In order to avoid bias, the value $\hat{\lambda}(x_i)$ should be estimated by kernel smoothing of the point pattern with the point x_i *deleted*. This “leave-one-out” estimator is implemented in `Kinhom` and is invoked when the argument `lambda` is not given:

```
> Ki2 <- Kinhom(bei)
> plot(Ki2, main = "Kinhom using leave-one-out")
```

	lty	col	key	label	
bord.modif	1	1	bord.modif	K[bordm](r)	
border	2	2	border	K[bord](r)	
theo	3	3	theo	K[pois](r)	
					meaning
bord.modif	modified		border-corrected	estimate of K(r)	
border			border-corrected	estimate of K(r)	
theo			theoretical	Poisson K(r)	

(the smoothing parameter σ can also be controlled.)

The inhomogeneous analogue of the L -function is defined by

$$\widehat{L}_{\text{inhom}}(r) = \sqrt{\frac{1}{\pi} \widehat{K}_{\text{inhom}}(r)}$$

This can be computed using `Linhom`. For an inhomogeneous Poisson process, $L_{\text{inhom}}(r) \equiv r$.

The inhomogeneous analogue of the pair correlation function can be defined, similarly to the homogeneous case, as

$$g_{\text{inhom}}(r) = \frac{K'_{\text{inhom}}(r)}{2\pi r}.$$

It has the same interpretation, namely, that $g_{\text{inhom}}(r)$ is the probability of observing a pair of points at certain locations separated by a distance r , divided by the corresponding probability for a Poisson process of the same (inhomogeneous) intensity.

The inhomogeneous pair correlation function is computed by `pcfinhom`:

```
> g <- pcfinhom(bei)
```

Incidentally there is also a function `pcf.fv` which will convert any K -function into a pair correlation function by numerical differentiation. Thus the following is an alternative:

```
> g <- pcf(Kinhom(bei))
```

25.2 Inhomogeneous cluster models

The inhomogeneous Poisson process was described in Section 15.1. We can also introduce spatial inhomogeneity into any of the non-Poisson models described in Section 22.

In the case of Poisson cluster processes (Section 22.1) we can introduce inhomogeneity in either the parent process or the offspring processes.

To make the *parents* inhomogeneous, we simply generate the parent points from an inhomogeneous Poisson process with some intensity function $\kappa(u)$.

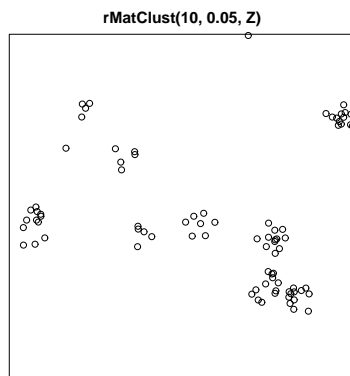
To make the *clusters* inhomogeneous, we use a clever construction by Waagepetersen [65]. For a parent point at location (x_0, y_0) , the offspring are generated from a Poisson process with intensity $\beta(x, y) = \mu(x, y)f(x - x_0, y - y_0)$, where $f(u, v)$ is either the Gaussian probability density (for the Thomas process) or the uniform probability density in a disc (for the Matérn cluster process), and $\mu(x, y)$ is the *reference* or *modulating* intensity. The number of offspring from a given parent (x_0, y_0) is a Poisson random variable with mean

$$B(x_0, y_0) = \int \beta(x, y) dx dy = \int f(x - x_0, y - y_0) \mu(x, y) dx dy.$$

The simulation algorithms `rMatClust` and `rThomas` allow both these options. If the parent intensity parameter `kappa` is given as a `function(x,y)` or a pixel image, then the parents are

Poisson with inhomogeneous intensity κ . If the offspring mean parameter μ is given as a `function(x,y)` or a pixel image, then this determines an inhomogeneous reference density for the clusters.

```
> Z <- as.im(function(x, y) {
+   6 * exp(2 * x - 1)
+ }, owin())
> plot(rMatClust(10, 0.05, Z))
```



25.3 Fitting inhomogeneous models by minimum contrast

Minimum contrast methods can be applied to inhomogeneous point process models.

In principle we could fit any model (homogeneous or inhomogeneous) by the method of minimum contrast using any summary statistic. However, the method works best when we have an exact formula for the true value of the summary function for the model, expressed as a function of the parameters of the model.

Waagepetersen [65] pointed out that, if we take a Thomas process or Matérn cluster process (or in general a Neyman-Scott process) with **homogeneous** parent intensity κ and **inhomogeneous** cluster reference density $\mu(u)$, then the overall intensity of the process is

$$\lambda(u) = \kappa \mu(u)$$

and the *inhomogeneous* K -function is the same as it would be if μ were constant.

Thus, we can fit a Thomas process or Matérn cluster process with inhomogeneous clusters as follows:

1. estimate the inhomogeneous intensity $\lambda(u)$ of the process.
2. derive an estimate of the inhomogeneous K -function.
3. use the method of minimum contrast to estimate the parent intensity κ and the cluster scale parameter (Gaussian standard deviation or disc radius), exactly as we would in the homogeneous case.

The command `kppm` performs this algorithm using a parametric model for the trend:

```
> data(bei)
> fit <- kppm(bei, ~elev + grad, "Thomas", covariates = bei.extra)
> fit
```

```
Inhomogeneous cluster point process model
Fitted to point pattern dataset bei
Fitted using the inhomogeneous K-function
Trend formula:~elev + grad
```

```
Fitted coefficients for trend formula:
```

```
(Intercept)      elev      grad
```

```
-8.55862210  0.02140987  5.84104065
```

```
Cluster model: Thomas process
```

```
Fitted parameters:
```

```
      kappa      sigma
```

```
0.0004290453 5.4110425537
```

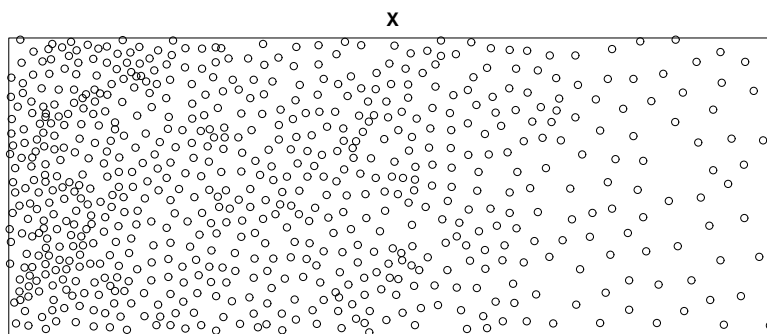
In this example, `kppm` first estimates the intensity by fitting the model `ppm(bei, ~elev+grad, covariates = bei.extra)`. Then `predict.ppm` is used to compute the predicted intensity at the data points, and this is passed to `Kinhom` to calculate the inhomogeneous K function. The parameters of the Thomas process are estimated from the inhomogeneous K function by minimum contrast.

The result of `kppm` can be printed, plotted, simulated and “enveloped” as before.

25.4 Local scaling

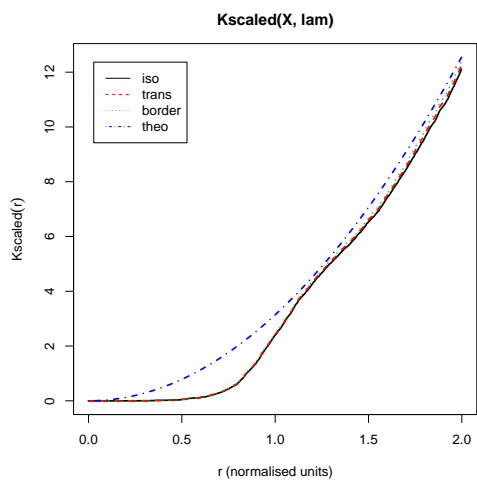
Locally-scaled point processes and summary functions [53] provide an alternative to the concept of locally-weighted K functions (Section 25.1). In essence, the point process is assumed to be equivalent, in small regions, to a rescaled version of a ‘template’ process, where the template process is stationary, and the rescaling factor can vary from place to place.

```
> data(bronzefilter)
> X <- unmark(bronzefilter)
> plot(X)
```



`Spatstat` provides the commands `Kscaled` and `Lscaled` which compute the locally-scaled K and L functions. Their syntax is similar to `Kinhom`.

```
> fit <- ppm(X, ~x)
> lam <- predict(fit)
> plot(Kscaled(X, lam), xlim = c(0, 2))
```



PART VI. GIBBS MODELS

Part VI of the workshop explains Gibbs point process models and how to apply them.

26 Gibbs models

One way to construct a statistical model (in any field of statistics) is to write down its probability density. Advantages of doing this are:

- the functional form of the density reflects its probabilistic properties.
- terms or factors in the density often have an interpretation as ‘components’ of the model.
- it is easy to introduce terms that represent the dependence of the model on covariates, etc.

This approach is useful provided the density *can* be written down, and provided the density is tractable.

Spatial point process models that are constructed by writing down their probability densities are called ‘**Gibbs processes**’. Good references on Gibbs point processes are [63, 31].

26.1 Probability densities

It is possible to define probability densities for spatial point processes that live inside a bounded window W .

The probability density will be a function $f(\mathbf{x})$ defined for each finite configuration $\mathbf{x} = \{x_1, \dots, x_n\}$ of points $x_i \in W$ for any $n \geq 0$. Notice that the number of points n is not fixed, and may be zero. Apart from this peculiarity, probability densities for point processes behave much like probability densities in more familiar contexts.

That’s all you need to know for applications. **If you’re interested in the mathematical technicalities, read on; otherwise, skip to section 26.2.**

A point process \mathbf{X} inside W is defined to have probability density f if and only if, for any nonnegative integrable function h ,

$$\mathbb{E}[h(\mathbf{X})] = e^{-|W|}h(\emptyset)f(\emptyset) + e^{-|W|} \sum_{n=1}^{\infty} \frac{1}{n!} \int_W \cdots \int_W h(\{x_1, \dots, x_n\})f(\{x_1, \dots, x_n\}) dx_1 \cdots dx_n \quad (31)$$

where $|W|$ denotes the area of W .

In particular, the probability that \mathbf{X} contains exactly n points is

$$p_n = \mathbb{P}\{n(\mathbf{X}) = n\} = \frac{e^{-|W|}}{n!} \int_W \cdots \int_W f(\{x_1, \dots, x_n\}) dx_1 \cdots dx_n$$

for $n \geq 1$ and $p_0 = \mathbb{P}\{n(\mathbf{X}) = 0\} = e^{-|W|}f(\emptyset)$. Given that there are exactly n points, the conditional joint density of the locations x_1, \dots, x_n is $f(\{x_1, \dots, x_n\})/p_n$.

26.2 Poisson processes

The uniform Poisson process with intensity 1 has probability density $f(\mathbf{x}) \equiv 1$.

The uniform Poisson process in W with intensity λ has probability density

$$f(\mathbf{x}) = \alpha \lambda^{n(\mathbf{x})} \quad (32)$$

where $n(\mathbf{x})$ is the number of points in the configuration \mathbf{x} , and the constant α is

$$\alpha = e^{(1-\lambda)|W|}.$$

The inhomogeneous Poisson process in W with intensity function $\lambda(u)$ has probability density

$$f(\mathbf{x}) = \alpha \prod_{i=1}^n \lambda(x_i). \quad (33)$$

where the constant α is

$$\alpha = \exp \left[\int_W (1 - \lambda(u)) \, du \right].$$

The densities (32) and (33) are products of terms associated with individual points x_i . This reflects the conditional independence property (PP4) of the Poisson process.

26.3 Pairwise interaction models

In order to construct spatial point processes which exhibit interpoint interaction (stochastic dependence between points), we need to introduce terms in the density that depend on more than one point. The simplest are *pairwise interaction models*, which have probability densities of the form

$$f(\mathbf{x}) = \alpha \left[\prod_{i=1}^{n(\mathbf{x})} b(x_i) \right] \left[\prod_{i < j} c(x_i, x_j) \right] \quad (34)$$

where α is a normalising constant, $b(u)$, $u \in W$ is the ‘first order’ term, and $c(u, v)$, $u, v \in W$ is the ‘second order’ or ‘pairwise interaction’ term. The pairwise interaction term introduces dependence between points. The interaction function must be symmetric, $c(u, v) = c(v, u)$. In principle we are free to choose any functions b and c , provided the resulting density is integrable (the right side of (31) should be finite when $h \equiv 1$).

26.3.1 Hard core process

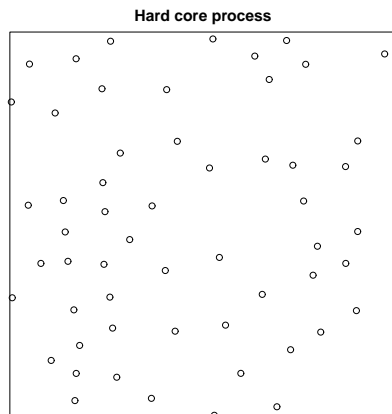
If we take $b(u) \equiv \beta$ and

$$c(u, v) = \begin{cases} 1 & \text{if } \|u - v\| > r \\ 0 & \text{if } \|u - v\| \leq r \end{cases} \quad (35)$$

where $\|u - v\|$ denotes the distance between u and v , and $r > 0$ is a fixed distance, then the density becomes

$$f(\mathbf{x}) = \begin{cases} \alpha \beta^{n(\mathbf{x})} & \text{if } \|x_i - x_j\| > r \text{ for all } i \neq j \\ 0 & \text{otherwise} \end{cases}$$

This is the density of the Poisson process of intensity β in W conditioned on the event that no two points of the pattern lie closer than r units apart. It is known as the (classical) *hard core process*.



26.3.2 Strauss process

Generalising the hard core process, suppose we take $b(u) \equiv \beta$ and

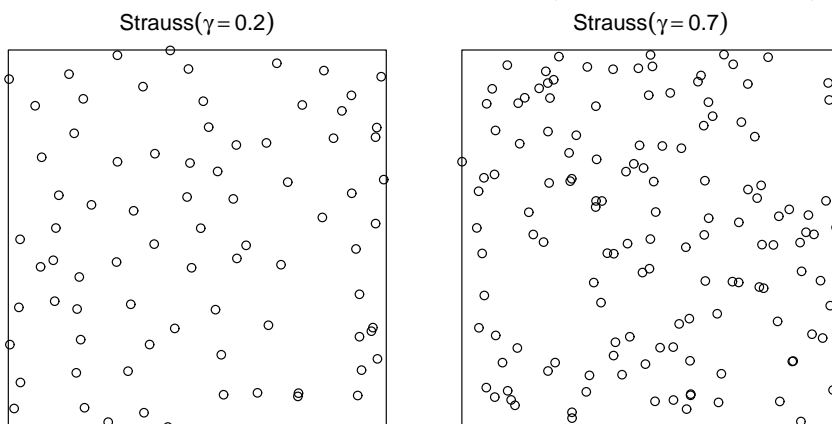
$$c(u, v) = \begin{cases} 1 & \text{if } \|u - v\| > r \\ \gamma & \text{if } \|u - v\| \leq r \end{cases} \quad (36)$$

where γ is a parameter. Then the density becomes

$$f(\mathbf{x}) = \alpha \beta^{n(\mathbf{x})} \gamma^{s(\mathbf{x})} \quad (37)$$

where $s(\mathbf{x})$ is the number of pairs of distinct points in \mathbf{x} that lie closer than r units apart.

The parameter γ controls the 'strength' of interaction between points. If $\gamma = 1$ the model reduces to a Poisson process with intensity β . If $\gamma = 0$ the model is a hard core process. For values $0 < \gamma < 1$, the process exhibits inhibition (negative association) between points.



For $\gamma > 1$, the density (37) is not integrable. Hence the Strauss process is defined only for $0 \leq \gamma \leq 1$ and is a model for inhibition between points. This is typical of most Gibbs models.

26.3.3 Other pairwise interaction models

Other pairwise interactions that are considered in `spatstat` include the *Strauss-hard core* interaction (with hard core distance $h > 0$ and interaction distance $r > h$)

$$c(u, v) = \begin{cases} 0 & \text{if } \|u - v\| \leq h \\ \gamma & \text{if } h < \|u - v\| \leq r \\ 1 & \text{if } \|u - v\| > r \end{cases} ,$$

the *soft-core* interaction (with scale $\sigma > 0$ and index $0 < \kappa < 1$)

$$c(u, v) = \left(\frac{\sigma}{\|u - v\|} \right)^{2/\kappa} ,$$

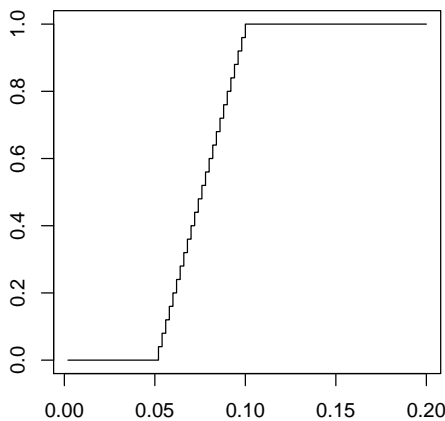
the *Diggle-Gates-Stibbard* interaction (with interaction range ρ)

$$c(u, v) = \begin{cases} \sin \left(\frac{\pi \|u - v\|}{2\rho} \right)^2 & \text{if } \|u - v\| \leq \rho \\ 1 & \text{if } \|u - v\| > \rho \end{cases} ,$$

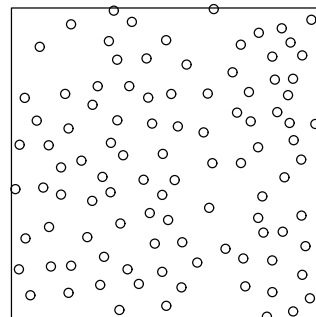
the *Diggle-Gratton* interaction (with hard core distance δ , interaction distance ρ and index κ)

$$c(u, v) = \begin{cases} 0 & \text{if } \|u - v\| \leq \delta \\ \left(\frac{\|u - v\| - \delta}{\rho - \delta} \right)^\kappa & \text{if } \delta < \|u - v\| \leq \rho \\ 1 & \text{if } \|u - v\| > \rho \end{cases} ,$$

and the general *piecewise constant* interaction in which $c(\|u - v\|)$ is a step function of $\|u - v\|$.



Piecewise constant interaction



26.4 Higher-order interactions

There are some useful Gibbs point process models which exhibit interactions of higher order, that is, in which the probability density has contributions from m -tuples of points, where $m > 2$.

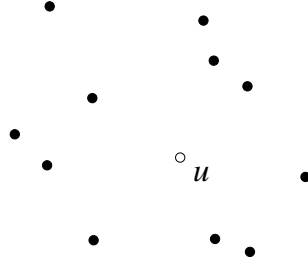
One example is the *area-interaction* or Widom-Rowlinson process [17] with probability density

$$f(\mathbf{x}) = \alpha \beta^{n(\mathbf{x})} \gamma^{-A(\mathbf{x})} \tag{38}$$

where α is the normalising constant, $\beta > 0$ is an intensity parameter, and $\gamma > 0$ is an interaction parameter. Here $A(\mathbf{x})$ denotes the area of the region obtained by drawing a disc of radius r centred at each point x_i , and taking the union of these discs. The value $\gamma = 1$ again corresponds to a Poisson process, while $\gamma < 1$ produces a regular process and $\gamma > 1$ a clustered process. This process has interactions of all orders. It can be used as a model for moderate regularity or clustering.

26.5 Conditional intensity

The main tool for analysing a Gibbs point process is its *conditional intensity* $\lambda(u, \mathbf{X})$. Intuitively this determines the conditional probability of finding a point of the process at the location u given complete information about the rest of the process. For formal definitions see [31]. Informally, the conditional probability of finding a point of the process inside an infinitesimal neighbourhood of the location u , given the complete point pattern at all other locations, is $\lambda(u, \mathbf{X}) du$.



For point processes in a bounded window, the conditional intensity at a location u given the configuration \mathbf{x} is related to the probability density f by

$$\lambda(u, \mathbf{x}) = \frac{f(\mathbf{x} \cup \{u\})}{f(\mathbf{x})} \quad (39)$$

(for $u \notin \mathbf{x}$), the ratio of the probability densities for the configuration \mathbf{x} with and without the point u added.

The homogeneous Poisson process with intensity λ has conditional intensity

$$\lambda(u, \mathbf{x}) = \lambda$$

while the inhomogeneous Poisson process with intensity function $\lambda(u)$ has conditional intensity

$$\lambda(u, \mathbf{x}) = \lambda(u)$$

. The conditional intensity for a Poisson process does not depend on the configuration \mathbf{x} , because the points of a Poisson process are independent.

For the general pairwise interaction process (34) the conditional intensity is

$$\lambda(u, \mathbf{x}) = b(u) \prod_{i=1}^{n(\mathbf{x})} c(u, x_i). \quad (40)$$

For the hard core process,

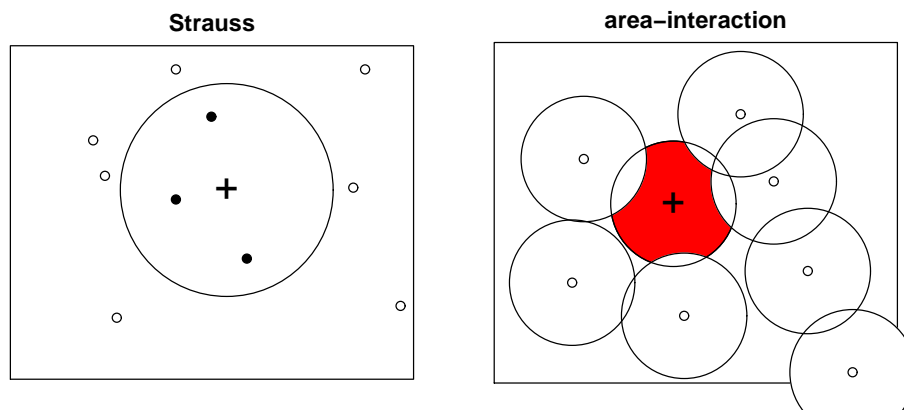
$$\lambda(u, \mathbf{x}) = \begin{cases} \beta & \text{if } \|u - x_i\| > r \text{ for all } i \\ 0 & \text{otherwise} \end{cases} \quad (41)$$

which has the nice interpretation that a point u is either ‘permitted’ or ‘not permitted’ depending on whether it satisfies the hard core requirement.

For the Strauss process

$$\lambda(u, \mathbf{x}) = \beta \gamma^{t(u, \mathbf{x})} \quad (42)$$

where $t(u, \mathbf{x}) = s(\mathbf{x} \cup \{u\}) - s(\mathbf{x})$ is the number of points of \mathbf{x} that lie within a distance r of the location u . For $\gamma < 1$, this has the interpretation that a random point is less likely to occur at the location u if there are many points in the neighbourhood.



For the area-interaction process,

$$\lambda(u, \mathbf{x}) = \beta \gamma^{-B(u, \mathbf{x})} \quad (43)$$

where $B(u, \mathbf{x}) = A(\mathbf{x} \cup \{u\}) - A(\mathbf{x})$ is the area of that part of the disc of radius r centred on u that is not covered by discs of radius r centred at the other points $x_i \in \mathbf{x}$. If the points represent trees or plants, we may imagine that each tree takes nutrients and water from the soil inside a circle of radius r . Then we may interpret $B(u, \mathbf{x})$ as the area of the ‘unclaimed zone’ where a new plant at location u would be able to draw nutrients and water without competition from other plants. For $\gamma < 1$ we can interpret (43) as saying that a random point is less likely to occur when the unclaimed area is small.

The conditional intensity of a point process determines the probability density, through (39). Hence we can use the conditional intensity to define a point process. The conditional intensity is the preferred modelling tool for Gibbs processes: it has a direct interpretation, and it is easier to handle than the probability density.

26.6 Simulating Gibbs models

Gibbs models can be simulated by Markov chain Monte Carlo algorithms. Indeed, MCMC algorithms were invented to simulate Gibbs processes [49, 55].

In brief, these algorithms simulate a Markov chain whose states are point patterns. The chain is designed so that its equilibrium distribution is the distribution of the point process we want to simulate. If the chain were run for an infinite time, the state would converge in distribution to the desired point process. In practice the chain is run for a long finite time. Further details are beyond the scope of this workshop; consult [51, 50] for more information.

Currently `spatstat` offers the function `rmh` which simulates Gibbs processes using the Metropolis-Hastings algorithm.

```
> rmh(model, start, control)
```

- `model` determines the point process model to be simulated (see `help(rmhmodel)`).
- `start` determines the initial state of the Markov chain (see `help(rmhstart)`).
- `control` specifies control parameters for running the Markov chain, such as the number of iteration steps (see `help(rmhcontrol)`).

In the simplest uses of `rmh`, the three arguments are lists of parameter values. To generate a simulated realisation of the Strauss process with parameters $\beta = 2, \gamma = 0.7, r = 0.7$ in a square of side 10,

```
> mo <- list(cif = "strauss", par = c(beta = 2, gamma = 0.2, r = 0.7),
+          w = square(10))
> X <- rmh(model = mo, start = list(n.start = 42), control = list(nrep = 1e+06))
```

The other arguments specify a random initial state of 42 points, and that the algorithm shall be run for a million iterations.

27 Fitting Gibbs models

27.1 Maximum pseudolikelihood

Maximum likelihood estimation is intractable for most point process models. At the very least it requires Monte Carlo simulation to evaluate the likelihood (or the score and the Fisher information).

A workable alternative, at least for investigative purposes, is to maximise the log *pseudolikelihood*

$$\log \text{PL}(\theta; \mathbf{x}) = \sum_i \log \lambda(x_i; \mathbf{x}) - \int_W \lambda(u, \mathbf{x}) \, du. \quad (44)$$

You may recognise this as being very similar to the likelihood (4) of the Poisson process. In general it is not a likelihood, but the analogue of the score equation

$$\frac{\partial}{\partial \theta} \log \text{PL}(\theta) = 0$$

is an unbiased estimating equation. Thus the maximum pseudolikelihood estimator is asymptotically unbiased, consistent and asymptotically normal under appropriate conditions.

The main advantage of maximum pseudolikelihood is that, at least for popular Gibbs models, the conditional intensity $\lambda(u, \mathbf{x})$ is easily computable, so that the pseudolikelihood is easy to compute and to maximise. The main disadvantage is the bias and inefficiency of maximum pseudolikelihood in small samples.

More computationally-intensive estimation procedures typically use the maximum pseudolikelihood estimate as their initial guess. We are implementing such procedures in `spatstat` as well.

27.2 Fitting Gibbs models in `spatstat`

We have already met the function `ppm` for fitting Poisson point process models. In fact this function will fit a wide class of Gibbs models.

`ppm` contains an implementation of the algorithm of Baddeley and Turner [9] for maximum pseudolikelihood (which extends the Berman-Turner device for Poisson processes to a general Gibbs process). The conditional intensity of the model, $\lambda_\theta(u, \mathbf{x})$, must be loglinear in the parameters θ :

$$\log \lambda_\theta(u, \mathbf{x}) = \theta \cdot S(u, \mathbf{x}), \quad (45)$$

generalising (5), where $S(u, \mathbf{x})$ is a real-valued or vector-valued function of location u and configuration \mathbf{x} . Parameters θ appearing in the loglinear form (45) are called ‘regular’ parameters, and

all other parameters are ‘irregular’ parameters. For example, the Strauss process conditional intensity (42) can be recast as

$$\log \lambda(u, \mathbf{x}) = \log \beta + (\log \gamma)t(u, \mathbf{x})$$

so that $\theta = (\log \beta, \log \gamma)$ are regular parameters, but the interaction distance r is an irregular parameter (technically called a ‘bloody nuisance parameter’).

In `spatstat` we split the conditional intensity into first-order and higher-order terms:

$$\log \lambda_\theta(u, \mathbf{x}) = \eta \cdot S(u) + \varphi \cdot V(u, \mathbf{x}). \quad (46)$$

The ‘first order term’ $S(u)$ describes spatial inhomogeneity and/or covariate effects. The ‘higher order term’ $V(u, \mathbf{x})$ describes interpoint interaction.

The model with conditional intensity (46) is fitted by calling `ppm` in the form

```
ppm(X, ~ terms, V)
```

The first argument `X` is the point pattern dataset. The second argument `~terms` is a model formula, specifying the first order term $S(u)$ in (46), in the manner described in Section 15. Thus the first order term $S(u)$ in (46) may take very general forms.

The third argument `V` is an object of the special class `"interact"` which describes the interpoint interaction term $V(u, \mathbf{x})$ in (46). It may be compared to the ‘family’ argument which determines the distribution of the responses in a linear model or generalised linear model. Only a limited number of canned interactions are available in `spatstat`, because they must be constructed carefully to ensure that the point process exists.

To fit the Strauss process to the cells data using `ppm`,

```
> data(cells)
> ppm(cells, ~1, Strauss(r = 0.1))
```

Stationary Strauss process

First order term:

```
beta
762.6005
```

Interaction: Strauss process

```
interaction distance:      0.1
Fitted interaction parameter gamma:      0.008
```

Relevant coefficients:

```
Interaction
-4.825006
```

Here `Strauss` is a special function that creates an ‘interaction’ object (class `"interact"`) describing the interaction structure of the Strauss process. Notice that we had to specify the value of the irregular parameter r (more about that later).

To fit the inhomogeneous Strauss process with conditional intensity

$$\lambda(u, \mathbf{x}) = b(u)\gamma^{t(u, \mathbf{x})}$$

where, say, $b(u)$ is loglinear in the Cartesian coordinates,

$$\log b((x, y)) = \beta_0 + \beta_1 x + \beta_2 y$$

we simply type

```
> ppm(cells, ~x + y, Strauss(r = 0.1))
```

Nonstationary Strauss process

Trend formula: $\tilde{x} + y$

Fitted coefficients for trend formula:

(Intercept)	x	y
6.2922384	0.5269869	0.1576416

Interaction: Strauss process

interaction distance: 0.1

Fitted interaction parameter gamma: 0.0082

Relevant coefficients:

Interaction
-4.805565

To fit an inhomogeneous Strauss process with log-quadratic first order term,

```
> ppm(cells, ~polynom(x, y, 2), Strauss(r = 0.1))
```

Nonstationary Strauss process

Trend formula: $\tilde{\text{polynom}}(x, y, 2)$

Fitted coefficients for trend formula:

(Intercept)	polynom(x, y, 2)[x]	polynom(x, y, 2)[y]
5.9747220	-0.9375707	3.4732733
polynom(x, y, 2)[x ²]	polynom(x, y, 2)[x.y]	polynom(x, y, 2)[y ²]
1.4970947	-0.1838987	-3.3696109

Interaction: Strauss process

interaction distance: 0.1

Fitted interaction parameter gamma: 0.0081

Relevant coefficients:

Interaction
-4.812711

27.3 Interpoint interactions

Instead of `Strauss` we may use any of the following functions to create an interaction:

<code>AreaInter()</code>	area-interaction process
<code>BadGey()</code>	hybrid Geyer saturation process
<code>DiggleGratton()</code>	Diggle-Gratton potential
<code>DiggleGatesStibbard()</code>	Diggle-Gates-Stibbard potential
<code>Fiksel()</code>	Fiksel double exponential potential
<code>Geyer()</code>	Geyer's saturation process
<code>Hardcore()</code>	hard core process
<code>LennardJones()</code>	Lennard-Jones potential
<code>Ord()</code>	Ord model, user-supplied potential
<code>OrdThresh()</code>	Ord process, threshold potential
<code>PairPiece()</code>	pairwise interaction, piecewise constant
<code>Pairwise()</code>	pairwise interaction, user-supplied potential
<code>Poisson()</code>	the Poisson point process (the default)
<code>Saturated()</code>	general saturated model, user-supplied potential
<code>SatPiece()</code>	multiscale saturation process
<code>Softcore()</code>	pairwise interaction, soft core potential
<code>Strauss()</code>	the Strauss process
<code>StraussHard()</code>	the Strauss/hard core point process

(There are three additional ones for multitype point processes, described in section 34.3.2.)

The area-interaction model and the Geyer saturation model are quite handy, as they can be used to model both clustering and regularity.

```
> data(redwood)
> ppm(redwood, ~1, Geyer(r = 0.07, sat = 2))
```

Stationary Geyer saturation process

First order term:

```
beta
12.39488
```

Interaction: Geyer saturation process

```
interaction distance:      0.07
saturation parameter:      2
Fitted interaction parameter gamma:      2.9004
```

Relevant coefficients:

```
Interaction
1.064845
```

```
> ppm(redwood, ~1, AreaInter(r = 0.03))
```

Stationary Area-interaction process

First order term:

```
beta
```


36.53100

```
Interaction: Area-interaction process
disc radius:      0.03
Fitted interaction parameter eta:      15.7515
```

```
Relevant coefficients:
Interaction
  2.756935
```

The printout for the area-interaction model uses the “scale-free” parameter `eta` defined by

$$\eta = \gamma^{\pi r^2}$$

where γ and r are the parameters appearing in the definition (43). Values of η greater than 1 suggest clustering.

For more detailed explanation of modelling, see [11].

27.4 Fitted point process models

The result of the `ppm` call is an object of class “`ppm`” (‘point process model’). This is very closely analogous to a fitted linear model (`lm`) or fitted generalised linear model (`glm`).

Standard R operations that are defined for fitted point process models (i.e. that have methods for the class “`ppm`”) include:

<code>print</code>	print basic information
<code>summary</code>	print detailed summary information
<code>plot</code>	plot the fitted (conditional) intensity
<code>predict</code>	fitted (conditional) intensity
<code>fitted</code>	fitted (conditional) intensity at data points
<code>update</code>	re-fit the model
<code>coef</code>	extract the fitted coefficient vector $\hat{\theta}$
<code>vcov</code>	variance-covariance matrix of $\hat{\theta}$
<code>anova</code>	analysis of deviance
<code>logLik</code>	evaluate log-pseudolikelihood
<code>model.matrix</code>	extract design matrix
<code>formula</code>	extract trend formula of model
<code>terms</code>	extract terms in model formula

(the methods for `anova` and `vcov` are only available for Poisson models). The following functions are also available:

<code>step</code>	stepwise model selection
<code>drop1</code>	one step backward in model selection
<code>model.images</code>	compute images of canonical covariates in model
<code>effectfun</code>	fitted intensity as function of one covariate

Plotting a fitted model generates a series of image and contour plots of

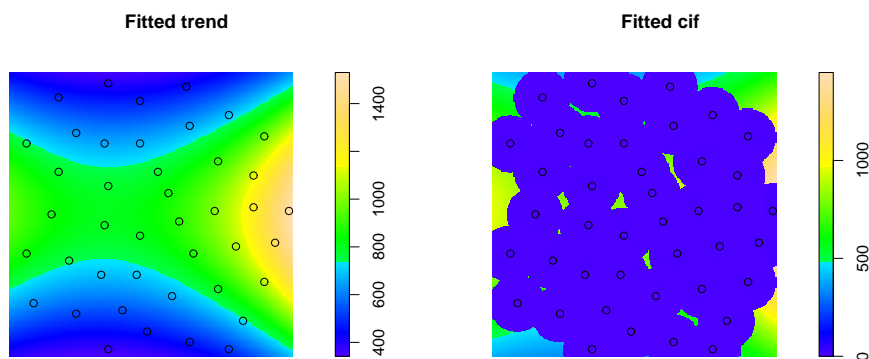
- the fitted first order term $\exp(\hat{\eta} \cdot S(u))$
- the fitted conditional intensity $\lambda_{\hat{\theta}}(u, \mathbf{x})$ evaluated for the data pattern \mathbf{x}

For Poisson models, the two plots are equivalent, and give the fitted intensity function.

```

> fit <- ppm(cells, ~polynom(x, y, 2), Strauss(r = 0.1))
> par(mfrow = c(1, 2))
> plot(fit, how = "image", ngrid = 256)

```

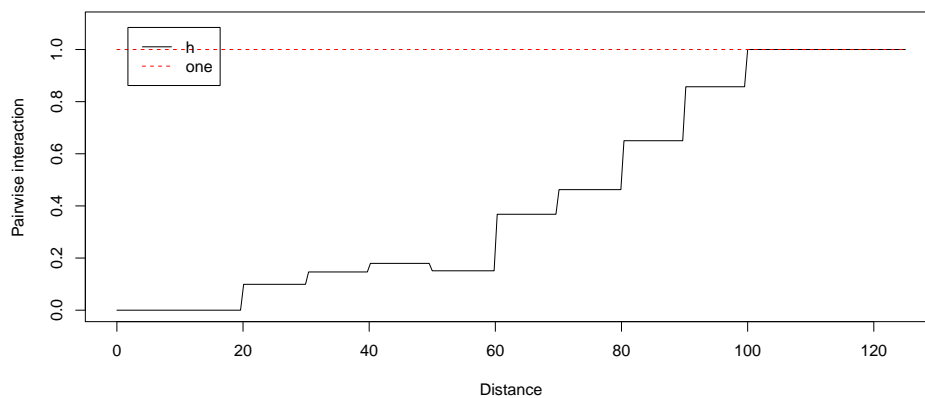


For non-Poisson models, it is also possible to extract and plot the interpoint interaction function, using `fitin`.

```

> model <- ppm(X, ~1, PairPiece(seq(10, 100, by = 10)))
> f <- fitin(model)
> plot(f)

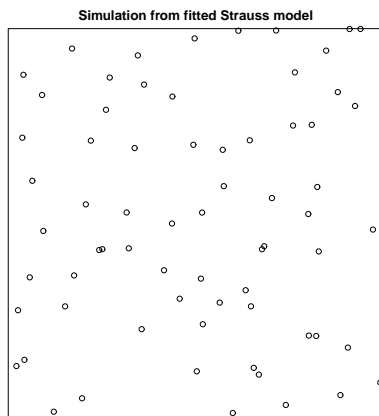
```



27.5 Simulation from fitted models

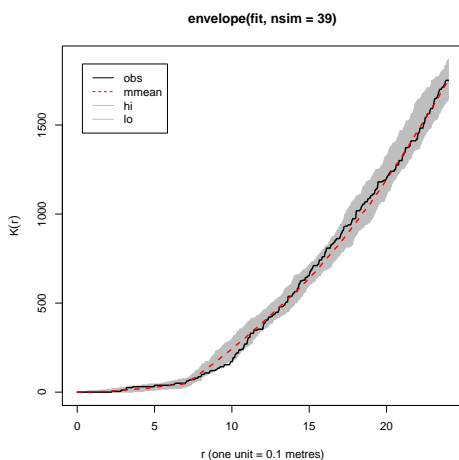
A fitted Gibbs model can also be simulated automatically using `rmh`.

```
> fit <- ppm(swedishpines, ~1, Strauss(r = 7))
> Xsim <- rmh(fit)
> plot(Xsim, main = "Simulation from fitted Strauss model")
```



The `envelope` command will also generate simulation envelopes for a fitted model.

```
> plot(envelope(fit, nsim = 39))
```



27.6 Dealing with nuisance parameters

Irregular parameters, such as the interaction radius r in the Strauss process, cannot be estimated directly using `ppm`. Indeed the statistical theory for estimating such parameters is unclear.

For some special cases, a maximum likelihood estimator of the nuisance parameter is available. For example, for the 'hard core process' (Strauss process with interaction parameter $\gamma = 0$) with interaction radius r , the maximum likelihood estimator is the minimum nearest-neighbour distance. Thus the following is a reasonable approach to the `cells` dataset:

```
> rhat <- min(nndist(cells))
> rhat <- rhat * 0.99999
> ppm(cells, ~1, Strauss(r = rhat))
```

Stationary Strauss process

First order term:

beta
301.0949

Interaction: Strauss process

interaction distance: 0.0836293018068393
Fitted interaction parameter gamma: 0

Relevant coefficients:

Interaction
-20.77031

The analogue of profile likelihood, *profile pseudolikelihood*, provides a general solution which may or may not perform well. If $\theta = (\phi, \eta)$ where ϕ denotes the nuisance parameters and η the regular parameters, define the profile log pseudolikelihood by

$$\text{PPL}(\phi, \mathbf{x}) = \max_{\eta} \log \text{PL}((\phi, \eta); \mathbf{x}).$$

The right hand side can be computed, for each fixed value of ϕ , by the algorithm `ppm`. Then we just have to maximise $\text{PPL}(\phi)$ over ϕ . This is done by the command `profilepl`:

```
> data(simdat)
> df <- data.frame(r = seq(0.05, 2, by = 0.025))
> pfit <- profilepl(df, Strauss, simdat, ~1)
```

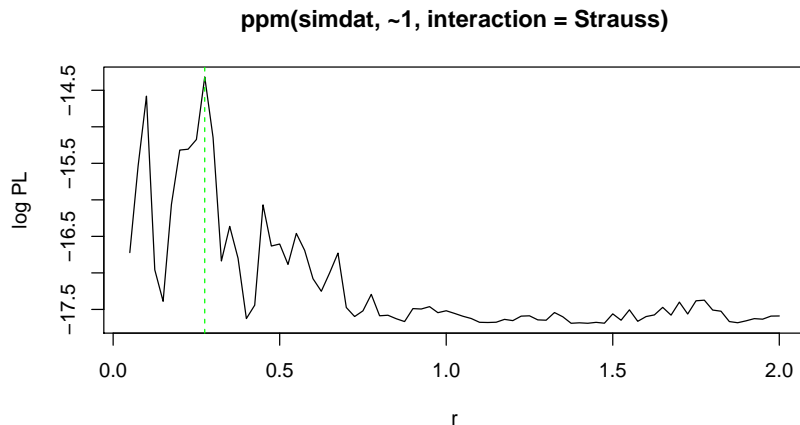
```
> pfit
```

Profile log pseudolikelihood values

```
for model: ppm(simdat, ~1, interaction = Strauss)
fitted with rbord= 2
Interaction: Strauss
with irregular parameter r in [0.05, 2]
Optimum value of irregular parameter: r = 0.275
```

The result is an object of class `profilepl` containing the profile log pseudolikelihood function, the optimised value of the irregular parameter r , and the final fitted model. To plot the profile log pseudolikelihood,

```
> plot(pfit)
```



To extract the final fitted model,

```
> pfit$fit
```

Stationary Strauss process

First order term:

```
beta
2.583110
```

Interaction: Strauss process

```
interaction distance:      0.275
Fitted interaction parameter gamma:      0.5631
```

Relevant coefficients:

```
Interaction
-0.5743608
```

There is a `summary` method for these objects as well.

27.7 Improvements over maximum pseudolikelihood

Maximum pseudolikelihood is quick and dirty. There are statistically more efficient alternatives, but they are computationally intensive.

Currently we have implemented the easiest of these alternatives, the Huang-Ogata [43] one-step approximation to maximum likelihood. Starting from the maximum pseudolikelihood estimate $\hat{\theta}_{PL}$, we simulate M independent realisations of the model with parameters $\hat{\theta}_{PL}$, evaluate the canonical sufficient statistics, and use them to form estimates of the score and Fisher information at $\theta = \hat{\theta}_{PL}$. Then we take one Newton-Raphson step, updating the value of θ . The rationale is that the log-likelihood is approximately quadratic in a neighbourhood of the maximum pseudolikelihood estimator, so that one Newton-Raphson step is almost enough.

To use the Huang-Ogata method instead of maximum pseudolikelihood, add the argument `method="ho"`.

```
> fit <- ppm(simdat, ~1, Strauss(r = 0.275), method = "ho")
```

```
> fit
```

Stationary Strauss process

First order term:

```
beta
2.42845
```

Interaction: Strauss process

```
interaction distance:      0.275
Fitted interaction parameter gamma:      0.5277
```

Relevant coefficients:

```
Interaction
-0.6392568
```

```
> vcov(fit)
```

```
          [,1]      [,2]
[1,]  0.01000399 -0.01255788
[2,] -0.01255788  0.04019740
```

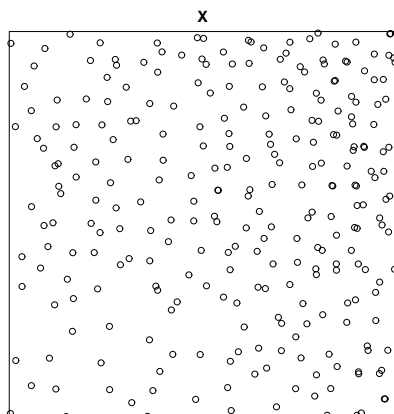
For models fitted by Huang-Ogata, the variance-covariance matrix returned by `vcov` is computed from the simulations.

28 Validation of fitted Gibbs models

Goodness-of-fit testing and model validation for Poisson models were described in Section 16. Checking a fitted Gibbs point process model is more difficult. There is little theory available to support goodness-of-fit tests and the like.

As an example, consider the following data:

```
> data(residualspaper)
> X <- residualspaper$Fig4b
> plot(X)
```



We fit a Strauss process model with a log-quadratic intensity term:

```
> fit <- ppm(X, ~polynom(x, y, 2), Strauss(0.05), correction = "isotropic")
```

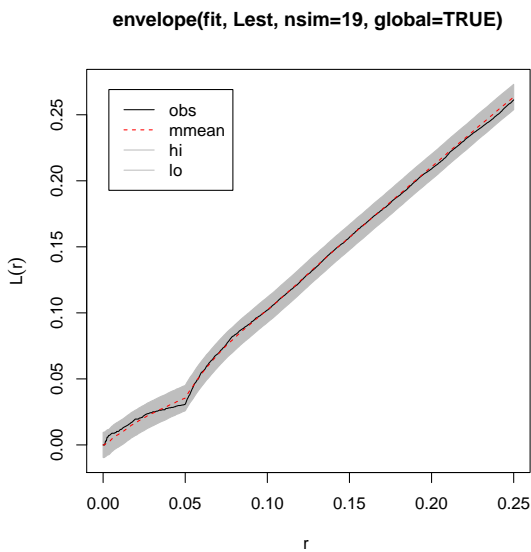
The question is how to confirm or validate this model.

28.1 Goodness-of-fit testing for Gibbs processes

For a fitted Gibbs process, no theory is available to support the χ^2 goodness-of-fit test or the Kolmogorov-Smirnov test. The predicted mean number of points in a given region is not known in closed form for a Gibbs process. Thus, the appropriate test statistic for a χ^2 test is not even available in closed form, let alone the null distribution of this statistic.

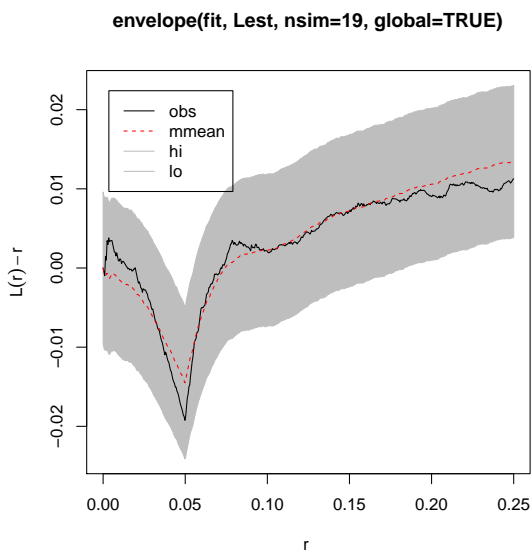
Instead, goodness-of-fit for fitted Gibbs models often relies on the summary functions K and G . The command `envelope` will accept as its first argument a fitted Gibbs model, and will simulate from this model to determine the critical envelope.

```
> plot(envelope(fit, Lest, nsim = 19, global = TRUE))
```



Let's subtract the theoretical Poisson value $L(r) = r$ to get a more readable plot:

```
> plot(envelope(fit, Lest, nsim = 19, global = TRUE), . - r ~ r)
```



This is fairly consistent with a Strauss process.

28.2 Residuals for Gibbs processes

28.2.1 Definition

Residuals for a general Gibbs model were defined only recently [12, 6]. The total residual in a region $B \subset \mathbb{R}^2$ is defined as

$$R(B) = n(\mathbf{x} \cap B) - \int_B \widehat{\lambda}(u, \mathbf{x}) \, du \quad (47)$$

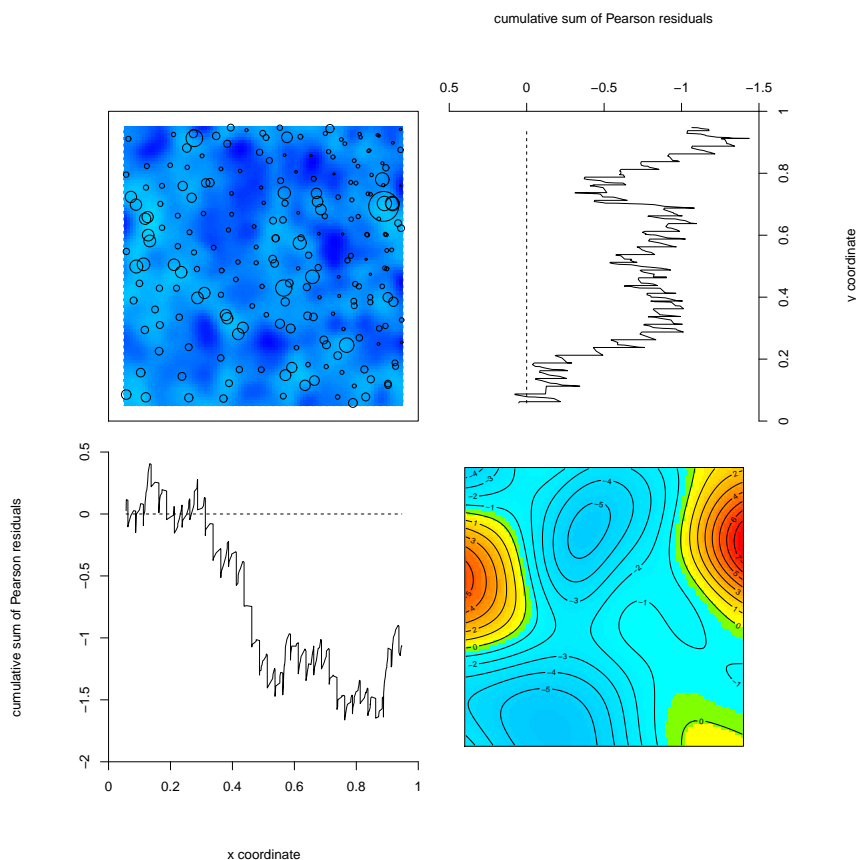
where again $n(\mathbf{x} \cap B)$ is the observed number of points in the region B , and $\widehat{\lambda}(u, \mathbf{x})$ is the **conditional** intensity of the fitted model, *evaluated for the data point pattern* \mathbf{x} . If the fitted model is correct, the residuals have mean zero.

This definition is similar to the definition of residuals for Poisson processes (Section 16.2) except that the intensity $\widehat{\lambda}(u)$ of the fitted Poisson process has been replaced by the *conditional* intensity $\widehat{\lambda}(u, \mathbf{x})$ of the fitted Gibbs process evaluated for the data point pattern \mathbf{x} .

28.2.2 Residual plots

Residuals for Gibbs processes can be plotted using the same techniques as in Section 16.2. Here is the four-panel plot:

```
> diagnose.ppm(fit, type = "Pearson")
```



At the time of writing, `spatstat` does not yet display 2σ significance bands for the lurking variable plots when the fitted model is not Poisson. The interpretation of the lurking variable plots is a little more difficult without the significance bands. One tends to place a little more emphasis on the smoothed residual field. The Pearson residuals should be *approximately* standardised, so that values which are much greater than 2 (in absolute value) suggest a lack of fit.

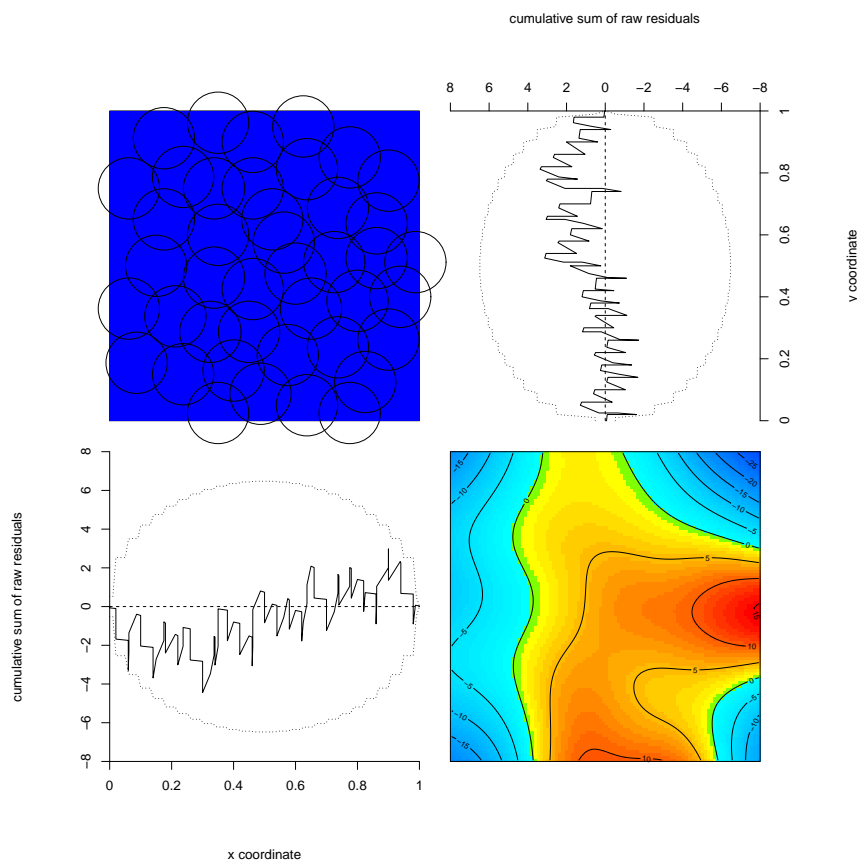
The four-panel plot above suggests that the model is a reasonable fit.

28.2.3 Q-Q plots

As we noted in Section 16.2.6, the four-panel residual plot and the lurking variable plot are useful for detecting misspecification of the *trend* in a fitted model. They are not very useful for checking misspecification of the *interaction* in a fitted model.

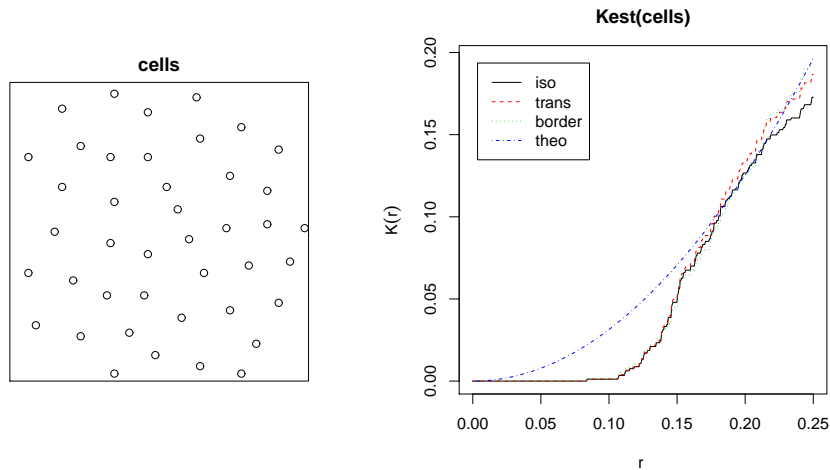
An extreme example is provided by the `cells` dataset. The residual plots for a uniform Poisson process fitted to the cells data suggest that this is a good model:

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



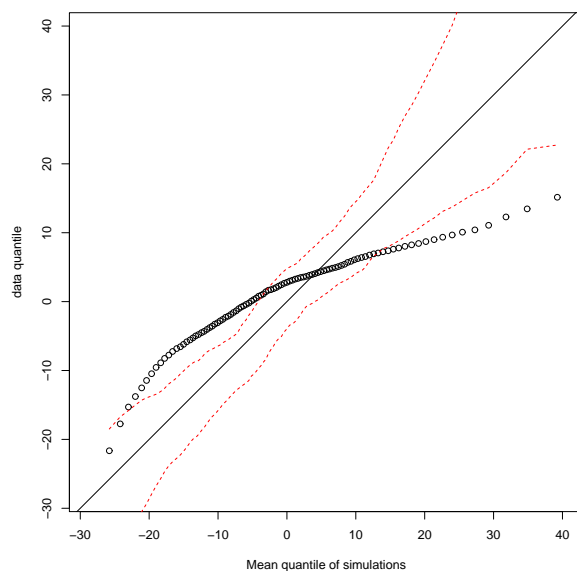
However, the K -function shows that the `cells` dataset is clearly not a Poisson pattern, but has strong inhibition:

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



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

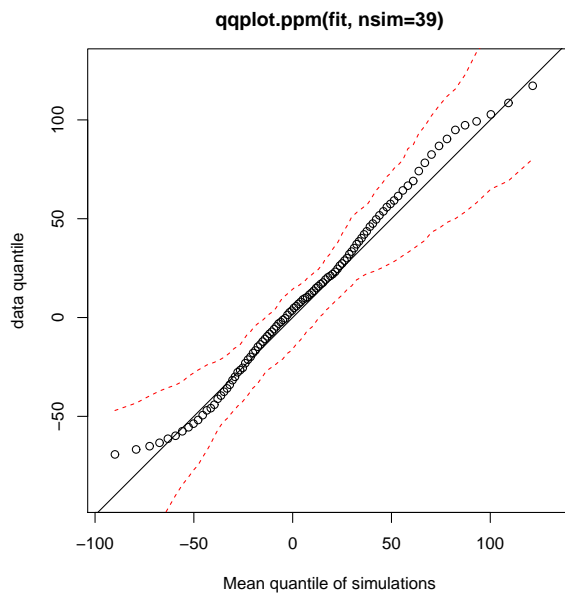
```
> qqplot.ppm(fitPois, nsim = 39)
```



This shows a $Q-Q$ plot of the smoothed residuals for a uniform Poisson model fitted to the `cells` data, with pointwise 5% critical envelopes from simulations of the fitted model. This indicates that the uniform Poisson model is grossly inappropriate for the `cells` data.

Returning to the model we fitted at the start of this chapter:

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



This shows a Q–Q plot of the smoothed residuals, with pointwise 5% critical envelopes from simulations of the fitted model. This suggests that the Strauss model is reasonable.

These validation techniques generalise and unify many existing exploratory methods. For particular models of interpoint interaction, the Q–Q plot is closely related to the summary functions F , G and K . See [12].

28.3 New methods

Several new diagnostic tools for validating the interaction term in a point process model will be published soon [8]. These methods will then be released in **spatstat**.

PART VII. MARKED POINT PATTERNS

Part VII of the workshop deals with marked point patterns.

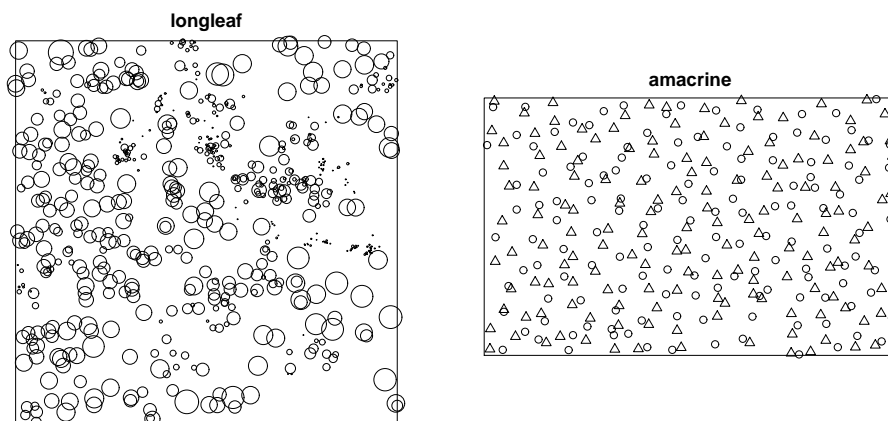
29 Marked point patterns

29.1 Marked point patterns

Each point in a spatial point pattern may carry additional information called a ‘mark’. For example, points which are classified into two or more different types (on/off, case/control, species, colour, etc) may be regarded as marked points, with a mark which identifies which type they are. Data recording the locations and heights of trees in a forest can be regarded as a marked point pattern where the mark attached to a tree’s location is the tree height.

Many of the functions in `spatstat` handle marked point patterns in which the mark attached to each point is either

- a **continuous variate** or “real number”. An example is the Longleaf Pines dataset (`longleaf`) in which each tree is marked with its diameter at breast height. The `marks` component must be a **numeric** vector such that `marks[i]` is the mark value associated with the *i*th point. We say the point pattern has *continuous marks*.
- a **categorical variate**. An example is the Amacrine Cells dataset (`amacrine`) in which each cell is identified as either “on” or “off”. Such point patterns may be regarded as consisting of points of different “types”. The `marks` component must be a **factor** such that `marks[i]` is the label or type of the *i*th point. We call this a *multitype point pattern* and the levels of the factor are the possible types.



Note that, in some other packages, a point pattern dataset consisting of points of two different types (A and B say) is represented by two datasets, one representing the points of type A and another containing the points of type B. In `spatstat` we take a different approach, in which all the points are collected together in one point pattern, and the points are then labelled by the type to which they belong. An advantage of this approach is that it is easy to deal with multitype point patterns with more than 2 types. For example the classic Lansing Woods dataset represents the positions of trees of 6 different species. This is available in `spatstat` as a single dataset, a marked point pattern, with the marks having 6 levels.

29.2 Formulation

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

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

A marked point pattern is an unordered set

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

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

29.3 Methodological issues

29.3.1 Should the data be treated as a marked point process?

In a marked point process the points are random. Treating the data as a point process is inappropriate if the locations are fixed, or if the locations are not part of the ‘response’.

Example 16 *Today’s maximum temperatures at 25 Australian cities are displayed on a map.*

This is not a point process in any useful sense. The cities are fixed locations. The temperatures are observations of a spatial variable at a fixed set of locations. See the R packages `sp`, `spdep`, `spgwr` for suitable methods.

Example 17 *A mineral exploration dataset records the map coordinates where 15 core samples were drilled, and for each core sample, the assayed concentration of iron in the sample.*

This typically should *not* be treated as a point process. The core sample locations were chosen by a geologist, and are part of the experimental design. The main interest is in the iron concentration at these locations. This should probably be analysed as a geostatistical dataset. See the R packages `geoR`, `geoRglm` for suitable methods.

29.3.2 Joint vs. conditional analysis

There are more choices for analysis (and more traps) when marks are present. Schematically, if we write X for the points and M for the marks, then a statistical model for the marked point pattern could be formulated in several ways:

- $[X] [M|X]$ — ‘conditional on locations’ — points X are first generated according to a spatial point process, then marks M are ‘assigned’ to the points by a random mechanism $[M|X]$;
- $[M] [X|M]$ — ‘conditional on marks’ or ‘split by marks’ — marks M are first generated according to some random mechanism $[M]$, then they are placed at certain locations X by point process(es) $[X|M]$;
- $[X, M]$ — ‘joint’ — marked points are generated according to a marked point process.

These approaches typically lead to different stochastic models and have different inferential interpretations. Correspondingly, there are different null hypotheses that can be tested:

- *random labelling*: given the locations X , the marks are conditionally independent and identically distributed;

- *independence of components*: the sub-processes \mathbf{X}_m of points of each mark m , are independent point processes;
- *complete spatial randomness and independence (CSRI)*: the locations \mathbf{X} are a uniform Poisson point process, and the marks are independent and identically distributed. (This implies both random labelling and independence of components).

These null hypotheses are not equivalent.

The properties of random labelling and independence of components are not equivalent. For example, take a point process \mathbf{X} where nearest neighbour distances are always larger than a threshold r , and attach random marks to the points. The resulting marked point process cannot be generated using the independence construction, because if points with different marks are independent, they can come arbitrarily close to one another.

Example 18 (Ant nests data) *Two species of ants build nests in a desert. We want to investigate ecological interaction between the species, and between different nests of the same species. The locations of all nests are mapped, and marked by the species.*

These data can be analysed as a marked point process consisting of two different types of points. The ‘mark’ attached to each point is its species (a categorical variable). The most natural kind of modelling and analysis is either joint $[X, M]$ or split by species $[M] [X|M]$. We could also treat one of the species as a covariate and analyse the other species conditional on it.

Example 19 *Trees in an orchard are examined and their disease status (infected/not infected) is recorded. We are interested in the spatial characteristics of the disease, such as contagion between neighbouring trees.*

These data probably should *not* be treated as a point process. The response is ‘disease status’. We can think of disease status as a label applied to the trees after their locations have been determined. Since we are interested in the spatial correlation of disease status, the tree locations are effectively fixed covariate values. It would probably be best to treat these data as a discrete random field (of disease status values) observed at a finite known set of sites (the trees).

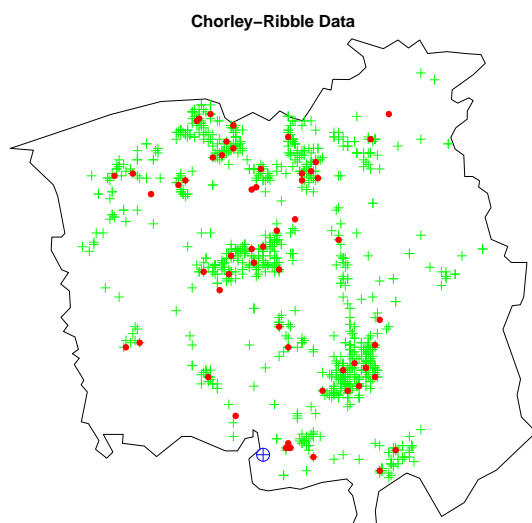
29.3.3 Grey areas

There are some ‘grey areas’ which permit several alternative choices of analysis. It could be appropriate either to analyse the locations and marks jointly (denoted $[X, M]$), or to analyse the marks conditional on the locations ($[M|X]$) or to analyse the locations given the marks ($[X|M]$).

One grey area occurs when the locations are random, but may be ancillary for the parameters of interest.

Example 20 *Case-control study of cancer [34, 38]. The domicile locations of all new cases of a rare cancer are mapped. To allow for spatial variation in the density of the susceptible population, domicile locations are recorded for a random sample of (matched) controls.*

This can be analysed either as a marked point pattern (where the mark is the case/control label) or, by conditioning on locations, as a random field of case/control values attached to the known domicile locations.



For further discussion of these issues, see [3].

30 Handling marked point pattern data

This section explains how to create a marked point pattern dataset in `spatstat`, and how to manipulate it.

30.1 Creating marked point pattern datasets

The marks attached to a point pattern may be stored in a vector (with one entry for each point) or in a matrix or data frame (with one row for each point and one column for each mark variable). The mark values can be of any atomic type: numeric, integer, character, factor, logical or complex.

A marked point pattern dataset can be created using any of the following tools:

<code>ppp</code>	create point pattern dataset
<code>as.ppp</code>	convert other data to point pattern
<code>superimpose</code>	combine several point patterns into a marked point pattern
<code>marks</code>	extract marks from a point pattern
<code>marks<-</code>	attach marks to a point pattern
<code>%mark%</code>	attach marks to a point pattern
<code>unmark</code>	delete marks from a point pattern
<code>scanpp</code>	read point pattern data from text file
<code>clickppp</code>	create a pattern using point-and-click on the screen

The command `ppp` can be used to create a marked point pattern dataset from raw data. The syntax is

```
> ppp(x, y, ..., marks = m)
```

where `x` and `y` are vectors of equal length containing the (x, y) coordinates, `m` is either a vector of the same length as `x` containing the mark values for each point, or a matrix or data frame with `nrow(m) = length(x)` containing the multivariate mark for each point, and `...` are arguments that determine the window for the point pattern.

Tip: If the marks are a vector and are intended to be a categorical variable (representing the types in a multitype point pattern),

- ensure that `m` is stored as a **factor** in R.
- when the point pattern `X` has been created, check that it is multitype using `is.multitype(X)`.
- check that the factor levels are as you intended, using `levels(m)` or `levels(marks(X))` where `X` is the marked point pattern. If the factor levels are character strings, they will be sorted into alphabetical order by default.
- be careful when performing equality/inequality comparisons involving a factor. Particular danger occurs when the factor levels are strings that represent integers.

The command `as.ppp` will convert data in another format (for example, a matrix or data frame) to a point pattern object of class "ppp". The third and subsequent columns of a matrix or data frame will be interpreted as containing the marks.

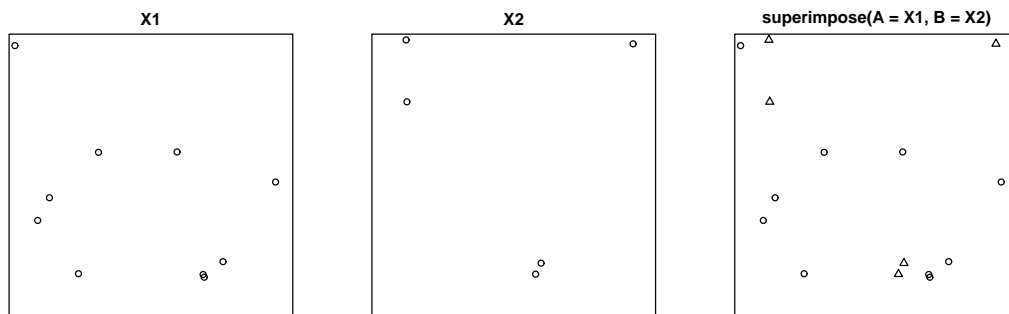
```
> mydata <- data.frame(x = runif(10), y = runif(10), m = sample(letters[1:3],
+ 10, replace = TRUE))
> as.ppp(mydata, square(1))
```

```
marked planar point pattern: 10 points
multitype, with levels = a      b      c
window: rectangle = [0, 1] x [0, 1] units
```

If point pattern data are stored in a text file, the command `scanpp` will read the data and create a point pattern object of class "ppp". The argument `multitype=TRUE` will ensure that the mark values are interpreted as a factor.

```
> X <- scanpp("myfile.txt", window = square(1), multitype = TRUE)
```

The command `superimpose` combines several point patterns within the same window. It can be used to create a multitype point pattern, if you have already created separate point patterns containing the points of each type. Suppose `X1` and `X2` are unmarked point patterns. Then `superimpose(A=X1, B=X2)` will create a multitype point pattern by attaching the mark `A` to each point of `X1`, attaching the mark `B` to each point of `X2`, and combining the points.



Marks can be attached to an existing point pattern `X` using the function `marks<-` as in

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

or using the binary operator `%mark%`,

```
> Y <- X %mark% m
```

These are convenient when you want to assign new marks to a dataset that are computed using another variable, or perhaps to randomise the marks in a dataset.

A multitype point pattern can also be created interactively using `clickppp`, using the argument `types` to specify the possible types.

30.2 Inspecting a marked point pattern

Basic tools for inspecting a marked point pattern include the `print`, `plot` and `summary` methods.

```
> data(amacrine)
> amacrine

marked planar point pattern: 294 points
multitype, with levels = off      on
window: rectangle = [0, 1.6012085] x [0, 1] units (one unit = 662 microns)
```

```
> summary(amacrine)
```

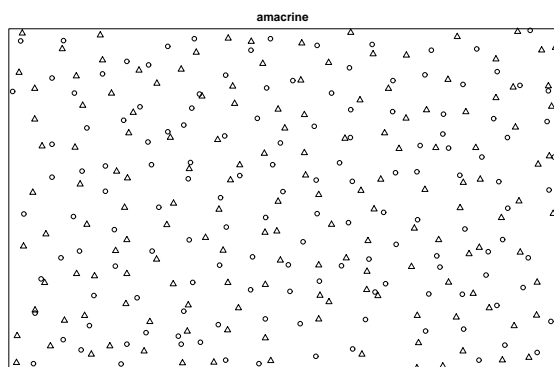
```
Marked planar point pattern: 294 points
Average intensity 184 points per square unit (one unit = 662 microns)
Multitype:
```

	frequency	proportion	intensity
off	142	0.483	88.7
on	152	0.517	94.9

```
Window: rectangle = [0, 1.6012085]x[0, 1]units
Window area = 1.60121 square units
Unit of length: 662 microns
```

```
> plot(amacrine)
```

```
off  on
  1   2
```

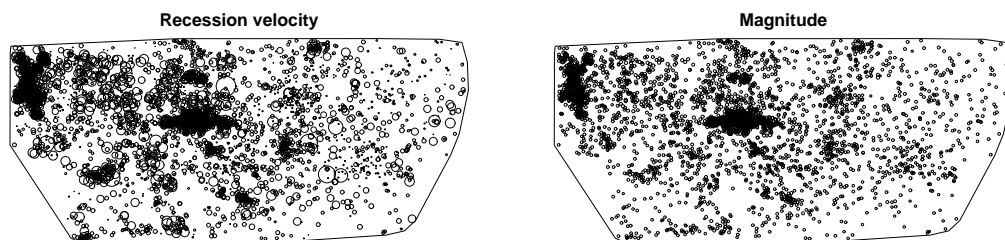


If the marks are a data frame (i.e. if there are several columns of marks), the first column of marks will be plotted by default. To change this, use the argument `which.marks` to specify another column.

```

> data(shapley)
> par(mfrow = c(1, 2))
> plot(shapley, which.marks = "V", maxsize = 0.5, main = "Recession velocity")
      -20000      0      20000      40000      60000      80000
-0.1303000  0.0000000  0.1303000  0.2605999  0.3908999  0.5211998
> plot(shapley, which.marks = "Mag", maxsize = 0.1, main = "Magnitude")
      0      5      10      15      20      25
0.0000000  0.0223914  0.0447828  0.0671742  0.0895656  0.1119570
> par(mfrow = c(1, 1))

```



You can also convert a marked point pattern into a data frame for closer inspection of the coordinates and mark values:

```

> as.data.frame(amacrine)
      x      y marks
1  0.0224 0.0243   on
2  0.0243 0.1028   on
3  0.1626 0.1477   on
.....

```

The marks can be extracted using the function `marks`:

```

> data(longleaf)
> m <- marks(longleaf)

```

Beware the possibility that two points with different marks may occupy the same spatial location. This is not currently detected by `ppp` since, for a marked point pattern, the function `duplicated.ppp` regards two points as identical only when their coordinates **and** mark values are identical. To detect duplication of the spatial locations, use `duplicated(unmark(X))`.

Further tools are presented in the next section.

30.3 Manipulating data

30.3.1 Manipulating marks

The following tools can manipulate the marks in a point pattern:

```

marks      extract marks
marks<-    attach marks to a point pattern
%mark%     attach marks to a point pattern
unmark     remove marks from point pattern

```

For example, the Lansing Woods data are tree locations marked by diameter at breast height (dbh) in centimetres. To convert the marks from diameters to circular areas,

```
> data(lansing)
> d <- marks(lansing)
> a <- (pi/4) * d^2
> marks(lansing) <- a
```

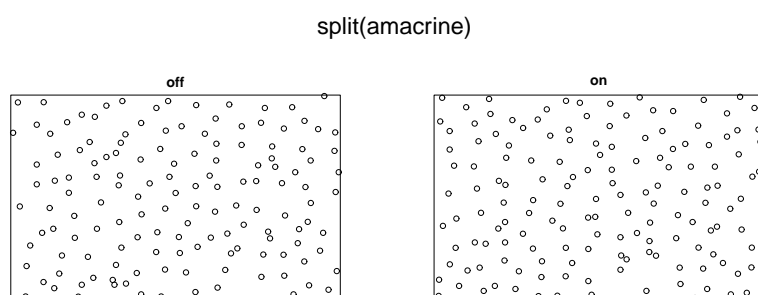
30.3.2 Separating points of different types

A *multitype* point pattern can be separated into the sub-patterns of points of each type, using the `split` command.

```
> data(amacrine)
> Y <- split(amacrine)
```

In fact `split` is a generic function and the commands above invoke the `split` method for the class of point patterns, `split.ppp`. The result `Y` is a list of point patterns, with names that correspond to the type labels. This list also belongs to the class "`splitppp`" which can be plotted automatically:

```
> plot(split(amacrine))
```



If the marks are a data frame, you will need to specify the splitting/grouping factor explicitly. For example the `nbfires` dataset records the location of forest fires, marked by 9 different variables. To split the fire locations by the cause of the fire,

```
> data(nbfires)
> Y <- split(nbfires, "cause")
```

30.3.3 Cutting the numerical scale into bands

For a point pattern with *numeric* marks, the marks can be converted to a factor, using a method for the generic function `cut`. The user specifies a series of cut-points on the numerical scale; all mark values between two cut-points are given the same label.

For example, the Longleaf Pines data are the locations of trees marked with their diameter at breast height, dbh, in centimetres. By convention we define “adult” trees to be those with dbh greater than 30 centimetres. To obtain the bivariate point pattern of adult and juvenile trees,

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

```
marked planar point pattern: 584 points
marks are numeric, of type double
window: rectangle = [0, 200] x [0, 200] metres
```

```
> X <- cut(longleaf, breaks = c(0, 30, 80), labels = c("juvenile",
+ "adult"))
> X
```

```
marked planar point pattern: 584 points
multitype, with levels = juvenile      adult
window: rectangle = [0, 200] x [0, 200] metres
```

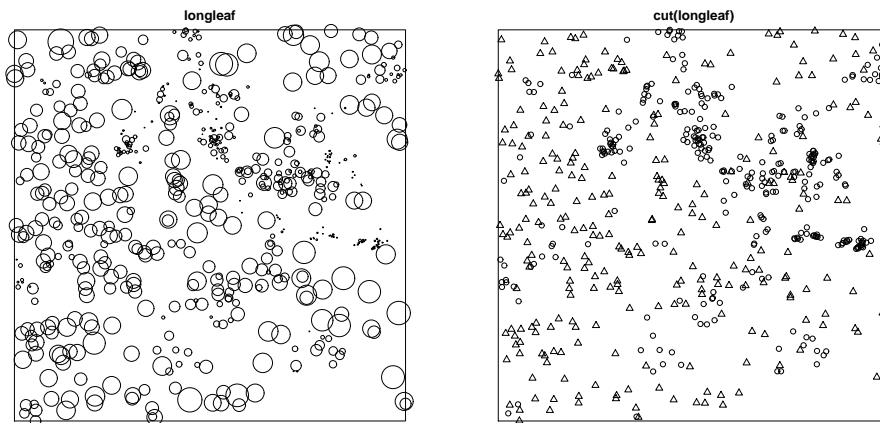
```
> par(mfrow = c(1, 2))
> plot(longleaf)
```

```
      0      20      40      60      80
0.000000 1.722522 3.445045 5.167567 6.890090
```

```
> plot(X, main = "cut(longleaf)")
```

```
juvenile  adult
      1      2
```

```
> par(mfrow = c(1, 1))
```



If the marks are a data frame, use the second argument `z` to specify which column should be used for the cut. For example, to classify New Brunswick fires into three groups by fire size,

```
> data(nbfires)
> Y <- cut(nbfires, "fnl.size", breaks = 4)
```

31 Exploratory tools for multitype point patterns

This section covers some tools for exploratory data analysis of multitype point patterns (i.e. where the marks are categorical).

31.1 Intensity

The Lansing Woods data give the locations of 6 species of trees in a forest in Michigan. Elementary estimates of the frequency distribution of species, and the intensity of each species, are available from `summary.ppp`.

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

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

```
*Pattern contains duplicated points*
```

```
Multitype:
```

	frequency	proportion	intensity
blackoak	135	0.0600	135
hickory	703	0.3120	703
maple	514	0.2280	514
misc	105	0.0466	105
redoak	346	0.1540	346
whiteoak	448	0.1990	448

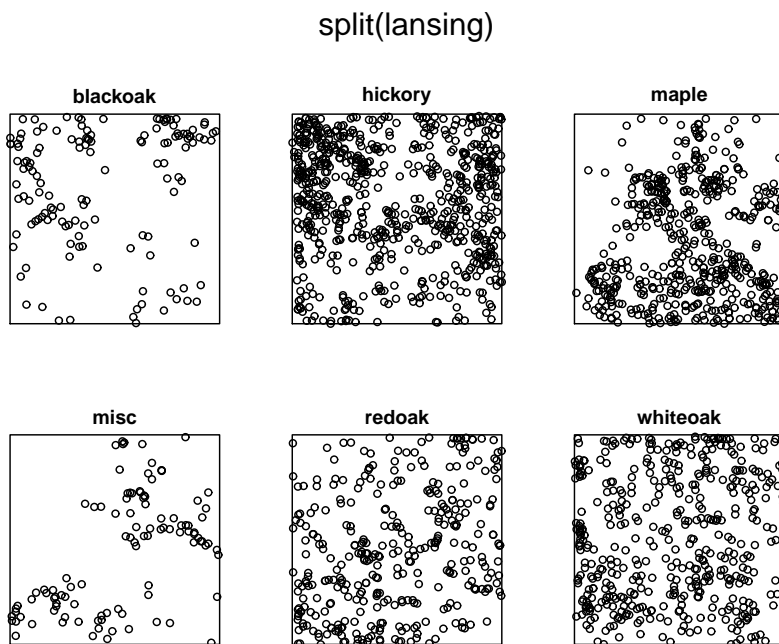
```
Window: rectangle = [0, 1]x[0, 1]units
```

```
Window area = 1 square unit
```

```
Unit of length: 924 feet
```

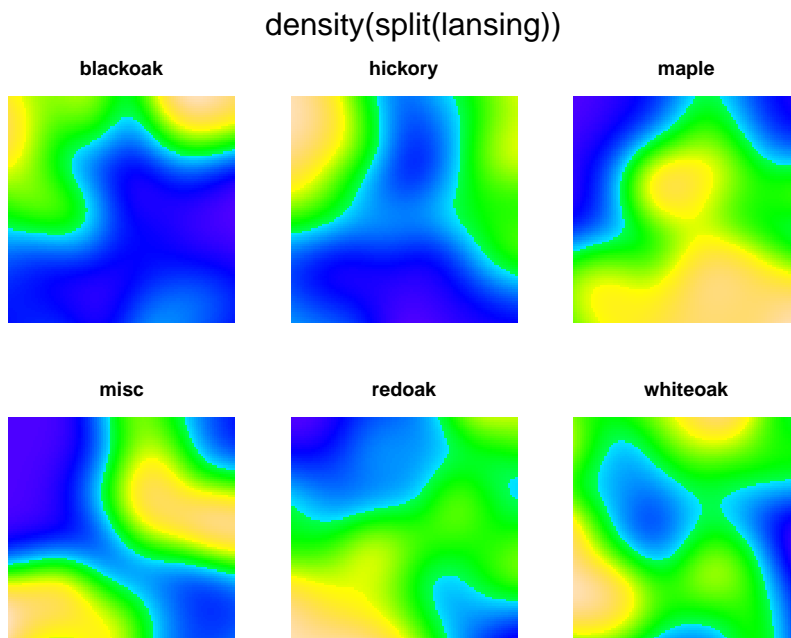
It's sensible to examine the sub-patterns of different types separately, using `split.ppp`.

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



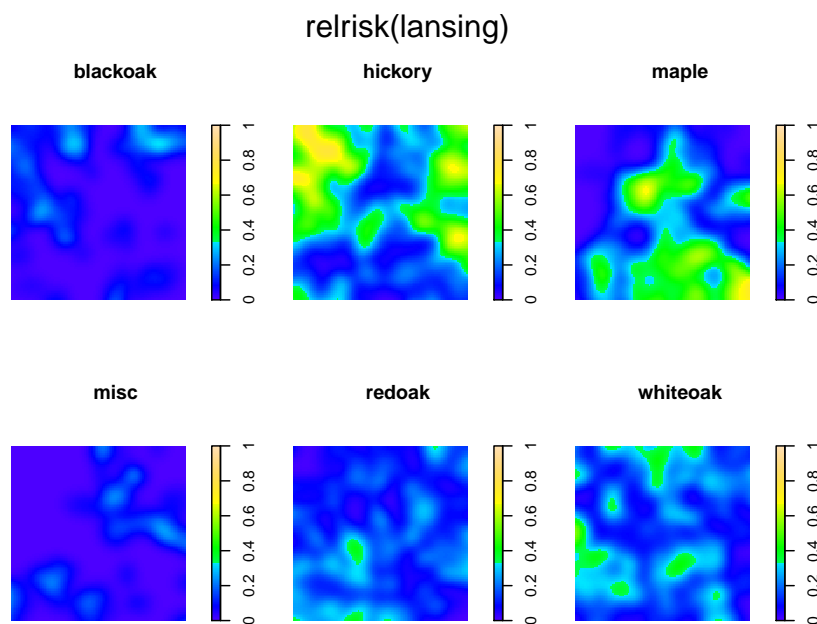
It would be useful to compute and plot a separate estimate of intensity for each type of tree. This is possible using the functions `density.splitppp` and `plot.listof`. They are invoked simply by typing

```
> plot(density(split(lansing)), ribbon = FALSE)
```



The relative proportions of intensity could then be computed by taking ratios of these densities, using `eval.im`. This is done more neatly using the command `relrisk` (which will also select the bandwidth automatically by cross-validation if it is not specified).

```
> plot(relrisk(lansing), zlim = c(0, 1))
```



In the context of the Lansing Woods data, these tools are searching for evidence of *segregation* between the different species of trees. Segregation occurs when the intensities of different species are negatively associated.

Parametric estimates of intensity can be obtained using `ppm`, fitting a Poisson model with an intensity function that may depend on location and/or on the marks. See below.

Evidence for segregation in a multitype point pattern can also be assessed using the Kolmogorov-Smirnov test `kstest` and other specialised tests.

31.2 Simple summaries of neighbouring marks

We are often interested in the marks that are attached to the close neighbours of a typical point.

For a multitype point pattern, the function `marktable` compiles a contingency table of the marks of all points within a given radius of each data point:

```
> data(amacrine)
> M <- marktable(amacrine, R = 0.1)
> M[1:10, ]
```

```
      mark
point off on
  1     1  1
  2     2  2
  3     4  3
  4     3  1
  5     4  1
  6     2  3
  7     3  2
  8     1  1
  9     3  1
 10     3  2
```


More general summaries of the marks of neighbours can be obtained using the function `markstat`. For example, to compute the average diameter of the 5 closest neighbours of each tree in the Longleaf Pines dataset,

```
> md <- markstat(longleaf, mean, N = 5)
> md[1:10]

[1] 43.40 43.40 48.58 21.70 48.38 53.32 40.28 29.82 24.92 21.70
```

31.3 Distance methods and summary functions

If X and Y are two point pattern objects, then

- `crossdist(X,Y)` computes the matrix of distances from each point of X to each point of Y ;
- `nncross(X,Y)` finds, for each point in X , the nearest point of Y and the distance to this point.

The summary functions F , G , J and K (and other functions derived from K , such as L and the pair correlation function) have been extended to multitype point patterns, using such distances.

31.3.1 A pair of types

Assume the multitype point process \mathbf{X} is stationary. Let \mathbf{X}_j denote the sub-pattern of points of type j , with intensity λ_j . Then for any pair of types i and j ,

- $F_j(r)$ is the empty space function for \mathbf{X}_j .
- $G_{ij}(r)$ is the distribution function of the distance from a point of type i to the nearest point of type j
- $K_{ij}(r)$ is $1/\lambda_j$ times the expected number of points of type j within a distance r of a typical point of type i .
- $L_{ij}(r)$ is the corresponding L -function

$$L_{ij}(r) = \sqrt{\frac{K_{ij}(r)}{\pi}}.$$

- $g_{ij}(r)$ is the corresponding analogue of the pair correlation function

$$g_{ij}(r) = \frac{K'_{ij}(r)}{2\pi r}$$

where $K'_{ij}(r)$ is the derivative of K_{ij} .

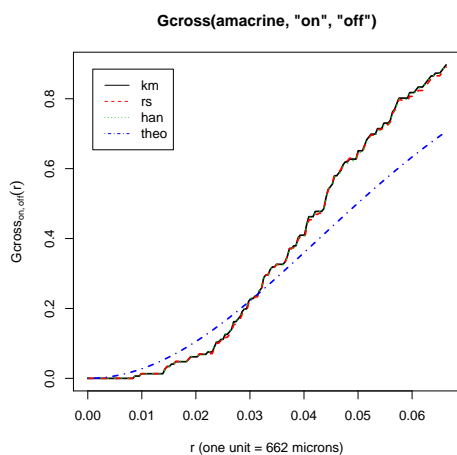
- J_{ij} is defined as

$$J_{ij}(r) = \frac{1 - G_{ij}(r)}{1 - F_j(r)}.$$

The functions G_{ij} , K_{ij} , L_{ij} , g_{ij} , J_{ij} are called “cross-type” or “ i -to- j ” summary functions. They are computed in `spatstat` by `Gcross`, `Kcross`, `Lcross`, `pcfcross` and `Jcross` respectively.

```
> data(amacrine)
> amacrine

> plot(Gcross(amacrine, "on", "off"))
```



The interpretation of the cross-type summary functions is similar, *but not identical*, to that of the original functions F , G , K etc:

- if \mathbf{X}_j is a uniform Poisson process (CSR), then $F_j(r) = 1 - \exp(-\lambda_j \pi r^2)$.
- if \mathbf{X}_j is a uniform Poisson process (CSR) *and is independent of* \mathbf{X}_i , then $G_{ij}(r) = 1 - \exp(-\lambda_j \pi r^2)$.
- if \mathbf{X}_i and \mathbf{X}_j are independent, then $K_{ij}(r) = \pi r^2$, $L_{ij}(r) = r$, $g_{ij}(r) = 1$, $G_{ij}(r) = F_{ij}(r)$ and $J_{ij}(r) = 1$.

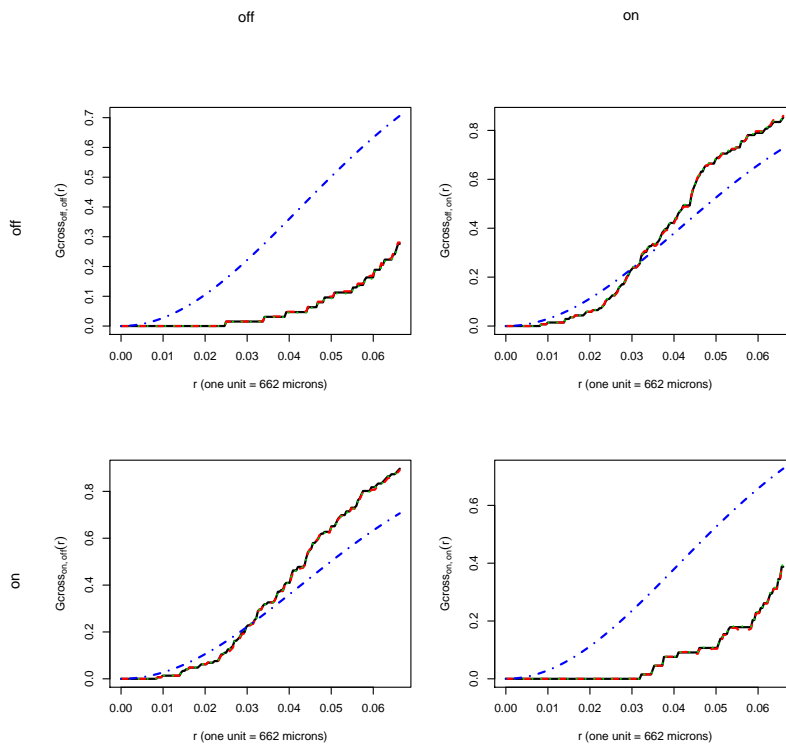
Here ‘independent’ means that the two point processes are probabilistically independent.

31.3.2 All pairs of types

The command `alltypes` enables the user to compute the cross-type summary functions between all pairs of types simultaneously. For example, to compute $G_{ij}(r)$ for all i and j in the amacrine cells data, we would use `alltypes(amacrine, "G")`. The result is automatically displayed as an array of plot panels.

```
> plot(alltypes(amacrine, "G"))
```

array of G functions for amacrine.



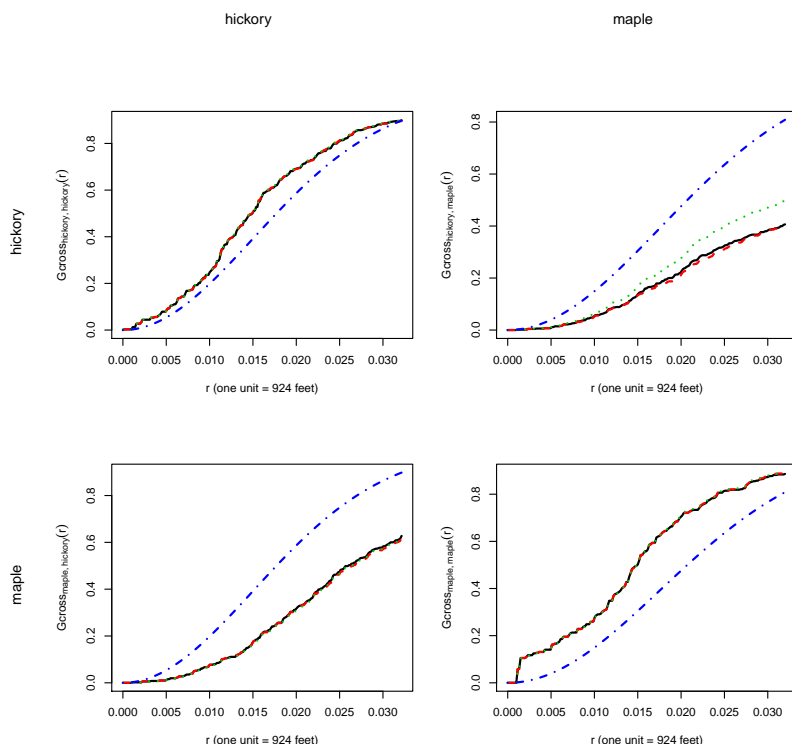
For example the top right panel shows the cumulative distribution function of the distance from an “off” cell to the nearest “on” cell.

The result of `alltypes` is a ‘function array’ (object of class “`fasp`”) which can be indexed by row and column subscripts. If the point pattern has a large number of possible types, you can compute the array of all possible pairwise G functions, then use the subscript operator to inspect a subset of the array.

```
> data(lansing)
> a <- alltypes(lansing, "G")
```

```
> plot(a[2:3, 2:3])
```

array of G functions for lansing.



31.3.3 One type to any type

Also defined are the “*i*-to-any” summaries

- $G_{i\bullet}(r)$, the distribution function of the distance from a point of type i to the nearest other point of any type;
- $K_{i\bullet}(r)$ is $1/\lambda$ times the expected number of points of any type within a distance r of a typical point of type i . Here $\lambda = \sum_j \lambda_j$ is the intensity of the entire process \mathbf{X} .
- $L_{i\bullet}(r)$ is the corresponding L -function

$$L_{i\bullet}(r) = \sqrt{\frac{K_{i\bullet}(r)}{\pi}}.$$

- $g_{i\bullet}(r)$ is the corresponding analogue of the pair correlation function

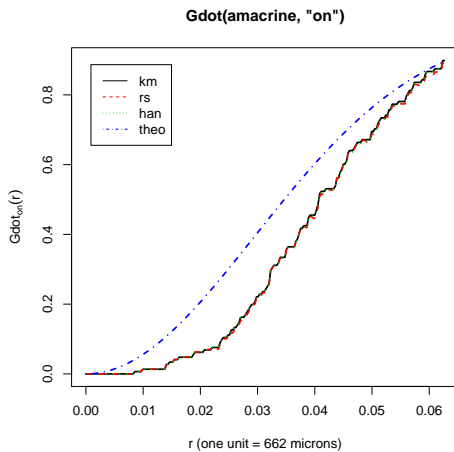
$$g_{i\bullet}(r) = \frac{K'_{i\bullet}(r)}{2\pi r}.$$

- $J_{i\bullet}$ is defined by

$$J_{i\bullet}(r) = \frac{1 - G_{i\bullet}(r)}{1 - F(r)}$$

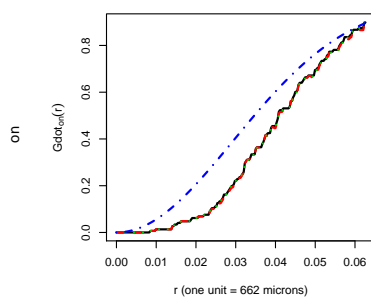
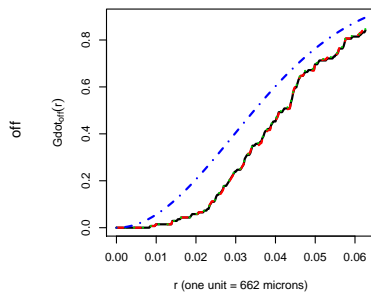
These are computed by `Gdot`, `Kdot`, `Ldot`, `pcfdot` and `Jdot` respectively, or using `alltypes`.

```
> plot(Gdot(amacrine, "on"))
```



```
> plot(alltypes(amacrine, "Gdot"))
```

array of Gdot functions for amacrine.



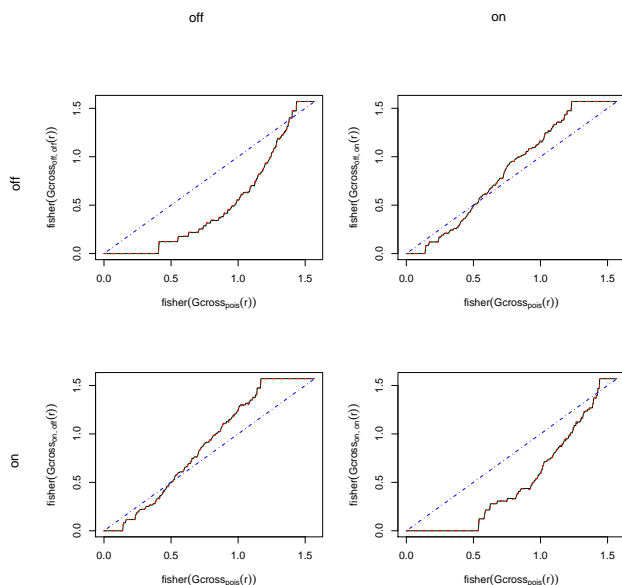
31.3.4 Plotting and manipulating function arrays

A function array (object of class "fasp") can be printed and plotted using methods for this class. It can also be manipulated in various ways.

The `plot` method is similar to `plot.fv` and allows the function values to be transformed:

```
> aG <- alltypes(amacrine, "G")
> fisher <- function(x) asin(sqrt(x))
> plot(aG, fisher(.) ~ fisher(theo))
```

array of G functions for amacrine.



As mentioned above, the function array can be indexed by array subscripts.

```
> data(lansing)
> a <- alltypes(lansing, "G")
> dim(a)
> b <- a[2:3, 2:3]
```

Calculations can be performed on all the functions in the array using `eval.fasp`.

```
> aGfish <- eval.fasp(asin(sqrt(aG)))
```

31.3.5 Mark connection function

The *mark connection function* between types i and j in a stationary multitype point process is

$$p_{ij}(r) = \frac{\lambda_i \lambda_j g_{ij}(r)}{\lambda^2 g(r)}.$$

This can be interpreted as the conditional probability, given that there is a point of the process at a location u and another point of the process at a location v separated by a distance $\|u - v\| = r$, that the first point is of type i and the second point is of type j .

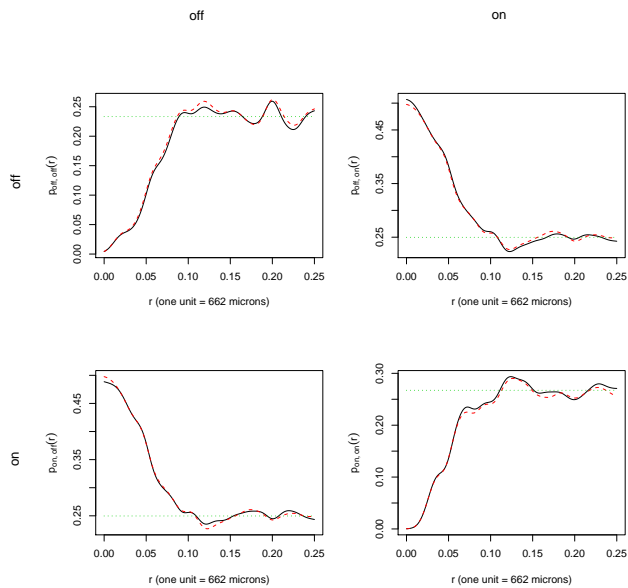
The command `markconnect` computes estimates of the mark connection function.

```
> data(amacrine)
> markconnect(amacrine, "on", "off")
```

We can use `alltypes` to compute the mark connection function p_{ij} for all pairs of types i and j :

```
> plot(alltypes(amacrine, markconnect))
```

array of markconnect functions for amacrine.



31.3.6 Mark equality function

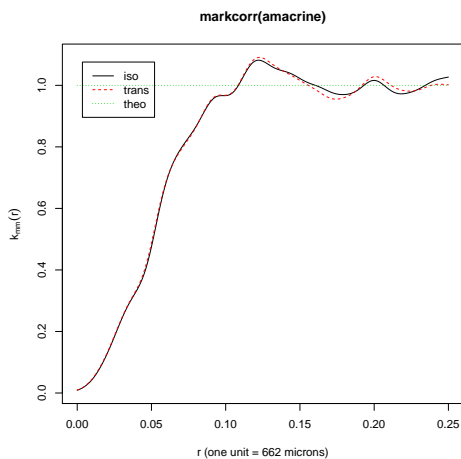
The composite

$$p(r) = \sum_i p_{ii}(r)$$

can be interpreted as the conditional probability, given that there is a point of the process at a location u and another point of the process at a location v separated by a distance $\|u - v\| = r$, that the two points have the *same* type.

This is sometimes called the *mark equality function*. It is a special case of a more general technique of “mark correlation” which we discuss in Section 32.2. To compute the mark equality function of a multitype point pattern, use `markcorr`.

```
> plot(markcorr(amacrine))
```



This plot indicates that nearby points tend to have different types.

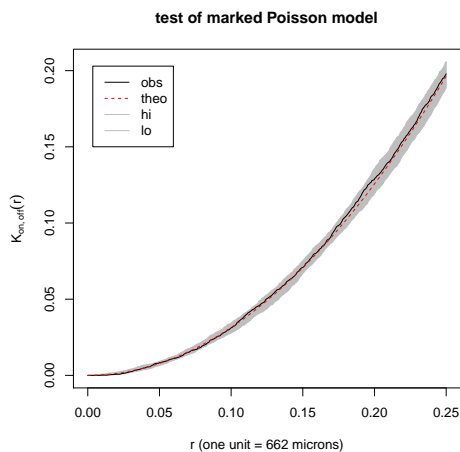
31.4 Randomisation tests

Simulation envelopes of summary functions can be used to test various null hypotheses for marked point patterns.

31.4.1 Poisson null

The null hypothesis of a homogeneous Poisson marked point process can be tested by direct simulation, using `envelope` as before. For example, using the cross-type K function as the test statistic,

```
> data(amacrine)
> E <- envelope(amacrine, Kcross, nsim = 39, i = "on", j = "off")
> plot(E, main = "test of marked Poisson model")
```



Notice that the arguments `i` and `j` here do not match any of the formal arguments of `envelope`, so they are passed to `Kcross`. This has the effect of calling `Kcross(X, i="on", j="off")` for each of the simulated point patterns X . Each simulated pattern is generated by the homogeneous Poisson point process with intensities estimated from the dataset `amacrine`.

31.4.2 Independence of components

It's also possible to test other null hypotheses by a randomisation test. We discussed two popular null hypotheses:

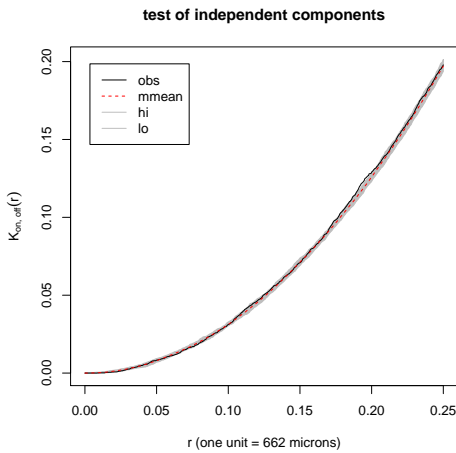
- *random labelling*: given the locations X , the marks are conditionally independent and identically distributed;
- *independence of components*: the sub-processes \mathbf{X}_m of points of each mark m , are independent point processes.

In a randomisation test of the independence-of-components hypothesis, the simulated patterns X are generated from the dataset by splitting the data into sub-patterns of points of one type, and randomly shifting these sub-patterns, independently of each other. The shifting is performed by `rshift`:

```
> E <- envelope(amacrine, Kcross, nsim = 39, i = "on", j = "off",
+   simulate = expression(rshift(amacrine, radius = 0.25)))
```



```
> plot(E, main = "test of independent components")
```



The independence-of-components hypothesis seems to be accepted in this example. Under the independence hypothesis,

$$\begin{aligned} K_{ij}(r) &= \pi r^2 \\ G_{ij}(r) &= F_j(r) \\ J_{ij}(r) &\equiv 1. \end{aligned}$$

while the “*i*-to-any” functions have complicated values. Thus, we would normally use K_{ij} or J_{ij} to construct a test statistic for independence of components.

31.4.3 Random labelling

In a randomisation test of the random labelling null hypothesis, the simulated patterns \mathbf{X} are generated from the dataset by holding the point locations fixed, and randomly resampling the marks, either with replacement (independent random sampling) or without replacement (randomly permuting the marks). The resampling operation is performed by `rlabel`.

Under random labelling,

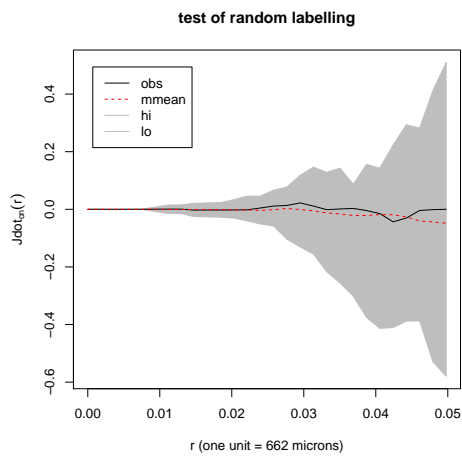
$$\begin{aligned} J_{i\bullet}(r) &= J(r) \\ K_{i\bullet}(r) &= K(r) \\ G_{i\bullet}(r) &= G(r) \end{aligned}$$

(where G, K, J are the summary functions for the point process without marks) while the other, cross-type functions have complicated values. Thus, we would normally use something like $K_{i\bullet}(r) - K(r)$ to construct a test statistic for random labelling.

To do this, cook up a little function to evaluate $J_{i\bullet}(r) - J(r)$:

```
> Jdif <- function(X, ..., i) {
+   Jidot <- Jdot(X, ..., i = i)
+   J <- Jest(X, ...)
+   dif <- eval.fv(Jidot - J)
+   return(dif)
+ }
> E <- envelope(amacrine, Jdif, nsim = 39, i = "on", simulate = expression(rlabel(amacrine)))
```

```
> plot(E, main = "test of random labelling")
```



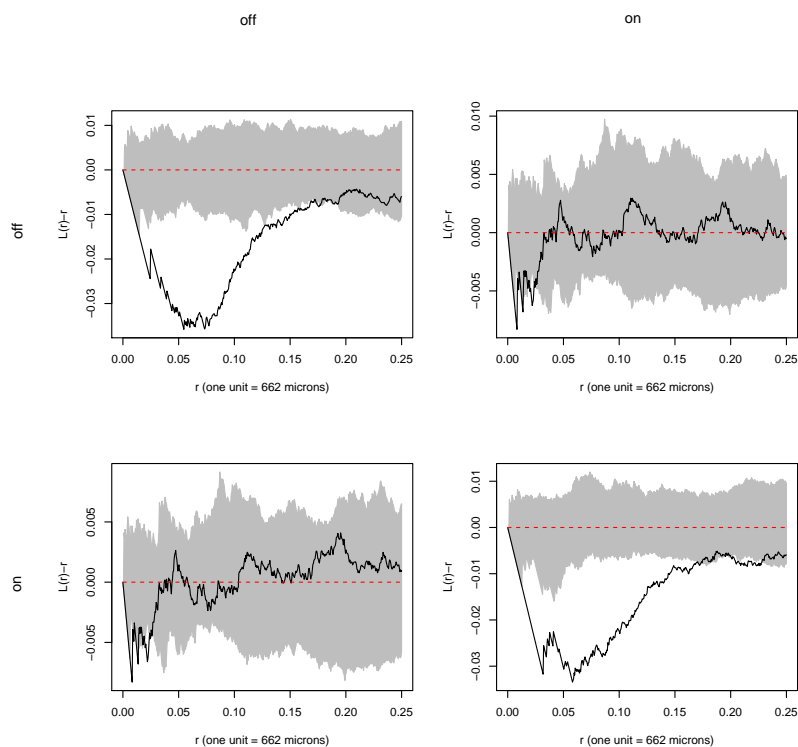
The random labelling hypothesis also seems to be accepted.

31.4.4 Arrays of envelopes

To compute a simulation envelope for the function K_{ij} for each pair of types i and j , use `alltypes` with the argument `envelope=TRUE`.

```
> aE <- alltypes(amacrine, Kcross, nsim = 39, envelope = TRUE)
> plot(aE, sqrt(./pi) - r ~ r, ylab = "L(r)-r")
```

array of envelopes of Kcross functions for amacrine.

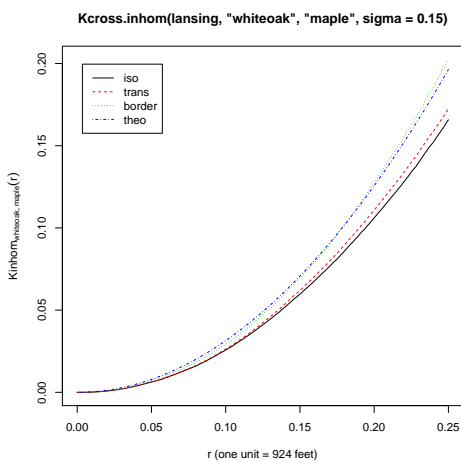


31.5 Adjusting for inhomogeneity

Inhomogeneous versions of the multitype K function, L function and pair correlation function also exist.

The command names are `Kdot.inhom`, `Kcross.inhom`, `Ldot.inhom`, `Lcross.inhom`, `pcfdot.inhom` and `pcfcross.inhom`. They require separate estimates of the intensities of the first and second types of points. Again these intensities can be estimated by kernel smoothing.

```
> data(lansing)
> plot(Kcross.inhom(lansing, "whiteoak", "maple", sigma = 0.15))
```

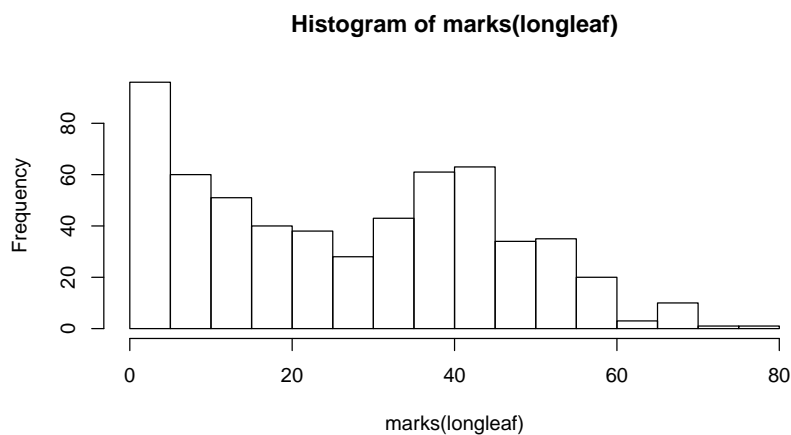


32 Exploratory tools for marked point patterns

32.1 Numeric marks: distribution and trend

For a point pattern with marks that are numeric (real numbers or integers) or logical values, the mark values can be extracted using the `marks` function and inspected using the histogram or kernel density estimate:

```
> data(longleaf)
> hist(marks(longleaf))
```



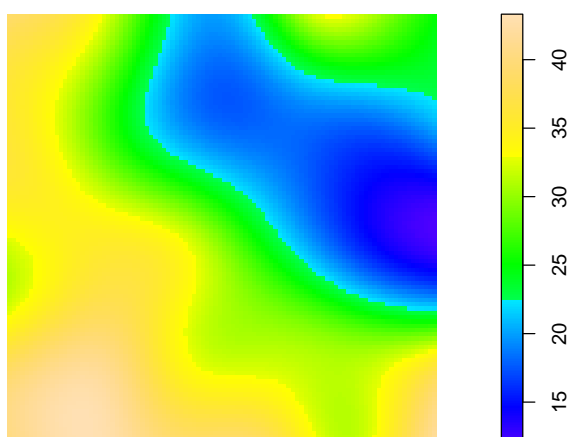
To assess spatial trend in the marks, one way is to form a kernel regression smoother. The smoothed mark value at location $u \in \mathbb{R}^2$ is

$$\hat{m}(u) = \frac{\sum_i m_i \kappa(u - x_i)}{\sum_i \kappa(u - x_i)}$$

where k is the smoothing kernel, and m_i is the mark value at data point x_i . This is computed by `smooth.ppp`:

```
> plot(smooth.ppp(longleaf))
```

smooth.ppp(longleaf)

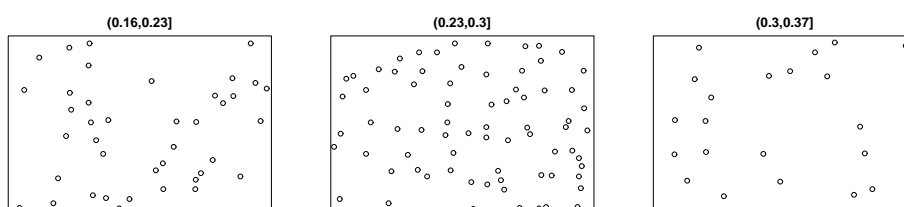


The plot shows that there is a region of younger trees in the northeast of the study region. If the marks are a data frame, the result of `smooth.ppp` will be a list of pixel images, one for each mark variable.

You can also use `cut.ppp` followed by `split.ppp` to look for spatial inhomogeneity of the marks:

```
> data(spruces)
> plot(split(cut(spruces, breaks = 3)))
```

split(cut(spruces, breaks = 3))



Other facilities include `markvar` which calculates a smoothed estimate of the local variance of the mark values.

32.2 Mark correlation function

The “mark correlation function” $\rho_f(r)$ of a stationary marked point process \mathbf{Y} is a measure of the dependence between the marks of two points of the process a distance r apart [61]. It is informally defined as

$$\rho_f(r) = \frac{\mathbb{E}[f(M_1, M_2)]}{\mathbb{E}[f(M, M')]}$$

where M_1, M_2 are the marks attached to two points of the process separated by a distance r , while M, M' are independent realisations of the marginal distribution of marks.

Here f is any function $f(m_1, m_2)$ with two arguments which are possible marks of the pattern, and which returns a nonnegative real value. Common choices of f are:

- for continuous real-valued marks, $f(m_1, m_2) = m_1 m_2$;
- for categorical marks (multitype point patterns), $f(m_1, m_2) = \mathbf{1}\{m_1 = m_2\}$;
- for marks taking values in $[0, 2\pi]$, $f(m_1, m_2) = \sin(m_1 - m_2)$.

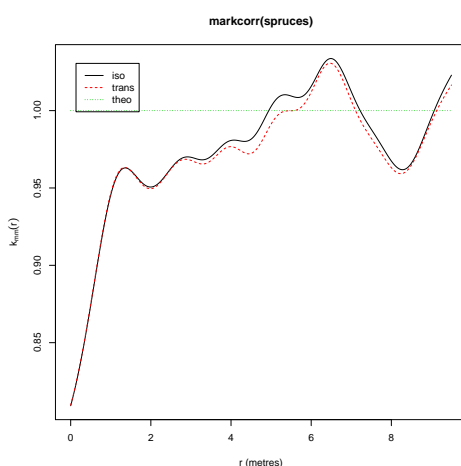
Note that $\rho_f(r)$ is not a “correlation” in the usual statistical sense. It can take any nonnegative real value. The value 1 suggests “lack of correlation”: under random labelling, $\rho_f(r) \equiv 1$. The interpretation of values larger or smaller than 1 depends on the choice of function f .

The mark correlation function is computed in `spatstat` by `markcorr`. It has the syntax

```
> markcorr(X, f)
```

where `X` is a point pattern and `f` is an R language function.

```
> data(spruces)
> plot(markcorr(spruces))
```



The plot suggests a slight negative association between the sizes of nearby trees. This is somewhat hard to interpret.

The cumulative version of the mark correlation is computed by `markcorrint`.

A concept closely related to the mark correlation, based on the nearest neighbour of each point rather than all neighbours at a given distance, is computed by `nncorr`.

32.3 Reverse conditional moments

Schlather et al [59] defined the functions $E(r)$ and $V(r)$ to be the conditional mean and conditional variance of the mark attached to a typical random point, given that there exists another random point at a distance r away from it. More formally,

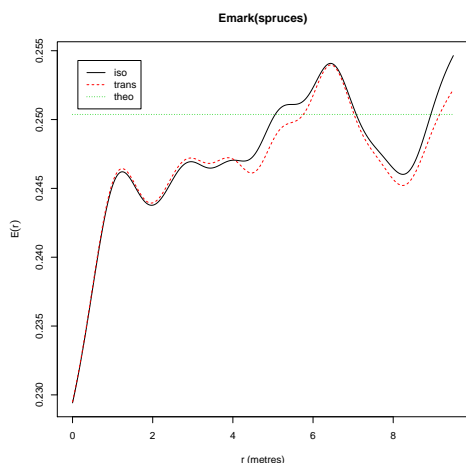
$$\begin{aligned} E(r) &= E_{0u}[M(0)] \\ V(r) &= E_{0u}[(M(0) - E(u))^2] \end{aligned}$$

where E_{0u} denotes the conditional expectation given that there are points of the process at the locations 0 and u separated by a distance r , and where $M(0)$ denotes the mark attached to the point 0.

These functions may serve as diagnostics for dependence between the points and the marks. If the points and marks are independent, then $E(r)$ and $V(r)$ should be constant (not depending on r).

These functions are computed using `Emark` and `Vmark`.

```
> plot(Emark(spruces))
```



Looking at the y axis scale we see that a *slight* drop in mean value (by about 10%) occurs near the origin. This plot suggests that a tree with a very close neighbour tends to have a diameter slightly smaller than average. The function `Emark` is simpler to interpret than the mark correlation function.

33 Multitype Poisson models

This section covers multitype Poisson process models: basic properties, simulation, and fitting models to data.

33.1 Theory

33.1.1 Complete spatial randomness and independence

A *uniform Poisson marked point process* in \mathbb{R}^2 with marks in \mathcal{M} can be defined in the following equivalent ways.

- randomly marked Poisson process (Poisson $[X]$, iid $[M|X]$): a Poisson point process of locations \mathbf{X} with intensity β is first generated. Then each point x_i is labelled with a random mark m_i , independently of other points, with distribution $\mathbb{P}\{M_i = m\} = p_m$ for $m \in \mathcal{M}$.
- superposition of independent Poisson processes (iid $[M]$, Poisson $[X|M]$): for each possible mark $m \in \mathcal{M}$, a Poisson process \mathbf{X}_m is generated, with intensity β_m . The points of \mathbf{X}_m are tagged with the mark m . Then the processes \mathbf{X}_m with different marks $m \in \mathcal{M}$ are superimposed, to yield a marked point process.
- Poisson marked point process (jointly Poisson $[X, M]$): a Poisson process on $\mathbb{R}^2 \times \mathcal{M}$ is generated, with intensity function $\lambda(u, m) = \beta_m$ at location u and mark m .

These constructions are *equivalent* when $\beta_m = p_m\beta$. See the lovely book by Kingman [45].

Since the established term CSR ('complete spatial randomness') is used to refer to the uniform Poisson point process, I propose that the uniform *marked* Poisson point process should be called 'complete spatial randomness and independence' (CSRI).

33.1.2 Inhomogeneous Poisson marked point processes

A *inhomogeneous* Poisson marked point process \mathbf{Y} with 'joint' intensity $\lambda(u, m)$ for locations u and mark values m is simply defined as an inhomogeneous Poisson point process on $\mathbb{R}^2 \times \mathcal{M}$ with intensity function $\lambda(u, m)$.

Let's restrict attention to the case of categorical marks, where \mathcal{M} is finite. Then the process \mathbf{Y} has the following properties:

- The locations \mathbf{X} , obtained by removing the marks, constitute an inhomogeneous Poisson process in \mathbb{R}^2 with intensity function

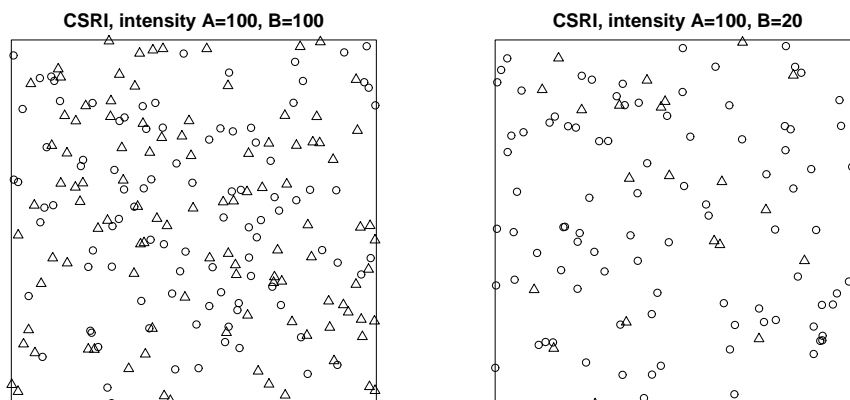
$$\beta(u) = \sum_m \lambda(u, m).$$

- Conditional on the locations \mathbf{X} , the marks attached to the points are independent. For a point x_i the conditional distribution of the mark m_i is $\mathbb{P}\{M_i = m\} = \lambda(x_i, m)/\beta(x_i)$.
- The sub-process \mathbf{X}_m of points with mark m , is an inhomogeneous Poisson point process with intensity $\beta_m(u) = \lambda(u, m)$.
- The sub-processes \mathbf{X}_m of points with different marks m are independent processes.

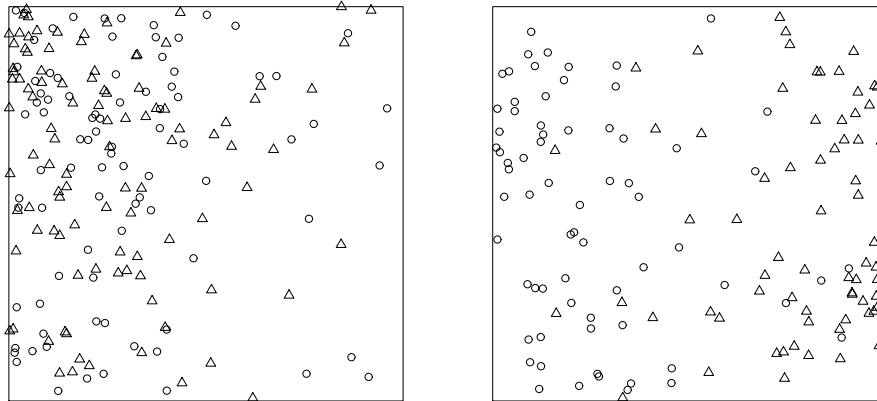
33.2 Simulation

Realisations of Poisson marked point processes can be generated by 'hand', using `rmppoispp`. The first argument of this command specifies the intensity or intensity function $\lambda(u, m)$. It can be a constant, a vector of constants, or an R function.

```
> par(mfrow = c(1, 2))
> Xunif <- rmppoispp(100, types = c("A", "B"), win = square(1))
> plot(Xunif, main = "CSRI, intensity A=100, B=100")
> Xunif <- rmppoispp(c(100, 20), types = c("A", "B"), win = square(1))
> plot(Xunif, main = "CSRI, intensity A=100, B=20")
> par(mfrow = c(1, 1))
```



```
> X1 <- rmppoispp(function(x, y, m) {
+   300 * exp(-3 * x)
+ }, types = c("A", "B"))
> lamb <- function(x, y, m) {
+   ifelse(m == "A", 300 * exp(-4 * x), 300 * exp(-4 * (1 - x)))
+ }
> X2 <- rmppoispp(lamb, types = c("A", "B"))
> par(mfrow = c(1, 2))
> plot(X1, main = "")
> plot(X2, main = "")
> par(mfrow = c(1, 1))
```

33.3 Fitting Poisson models

Poisson marked point process models may be fitted to point pattern data using ppm. Currently the methods are only available for multitype point processes (categorical marks).

33.3.1 Probability densities

Let $W \subset \mathbb{R}^2$ be the study region, and \mathcal{M} the (finite) set of possible marks. Then a marked point pattern is a set

$$\mathbf{y} = \{(x_1, m_1), \dots, (x_n, m_n)\}, \quad x_i \in W, \quad m_i \in \mathcal{M}, \quad n \geq 0$$

of pairs (x_i, m_i) of locations x_i with marks m_i . It can be viewed as a point pattern in the Cartesian product $W \times \mathcal{M}$.

The probability density of a marked point process is a function $f(\mathbf{y})$ defined for all marked point patterns \mathbf{y} including the empty pattern \emptyset .

The process with probability density $f(\mathbf{y}) \equiv 1$ is the uniform Poisson marked point process with intensity 1 **for each mark**. That is, for this model, the sub-process of points with mark $m_i = m$ is a uniform Poisson process with intensity 1. If the marks are removed, we obtain a Poisson point process with intensity equal to $|\mathcal{M}|$, the number of possible types.

The uniform Poisson marked point process with intensity $\lambda(u, m) = \beta_m$ has probability density

$$\begin{aligned} f(\mathbf{y}) &= \exp\left(\sum_{m \in \mathcal{M}} (1 - \beta_m)|W|\right) \prod_{i=1}^{n(\mathbf{y})} \beta_{m_i} \\ &= \exp\left(\sum_{m \in \mathcal{M}} (1 - \beta_m)|W|\right) \prod_{m \in \mathcal{M}} \beta_m^{n_m(\mathbf{y})} \end{aligned}$$

where $n_m(\mathbf{y})$ is the number of points in \mathbf{y} having mark value m .

The inhomogeneous Poisson marked point process with intensity function $\lambda(u, m)$, at location $u \in W$ and mark $m \in \mathcal{M}$, has probability density

$$f(\mathbf{y}) = \exp\left(\sum_{m \in \mathcal{M}} \int_W (1 - \lambda(u, m)) du\right) \prod_{i=1}^{n(\mathbf{y})} \lambda(x_i, m_i). \quad (48)$$

33.3.2 Maximum likelihood

For the multitype Poisson process with intensity function $\lambda(u, m)$ at location $u \in W$ and mark $m \in \mathcal{M}$, the loglikelihood is, up to a constant,

$$\log L = \sum_{i=1}^n \log \lambda(x_i, m_i) - \sum_{m \in \mathcal{M}} \int_W \lambda(u, m) \, du. \quad (49)$$

where m_i is the mark attached to data point x_i . This is formally equivalent to the loglikelihood of a Poisson loglinear regression, so the Berman-Turner algorithm can again be used to maximise the loglikelihood.

33.3.3 Model-fitting in spatstat

Poisson marked point process models are fitted to data using `ppm`.

The trend formula in the call to `ppm` may involve the reserved name `marks` as a variable. This refers to the marks of the points. Since the marks are categorical, `marks` is treated as a factor variable for modelling purposes.

To fit the homogeneous multitype Poisson process (CSRI), equation (50), we call

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

The formula `~marks` indicates that the trend depends only on the marks, and not on spatial location; since `marks` is a factor, the trend has a separate constant value for each level of `marks`. This is the model (50).

Note that if we had typed

```
> ppm(X, ~1)
```

this would have fitted the special case of CSRI where the intensities β_m are equal, $\beta_m \equiv \alpha$ say, for all possible marks. That model is only appropriate if we believe that all mark values are equally likely.

For the Lansing Woods data, the minimal model that makes sense is (50), so we call

```
> ppm(lansing, ~marks)
```

Stationary multitype Poisson process

Possible marks:

```
blackoak hickory maple misc redoak whiteoak
```

Trend formula: ~marks

Intensities:

beta_blackoak	beta_hickory	beta_maple	beta_misc	beta_redoak
135	703	514	105	346
beta_whiteoak				
448				

Since `lansing` is a multitype point pattern (its marks are categorical), the variable `marks` in the formula is a factor. The model has one parameter/coefficient for each level of the factor, i.e. one coefficient for each type of point. In other words, this is the homogeneous Poisson marked point process with intensity β_m for points of mark m .

You'll notice that the parameter estimates $\hat{\beta}_m$ coincide with those obtained from `summary.ppp` above. That is a consequence of the fact that the maximum likelihood estimates (obtained by `ppm`) are also the method-of-moments estimates (obtained by `summary.ppp`).

A more complicated example is

```
> ppm(lansing, ~marks + x)
```

```
Nonstationary multitype Poisson process
```

```
Possible marks:
```

```
blackoak hickory maple misc redoak whiteoak
```

```
Trend formula: ~marks + x
```

```
Fitted coefficients for trend formula:
```

(Intercept)	markshickory	marksmaple	marksmisc	marksredoak
4.94294727	1.65008211	1.33694849	-0.25131442	0.94116400
markswiteoak	x			
1.19951845	-0.07581624			

This is the marked Poisson process whose intensity function $\lambda((x, y, m))$ at location (x, y) and mark m satisfies

$$\log \lambda((x, y, m)) = \alpha_m + \beta x$$

where $\alpha_1, \dots, \alpha_6$ and β are parameters. The intensity is loglinear in x , with a different intercept for each mark, but the same slope ("parallel loglinear regression"). In the printout above, the fitted slope parameter β is $\hat{\beta} = -0.07581624$. As discussed in Section 15.3 on page 98, the fitted coefficients α_m for the categorical mark are interpreted in the light of the 'contrasts' in force. The default is the treatment contrasts, and the first level of the mark is **blackoak**, so in this case the fitted coefficient for $m=\text{blackoak}$ is 4.942947, while the fitted coefficient for $m=\text{hickory}$ is $4.942947 + 1.650082 = 6.593029$ and so on.

```
> ppm(lansing, ~marks * x)
```

```
Nonstationary multitype Poisson process
```

```
Possible marks:
```

```
blackoak hickory maple misc redoak whiteoak
```

```
Trend formula: ~marks * x
```

```
Fitted coefficients for trend formula:
```

(Intercept)	markshickory	marksmaple	marksmisc	marksredoak
5.2378062	1.4424915	0.6795604	-0.8482907	0.6916392
markswiteoak	x	markshickory:x	marksmaple:x	marksmisc:x
1.0901772	-0.7063987	0.4511157	1.3243326	1.2138278
marksredoak:x	markswiteoak:x			
0.5380413	0.2421379			

The symbol `*` here is an 'interaction' in the usual sense for linear models. The fitted model is the marked Poisson process with

$$\log \lambda((x, y, m)) = \alpha_m + \beta_m x$$

where $\alpha_1, \dots, \alpha_6$ and β_1, \dots, β_6 are parameters. The intensity is loglinear in x with a different slope and intercept for each mark.

The result of `ppm` is again an object of class "ppm" representing a fitted point process model. To plot the fitted intensity and conditional intensity of the fitted model, use `plot.ppm`. For a multitype point process you will get a separate plot for each possible mark value.

More complicated examples are:

```
> ppm(lansing, ~marks * polynom(x, y, 2))
> ppm(lansing, ~marks * harmonic(x, y, 2))
```

33.3.4 Facilities available

A fitted multitype Poisson process model can be manipulated using any of the methods available for the class `ppm`:

<code>print</code>	print basic information
<code>summary</code>	print detailed summary information
<code>plot</code>	plot the fitted (conditional) intensity
<code>predict</code>	fitted (conditional) intensity
<code>fitted</code>	fitted (conditional) intensity at data points
<code>update</code>	re-fit the model
<code>coef</code>	extract the fitted coefficient vector $\hat{\theta}$
<code>vcov</code>	variance-covariance matrix of $\hat{\theta}$
<code>anova</code>	analysis of deviance
<code>logLik</code>	evaluate log-pseudolikelihood
<code>model.matrix</code>	extract design matrix
<code>formula</code>	extract trend formula of model
<code>terms</code>	extract terms in model formula

The following functions are also available:

<code>step</code>	stepwise model selection
<code>drop1</code>	one step backward in model selection
<code>model.images</code>	compute images of canonical covariates in model
<code>effectfun</code>	fitted intensity as function of one covariate

A fitted multitype Poisson process model can be simulated automatically using `rmh.ppm` or `simulate.ppm`.

34 Gibbs models for multitype point patterns

Gibbs point process models (section 26) are also available for marked point processes, and can be fitted to data using `ppm`. Currently the methods are only implemented for *multitype* point processes (categorical marks), so we restrict attention to this case.

34.1 Gibbs models

Much of the theory of Gibbs models described in Section 26 carries over immediately to *multitype* point processes.

34.1.1 Conditional intensity

The conditional intensity $\lambda(u, \mathbf{X})$ of an (unmarked) point process \mathbf{X} at a location u was defined in section 26.5. Roughly speaking $\lambda(u, \mathbf{x}) du$ is the conditional probability of finding a point near u , given that the rest of the point process \mathbf{X} coincides with \mathbf{x} .

For a marked point process \mathbf{Y} the conditional intensity is a function $\lambda((u, m), \mathbf{Y})$ giving a value at a location u for each possible mark m . For a *finite* set of marks M , we can interpret $\lambda((u, m), \mathbf{y}) du$ as the conditional probability finding a point *with mark* m near u , given the rest of the marked point process.

The conditional intensity is related to the probability density $f(\mathbf{y})$ by

$$\lambda((u, m), \mathbf{y}) = \frac{f(\mathbf{y} \cup \{u\})}{f(\mathbf{y})}$$

for $(u, m) \notin \mathbf{y}$.

For Poisson processes, the conditional intensity $\lambda((u, m), \mathbf{y})$ coincides with the intensity function $\lambda(u, m)$ and does not depend on the configuration \mathbf{y} . For example, the homogeneous Poisson multitype point process or ‘‘CSRI’’ (Section 33.1.1) has conditional intensity

$$\lambda((u, m), \mathbf{y}) = \beta_m \tag{50}$$

where $\beta_m \geq 0$ are constants which can be interpreted in several equivalent ways (section 26.5). The sub-process consisting of points of type m *only* is Poisson with intensity β_m . The process obtained by ignoring the types, and combining all the points, is Poisson with intensity $\beta = \sum_m \beta_m$. The marks attached to the points are i.i.d. with distribution $p_m = \beta_m/\beta$.

34.1.2 Pairwise interactions

A *multitype* pairwise interaction process is a Gibbs process with probability density of the form

$$f(\mathbf{y}) = \alpha \left[\prod_{i=1}^{n(\mathbf{y})} b_{m_i}(x_i) \right] \left[\prod_{i < j} c_{m_i, m_j}(x_i, x_j) \right] \tag{51}$$

where $b_m(u), m \in \mathcal{M}$ are functions determining the ‘first order trend’ for points of each type, and $c_{m, m'}(u, v), m, m' \in \mathcal{M}$ are functions determining the interaction between a pair of points of given types m and m' . The interaction functions must be symmetric, $c_{m, m'}(u, v) = c_{m, m'}(v, u)$ and $c_{m, m'} \equiv c_{m', m}$. The conditional intensity is

$$\lambda((u, m); \mathbf{y}) = b_m(u) \prod_{i=1}^{n(\mathbf{y})} c_{m, m_i}(u, x_i). \tag{52}$$

34.1.3 Pairwise interactions not depending on marks

The simplest examples of multitype pairwise interaction processes are those in which the interaction term $c_{m,m'}(u, v)$ does not depend on the marks m, m' . For example, we can take any of the interaction functions $c(u, v)$ described in section 26.3 and use it to construct a marked point process.

Such processes can be constructed equivalently as follows [14]:

- an *unmarked* Gibbs process is generated with first order term $b(u) = \sum_{m \in \mathcal{M}} b_m(u)$ and pairwise interaction $c(u, v)$.
- each point x_i of this unmarked process is labelled with a mark m_i with probability distribution $\mathbb{P}\{m_i = m\} = b_i(x_i)/b(x_i)$ independent of other points.

If additionally the intensity functions are constant, $b_m(u) \equiv \beta_m$, then such a point process has the random labelling property.

34.1.4 Mark-dependent pairwise interactions

Various complex kinds of behaviour can be created by postulating a pairwise interaction that does depend on the marks.

A simple example is the *multitype hard core process* in which $\beta_m(u) \equiv \beta$ and

$$c_{m,m'}(u, v) = \begin{cases} 1 & \text{if } \|u - v\| > r_{m,m'} \\ 0 & \text{if } \|u - v\| \leq r_{m,m'} \end{cases} \quad (53)$$

where $r_{m,m'} = r_{m',m} > 0$ is the hard core distance for type m with type m' . In this process, two points of type m and m' respectively can never come closer than the distance $r_{m,m'}$.

By setting $r_{m,m'} = 0$ for a particular pair of marks m, m' we effectively remove the interaction term between points of these types. If there are only two types, say $\mathcal{M} = \{1, 2\}$, then setting $r_{1,2} = 0$ implies that the sub-processes \mathbf{X}_1 and \mathbf{X}_2 , consisting of points of types 1 and 2 respectively, are independent point processes. In other words the process satisfies the independence-of-components property.

The *multitype Strauss process* has pairwise interaction term

$$c_{m,m'}(u, v) = \begin{cases} 1 & \text{if } \|u - v\| > r_{m,m'} \\ \gamma_{m,m'} & \text{if } \|u - v\| \leq r_{m,m'} \end{cases} \quad (54)$$

where $r_{m,m'} > 0$ are interaction radii as above, and $\gamma_{m,m'} \geq 0$ are interaction parameters.

In contrast to the unmarked Strauss process, which is well-defined only when its interaction parameter γ is between 0 and 1, the multitype Strauss process allows some of the interaction parameters $\gamma_{m,m'}$ to exceed 1 for $m \neq m'$, provided one of the relevant types has a hard core ($\gamma_{m,m} = 0$ or $\gamma_{m',m'} = 0$).

If there are only two types, say $\mathcal{M} = \{1, 2\}$, then setting $\gamma_{1,2} = 1$ implies that the sub-processes \mathbf{X}_1 and \mathbf{X}_2 , consisting of points of types 1 and 2 respectively, are independent Strauss processes.

The *multitype Strauss-hard core process* has pairwise interaction term

$$c_{m,m'}(u, v) = \begin{cases} 0 & \text{if } \|u - v\| < h_{m,m'} \\ \gamma_{m,m'} & \text{if } h_{m,m'} \leq \|u - v\| \leq r_{m,m'} \\ 1 & \text{if } \|u - v\| > r_{m,m'} \end{cases} \quad (55)$$

where $r_{m,m'} > 0$ are interaction distances and $\gamma_{m,m'} \geq 0$ are interaction parameters as above, and $h_{m,m'}$ are hard core distances satisfying $h_{m,m'} = h_{m',m}$ and $0 < h_{m,m'} < r_{m,m'}$.

34.2 Pseudolikelihood for multitype Gibbs processes

Models can be fitted by maximum pseudolikelihood. For a multitype Gibbs point process with conditional intensity $\lambda((u, m); \mathbf{y})$, the log pseudolikelihood is

$$\log \text{PL} = \sum_{i=1}^{n(\mathbf{y})} \log \lambda((x_i, m_i); \mathbf{y}) - \sum_{m \in \mathcal{M}} \int_W \lambda((u, m); \mathbf{y}) \, du. \quad (56)$$

The pseudolikelihood can be maximised using an extension of the Berman-Turner device [9].

34.3 Fitting Gibbs models to multitype data

Marked point process models may be fitted to point pattern data using `ppm`. Currently the methods are only available for multitype point processes (categorical marks).

34.3.1 Interactions not depending on marks

The model-fitting function `ppm` expects an argument `interaction` that specifies the interpoint interaction structure of the point process. The default is ‘no interaction’, corresponding to a Poisson process.

On page 165 there is a list of interpoint interactions for modelling *unmarked* point patterns. These interactions can also be used, without modification, to fit models to *multitype* point patterns.

For example

```
> ppm(lansing, ~marks, Strauss(0.07))
```

fits a multitype version of the Strauss process (section 26.3.2) in which the conditional intensity is

$$\lambda((u, m), \mathbf{y}) = \beta_m \gamma^{t(u, \mathbf{y})}. \quad (57)$$

Here β_m are constants which account for the unequal abundance of the different species of tree. The other quantities are the same as in (42). The interaction between two trees is assumed to be the same for all species, and is controlled by the interaction parameter γ and interaction radius $r = 0.07$. For example, this includes the case $\gamma = 0$ where no two trees (whatever species they belong to) come closer than 0.07 units apart, a ‘multitype hard core process’.

34.3.2 Interactions depending on marks

There are two additional interpoint interactions defined in `spatstat` for multitype point patterns:

`MultiStrauss` the multitype Strauss process
`MultiStraussHard` multitype hybrid hard core / Strauss process

In these models, the interaction between two points depends on the types of the points as well as their separation.

In the multitype Strauss process (54), for each pair of types i and j there is an interaction radius r_{ij} and interaction parameter γ_{ij} . In simple terms, each pair of points, with marks i and j say, contributes an interaction term $\gamma_{i,j}$ if the distance between them is less than the interaction distance $r_{i,j}$. These parameters must satisfy $r_{ij} = r_{ji}$ and $\gamma_{ij} = \gamma_{ji}$. The conditional intensity is

$$\lambda((u, i), \mathbf{y}) = \beta_i \prod_j \gamma_{i,j}^{t_{i,j}(u, \mathbf{y})} \quad (58)$$

where $t_{i,j}(u, \mathbf{y})$ is the number of points in \mathbf{y} , with mark equal to j , lying within a distance $r_{i,j}$ of the location u .

To fit the stationary multitype Strauss process to the dataset `betacells`, we must specify the matrix of interaction radii r_{ij} :

```
> data(betacells)
> r <- 30 * matrix(c(1, 2, 2, 1), nrow = 2, ncol = 2)
> ppm(betacells, ~1, MultiStrauss(c("off", "on"), r), rbord = 60)
```

Stationary Multitype Strauss process

Possible marks:

off on

First order terms:

```
      beta_off      beta_on
0.0001373652 0.0001373652
```

Interaction: Pairwise interaction family

Interaction: Multitype Strauss process

2 types of points

Possible types:

```
[1] "off" "on"
```

Interaction radii:

```
      off on
off 30 60
on  60 30
```

Fitted interaction parameters γ_{ij} :

```
      off      on
off 0.0000 0.8303
on  0.8303 0.0000
```

Relevant coefficients:

```
markoffxoff markoffxon markonxon
-17.2378706 -0.1860184 -17.2138383
```

To fit a nonstationary multitype Strauss process with log-cubic polynomial trend:

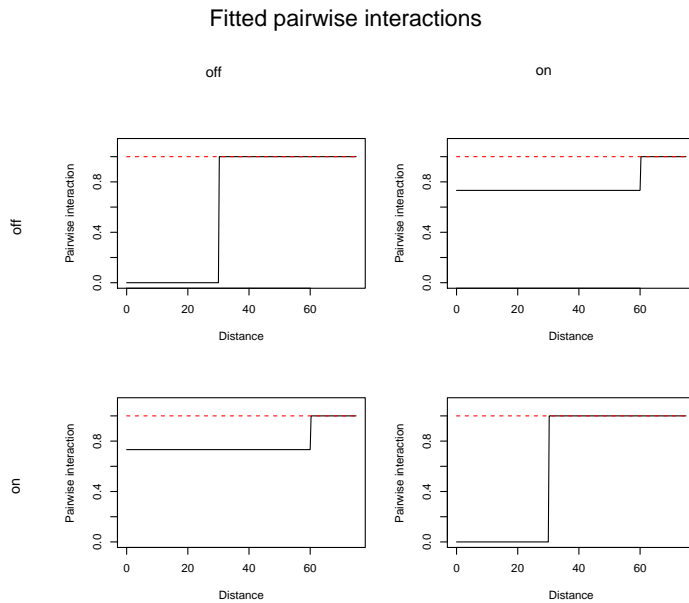
```
> ppm(betacells, ~polynom(x, y, 3), MultiStrauss(c("off", "on"),
+      r), rbord = 60)
```

For more detailed explanation and examples of modelling and the interpretation of model formulae for point processes, see [11].

34.3.3 Plotting the fitted interaction

The fitted pairwise interaction in a point process model can be plotted using `fitin`. The value returned by `fitin` is a function array (class "fasp").

```
> model <- ppm(betacells, ~polynom(x, y, 3), MultiStrauss(c("off",
+ "on"), r), rbord = 60)
> plot(fitin(model))
```



34.3.4 Simulating a Gibbs model

A fitted Gibbs point process model can be simulated using `rmh.ppm` or `simulate.ppm`.

```
> rmh(model, verbose = FALSE)
```

```
marked planar point pattern: 139 points
multitype, with levels = off      on
window: rectangle = [28.08, 778.08] x [16.2, 1007.02] microns
```

It's also possible to simulate any specified Gibbs model using `rmh.default` with the model specified 'by hand' using `rmhmodel.default`.

PART VIII. HIGHER DIMENSIONS AND OTHER SPATIAL DATA

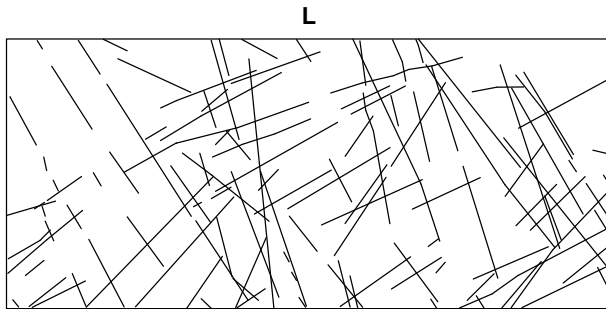
Part VIII of the workshop deals with point patterns in 3D, space-time, and multi-dimensional space time. It also covers line segment patterns and stochastic geometry techniques.

35 Line segment data

`spatstat` also has some facilities for handling spatial patterns of *line segments*.

For example, the `copper` dataset in `spatstat` contains a dataset `copper$Lines` that records the locations of geological faults in a survey region.

```
> data(copper)
> L <- copper$Lines
> L <- rotate(L, pi/2)
> plot(L)
```



A spatial pattern of line segments is represented by an object of class "`psp`". It consists of a list of line segments (given by the coordinates of their two endpoints), and a window in which the line segments were observed. The line segments may also carry marks.

Objects of class "`psp`" can be created by the function `psp` or obtained by converting other data using the function `as.psp`.

Capabilities available for this class include:

<code>is.psp</code>	check whether an object is a <code>psp</code>
<code>[.psp</code>	subset operator (also performs clipping)
<code>marks.psp</code>	extract marks
<code>marks<-.psp</code>	assign marks
<code>as.data.frame.psp</code>	extract coordinates and marks
<code>endpoints.psp</code>	extract endpoints of line segments
<code>midpoints.psp</code>	compute midpoints of line segments
<code>unitname.psp</code>	determine units of length
<code>rescale.psp</code>	change units of length
<code>lengths.psp</code>	compute lengths of line segments
<code>angles.psp</code>	compute orientation angles for line segments
<code>rotate.psp</code>	rotate a line segment pattern
<code>shift.psp</code>	shift a line segment pattern
<code>affine.psp</code>	apply affine transformation
<code>crossing.psp</code>	find intersection points between line segments
<code>selfcrossing.psp</code>	find intersection points between line segments
<code>density.psp</code>	kernel-smoothed intensity image
<code>rshift.psp</code>	apply random shift to each line segment
<code>superimposePSP</code>	combine several line segment patterns
<code>identify.psp</code>	point-and-click to identify line segments
<code>as.mask.psp</code>	convert line segments to binary mask
<code>pixellate.psp</code>	measure the length of line in each pixel

There are also the usual methods

```
plot.psp      plot a line segment pattern
print.psp     print information on a line segment pattern
summary.psp   compute summary of a line segment pattern
```

```
> summary(L)
```

```
146 line segments
```

```
Lengths:
```

```
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.09242 6.61400 12.18000 15.02000 19.95000 65.48000
```

```
Total length: 2192.57251480451 km
```

```
Length per unit area: 0.196937548404655
```

```
Angles (radians):
```

```
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.008107 0.549500 1.747000 1.378000 2.113000 2.912000
```

```
Window: rectangle = [-158.233, -0.19]x[-0.335, 70.11]km
```

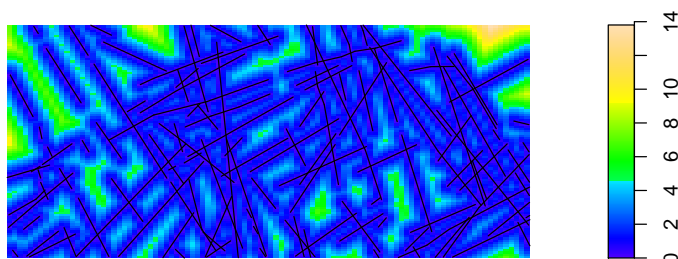
```
Window area = 11133.3 square km
```

```
Unit of length: 1 km
```

```
> plot(distmap(L))
```

```
> plot(L, add = TRUE)
```

distmap(L)



For measuring distances to and from line segments, there are the following facilities:

```
pairedist.psp  distances between line segments
crossdist.psp  distances between two sets of line segments
nndist.psp     closest distances between line segments
nncross        closest distances between points and line segments
nearestsegment closest line segment to each point
project2segment point on line segment closest to specified point
```

It's also possible to generate random points that lie on a set of line segments:

```
runifpointOnLines  fixed number of random points on lines
rpoisppOnLines     Poisson process on lines
```

36 Point patterns in 3D

Basic support for three-dimensional point patterns has recently been added to `spatstat`.

A point pattern in 3D is an object of class "pp3". It is created by the function `pp3`.

```
> X <- pp3(x = runif(100), y = runif(100), z = runif(100), box3(c(0,
+   1)))
> summary(X)
```

```
Three-dimensional point pattern
100 points
Box: [0, 1] x [0, 1] x [0, 1] units
Volume 1 cubic unit
Average intensity 100 points per cubic unit
```

The window containing the point pattern is currently required to be a 3D rectangular box, stored as an object of class "box3", created by the function `box3`. Facilities available for `box3` objects include `print`, `summary`, `diameter`, `volume`, `shortside` and `eroded.volumes`.

Facilities available for `pp3` objects include:

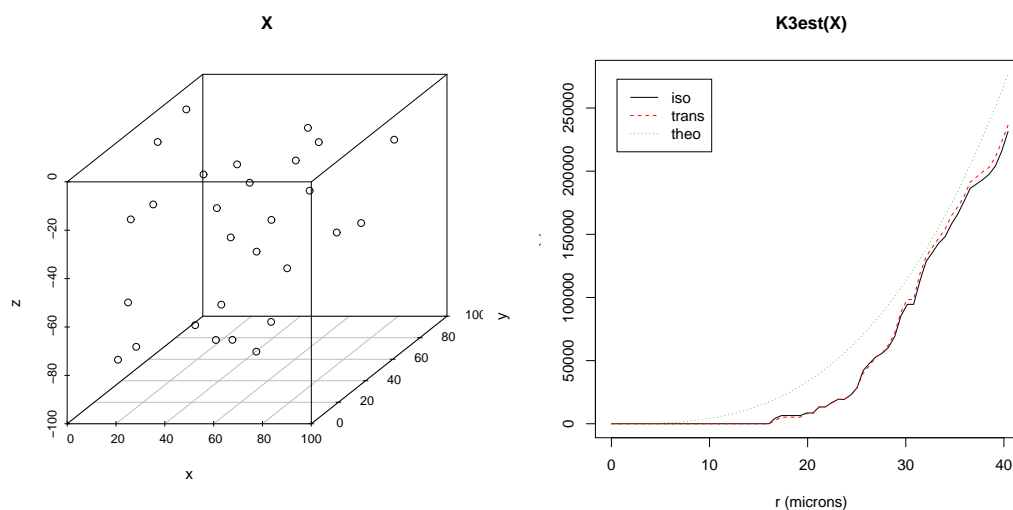
<code>print</code>	print basic information
<code>summary</code>	print detailed summary of data
<code>plot</code>	plot 3D points
<code>as.data.frame</code>	extract coordinates and marks
<code>npoints</code>	number of points
<code>coords</code>	extract coordinates
<code>coords<-</code>	change coordinates
<code>marks</code>	extract marks
<code>marks<-</code>	change marks
<code>unmark</code>	remove marks
<code>unitname</code>	extract name of unit of length
<code>unitname<-</code>	set name of unit of length
<code>pairdist</code>	matrix of distances between all pairs of points
<code>crossdist</code>	distances between all pairs of points in two patterns
<code>nndist</code>	nearest neighbour distance
<code>nnwhich</code>	identify nearest neighbour
<code>F3est</code>	three-dimensional empty space function
<code>G3est</code>	three-dimensional nearest neighbour function
<code>K3est</code>	three-dimensional K -function
<code>pcf3est</code>	three-dimensional pair correlation function
<code>envelope</code>	simulation envelopes of summary functions
<code>nnclean</code>	Byers-Raftery nearest neighbour cleaning

Random 3D point patterns can be generated by `runifpoint3` and `rpoispp3`. Currently the only 3D point pattern data installed in `spatstat` is the dataset `osteo`.

```
> data(osteo)
> X <- osteo$pts[[36]]
> par(mfrow = c(1, 2))
> plot(X)
> plot(K3est(X))
```

	lty	col	key	label	meaning
iso	1	1	iso	K3[iso](r)	isotropic-corrected estimate of K3(r)
trans	2	2	trans	K3[trans](r)	translation-corrected estimate of K3(r)
theo	3	3	theo	K3[pois](r)	theoretical Poisson K3(r)

```
> par(mfrow = c(1, 1))
```



37 Point patterns in multi-dimensional space-time

Experimental support for multi-dimensional space-time point patterns has recently been added to `spatstat`. These objects belong to the class "ppx" and are created by the function `ppx`. There may be any number of dimensions of space, and any number of dimensions of time.

```
> df <- data.frame(x = runif(100, max = 3), y = runif(100, max = 3),
+   z = runif(100, max = 2), t = runif(100))
> bb <- boxx(c(0, 3), c(0, 3), c(0, 2), c(0, 1))
> X <- ppx(data = df, domain = bb, temporal = "t")
> summary(X)
```

```
Multidimensional point pattern
100 points
3-dimensional space coordinates (x,y,z)
1-dimensional time coordinates (t)
Domain:
  4-dimensional box:
[0, 3] x [0, 3] x [0, 2] x [0, 1] units
```

The point pattern may have marks of any type, stored as a `hyperframe`.

```
> marks(X) <- with(as.hyperframe(df), disc(centre = c(x, y)))
> X
```

```

Multidimensional point pattern
100 points
1 column of marks: marks
Domain:
    4-dimensional box:
[0, 3] x [0, 3] x [0, 2] x [0, 1] units

```

The window containing the point pattern is currently required to be a rectangular box, stored as an object of class "boxx", created by the function `boxx`. Facilities available for `boxx` objects include `print`, `summary`, `diameter`, `volume`, `shortside` and `eroded.volumes`.

Facilities currently available for `ppx` objects include:

```

print          print basic information
summary       print detailed summary of data
as.data.frame extract coordinates and marks
npoints       number of points
coords        extract coordinates
coords<-      change coordinates
marks         extract marks
marks<-      change marks
unmark        remove marks
unitname      extract name of unit of length
unitname<-   set name of unit of length
pairdist      matrix of distances between all pairs of points
crossdist     distances between all pairs of points in two patterns
nndist        nearest neighbour distance
nnwhich       identify nearest neighbour

```

Random hyperdimensional point patterns can be generated by `runifpointx` and `rpoisppx`.

There is also code to compute the theoretical distribution of the nearest neighbour distances, i.e. the distance from a reference point to the k -th nearest neighbour in a uniform Poisson point process in m dimensions:

```

dknn  probability density
pknn  cumulative probability
qknn  quantiles
rknn  random generator

```

38 Replicated data and hyperframes

Sometimes the data may consist of several point patterns. We may have *replicated* point pattern data obtained by repeating an experiment, such as the locations of cell nuclei in 10 different samples of tissue. These data could be stored in a simple list Z , where the i th entry in the list, $Z[[i]]$, is the i th point pattern object.

A *hyperframe* is like a data frame, except that the entries can be any type of object. For example, a hyperframe can include a column, each of whose entries is a point pattern. The only constraint is that all the entries in a column must be of the same type.

A hyperframe is a convenient way to store replicated point pattern data together with auxiliary data. A hyperframe can store the results of a designed experiment where the “response” is a point pattern and the covariates are other types of data.

The following facilities are available:

<code>hyperframe</code>	create a hyperframe
<code>as.hyperframe</code>	convert other data to a hyperframe
<code>print.hyperframe</code>	print a representation of a hyperframe
<code>dim.hyperframe</code>	dimensions of hyperframe
<code>\$.hyperframe</code>	extract column of hyperframe
<code>[.hyperframe</code>	extract subset of hyperframe
<code>[<-.hyperframe</code>	replace subset of hyperframe
<code>as.data.frame.hyperframe</code>	convert hyperframe to data frame
<code>cbind.hyperframe</code>	combine several hyperframes
<code>rbind.hyperframe</code>	combine several hyperframes
<code>with.hyperframe</code>	compute an expression in each row of hyperframe

The dataset `osteo` installed in `spatstat` contains three-dimensional point patterns recorded in several sampling volumes in each of several bone samples [15]. It is a hyperframe with the following columns:

<code>id</code>	string identifier of bone sample
<code>shortid</code>	last numeral in <code>id</code>
<code>brick</code>	serial number of sampling volume within bone sample
<code>pts</code>	three dimensional point pattern (class <code>pp3</code>)
<code>depth</code>	the depth of the brick in microns

```
> data(osteo)
> osteo[1:5, ]
```

Hyperframe:

```
   id shortid brick  pts depth
1 c77za4     4     1 (pp3)  45
2 c77za4     4     2 (pp3)  60
3 c77za4     4     3 (pp3)  55
4 c77za4     4     4 (pp3)  60
5 c77za4     4     5 (pp3)  85
```

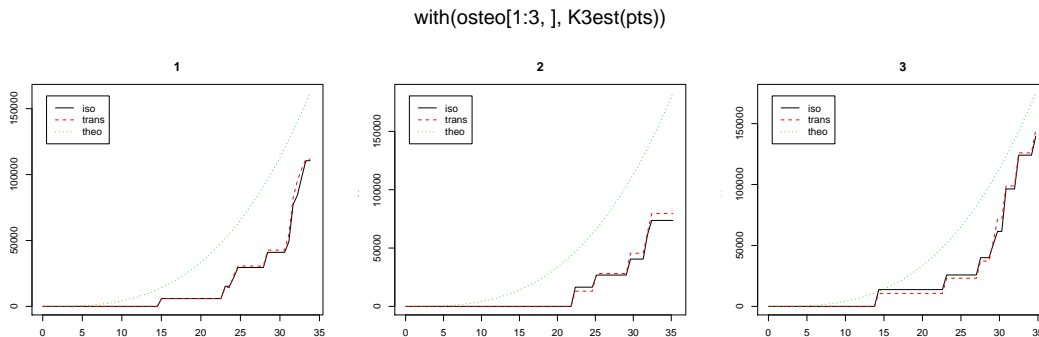
The command `with.hyperframe` can be used to evaluate an expression in each row of the hyperframe. For example, to count the number of points in each of the first 10 patterns:

```
> with(osteo, npoints(pts))[1:10]

 1  2  3  4  5  6  7  8  9 10
13 11 11 12 14 12 16 15 18 16
```


To plot the three-dimensional K -function estimates of the first 3 patterns:

```
> plot(with(osteo[1:3, ], K3est(pts)))
```

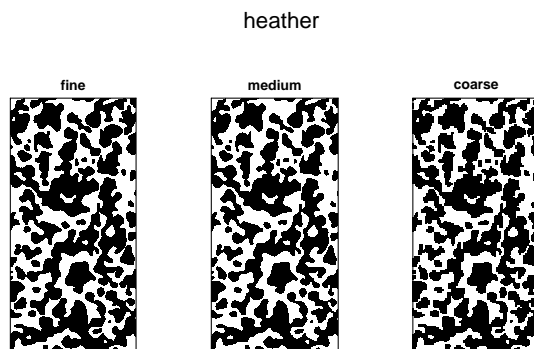


39 Stochastic geometry

`Spatstat` provides some basic facilities for stochastic geometry (model simulation and tools for image analysis).

The dataset `heather` contains Diggle's [32] heather data, a binary pixel mask of the presence/absence of heather (*Calluna vulgaris*) in a survey plot in Sweden. Three versions of the data are given.

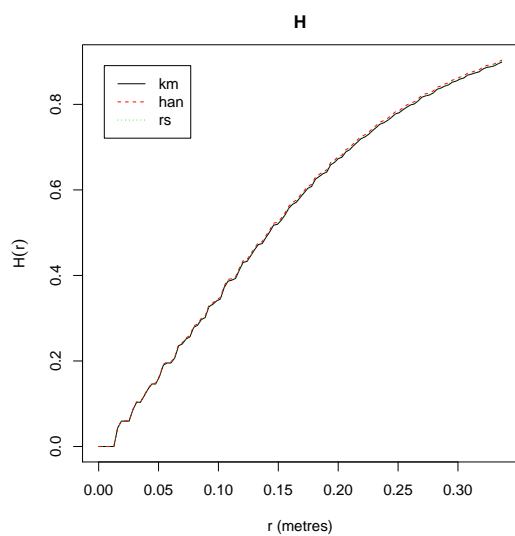
```
> data(heather)
> plot(heather)
```



Elementary morphological operations can be applied using `erosion`, `dilation`, `opening` and `closing` along with geometrical operations such as `complement.owin`, `intersect.owin`, `union.owin`, `setminus.owin`. The areas of dilations and erosions can be computed using either `distmap` or `eroded.areas`, `dilated.areas`. Connected components can be identified using `connected`.

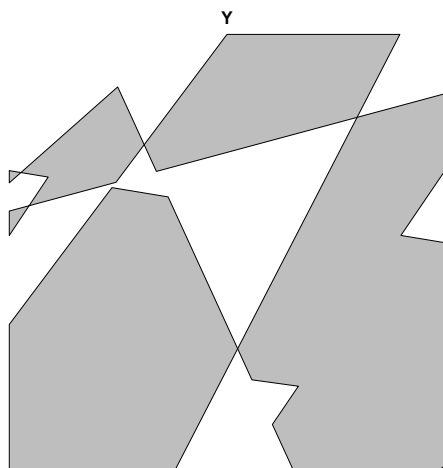
The empirical *spherical contact distribution* or empty space function of such a dataset can be computed using `Hest`:

```
> X <- heather$fine
> H <- Hest(X)
> plot(H)
```



Patterns of line segments are represented by objects of the class "psp" described elsewhere. Patterns of *infinite* lines in the two-dimensional plane are represented by objects of the class "infinite" created by the function `infinite`. The command `rpoisline` creates a random tessellation delineated by a Poisson line process. Switzer's random set is generated by `rMosaicSet` and the analogous random field is generated by `rMosaicField`.

```
> X <- rpoislinetess(5)
> Y <- rMosaicSet(X, p = 0.5)
> plot(Y, col = "grey")
```



40 Further information on spatstat

Help files

For information on a particular command in `spatstat`, consult the online help file by typing `help(command)`. The help files are detailed and extensive. The complete manual is over 700 pages long.

For examples of the use of a particular command, read the examples section in the help file, or type `example(command)` to see the examples executed.

Quick reference

Type `help(spatstat)` for a quick-reference overview of all the functions available in the package.

For a demonstration of many of the capabilities of `spatstat`, type `demo(spatstat)`.

For a visual display of all the datasets supplied in `spatstat`, type `demo(data)`.

Website

The website www.spatstat.org contains information on recent updates to the package, frequently-asked questions, bug fixes, literature and other developments.

Updates

`Spatstat` is updated approximately once a month. Descriptions of the updates are given on the spatstat.org website, and can also be accessed from within the package, by typing `latest.news()` or `news(package="spatstat")`.

Modelling

For more `spatstat` examples on fitting point process models, see [11]. For more discussion on point process modelling strategies, see [3].

Citation

If you use `spatstat` in a research publication, it would be much appreciated if you could cite the paper [10], or mention `spatstat` in the acknowledgements.

In doing so, you will help us to justify the expenditure of time and effort on maintaining and developing the package.

Citation details are also available in the package by typing `citation(package="spatstat")`.

Queries and requests

If you have difficulty in getting the package to do what you want, or if you have a suggestion for additional features that could be added, please contact the package authors, Adrian.Baddeley@csiro.au and r.turner@auckland.ac.nz, or email the R special interest group in spatial and geographical statistics, r-sig-geo@stat.math.ethz.ch.

References

- [1] 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] L. Anselin. Local indicators of spatial association – LISA. *Geographical Analysis*, 27:93–115, 1995.
- [3] 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.
- [4] A. Baddeley, M. Berman, N.I. Fisher, A. Hardegen, R.K. Milne, D. Schuhmacher, R. Shah, and R. Turner. Spatial logistic regression and change-of-support for spatial Poisson point processes. *Electronic Journal of Statistics*, 4:1151–1201, 2010. doi: 10.1214/10-EJS581.
- [5] A. Baddeley, Y.-M. Chang, Y. Song, and R. Turner. Diagnostics for transformation of covariates in spatial Poisson point process models. Submitted for publication.
- [6] 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.
- [7] A. Baddeley, J. Møller, and R. Waagepetersen. Non- and semiparametric estimation of interaction in inhomogeneous point patterns. *Statistica Neerlandica*, 54(3):329–350, November 2000.
- [8] A. Baddeley, E. Rubak, and J. Møller. Score, pseudo-score and residual diagnostics for goodness-of-fit of spatial point process models. Submitted for publication.
- [9] 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.
- [10] 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.
- [11] 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.
- [12] 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.
- [13] 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.
- [14] A.J. Baddeley and J. Møller. Nearest-neighbour Markov point processes and random sets. *International Statistical Review*, 57:89–121, 1989.

-
- [15] 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.
- [16] 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.
- [17] A.J. Baddeley and M.N.M. van Lieshout. Area-interaction point processes. *Annals of the Institute of Statistical Mathematics*, 47:601–619, 1995.
- [18] 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.
- [19] M. Bell and G. Grunwald. Mixed models for the analysis of replicated spatial point patterns. *Biostatistics*, 5:633–648, 2004.
- [20] M. Berman. Testing for spatial association between a point process and another stochastic process. *Applied Statistics*, 35:54–62, 1986.
- [21] M. Berman and T.R. Turner. Approximating point process likelihoods with GLIM. *Applied Statistics*, 41:31–38, 1992.
- [22] J. Besag and P.J. Diggle. Simple Monte Carlo tests for spatial pattern. *Applied Statistics*, 26:327–333, 1977.
- [23] J.E. Besag and P. Clifford. Generalized Monte Carlo significance tests. *Biometrika*, 76:633–642, 1989.
- [24] R. Bivand, E.J. Pebesma, and V. Gómez-Rubio. *Applied spatial data analysis with R*. Springer, 2008.
- [25] D.R. Brillinger. Comparative aspects of the study of ordinary time series and of point processes. In P.R. Krishnaiah, editor, *Developments in Statistics*, pages 33–133. Academic Press, 1978.
- [26] 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.
- [27] 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.
- [28] 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.
- [29] 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.
- [30] N.A.C. Cressie. *Statistics for Spatial Data*. John Wiley and Sons, New York, 1991.
- [31] D.J. Daley and D. Vere-Jones. *An Introduction to the Theory of Point Processes*. Springer Verlag, New York, 1988.

- [32] P.J. Diggle. Binary mosaics and the spatial pattern of heather. *Biometrics*, 37:531–539, 1981.
- [33] P.J. Diggle. *Statistical analysis of spatial point patterns*. Academic Press, London, 1983.
- [34] 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.
- [35] P.J. Diggle. *Statistical Analysis of Spatial Point Patterns*. Hodder Arnold, London, second edition, 2003.
- [36] P.J. Diggle, N. Lange, and F. M. Benes. Analysis of variance for replicated spatial point patterns in clinical neuroanatomy. *Journal of the American Statistical Association*, 86:618–625, 1991.
- [37] P.J. Diggle, J. Mateu, and H.E. Clough. A comparison between parametric and non-parametric approaches to the analysis of replicated spatial point patterns. *Advances in Applied Probability (SGSA)*, 32:331–343, 2000.
- [38] 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.
- [39] M. Dwass. Modified randomization tests for nonparametric hypotheses. *Annals of Mathematical Statistics*, 28:181–187, 1957.
- [40] 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.
- [41] A.C.A. Hope. A simplified Monte Carlo significance test procedure. *Journal of the Royal Statistical Society, series B*, 30:582–598, 1968.
- [42] C.V. Howard, S. Reid, A.J. Baddeley, and A. Boyde. Unbiased estimation of particle density in the tandem-scanning reflected light microscope. *Journal of Microscopy*, 138:203–212, 1985.
- [43] F. Huang and Y. Ogata. Improvements of the maximum pseudo-likelihood estimators in various spatial statistical models. *Journal of Computational and Graphical Statistics*, 8(3):510–530, 1999.
- [44] J. Illian, A. Penttinen, H. Stoyan, and D. Stoyan. *Statistical Analysis and Modelling of Spatial Point Patterns*. John Wiley and Sons, Chichester, 2008.
- [45] J.F.C. Kingman. *Poisson Processes*. Oxford University Press, New York, 1993.
- [46] G.M. Laslett. Censoring and edge effects in areal and line transect sampling of rock joint traces. *Mathematical Geology*, 14:125–140, 1982.
- [47] 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.
- [48] J.K. Lindsey. *The analysis of stochastic processes using GLIM*. Springer, Berlin, 1992.

- [49] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.
- [50] J. Møller and R. P. Waagepetersen. Modern spatial point process modelling and inference (with discussion). *Scandinavian Journal of Statistics*, 34:643–711, 2007.
- [51] J. Møller and R.P. Waagepetersen. *Statistical Inference and Simulation for Spatial Point Processes*. Chapman and Hall/CRC, Boca Raton, 2004.
- [52] Y. Ogata. Statistical models for earthquake occurrences and residual analysis for point processes. *Journal of the American Statistical Association*, 83:9–27, 1988.
- [53] 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.
- [54] B.D. Ripley. Modelling spatial patterns (with discussion). *Journal of the Royal Statistical Society, series B*, 39:172–212, 1977.
- [55] B.D. Ripley. Simulating spatial patterns: dependent samples from a multivariate density. *Applied Statistics*, 28:109–112, 1979.
- [56] B.D. Ripley. *Spatial Statistics*. John Wiley and Sons, New York, 1981.
- [57] B.D. Ripley. *Statistical Inference for Spatial Processes*. Cambridge University Press, 1988.
- [58] 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.
- [59] 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.
- [60] D. Stoyan and P. Grabarnik. Second-order characteristics for stochastic structures connected with Gibbs point processes. *Mathematische Nachrichten*, 151:95–100, 1991.
- [61] D. Stoyan and H. Stoyan. *Fractals, Random Shapes and Point Fields*. John Wiley and Sons, Chichester, 1995.
- [62] 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.
- [63] M.N.M. van Lieshout. *Markov Point Processes and their Applications*. Imperial College Press, London, 2000.
- [64] M.N.M. van Lieshout and A.J. Baddeley. A nonparametric measure of spatial interaction in point patterns. *Statistica Neerlandica*, 50:344–361, 1996.
- [65] R. Waagepetersen. An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics*, 63:252–258, 2007.

Index

- analysis of deviance, 103
- area-interaction process, 159
- binary mask, 33, 48
- circular windows, 46
- classes, 32
 - in R, 32
 - in spatstat, 32
- cluster models
 - fitting, 144, 152
 - inhomogeneous, 151
 - fitting, 152
- complete spatial randomness, 88
 - and independence, 179, 204
 - definition, 88
 - Kolmogorov-Smirnov test, 91
 - quadrat counting test, 89
- conditional intensity, 160
 - for marked point processes, 210
- contrasts, 98, 208
- covariate effects, 9
- covariates, 7, 16, 98
 - in ppm, 98
- Cox process, 141
- CSRI, 179, 204
 - conditional intensity, 210
 - fitting to data, 207
 - simulating, 205
- data entry, 38
 - checking, 43
 - GIS formats, 45, 49
 - marked point patterns, 181
 - marks, 40
 - point-and-click, 44
- data sharpening, 148
- datasets
 - inspecting, 21
 - provided in spatstat, 30
- dispatching, 32
- distance methods, 115
- distances
 - empty space, 115, 116
 - nearest neighbour, 115, 122
 - pairwise, 115, 125
- distmap, 115
- edge effects, 116
- empty space distances, 115, 116
- empty space function, 117, 222
- envelopes, 132
 - and Monte Carlo tests, 132
 - for any fitted model, 136
 - for any simulation procedure, 137
 - in spatstat, 133
 - of summary functions, 132
- exploratory data analysis, 23
 - for marked point patterns, 200
 - for multitype point patterns, 187
- fitted model, 166
 - goodness-of-fit, 106, 172
 - interpretation of coefficients, 98
 - methods for, 99
 - residuals, 107, 173
 - simulation of, 104
- fitting models
 - by Huang-Ogata method, 170
 - kppm, 144, 152
 - maximum pseudolikelihood, 162
 - to marked point patterns, 207, 212
 - via summary statistics, 144
- fv, 37
- geometrical transformations, 57
- Gibbs models, 156
 - area-interaction, 159
 - Diggle-Gates-Stibbard, 159
 - Diggle-Gratton, 159
 - fitting, 162
 - by Huang-Ogata method, 170
 - maximum pseudolikelihood, 162
 - ppm, 162
 - fitting to marked point patterns, 212
 - goodness-of-fit, 172
 - hard core process, 157
 - in spatstat, 165
 - infinite order interaction, 159
 - multitype, 210
 - maximum pseudolikelihood, 212
 - multitype pairwise interaction, 210
 - pairwise interaction, 159
 - residuals, 173
 - simulation, 161

- simulation of fitted model, 168
- soft core, 159
- Strauss process, 158
- Strauss-hard core, 159
- GIS formats, 45, 49
- goodness-of-fit, 106
 - for fitted Gibbs model, 172
 - for Poisson models, 106
- hard core process, 157
 - multitype, 211
- heather data, 222
- Huang-Ogata method, 170
- hyperframe, 221
- hyperframe**, 32
- im**, 32, 63
- images, 63
 - computing with, 70
 - creating, 63
 - from raw data, 63
 - exploratory inspection of, 68
 - extracting subset, 69
 - plotting, 66
 - returned by a function, 65
- independence of components, 179, 197
- intensity
 - function, 79
 - kernel estimator, 79
 - homogeneous, 78
 - inhomogeneous, 79
 - investigation of, 78
 - measure, 79
 - of multitype point process, 187
- interaction, 8, 11
 - distance methods, 115
 - exploratory methods, 114
 - in **spatstat**, 165
 - multitype, 210, 212
 - in **spatstat**, 212
 - plotting a fitted interaction, 214
 - Q–Q plot, 175
 - simple models, 139
 - summary functions, 115
- K* function, 24, 125
 - for multitype point pattern, 190
 - inhomogeneous, 149
- kernel estimator of intensity, 79, 80
- kernel smoothing of marks, 200
- Kolmogorov-Smirnov test
 - of CSR, 91
 - of inhomogeneous Poisson, 107
- kppm**, 144, 152
- line segments, 216
- LISA, 148
- local features, 148
- locally scaled point process, 153
- lurking variable plot, 109
- maptools** package, 45
- mark connection function, 195
- mark correlation function, 202
- marked point patterns
 - cutting marks into bands, 185
 - data entry, 181
 - exploratory data analysis, 200
 - exploring marks, 200
 - inspecting, 183
 - joint and conditional analysis, 179
 - manipulating, 184
 - methodological issues, 179
 - model-fitting, 207, 212
 - probabilistic formulation, 178
 - randomisation tests, 179
 - separating into types, 185
 - summary functions, 190
- marks, 6, 16, 178
 - categorical, 41
 - data entry, 40
 - exploratory data analysis, 200
 - manipulating, 184
 - operations on, 56
 - smoothing, 200
 - spatial trend in, 200
 - versus covariates, 15
- markstat**, 190
- marktable**, 189
- Matern cluster process, 140
- maximum likelihood, 95
- maximum pseudolikelihood, 162, 212
 - for multitype Gibbs models, 212
 - improvements over, 170
- methods, 32
 - default method, 34
 - dispatch, 32
- minimum contrast, 144
- model validation, 106, 171

- models, 25, 224
- Monte Carlo test, 132
 - pointwise, 133
 - simultaneous, 134
- multidimensional point pattern, 219
- multitype hard core process, 211
- multitype point pattern, 10, 11, 27, 41
- multitype point patterns
 - exploratory data analysis, 187
 - separating into types, 185
 - summary functions, 190
- multitype point process
 - intensity, 187
- multitype Strauss process, 211

- nearest neighbour cleaning, 148
- nearest neighbour distances, 115, 122
- `nndist`, 115
- nuisance parameters, 168

- `owin`, 32, 46

- `pairdist`, 115
- pairwise distances, 115, 125
- pairwise interaction process, 157
- point pattern, 6
 - data entry, 38
 - in many dimensions, 219
 - in space-time, 219
 - in three dimensions, 218
 - marked, 178
 - marks, 6, 16
 - multitype, 10, 11
 - needs window, 54
 - point process model for, 13
 - standard model, 14
- point process, 13
- point process models, 25
 - area-interaction, 159
 - Diggle-Gates-Stibbard, 159
 - Diggle-Gratton, 159
 - Gibbs, 156
 - hard core, 157
 - infinite order interaction, 159
 - pairwise interaction, 157, 159
 - soft core, 159
 - Strauss, 158
 - Strauss-hard core, 159
- Poisson cluster processes, 140
- Poisson line process, 223
- Poisson line tessellation, 223
- Poisson models
 - fitting, 96
 - goodness-of-fit, 106
 - homogeneous, 88
 - inhomogeneous, 95
 - log-likelihood, 96
 - marked, 204
 - maximum likelihood, 95
 - residuals, 107
- Poisson point process
 - homogeneous
 - definition, 88
 - simulation, 88
 - inhomogeneous
 - definition, 95
 - fitting, 96
 - likelihood, 96
 - motivation, 95
 - simulation, 95
- Poisson-derived models, 139
- polygonal windows, 33, 47
- `pp3`, 32, 218
- `ppm`, 99, 166
 - marked Gibbs point process models, 212
 - marked Poisson point process models, 207
 - methods for, 100
- `ppp`, 32
 - combining several, 61
 - extracting subset, 55
 - format, 53
 - geometrical transformations, 57
 - in arbitrary window, 51
 - manipulating, 53
 - needs window, 54
 - operations on, 55
 - random perturbations, 58
 - ways to make, 44
- `ppx`, 32, 219
- probability density, 156
- profile pseudolikelihood, 168
- pseudolikelihood, 162
 - profile pseudolikelihood, 168
- `psp`, 32, 216

- quadrat counting, 23, 79
- quadrat counting test
 - of CSR, 89
- quadrat test

- of inhomogeneous Poisson, 106
- R, 18
 - contributed packages, 19
 - for spatial data formats, 45
 - for spatial statistics, 19
 - where to get, 18
- random labelling, 179, 198
- random perturbations, 58
- random thinning, 95
- randomisation tests, 179, 197
 - for marked point patterns, 197
- rectangular windows, 33, 46
- replicated point patterns, 221
- residuals, 107, 173
 - for fitted Gibbs model, 173
 - for Poisson models, 107
 - lurking variable plot, 109
 - Q–Q plot, 174
 - smoothed residual field, 108
- return value, 35
- `rpoispp`, 88, 95
- `runifpoint`, 89
- sequential models, 142
- shapefiles, 45
- `shapefiles` package, 45
- simulation
 - of fitted Gibbs model, 168
 - of fitted Poisson model, 104
- smoothed residual field, 108
- `sp` package, 45
- `spatstat`, 20, 224
 - citing, 20, 224
 - getting started, 20
 - help files, 224
 - installing, 20
 - queries and requests, 224
 - updates, 224
 - website, 224
- spherical contact distribution, 222
- `split`, 28
- standard model, 14
- stochastic geometry, 222
- Strauss process, 158
 - fitting to data, 163
 - multitype, 211
- summary functions, 115
 - and Monte Carlo tests, 132
 - critique, 130
 - edge effects, 116
 - envelopes, 132
 - F , 117
 - for multitype point patterns, 190
 - G , 122
 - inference using, 132
 - inhomogeneous K , 149
 - J , 128
 - K , 125
 - L , 126
 - mark connection, 195
 - mark correlation, 202
 - model-fitting with, 144
 - pair correlation, 126
 - Switzer's random set, 223
- `tess`, 32
- tests
 - χ^2 quadrat counting, 89
 - Kolmogorov-Smirnov, 91, 107
 - Monte Carlo, 132
- thinning, 141
- Thomas process, 140
- three dimensional point pattern, 218
- tips, 32, 36, 42, 56, 116, 119, 134, 181
- treatment contrasts, 98
- `unitname`, 43
- units of length, 43
- validation, 106, 171
- windows, 46
 - binary mask, 33, 48
 - circular, 46
 - GIS formats, 49
 - needed in any point pattern, 54
 - operations on, 50
 - polygonal, 33, 47
 - rectangular, 33, 46
 - returned by functions, 49
- χ^2 quadrat counting test, 89

Practice Session 1

This session gives a general orientation to R and `spatstat`.

If you have not already done so, you'll need to

- install the R system on your computer (see information sheet *How to install R*)
 - install the `spatstat` package in R (see information sheet *How to install spatstat*).
 - Start R.
 - Load the `spatstat` package by typing `library(spatstat)`. Check that *version 1.21-2 or later* is loaded.
-

1. We will study a dataset that records the locations of Ponderosa Pine trees (*Pinus ponderosa*) in a study region in the Klamath National Forest in northern California. The data are included with `spatstat` as the dataset `ponderosa`.
 - (a) Type `data(ponderosa)` to access the data;
 - (b) assign the data to a shorter name, like `X` or `P`;
 - (c) plot the data;
 - (d) find out how many trees are recorded;
 - (e) find the dimensions of the study region;
 - (f) obtain an estimate of the average intensity of trees (number of trees per unit area).
2. The Ponderosa data, continued:
 - (a) When you type `plot(ponderosa)`, the command that is actually executed is `plot.ppp`, the plot method for point patterns. Read the help file for the function `plot.ppp`, and find out which argument to the function can be used to control the main title for the plot;
 - (b) plot the Ponderosa data with the title "*Ponderosa Pine Trees*" above it;
 - (c) from your reading of the help file, predict what will happen if we type

```
plot(ponderosa, chars="X", cols="green")
```

then check that your guess was correct;
 - (d) try different values of the argument `chars`, for example, one of the integers 0 to 25, or a letter of the alphabet. (Note the difference between `chars=3` and `chars="+`", and the difference between `chars=4` and `chars="X"`).

3. The following table records the locations of 10 scintillation events observed under a microscope. Coordinates are given in microns, and the study region was 30×30 microns, with the origin at the bottom left corner.

x	y
13	3
15	15
27	7
17	11
8	10
8	17
1	29
14	22
19	19
23	29

Enter the data into R as two vectors x and y . Create a point pattern X from the data, and plot the point pattern.

Supplementary Exercises

4. The dataset `longleaf` contains the Longleaf Pines dataset giving the locations of trees and their diameters at breast height.
- Read the help file for the data;
 - access the dataset and plot it;
 - re-plot the data so that the tree diameters are displayed at a physical scale that is 10 times the physical scale of the location coordinates.
5. The file `anthills.txt` is available on a USB stick from the workshop demonstrators, or at this URL

<http://school.maths.uwa.edu.au/homepages/adrian/anthills.txt>

It records the locations of anthills recorded in a 1200×1500 metre study region in northern Australia. Coordinates are given in metres, along with a letter code recording the ecological 'status' of each anthill.

- read the data into R as a data frame, using the R function `read.table`.
Since the input file has a header line, you will need to use the argument `header=TRUE` when you call `read.table`.
It may be useful to change the working directory first: use the `setwd` command, or the pull-down menu `File > Change Dir`.
- check the data for any peculiarities.
- create a point pattern `hills` containing these data. Ensure that the marks are a factor, and that the unit of length is given its correct name.
- plot the data.

Supplementary Exercises

6. (a) Compute a kernel estimate of the intensity for the Japanese Pines data using a Gaussian kernel with standard deviation $\sigma = 0.15$.
- (b) Find the maximum and minimum values of the intensity estimate over the study region. [Hint: use `summary.im` or `range.im`]
- (c) The kernel estimate of intensity is defined so that its integral over the entire study region is equal to the number of points in the data pattern, ignoring edge effects. Check whether this is approximately true in this example. [Hint: use `summary.im` or `integral.im`]
7. The command `rpoispp(2, win=square(10))` generates realisations of the Poisson process with intensity $\lambda = 2$ in the square $[0, 10] \times [0, 10]$. **Warning:** the argument `win` is not the second argument so it must be named explicitly.
- (a) Repeat the command `plot(rpoispp(2, win=square(10)))` several times to build your intuition about the appearance of a completely random pattern of points.
- (b) What is the expected number of points in the random pattern that this command will generate?
8. The `update` command can be used to re-fit a point process model using a different model formula.
- (a) Type the following commands and interpret the results:
- ```
fit0 <- ppm(japanesepines, ~1)
fit1 <- update(fit0, ~x)
fit1
fit2 <- update(fit1, ~ . + y)
fit2
```
- (b) Now type `step(fit2)` and interpret the results.
9. Fit Poisson point process models to the Japanese Pines data, with the following trend formulas. Read off an expression for the fitted intensity function in each case.
- | TREND FORMULA                    | FITTED INTENSITY FUNCTION |
|----------------------------------|---------------------------|
| <code>~1</code>                  |                           |
| <code>~x</code>                  |                           |
| <code>~sin(x)</code>             |                           |
| <code>~x+y</code>                |                           |
| <code>~polynom(x,y,2)</code>     |                           |
| <code>~factor(x &lt; 0.4)</code> |                           |
10. Make image plots of the fitted intensities for the inhomogeneous models above.

---

## Practice Session 2

---

This session covers tools for investigating intensity, including nonparametric methods (kernel smoothing) and parametric modelling.

You'll need to start R, load the `spatstat` library and set `spatstat.options(gpclip=TRUE)`.

1. The dataset `japanesepines` contains the locations of Japanese Black Pine trees in a study region.
  - (a) Plot the `japanesepines` data.
  - (b) Use the command `quadratcount` to divide the study region of the Japanese Pines data into a  $3 \times 3$  array of equal quadrats, and count the number of trees in each quadrat.
  - (c) Most plotting commands will accept the argument `add=TRUE` and interpret it to mean that the plot should be drawn over the existing display, without clearing the screen beforehand. Use this to plot the Japanese Pines data, and superimposed on this, the  $3 \times 3$  array of quadrats, with the quadrat counts also displayed.
  - (d) Use the command `quadrat.test` to perform the  $\chi^2$  test of CSR on the Japanese Pines data.
  - (e) Plot the Japanese Pines data, and superimposed on this, the  $3 \times 3$  array of quadrats and the observed, expected and residual counts. Use the argument `cex` to make the numerals larger and `col` to display them in another colour.
2. Japanese Pines, continued:
  - (a) Using `density.ppp`, compute a kernel estimate of the spatially-varying intensity function for the Japanese pines data, using a Gaussian kernel with standard deviation  $\sigma = 0.1$  units, and store the estimated intensity in an object `D` say.
  - (b) Plot a colour image of the kernel estimate `D`.
  - (c) Plot a colour image of the kernel estimate `D` with the original Japanese Pines data superimposed.
  - (d) Plot the kernel estimate without the 'colour ribbon'.  
[Hint: consult the help file for `plot.im`, the `plot` method for pixel images (objects of class "`im`").]
  - (e) Try the following command

```
persp(D, theta=30, phi=45, shade=0.4)
```

and find the documentation for the arguments `theta`, `phi` and `shade`.

3. The `murchison` dataset gives the locations of gold deposits and geological faults in a survey area.

(a) Extract the data and rescale them to kilometres by typing

```
data(murchison)
attach(murchison)
gold <- rescale(gold, 1000)
faults <- rescale(faults, 1000)
unitname(gold) <- unitname(faults) <- "km"
```

(b) Plot the `gold` and `faults` datasets together on the same plot.

(c) Compute the distance function of the geological faults as `d <- distfun(faults)`

(d) Plot the distance function `d` and check that it is what you expected

(e) Assuming that the intensity of gold deposits is a function  $\lambda(u) = \rho(d(u))$  where  $d(u)$  is the distance function of the faults, plot a nonparametric estimate of the function  $\rho$  by `plot(rhohat(gold, d))`.

4. Returning to the Japanese Pines data,

(a) Fit the uniform Poisson point process model to the Japanese Pines data,  
`ppm(japanesepines, ~1)`

(b) Read off the fitted intensity. Check that this is the correct value of the maximum likelihood estimate of the intensity (see the Notes, page 96, below equation (3)).

(c) Fit the Poisson point process models with loglinear intensity (trend formula `~x+y`) and log-quadratic intensity (trend formula `~polynom(x,y,2)`) to the Japanese Pines data.

(d) extract the fitted coefficients for these models using `coef`.

(e) perform the Likelihood Ratio Test for the null hypothesis of a loglinear intensity against the alternative of a log-quadratic intensity, using `anova`.

(f) Generate 10 simulated realisations of the fitted log-quadratic model, and plot them, using `plot(simulate(fit, nsim=10))` where `fit` is the fitted model.

5. Murchison data, continued:

(a) Fit a Poisson point process model to the Murchison gold deposit data which assumes that the intensity is a loglinear function of distance to the nearest fault,  
`ppm(gold, ~dfault, covariates=list(dfault=d))`

(b) Read off the fitted coefficients and write down the fitted intensity function.

(c) Plot the fitted intensity as a colour image.

(d) extract the estimated variance-covariance matrix of the coefficient estimates, using `vcov`.

(e) Plot the standard error of the fitted intensity as a colour image.



6. [The following questions are based on Section 8 of the Workshop Notes.] The file `region.txt` is available on a USB stick from the demonstrators, or at this URL

<http://school.maths.uwa.edu.au/homepages/adrian/region.txt>

It contains the coordinates (in kilometres) of vertices of the polygonal boundary of a study region.

- (a) read the data into R as a data frame, using the R function `read.table`.
  - (b) create a window object representing the polygon.
  - (c) plot the region and the position of its centroid.
  - (d) obtain the bounding box of the region (the smallest rectangle containing the region).
  - (e) find the area, diameter and perimeter length of the polygon.
7. First type `spatstat.options(gpclip=TRUE)` to ensure that polygon computations are enabled. Create and plot a window object (object of class "owin") representing each of the following:
- (a) the rectangle  $[0, 10] \times [0, 5]$  ;
  - (b) the disc of radius 4 centred at  $(5, 2)$ ;
  - (c) the intersection of these two windows.