Analysing replicated point patterns in `spatstat`

Adrian Baddeley

For `spatstat` version `3.6-4`

Abstract

This document describes `spatstat`'s capabilities for fitting models to replicated point patterns. More generally it applies to data from a designed experiment in which the response from each unit is a spatial point pattern.

Contents

1 Introduction **3**

2 Overview of software **3**

3 Formulating the problem **3**

4 Installed datasets **4**

5 Lists of objects **4**
5.1 Waterstriders data 4
5.2 The class `solist` 5
5.3 Creating a `solist` 6
5.4 The class `anylist` 7

6 Hyperframes **7**
6.1 Creating hyperframes 7
6.2 Hyperframes of data 8
6.3 Columns of a hyperframe 10
6.4 Subsets of a hyperframe 11

7 Plotting **12**
7.1 Plotting a `solist` object 12
7.2 Plotting a hyperframe 12
7.2.1 Plotting one column 12
7.2.2 Complex plots 13

8 Data analysis **14**
8.1 Computing with hyperframes 14
8.2 Summary statistics 15
8.3 Generating new columns 17
8.4 Simulation 17

9 Exploratory data analysis **18**
9.1 Exploring spatial trend and covariate effects . . . 18
9.2 Exploring interpoint interaction . . . 19

10 Fitting models of spatial trend **19**
10.1 Trend formula . . . 19
10.1.1 Design covariates . . . 20
10.1.2 Spatial covariates . . . 21

11 Interpoint interaction **21**
11.1 Same interaction for all patterns . . . 22
11.2 Hyperframe of interactions . . . 22
11.3 Interaction formula . . . 23
11.3.1 Selecting one column . . . 23
11.3.2 Interaction depending on design . . . 24
11.3.3 Completely different interactions for different cases . . . 27

12 Studying the fitted model **29**
12.1 Fits for each pattern . . . 29
12.1.1 Subfits . . . 29
12.1.2 Fitting separately to each pattern . . . 29
12.2 Residuals . . . 30
12.2.1 Point process residuals . . . 30
12.2.2 Sums of residuals . . . 32
12.2.3 Four-panel diagnostic plots . . . 32
12.2.4 Residuals of the parameter estimates . . . 33
12.3 Goodness-of-fit tests . . . 34
12.3.1 Quadrat count test . . . 34
12.3.2 Kolmogorov-Smirnov test . . . 35

Bibliography **36**

1 Introduction

'Replicated point patterns' are datasets consisting of several point patterns which can be regarded as independent repetitions of the same experiment. For example, three point patterns taken from micrographs of three pipette samples of the same jug of milk, could be assumed to be replicated observations.

More generally we could have several experimental groups, with replicated point pattern data in each group. For example there may be two jugs of milk that were treated differently, and we take three pipette samples from each jug.

Even more generally our point patterns could be the result of a designed experiment involving control and treatment groups, covariates such as temperature, and even spatial covariates (such as image data).

This document describes some capabilities available in the `spatstat` package for analysing such data. **For further detail, see Chapter 16 of the spatstat book [2].**

2 Overview of software

The main components needed are:

- the model-fitting function `mppm`, an extension of the `spatstat` function `ppm`, that will fit Gibbs point process models to multiple point pattern datasets;
- support for the class `"mppm"` of point process models fitted by `mppm` (e.g. functions to print and plot the fitted model, analysis of deviance for Poisson models)
- some tools for exploratory data analysis;
- basic support for the data from such experiments by storing the data in a *"hyperframe"*. A hyperframe is like a data frame, except that each entry in a column can be a point pattern or a pixel image, as well as a single number or categorical value.
- four example datasets.

3 Formulating the problem

We view the experiment as involving a series of *'units'*. Each unit is subjected to a known set of experimental conditions (described by the values of the *covariates*), and each unit yields a *response* which is a spatial point pattern. The value of a particular covariate for each unit can be either a single value (numerical, logical or factor), or a pixel image.

Three important cases are:

independent replicates: We observe n different point patterns that can be regarded as independent replicates, i.e. independent realisations of the same point process. The 'responses' are the point patterns; there are no covariates.

replication in groups: there are K different experimental groups (e.g. control, aspirin, nurofen). In group k ($k = 1, \ldots, K$) we observe n_k point patterns which can be regarded as independent replicates within this group. We regard this as an experiment with $n = \sum_k n_k$ units. The responses are the point patterns; there is one covariate which is a factor (categorical variable) identifying which group each point pattern belongs to.

general case: there are covariates other than factors that influence the response. The point patterns are assumed to be independent, but no two patterns have the same distribution.

Examples of these three cases are given in the datasets `waterstriders`, `pyramidal` and `demohyper` respectively, which are installed in `spatstat`.

4 Installed datasets

The following datasets are currently installed in `spatstat`.

- `waterstriders`: Penttinen's [7] waterstriders data recording the locations of insect larvae on a pond in 3 independent experiments.
- `pyramidal`: data from Diggle, Lange and Benes [6] on the locations of pyramidal neurons in human brain, 31 human subjects grouped into 3 groups (controls, schizoaffective and schizophrenic).
- `flu`: data from Chen et al [5] giving the locations of two different virus proteins on the membranes of cells infected with influenza virus; 41 multitype point patterns divided into two virus types (wild and mutant) and two stain types.
- `simba`: simulated data from an experiment with two groups and 5 replicate point patterns per group.
- `demohyper`: simulated data from an experiment with two groups in which each experimental unit has a point pattern response and a pixel image covariate.

5 Lists of objects

First we need a convenient way to store the *responses* from all the units in an experiment.

An individual point pattern is stored as an object of class `"ppp"`. The easiest way to store all the responses is to form a list of `"ppp"` objects.

5.1 Waterstriders data

The `waterstriders` data are an example of this type. The data consist of 3 independent point patterns representing the locations of insect larvae on a pond. See `help(waterstriders)`.

```
> waterstriders

List of point patterns

Component 1:
Planar point pattern: 38 points
window: rectangle = [0, 48.1] x [0, 48.1] cm

Component 2:
Planar point pattern: 36 points
window: rectangle = [0, 48.8] x [0, 48.8] cm

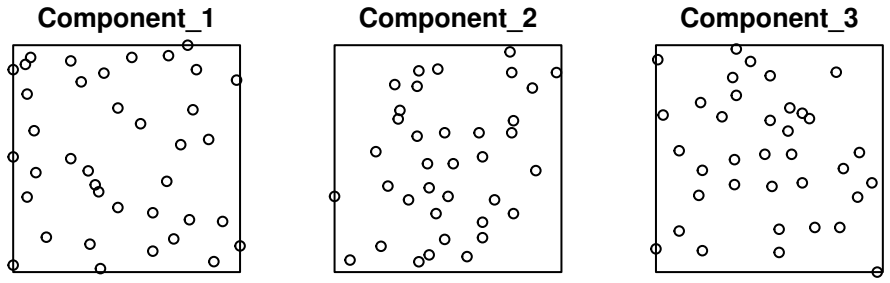
Component 3:
Planar point pattern: 36 points
window: rectangle = [0, 46.4] x [0, 46.4] cm
```

The `waterstriders` dataset is a list of point patterns. It is a list, each of whose entries is a point pattern (object of class `"ppp"`). Note that the observation windows of the three point patterns are `not` identical.

5.2 The class `solist`

For convenience, the `waterstriders` dataset also belongs to the class `"solist"` (spatial object list). This is a simple mechanism to allow us to handle the list neatly — for example, we can provide special methods for printing, plotting and summarising the list.

```
> plot(waterstriders, main="")
```



Notice that the plot method displays each entry of the list in a separate panel. There's also the summary method:

```
> summary(waterstriders)

Summary of 3 point patterns

Component 1:
Planar point pattern:  38 points
Average intensity 0.01642455 points per square cm

Coordinates are given to 2 decimal places
i.e. rounded to the nearest multiple of 0.01 cm

Window: rectangle = [0, 48.1] x [0, 48.1] cm
Window area = 2313.61 square cm
Unit of length: 1 cm

Component 2:
Planar point pattern:  36 points
Average intensity 0.0151169 points per square cm

Coordinates are given to 2 decimal places
i.e. rounded to the nearest multiple of 0.01 cm

Window: rectangle = [0, 48.8] x [0, 48.8] cm
Window area = 2381.44 square cm
```

```
Unit of length: 1 cm

Component 3:
Planar point pattern:  36 points
Average intensity 0.01672117 points per square cm

Coordinates are given to 2 decimal places
i.e. rounded to the nearest multiple of 0.01 cm

Window: rectangle = [0, 46.4] x [0, 46.4] cm
Window area = 2152.96 square cm
Unit of length: 1 cm
```

5.3 Creating a `solist`

For example, here is a simulated dataset containing three independent realisations of the Poisson process with intensity 100.

```
> X <- solist(rpoispp(100), rpoispp(100), rpoispp(100))
```

Then it can be printed and plotted.

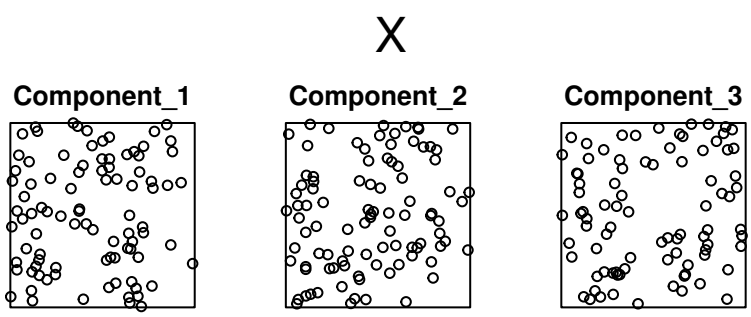
```
> plot(X)
> X

List of point patterns

Component 1:
Planar point pattern: 91 points
window: rectangle = [0, 1] x [0, 1] units

Component 2:
Planar point pattern: 95 points
window: rectangle = [0, 1] x [0, 1] units

Component 3:
Planar point pattern: 86 points
window: rectangle = [0, 1] x [0, 1] units
```



Each element of a `solist` must be an object which has two-dimensional spatial coordinates: for example a point pattern (class `ppp`), a window (class `owin`), a pixel image (class `im`) and so on. The different elements of a `solist` can be different types of objects:

```
> A <- solist(cells, density(cells), quadratcount(cells))
```

To convert an existing list to the class `solist`, use `as.solist`. If some elements of the list are not spatial objects, the default action is to terminate with an error. However if `demote=TRUE`, no error occurs, and the result is an `anylist` as described below.

5.4 The class `anylist`

An `anylist` is a list of objects which do not necessarily contain spatial coordinates. This is a convenient trick for handling multiple objects which belong to a non-spatial data type. For example, the estimated K-functions of each of the water strider point patterns can be saved as

```
> K <- as.anylist(lapply(waterstriders, Kest))
```

Each element `K[[i]]` is a function value table object of class `"fv"`. These three K-functions can be printed, plotted and summarised, for example

```
> plot(K)
```

An `anylist` can be created by the function `anylist()`. To convert an existing list to an `anylist`, use `as.anylist`.

6 Hyperframes

A *hyperframe* is like a data frame, except that its entries can be objects of any kind. A hyperframe is effectively a two-dimensional array in which each column consists of values of one type (as in a data frame) or consists of objects of one class.

The entries in a hyperframe can be point patterns, pixel images, windows, or any other objects.

To analyse an experiment, we will store **all** the data from the experiment in a single hyperframe. The rows of the hyperframe will correspond to different experimental units, while the columns represent different variables (response variables or covariates).

6.1 Creating hyperframes

The function `hyperframe` will create a hyperframe.

```
> hyperframe(...)
```

The arguments `...` are any number of arguments of the form `tag=value`. Each `value` will become a column of the array. The `tag` determines the name of the column.

Each `value` can be either

- an atomic vector or factor (i.e. numeric vector, integer vector, character vector, logical vector, complex vector or factor)
- a list of objects which are all of the same class
- one atomic value, which will be replicated to make an atomic vector or factor

- one object, which will be replicated to make a list of identical objects.

All columns (vectors, factors and lists) must be of the same length, if their length is greater than 1.

For example, here is a hyperframe containing a column of numbers and a column of *functions*:

```
> H <- hyperframe(X=1:3, Y=list(sin,cos,tan))
> H

Hyperframe:
  X          Y
1 1 (function)
2 2 (function)
3 3 (function)
```

Note that a column of character strings will be converted to a factor, unless you set `stringsAsFactors=FALSE` in the call to `hyperframe`. This is the same behaviour as for the function `data.frame`.

```
> G <- hyperframe(X=1:3, Y=letters[1:3], Z=factor(letters[1:3]),
+                 W=list(rpoispp(100),rpoispp(100), rpoispp(100)),
+                 U=42,
+                 V=rpoispp(100), stringsAsFactors=FALSE)
> G

Hyperframe:
  X Y Z     W  U     V
1 1 a a (ppp) 42 (ppp)
2 2 b b (ppp) 42 (ppp)
3 3 c c (ppp) 42 (ppp)
```

This hyperframe has 3 rows and 6 columns. The columns named `U` and `V` are constant (all entries in a column are the same). The column named `Y` is a character vector while `Z` is a factor.

6.2 Hyperframes of data

To analyse an experiment, we will store **all** the data from the experiment in a single hyperframe. The rows of the hyperframe will correspond to different experimental units, while the columns represent different variables (response variables or covariates).

Several examples of hyperframes are provided with the package, including `demohyper`, `flu`, `simba` and `pyramidal`, described above.

The `simba` dataset contains simulated data from an experiment with a 'control' group and a 'treatment' group, each group containing 5 experimental units. The responses in the control group are independent Poisson point patterns with intensity 80. The responses in the treatment group are independent realisations of a Strauss process (see `help(simba)` for details). The `simba` dataset is a hyperframe with 10 rows and 2 columns: `Points` (the point patterns) and `group` (a factor with levels `control` and `treatment`).

```
> simba
```

```
Hyperframe:
   Points     group
1   (ppp)   control
2   (ppp)   control
3   (ppp)   control
4   (ppp)   control
5   (ppp)   control
6   (ppp) treatment
7   (ppp) treatment
8   (ppp) treatment
9   (ppp) treatment
10  (ppp) treatment
```

The `pyramidal` dataset contains data from Diggle, Lange and Benes [6] on the locations of pyramidal neurons in human brain. One point pattern was observed in each of 31 human subjects. The subjects were classified into 3 groups (controls, schizoaffective and schizophrenic). The `pyramidal` dataset is a hyperframe with 31 rows and 2 columns: `Neurons` (the point patterns) and `group` (a factor with levels `control`, `schizoaffective` and `schizophrenic`).

```
> pyramidal
```

```
Hyperframe:
   Neurons           group
1    (ppp)         control
2    (ppp)         control
3    (ppp)         control
4    (ppp)         control
5    (ppp)         control
6    (ppp)         control
7    (ppp)         control
8    (ppp)         control
9    (ppp)         control
10   (ppp)         control
11   (ppp)         control
12   (ppp)         control
13   (ppp) schizoaffective
14   (ppp) schizoaffective
15   (ppp) schizoaffective
16   (ppp) schizoaffective
17   (ppp) schizoaffective
18   (ppp) schizoaffective
19   (ppp) schizoaffective
20   (ppp) schizoaffective
21   (ppp) schizoaffective
22   (ppp)   schizophrenic
23   (ppp)   schizophrenic
24   (ppp)   schizophrenic
25   (ppp)   schizophrenic
26   (ppp)   schizophrenic
```

```
27   (ppp)   schizophrenic
28   (ppp)   schizophrenic
29   (ppp)   schizophrenic
30   (ppp)   schizophrenic
31   (ppp)   schizophrenic
```

The `waterstriders` dataset is not a hyperframe; it's just a list of point patterns. It can easily be converted into a hyperframe:

```
> ws <- hyperframe(Striders=waterstriders)
```

6.3 Columns of a hyperframe

Individual columns of a hyperframe can be extracted using `$`:

```
> H$X
```

```
[1] 1 2 3
```

```
> H$Y
```

```
1:
function (x)  .Primitive("sin")

2:
function (x)  .Primitive("cos")

3:
function (x)  .Primitive("tan")
```

The result of `$` is a vector or factor if the column contains atomic values; otherwise it is a list of objects (with class `"solist"` or `"anylist"` to make it easier to print and plot).

Individual columns can also be assigned (overwritten or created) using `$<-`:

```
> H$U <- letters[1:3]
> H
```

```
Hyperframe:
  X          Y U
1 1 (function) a
2 2 (function) b
3 3 (function) c
```

This can be used to build up a hyperframe column-by-column:

```
> G <- hyperframe()
> G$X <- waterstriders
> G$Y <- 1:3
> G
```

```
Hyperframe:
      X Y
1 (ppp) 1
2 (ppp) 2
3 (ppp) 3
```

6.4 Subsets of a hyperframe

Other subsets of a hyperframe can be extracted with `[`:

```
> H[,1]

Hyperframe:
  X
1 1
2 2
3 3

> H[2,]

Hyperframe:
  X          Y U
2 2 (function) b

> H[2:3, ]

Hyperframe:
  X          Y U
2 2 (function) b
3 3 (function) c

> H[1,1]

Hyperframe:
  X
1 1
```

The result of `[` is a hyperframe, unless you set `drop=TRUE` and the subset consists of only one element or one column:

```
> H[,1,drop=TRUE]

[1] 1 2 3

> H[1,1,drop=TRUE]

[1] 1

> H[1,2,drop=TRUE]

function (x)  .Primitive("sin")
```

There is also a method for `[<-` that allows you to assign values to a subset of a hyperframe.

7 Plotting

7.1 Plotting a `solist` object

The plot method for `solist` objects has formal arguments

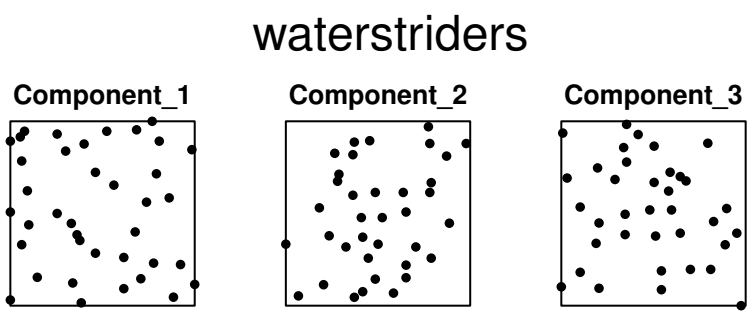
```
> plot.solist(x, ..., main, arrange = TRUE, nrows = NULL, ncols = NULL,
+             main.panel = NULL, mar.panel = c(2, 1, 1, 2), hsep = 0, vsep = 0,
+             panel.begin = NULL, panel.end = NULL, panel.args = NULL,
+             panel.begin.args = NULL, panel.end.args = NULL, panel.vpad = 0.2,
+             plotcommand = "plot", do.plot = TRUE, adorn.left = NULL,
+             adorn.right = NULL, adorn.top = NULL, adorn.bottom = NULL,
+             adorn.size = 0.2, adorn.args = list(), equal.scales = FALSE,
+             halign = FALSE, valign = FALSE)
```

where `main` is a title for the entire page, and `main.panel` is a character vector of titles for each plot panel.

If `arrange=TRUE` then the entries of the list are displayed in separate panels on the same page (with `nrows` rows and `ncols` columns of panels), while if `arrange=FALSE` then the entries are just plotted as a series of plot frames.

The extra arguments `...` will be passed to the plot method that displays each entry of the list. Suitable arguments depend on the type of entries.

```
> plot(waterstriders, pch=16, nrows=1)
```



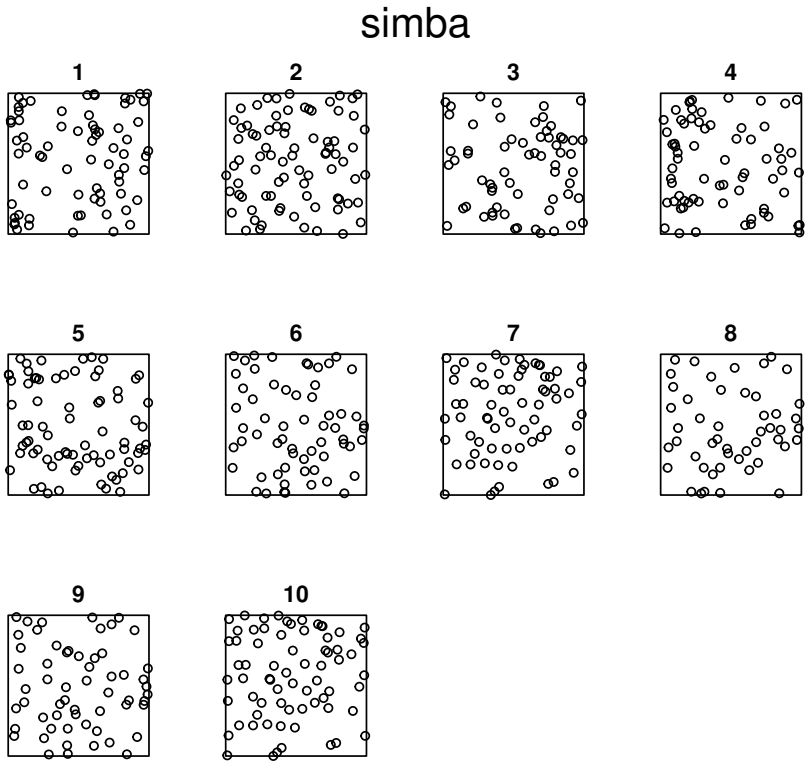
For the meaning of the other arguments, see the help for `plot.solist`.

7.2 Plotting a hyperframe

7.2.1 Plotting one column

If `h` is a hyperframe, then the default action of `plot(h)` is to extract the first column of `h` and plot each of the entries in a separate panel on one page (actually using the plot method for class `"solist"` or `"anylist"`).

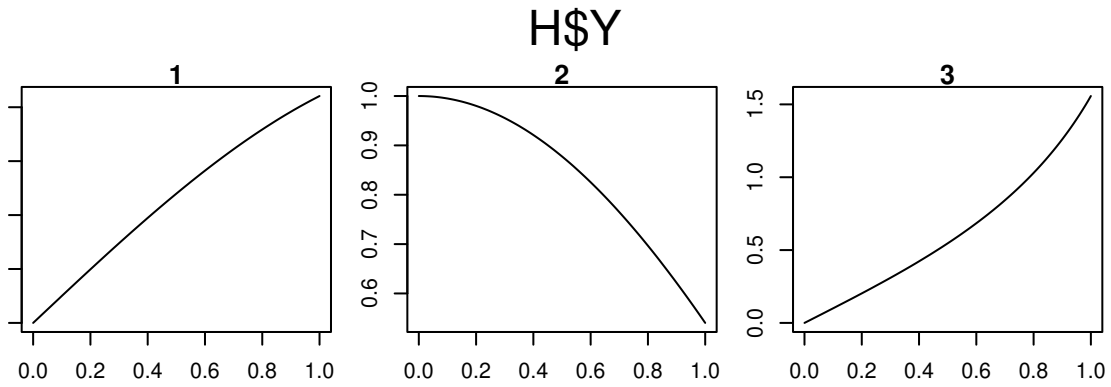
```
> plot(simba)
```



This only works if the entries in the first column are objects for which a plot method is defined (for example, point patterns, images, windows).

To select a different column, use `$` or `[`:

```
> H <- hyperframe(X=1:3, Y=list(sin,cos,tan))
> plot(H$Y)
```



Since `H$Y` is an `anylist`, the plot can be controlled using the arguments for `plot.anylist` (and, in this case, `plot.function`, since each entry is a function).

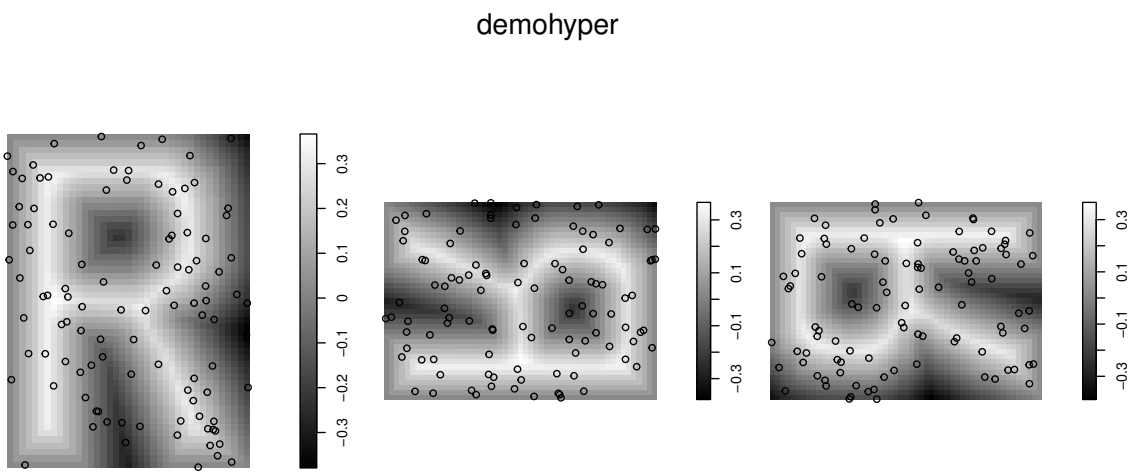
7.2.2 Complex plots

More generally, we can display any kind of higher-order plot involving one or more columns of a hyperframe:

```
> plot(h, e)
```

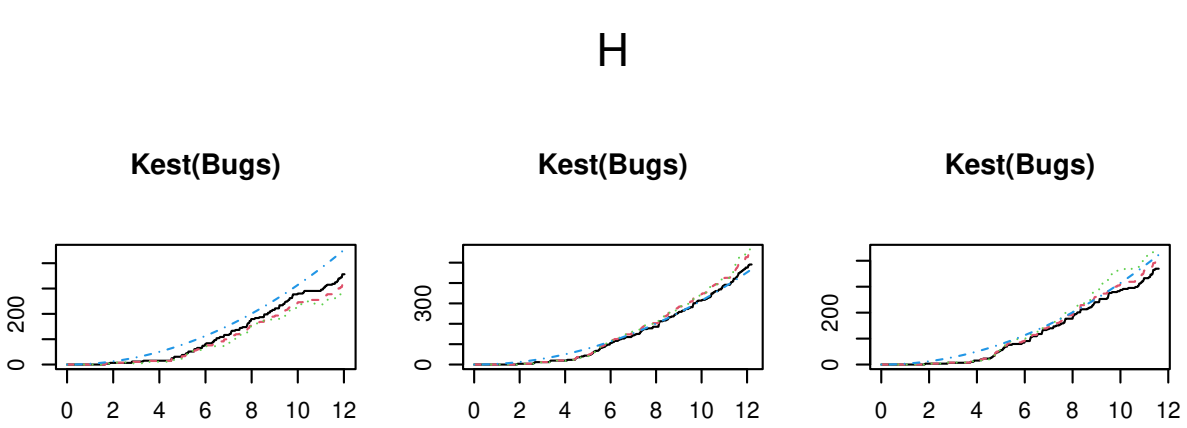
where `h` is a hyperframe and `e` is an R language call or expression that must be evaluated in each row to generate each plot panel.

```
> plot(demohyper, quote({ plot(Image, main=""); plot(Points, add=TRUE) }))
```



Note the use of `quote`, which prevents the code inside the braces from being evaluated immediately.
To plot the K-functions of each of the patterns in the `waterstriders` dataset,

```
> H <- hyperframe(Bugs=waterstriders)
> plot(H, quote(plot(Kest(Bugs), legend=FALSE)), marsize=2)
```



8 Data analysis

8.1 Computing with hyperframes

Often we want to perform some computation on each row of a hyperframe.

In a data frame, this can be done using the command `with`:

```
> df <- data.frame(A=1:10, B=10:1)
> with(df, A-B)

 [1] -9 -7 -5 -3 -1  1  3  5  7  9
```

In this example, the expression `A-B` is evaluated in each row of the data frame, and the result is a vector containing the computed values for each row. The function `with` is generic, and has a method for data frames, `with.data.frame`. The computation above was executed by `with.data.frame`.

The same syntax is available for hyperframes using the method `with.hyperframe`:

```
> with(h,e)
```

Here `h` is a hyperframe, and `e` is an R language construct involving the names of columns in `h`. For each row of `h`, the expression `e` will be evaluated in such a way that each entry in the row is identified by its column name.

```
> H <- hyperframe(Bugs=waterstriders)
> with(H, npoints(Bugs))

 1  2  3
38 36 36

> with(H, distmap(Bugs))

List of pixel images

1:
real-valued pixel image
128 x 128 pixel array (ny, nx)
enclosing rectangle: [0, 48.1] x [0, 48.1] cm

2:
real-valued pixel image
128 x 128 pixel array (ny, nx)
enclosing rectangle: [0, 48.8] x [0, 48.8] cm

3:
real-valued pixel image
128 x 128 pixel array (ny, nx)
enclosing rectangle: [0, 46.4] x [0, 46.4] cm
```

The result of `with.hyperframe` is a list of objects (of class `"solist"` or `"anylist"`), or a vector or factor if appropriate.

Notice that (unlike the situation for data frames) the operations in the expression `e` do not have to be vectorised. For example, `distmap` expects a single point pattern, and is not vectorised to deal with a list of point patterns. Instead, the expression `distmap(Bugs)` is evaluated separately in each row of the hyperframe.

8.2 Summary statistics

One application of `with.hyperframe` is to calculate summary statistics for each row of a hyperframe.

For example, the number of points in a point pattern `X` is returned by `npoints(X)`. To calculate this for each of the responses in the `simba` dataset,

```
> with(simba, npoints(Points))
```

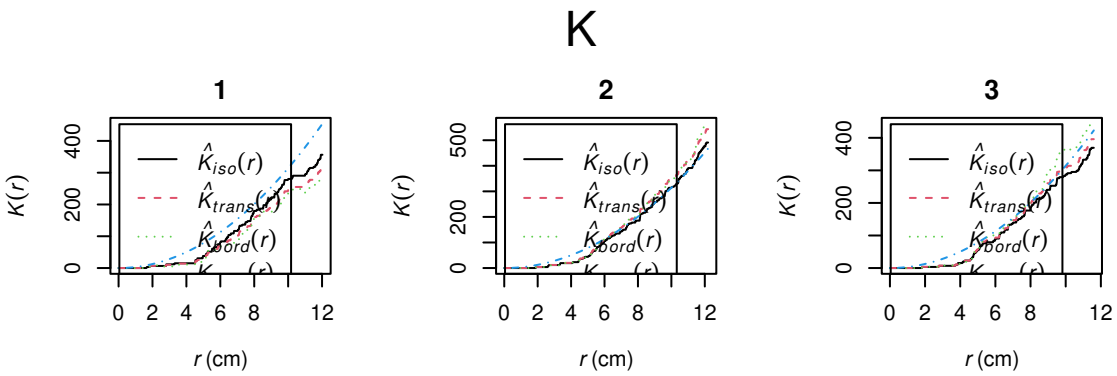
```
 1  2  3  4  5  6  7  8  9 10
71 80 63 68 75 61 68 49 58 70
```

The summary statistic can be any kind of object. For example, to compute the empirical K-functions for each of the patterns in the `waterstriders` dataset,

```
> H <- hyperframe(Bugs=waterstriders)
> K <- with(H, Kest(Bugs))
```

To plot these K-functions you can then just type

```
> plot(K)
```



The summary statistic for each row could be a numeric vector:

```
> H <- hyperframe(Bugs=waterstriders)
> with(H, nndist(Bugs))
```

```
1:
 [1] 9.190609 5.850214 2.817836 1.850000 5.500445 5.966071 1.850000 7.107468 5.500445 9.190609
[11] 4.516426 4.964071 4.964071 3.309381 5.635672 1.654690 1.654690 5.635672 5.258688 5.850214
[21] 6.792297 5.850214 5.140204 7.107468 7.288155 5.140204 6.023164 5.258688 7.117092 5.500445
[31] 6.023164 6.472357 6.365721 8.794800 4.636087 6.365721 5.153504 4.636087

2:
 [1]  7.288155  7.840159  7.288155  5.336216  4.824210  1.886637  1.886637  5.140204  3.350493
[10]  5.500445  2.668108  3.350493  5.193197  4.470727  2.668108  4.516426  4.086784  4.709331
[19]  5.966071  4.470727  5.550000  5.258688  5.232590  5.232590  3.330000  3.330000  5.032544
[28]  2.616295  2.616295  5.032544  4.455390  4.455390  5.550000  9.648420  6.158027 11.682864

3:
 [1]  5.990409 11.372357  8.007128  5.990409  6.208679  5.171779  5.258564  5.171779  6.215143
[10]  5.258564  5.110000  5.110000  3.724258  3.724258  3.869186  3.869186  5.470000  4.251459
[19]  4.968581  6.242828  4.740000  4.740000  4.251459  2.773193  4.805767  6.237123  1.828004
[28]  1.828004  5.110000 10.888329  5.110000  4.638620  4.136581  4.638620  4.136581 11.917739
```

The result is a list, each entry being a vector of nearest neighbour distances. To find the minimum interpoint distance in each pattern:

```
> with(H, min(nndist(Bugs)))
```

```
       1        2        3
1.654690 1.886637 1.828004
```

8.3 Generating new columns

New columns of a hyperframe can be created by computation from the existing columns.

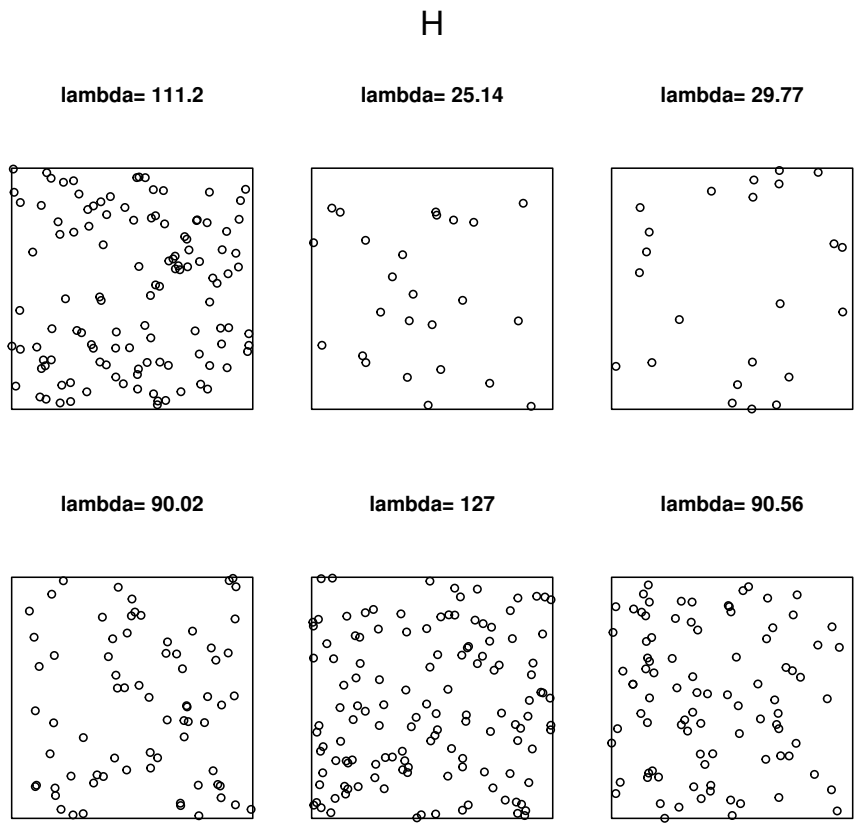
For example, I can add a new column to the `simba` dataset that contains pixel images of the distance maps for each of the point pattern responses.

```
> simba$Dist <- with(simba, distmap(Points))
```

8.4 Simulation

This can be useful for simulation. For example, to generate Poisson point patterns with different intensities, where the intensities are given by a numeric vector `lambda`:

```
> lambda <- rexp(6, rate=1/50)
> H <- hyperframe(lambda=lambda)
> H$Points <- with(H, rpoispp(lambda))
> plot(H, quote(plot(Points, main=paste("lambda=", signif(lambda, 4)))))
```



It's even simpler to generate 10 independent Poisson point patterns with the *same* intensity 50, say:

```
> H$X <- with(H, rpoispp(50))
```

The expression `rpoispp(50)` is evaluated once in each row, yielding a different point pattern in each row because of the randomness.

9 Exploratory data analysis

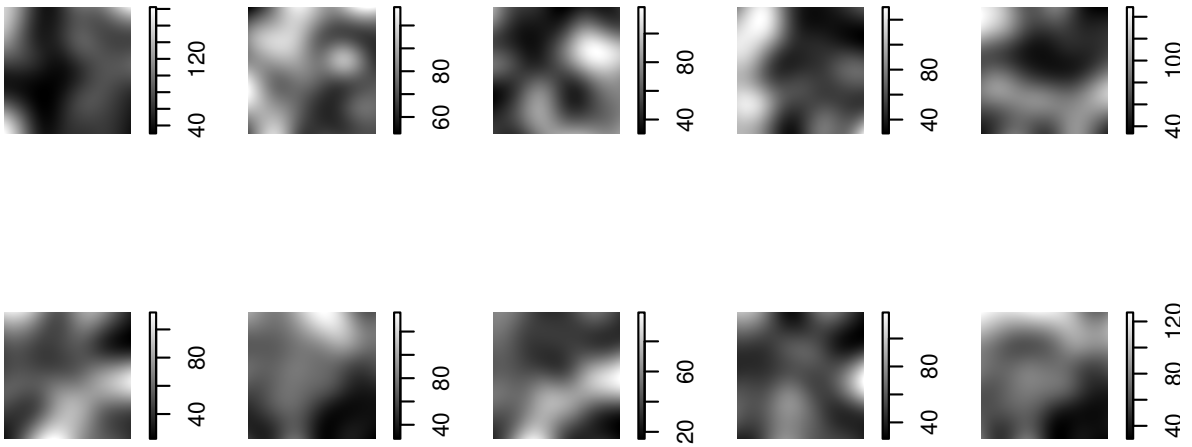
Before fitting models to the data, it is prudent to explore the data to detect unusual features and to suggest appropriate models.

9.1 Exploring spatial trend and covariate effects

Points may be distributed non-uniformly either because they are intrinsically non-uniform ("spatial trend") or because their abundance depends on a spatial covariate ("covariate effects").

Non-uniformity of a point pattern can be investigated using the kernel smoothed intensity. This is the convolution of the point pattern with a smooth density called the kernel. Effectively each point in the pattern is replaced by a copy of the kernel, and the sum of all copies of the kernel is the kernel-smoothed intensity function. It is computed by `density.ppp` separately for each point pattern.

```
> plot(simba, quote(plot(density(Points), main="")),
+      main="", nrows=2, marsize=1.5)
```

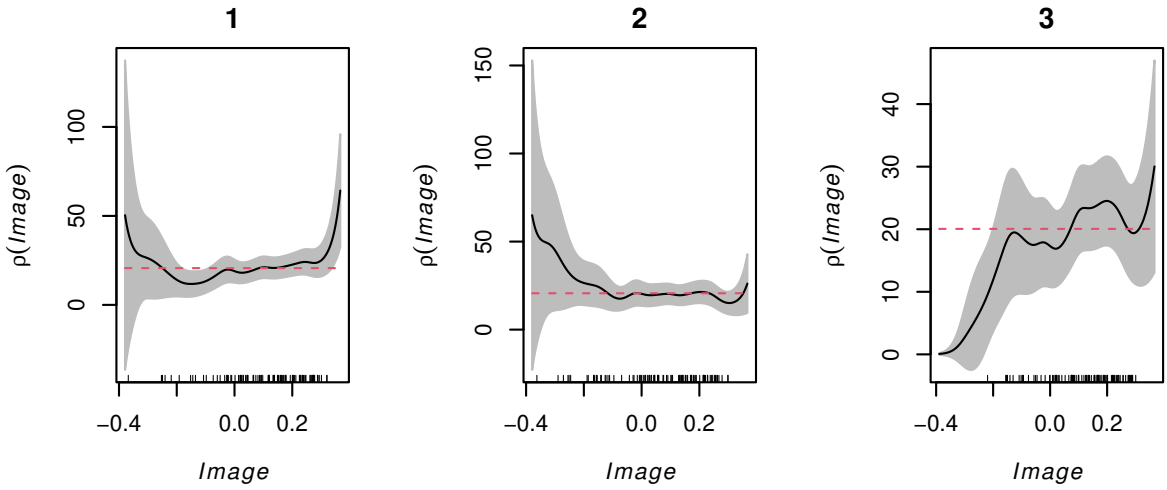


Covariate effects due to a real-valued spatial covariate (a real-valued pixel image) can be investigated using the command `rhohat`. This uses a kernel smoothing technique to fit a model of the form

$$\lambda(u) = \rho(Z(u))$$

where $\lambda(u)$ is the point process intensity at a location u, and $Z(u)$ is the value of the spatial covariate at that location. Here ρ is an unknown, smooth function which is to be estimated. The function ρ expresses the effect of the spatial covariate on the point process intensity. If ρ turns out to be constant, then the covariate has no effect on point process intensity (and the constant value of ρ is the constant intensity of the point process).

```
> rhos <- with(demohyper, rhohat(Points, Image))
> plot(rhos, legend=FALSE, main="")
```



9.2 Exploring interpoint interaction

See Chapter 16 of [?].

10 Fitting models of spatial trend

The command `mppm` fits models to multiple point patterns. Its syntax is very similar to that of `lm` and `glm`:

```
> mppm(formula, data, interaction, ...)
```

where `formula` is a formula describing the systematic trend part of the model, `data` is a hyperframe containing all the data (responses and covariates), and `interaction` determines the stochastic interpoint interaction part of the model.

For example:

```
> mppm(Points ~ group, simba, Poisson())
```

Note that the formula has a left hand side, which identifies the response. This should be the name of a column of `data`.

10.1 Trend formula

The right side of `formula` is an expression for the linear predictor (effectively the **logarithm** of the spatial trend).

The variables appearing in the right hand side of `formula` should be either

- names of columns in `data`
- objects in the R global environment (such as `pi` and `log`)
- the reserved names `x`, `y` (representing Cartesian coordinates), `marks` (representing mark values attached to points) or `id` (a factor representing the row number in the hyperframe).

10.1.1 Design covariates

The variables in the trend could be 'design covariates'.

For example, to fit a model to the `simba` dataset in which all patterns are independent replicates of the same uniform Poisson process, with the same constant intensity:

```
> mppm(Points ~ 1, simba)

Point process model fitted to 10 point patterns
Call: mppm(Points ~ 1, simba)
Log trend formula: ~1

Fitted trend coefficients:
(Intercept)
    4.19419

Interaction for all patterns:        Poisson process
```

To fit a model in which the two groups of patterns (control and treatment groups) each consist of independent replicates of a uniform Poisson process, but with possibly different intensity in each group:

```
> mppm(Points ~ group, simba)

Point process model fitted to 10 point patterns
Call: mppm(Points ~ group, simba)
Log trend formula: ~group

Fitted trend coefficients:
   (Intercept) grouptreatment
     4.2682979     -0.1541507

Interaction for all patterns:        Poisson process
```

To fit a uniform Poisson process to each pattern, with different intensity for each pattern:

```
> mppm(Points ~ id, simba)

Point process model fitted to 10 point patterns
Call: mppm(Points ~ id, simba)
Log trend formula: ~id

Fitted trend coefficients:
(Intercept)         id2         id3         id4         id5         id6         id7         id8
 4.26267988  0.11934676 -0.11954515 -0.04317217  0.05480824 -0.15180601 -0.04317217 -0.37085958
        id9        id10
-0.20223687 -0.01418463

Interaction for all patterns:        Poisson process
```

10.1.2 Spatial covariates

The variables in the trend could be 'spatial covariates'.

For example, the `demohyper` dataset has a column `Image` containing pixel images.

```
> mppm(Points ~ Image, data=demohyper)

Point process model fitted to 3 point patterns
Call: mppm(Points ~ Image, data = demohyper)
Log trend formula: ~Image

Fitted trend coefficients:
(Intercept)       Image
  2.9840789   0.3924519

Interaction for all patterns:        Poisson process
```

This model postulates that each pattern is a Poisson process with intensity of the form

$$\lambda(u) = \exp(\beta_0 + \beta_1 Z(u))$$

at location u, where β_0, β_1 are coefficients to be estimated, and $Z(u)$ is the value of the pixel image `Image` at location u.

It may or may not be appropriate to assume that the intensity of the points is an exponential function of the image pixel value Z. If instead we wanted the intensity $\lambda(u)$ to be *proportional* to $Z(u)$, the appropriate model is

```
> mppm(Points ~ offset(log(Image)), data=demohyper)
```

which corresponds to an intensity proportional to `Image`,

$$\lambda(u) = \exp(\beta_0 + \log Z(u)) = e^{\beta_0}\ Z(u).$$

The `offset` indicates that there is no coefficient in front of $\log Z(u)$.

Alternatively we could allow a coefficient:

```
> mppm(Points ~ log(Image), data=demop)
```

which corresponds to a gamma transformation of `Image`,

$$\lambda(u) = \exp(\beta_0 + \beta_1 \log Z(u)) = e^{\beta_0}\ Z(u)^{\beta_1}.$$

11 Interpoint interaction

The stochastic interpoint interaction in a point process model is specified by the arguments `interaction` and (optionally) `iformula` in

```
> mppm(formula, data, interaction, ..., iformula=NULL)
```

11.1 Same interaction for all patterns

In the simplest case, the argument `interaction` is one of the familiar objects that describe the point process interaction structure. It is an object of class `"interact"` created by calling one of the functions

`Poisson()`	the Poisson point process
`Hardcore()`	the hard core process
`Strauss()`	the Strauss process
`StraussHard()`	the Strauss/hard core point process
`Softcore()`	pairwise interaction, soft core potential
`PairPiece()`	pairwise interaction, piecewise constant
`DiggleGatesStibbard()`	Diggle-Gates-Stibbard pair potential
`DiggleGratton()`	Diggle-Gratton pair potential
`Fiksel()`	Fiksel pair potential
`LennardJones()`	Lennard-Jones pair potential
`Pairwise()`	pairwise interaction, user-supplied potential
`AreaInter()`	area-interaction potential
`Geyer()`	Geyer's saturation process
`BadGey()`	multiscale Geyer saturation process
`Saturated()`	Saturated pair model, user-supplied potential
`OrdThresh()`	Ord process, threshold potential
`Ord()`	Ord model, user-supplied potential
`MultiStrauss()`	multitype Strauss process
`MultiStraussHard()`	multitype Strauss/hard core process
`Concom()`	connected component interaction
`Hybrid()`	hybrid of several interactions

In this 'simple' usage of `mppm`, the point process model assumes that all point patterns have exactly the same interpoint interaction, (with the same interaction parameters), and only differ in their spatial trend.

11.2 Hyperframe of interactions

More generally the argument `interaction` can be a hyperframe containing objects of class `"interact"`.

For example, we might want to fit a Strauss process to each point pattern, but with a different Strauss interaction radius for each pattern.

Then `radii` is a vector of numbers which we could use as the values of the interaction radius for each case. First we need to make the interaction objects:

```
> Rad <- hyperframe(R=radii)
> Str <- with(Rad, Strauss(R))
```

Then we put them into a hyperframe and fit the model:

```
> Int <- hyperframe(str=Str)
> mppm(Points ~ 1, simba, interaction=Int)

Point process model fitted to 10 point patterns
Call: mppm(Points ~ 1, simba, interaction = Int)
Log trend formula: ~1
```

```
Fitted trend coefficients:
(Intercept)
   4.372376

Fitted coefficients of interpoint interaction:
       str
-0.2035399

Interaction for each pattern:        Strauss process

Interaction on row 1:        Strauss(r = 0.06525, gamma = 0.8158)

Interaction on row 2:        Strauss(r = 0.06588, gamma = 0.8158)

Interaction on row 3:        Strauss(r = 0.06567, gamma = 0.8158)

Interaction on row 4:        Strauss(r = 0.06511, gamma = 0.8158)

Interaction on row 5:        Strauss(r = 0.05764, gamma = 0.8158)

Interaction on row 6:        Strauss(r = 0.07948, gamma = 0.8158)

Interaction on row 7:        Strauss(r = 0.07608, gamma = 0.8158)

Interaction on row 8:        Strauss(r = 0.09014, gamma = 0.8158)

Interaction on row 9:        Strauss(r = 0.07943, gamma = 0.8158)

Interaction on row 10:        Strauss(r = 0.07684, gamma = 0.8158)
```

An important constraint is that all of the interaction objects in one column must be *instances of the same process* (e.g. Strauss) albeit possibly having different parameter values. For example, you cannot put Poisson and Strauss processes in the same column.

11.3 Interaction formula

If `interaction` is a hyperframe, then the additional argument `iformula` may be used to fully specify the interaction.

(An `iformula` is also required if `interaction` has more than one column.)

The `iformula` should be a formula without a left hand side. Variables on the right hand side are typically the names of columns in `interaction`.

11.3.1 Selecting one column

If the right hand side of `iformula` is a single name, then this identifies the column in `interaction` to be used as the interpoint interaction structure.

```
> h <- hyperframe(Y=waterstriders)
> g <- hyperframe(po=Poisson(), str4 = Strauss(4), str7= Strauss(7))
> mppm(Y ~ 1, data=h, interaction=g, iformula=~str4)
```

```
Point process model fitted to 3 point patterns
Call: mppm(Y ~ 1, data = h, interaction = g, iformula = ~str4)
Log trend formula: ~1

Fitted trend coefficients:
(Intercept)
  -3.522586

Fitted coefficients of interpoint interaction:
     str4
-1.364244

Interaction for each pattern:        Strauss process

Summary table of interactions:
  row interaction
1   1           A
2   2           A
3   3           A

key to interaction table:
$A
Strauss process
Interaction distance:        4
Fitted interaction parameter gamma:        0.2555737


=== Interactions on each row ====:

Interaction on row 1:        Strauss(r = 4, gamma = 0.2556)

Interaction on row 2:        Strauss(r = 4, gamma = 0.2556)

Interaction on row 3:        Strauss(r = 4, gamma = 0.2556)
```

11.3.2 Interaction depending on design

The `iformula` can also involve columns of `data`, but only those columns that are vectors or factors. This allows us to specify an interaction that depends on the experimental design. [This feature is **experimental**.] For example

```
> fit <- mppm(Points ~ 1, simba, Strauss(0.07), iformula = ~Interaction*group)
```

Since `Strauss(0.1)` is not a hyperframe, it is first converted to a hyperframe with a single column named `Interaction`.

The `iformula = Interaction*group` specifies (since `group` is a factor) that the interpoint interaction shall have a different coefficient in each experimental group. That is, we fit a model which has two different values for the Strauss interaction parameter γ, one for the control group and one for the treatment group.

When you print the result of such a fit, the package tries to do 'automatic interpretation' of the fitted model (translating the fitted interaction coefficients into meaningful numbers like γ). This will be successful in *most* cases:

```
> fit

Point process model fitted to 10 point patterns
Call: mppm(Points ~ 1, simba, Strauss(0.07), iformula = ~Interaction *
Log trend formula: ~1

Fitted trend coefficients:
   (Intercept) grouptreatment
     4.1979786      0.4389735

Fitted coefficients of interpoint interaction:
               Interaction Interaction:grouptreatment
                0.06839639                -0.95458813

Interaction for each pattern:        Strauss process

Interaction depends on design covariate:        'group'

Summary table of interactions:
   row interaction
1    1           A
2    2           A
3    3           A
4    4           A
5    5           A
6    6           B
7    7           B
8    8           B
9    9           B
10  10           B

key to interaction table:
$A
Strauss process
Interaction distance:        0.07
Fitted interaction parameter gamma:        1.0707897

$B
Strauss process
Interaction distance:        0.07
Fitted interaction parameter gamma:        0.4122226


=== Interactions on each row ====:
```

```
Interaction on row 1:        Strauss(r = 0.07, gamma = 1.071)

Interaction on row 2:        Strauss(r = 0.07, gamma = 1.071)

Interaction on row 3:        Strauss(r = 0.07, gamma = 1.071)

Interaction on row 4:        Strauss(r = 0.07, gamma = 1.071)

Interaction on row 5:        Strauss(r = 0.07, gamma = 1.071)

Interaction on row 6:        Strauss(r = 0.07, gamma = 0.4122)

Interaction on row 7:        Strauss(r = 0.07, gamma = 0.4122)

Interaction on row 8:        Strauss(r = 0.07, gamma = 0.4122)

Interaction on row 9:        Strauss(r = 0.07, gamma = 0.4122)

Interaction on row 10:        Strauss(r = 0.07, gamma = 0.4122)
```

Thus we see that the estimate of the Strauss parameter γ for the control group is 1.071, and for the treatment group 0.4122 (the correct values in this simulated dataset were 1 and 0.5).

The fitted model can also be interpreted directly from the fitted canonical coefficients:

```
> coef(fit)
```

```
               (Intercept)                Interaction             grouptreatment
                4.19797857                 0.06839639                 0.43897349
Interaction:grouptreatment
               -0.95458813
```

The last output shows all the coefficients β_j in the linear predictor for the (log) conditional intensity.

The interpretation of the model coefficients, for any fitted model in R, depends on the *contrasts* which were applicable when the model was fitted. This is part of the core R system: see `help(contrasts)` or `options(contrasts)`. If you did not specify otherwise, the default is to use *treatment contrasts*. This means that, for an explanatory variable which is a `factor` with N levels, the first level of the factor is used as a baseline, and the fitted model coefficients represent the factor levels $2, 3, \ldots, N$ relative to this baseline.

In the output above, there is a coefficient for `(Intercept)` and one for `grouptreatment`. These are coefficients related to the `group` factor. According to the "treatment contrasts" rule, the `(Intercept)` coefficient is the estimated effect for the control group, and the `grouptreatment` coefficient is the estimated difference between the treatment and control groups. Thus the fitted first order trend is $\exp(4.198) = 66.55$ for the control group and $\exp(4.198 + 0.439) = 103.2$ for the treatment group. The correct values in this simulated dataset were 80 and 100.

The remaining coefficients in the output are `Interaction` and `Interaction:grouptreatment`. Recall that the Strauss process interaction term is $\gamma^{t(u,\boldsymbol{x})} = \exp(t(u,\boldsymbol{x}) \log \gamma)$ at a spatial location u, for a point pattern $\boldsymbol{x}$. Since we're using treatment contrasts, the coefficient `Interaction` is the estimate of $\log \gamma$ for the control group. The coefficient `Interaction:grouptreatment` is the estimate of the difference in $\log \gamma$ between the treatment and control groups. Thus the estimated Strauss interaction parameter γ is $\exp(0.0684) = 1.071$ for the control group and $\exp(0.0684 + (-0.9546)) = 0.4122$ for the treatment group. The correct values were 1 and 0.5.

11.3.3 Completely different interactions for different cases

In the previous example, when we fitted a Strauss model to all point patterns in the `simba` dataset, the fitted model for the patterns in the control group was close to Poisson ($\gamma \approx 1$). Suppose we now want to fit a model which *is* Poisson in the control group, and Strauss in the treatment group. The Poisson and Strauss interactions must be given as separate columns in a hyperframe of interactions:

```
> interaction=hyperframe(po=Poisson(), str=Strauss(0.07))
```

What do we write for the `iformula`? The following *will not* work:

```
> iformula=~ifelse(group=="control", po, str)
```

This does not work because the Poisson and Strauss models are 'incompatible' inside such expressions. The canonical sufficient statistics for the Poisson and Strauss processes do not have the same dimension. Internally in `mppm` we translate the symbols `po` and `str` into matrices; the dimensions of these matrices are different, so the `ifelse` expression cannot be evaluated.

Instead we need something like the following:

```
> iformula=~I((group=="control")*po) + I((group=="treatment") * str)
```

The letter `I` here is a standard R function that prevents its argument from being interpreted as a formula (thus the `*` is interpreted as multiplication instead of a model interaction). The expression `(group=="control")` is logical, and when multiplied by the matrix `po`, yields a matrix.

So the following does work:

```
> g <- hyperframe(po=Poisson(), str=Strauss(0.07))
> fit2 <- mppm(Points ~ 1, simba, g,
+              iformula=~I((group=="control")*po)
+                      + I((group=="treatment") * str))
> fit2

Point process model fitted to 10 point patterns
Call: mppm(Points ~ 1, simba, g, iformula = ~I((group == "control") *
Log trend formula: ~1

Fitted trend coefficients:
(Intercept)
   4.391413

Fitted coefficients of interpoint interaction:
   I((group == "control") * po) I((group == "treatment") * str)
                             NA                       -0.6994845

Active interactions:        (10 x 2 matrix)

Summary table of interactions:
   row interaction
1    1           A
2    2           A
3    3           A
```

```
4    4           A
5    5           A
6    6           B
7    7           B
8    8           B
9    9           B
10  10           B

key to interaction table:
$A
Poisson process


$B
Strauss process
Interaction distance:        0.07
Fitted interaction parameter gamma:        0.4968413


=== Interactions on each row ====:

Interaction on row 1:        Poisson process


Interaction on row 2:        Poisson process


Interaction on row 3:        Poisson process


Interaction on row 4:        Poisson process


Interaction on row 5:        Poisson process


Interaction on row 6:        Strauss(r = 0.07, gamma = 0.4968)

Interaction on row 7:        Strauss(r = 0.07, gamma = 0.4968)

Interaction on row 8:        Strauss(r = 0.07, gamma = 0.4968)

Interaction on row 9:        Strauss(r = 0.07, gamma = 0.4968)

Interaction on row 10:        Strauss(r = 0.07, gamma = 0.4968)
```

12 Studying the fitted model

Fitted models produced by `mppm` can be examined and validated in many ways.

12.1 Fits for each pattern

12.1.1 Subfits

The command `subfits` takes an `mppm` object and extracts, for each individual point pattern, the fitted point process model for that pattern *that is implied by the overall fit*. It returns a list of objects of class `ppm`.

```
> H <- hyperframe(W=waterstriders)
> fit <- mppm(W ~ 1, H)
> subfits(fit)

1:
Stationary Poisson process
Fitted to point pattern dataset 'Y'
Intensity: 0.01606306
For standard errors, type coef(summary(x))

2:
Stationary Poisson process
Fitted to point pattern dataset 'Y'
Intensity: 0.01606306
For standard errors, type coef(summary(x))

3:
Stationary Poisson process
Fitted to point pattern dataset 'Y'
Intensity: 0.01606306
For standard errors, type coef(summary(x))
```

In this example the result is a list of three `ppm` objects representing the implied fits for each of the three point patterns in the `waterstriders` dataset. Notice that **the fitted coefficients are the same** in all three models.

Note that there are some unresolved difficulties with the implementation of `subfits`. Two completely different implementations are supplied in the package; they are called `subfits.old` and `subfits.new`.The old version would occasionally crash. Unfortunately the newer version `subfits.new` is quite memory-hungry and sometimes causes R to hang. We're still working on this problem. So for the time being, `subfits` is the same as `subfits.old`. You can change this simply by reassigning, e.g.

```
> subfits <- subfits.new
```

12.1.2 Fitting separately to each pattern

For comparison, we could fit a point process model separately to each point pattern dataset using `ppm`. The easy way to do this is with `with.hyperframe`.

To fit a *separate* uniform Poisson point process to each of the three waterstriders patterns,

```
> H <- hyperframe(W=waterstriders)
> with(H, ppm(W))

1:
Stationary Poisson process
Fitted to point pattern dataset 'W'
Intensity: 0.01642455
             Estimate      S.E.   CI95.lo  CI95.hi Ztest      Zval
log(lambda) -4.108978 0.1622214 -4.426926 -3.79103   *** -25.32944

2:
Stationary Poisson process
Fitted to point pattern dataset 'W'
Intensity: 0.0151169
             Estimate      S.E.   CI95.lo   CI95.hi Ztest      Zval
log(lambda) -4.191942 0.1666667 -4.518602 -3.865281   *** -25.15165

3:
Stationary Poisson process
Fitted to point pattern dataset 'W'
Intensity: 0.01672117
            Estimate      S.E.   CI95.lo   CI95.hi Ztest      Zval
log(lambda) -4.09108 0.1666667 -4.417741 -3.764419   *** -24.54648
```

The result is again a list of three fitted point process models (objects of class `ppm`), but now the fitted coefficients are different.

12.2 Residuals

One standard way to check a fitted model is to examine the residuals.

12.2.1 Point process residuals

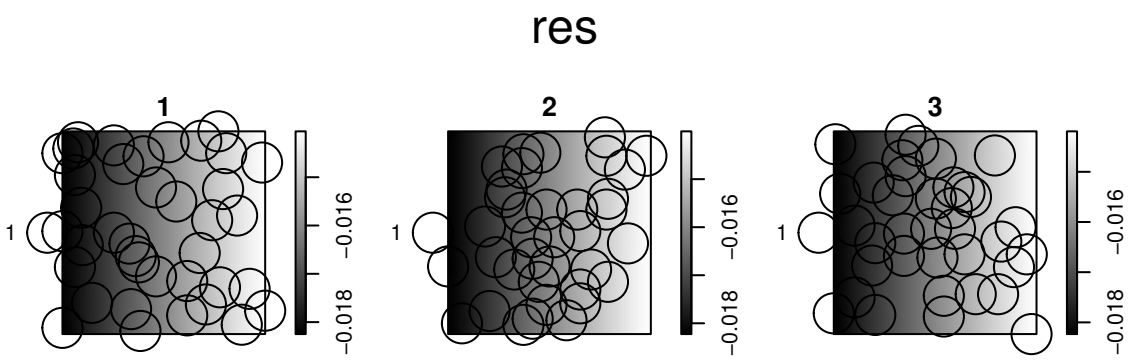
Some recent papers [4, 1] have defined residuals for a fitted point process model (fitted to a *single* point pattern). These residuals are implemented in `spatstat` as `residuals.ppm` and apply to an object of class `ppm`, that is, a model fitted to a *single* point pattern.

The command `residuals.mppm` computes the point process residuals for an `mppm` object.

```
> fit <- mppm(P ~ x, hyperframe(P=waterstriders))
> res <- residuals(fit)
```

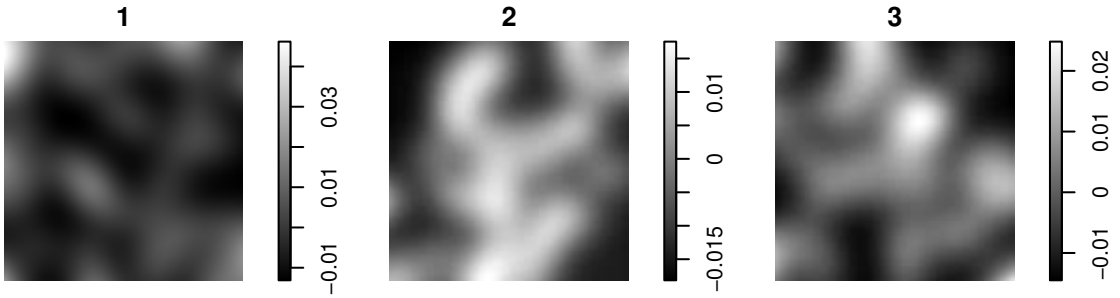
The result is a list, with one entry for each of the point pattern datasets. Each list entry contains the point process residuals for the corresponding point pattern dataset. Each entry in the list is a signed measure (object of class `"msr"`) as explained in the help for `residuals.ppm`). It can be plotted:

```
> plot(res)
```



You probably want the smoothed residual field:

```
> smor <- with(hyperframe(res=res), Smooth(res, sigma=4))
> plot(smor, main="")
```



12.2.2 Sums of residuals

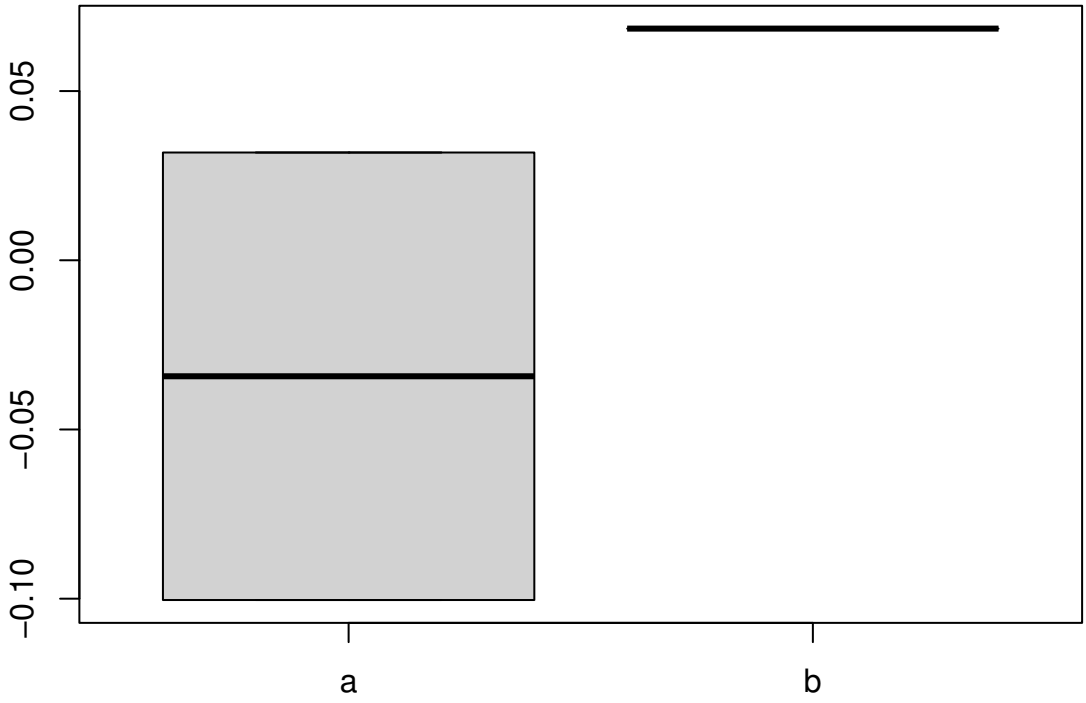
It would be useful to have a residual that is a single value for each point pattern (representing how much that point pattern departs from the model fitted to all the point patterns).

That can be computed by *integrating* the residual measures using the function `integral.msr`:

```
> fit <- mppm(P ~ x, hyperframe(P=waterstriders))
> res <- residuals(fit)
> totres <- sapply(res, integral.msr)
```

In designed experiments we can plot these total residuals against the design covariates:

```
> fit <- mppm(Points~Image, data=demohyper)
> resids <- residuals(fit, type="Pearson")
> totres <- sapply(resids, integral.msr)
> areas <- with(demohyper, area.owin(as.owin(Points)))
> df <- as.data.frame(demohyper[, "Group"])
> df$resids <- totres/areas
> par(mar=rep(2, 4))
> plot(resids~Group, df)
```



12.2.3 Four-panel diagnostic plots

Sometimes a more useful tool is the function `diagnose.ppm` which produces a four-panel diagnostic plot based on the point process residuals. However, it is only available for `ppm` objects.

To obtain a four-panel diagnostic plot for each of the point patterns, do the following:

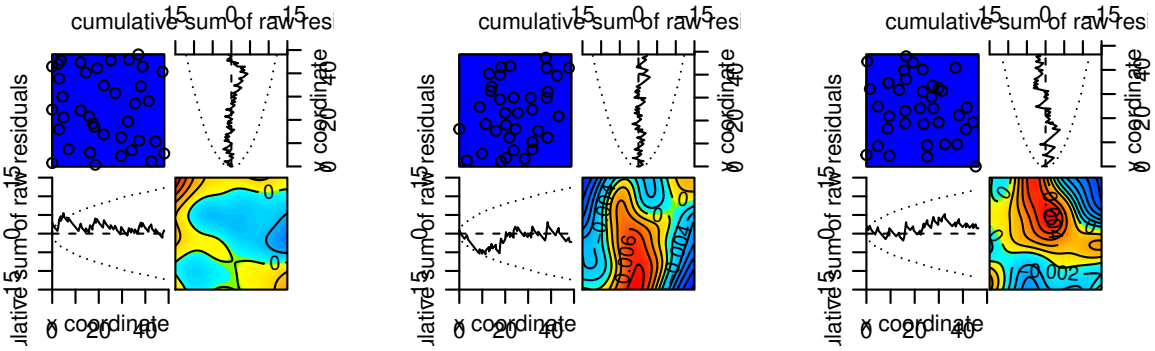
1. fit a model to multiple point patterns using `mppm`.

2. extract the individual fits using `subfits`.

3. plot the residuals of the individual fits.

For example:

```
> fit <- mppm(P ~ 1, hyperframe(P=waterstriders))
> sub <- hyperframe(Model=subfits(fit))
> plot(sub, quote(diagnose.ppm(Model)), main="", marsize=1.5)
```



(One could also do this for models fitted separately to the individual point patterns.)

12.2.4 Residuals of the parameter estimates

We can also compare the parameter estimates obtained by fitting the model simultaneously to all patterns (using `mppm`) with those obtained by fitting the model separately to each pattern (using `ppm`).

```
> H <- hyperframe(P = waterstriders)
> fitall <- mppm(P ~ 1, H)
> together <- subfits(fitall)
> separate <- with(H, ppm(P))
> Fits <- hyperframe(Together=together, Separate=separate)
> dr <- with(Fits, unlist(coef(Separate)) - unlist(coef(Together)))
> dr
          1           2           3
 0.02225482 -0.06070868  0.04015303

> exp(dr)
        1         2         3
1.0225043 0.9410974 1.0409701
```

One could also try deletion residuals, etc.

12.3 Goodness-of-fit tests

12.3.1 Quadrat count test

The χ^2 goodness-of-fit test based on quadrat counts is implemented for objects of class `ppm` (in `quadrat.test.ppm`) and also for objects of class `mppm` (in `quadrat.test.mppm`).

This is a goodness-of-fit test for a fitted **Poisson** point process model only. The model could be uniform or non-uniform and the intensity might depend on covariates.

```
> H <- hyperframe(X=waterstriders)
> # Poisson with constant intensity for all patterns
> fit1 <- mppm(X~1, H)
> quadrat.test(fit1, nx=2)

        Chi-squared test of fitted Poisson model 'fit1' using quadrat counts

data:  fit1
X2 = 1.8596, df = 11, p-value = 0.002137
alternative hypothesis: two.sided

Pooled test
Quadrats of component tests:
List of spatial objects

1:
2 by 2 grid of tiles

2:
2 by 2 grid of tiles

3:
2 by 2 grid of tiles

> # uniform Poisson with different intensity for each pattern
> fit2 <- mppm(X ~ id, H)
> quadrat.test(fit2, nx=2)

        Chi-squared test of fitted Poisson model 'fit2' using quadrat counts

data:  fit2
X2 = 1.8596, df = 9, p-value = 0.01301
alternative hypothesis: two.sided

Pooled test
Quadrats of component tests:
List of spatial objects

1:
2 by 2 grid of tiles
```

```
2:
2 by 2 grid of tiles

3:
2 by 2 grid of tiles
```

See the help for `quadrat.test.ppm` and `quadrat.test.mppm` for further details.

12.3.2 Kolmogorov-Smirnov test

The Kolmogorov-Smirnov test of goodness-of-fit of a Poisson point process model compares the observed and predicted distributions of the values of a spatial covariate.

We want to test the null hypothesis H_0 that the observed point pattern $\mathbf{x}$ is a realisation from the Poisson process with intensity function $\lambda(u)$ (for locations u in the window W). Let $Z(u)$ be a given, real-valued covariate defined at each spatial location u. Under H_0, the *observed* values of Z at the data points, $Z(x_i)$ for each $x_i \in \mathbf{x}$, are independent random variables with common probability distribution function

$$F_0(z) = \frac{\int_W \lambda(u)\mathbf{1}\{Z(u) \le z\}\,\mathrm{d}u}{\int_W \lambda(u)\,\mathrm{d}u}.$$

We can therefore apply the Kolmogorov-Smirnov test of goodness-of-fit. This compares the empirical cumulative distribution of the observed values $Z(x_i)$ to the predicted c.d.f. F_0.

The test is implemented as `kstest.ppm`. The syntax is

```
> kstest.mppm(model, covariate)
```

where `model` is a fitted model (of class `"mppm"`) and `covariate` is either

- a `function(x,y)` making it possible to compute the value of the covariate at any location `(x,y)`
- a pixel image containing the covariate values
- a list of functions, one for each row of the hyperframe of original data
- a list of pixel images, one for each row of the hyperframe of original data
- a hyperframe with one column containing either functions or pixel images.

See Chapter 16 of [**?**] for further information.

References

[1] 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.

[2] A. Baddeley, E. Rubak, and R. Turner. *Spatial Point Patterns: Methodology and Applications with R.* Chapman & Hall/CRC Press, 2015.

[3] A. Baddeley, I. Sintorn, L. Bischof, R. Turner, and S. Heggarty. Analysing designed experiments where the response is a spatial point pattern. In preparation.

[4] 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.

[5] B.J. Chen, G.P. Leser, D. Jackson, and R.A. Lamb. The influenza virus M2 protein cytoplasmic tail interacts with the M1 protein and influences virus assembly at the site of virus budding. *Journal of Virology*, 82:10059–10070, 2008.

[6] 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.

[7] A. Penttinen. *Modelling Interaction in Spatial Point Patterns: Parameter Estimation by the Maximum Likelihood Method.* Number 7 in Jyväskylä Studies in Computer Science, Economics and Statistics. University of Jyväskylä, 1984.