A guide to function objects (class "fv" and "envelope") in spatstat

Adrian Baddeley, Rolf Turner and Ege Rubak

For spatstat version 3.6-4

Abstract

This vignette explains how to use and manipulate function objects (objects of class "fv") and envelope objects (objects of class "envelope") in the spatstat package.

Contents

1 Introduction **2**

2 Plotting **6**

3 Calculating with an "fv" object **11**

4 Calculating with several "fv" objects **13**

5 Creating "fv" objects from raw data **14**

6 Editing the auxiliary information in "fv" objects **18**

7 Pooling several function estimates **20**

8 Structure of objects of class "fv" **21**

9 Structure of objects of class "envelope" **23**

1 Introduction

1.1 Functional summary statistics

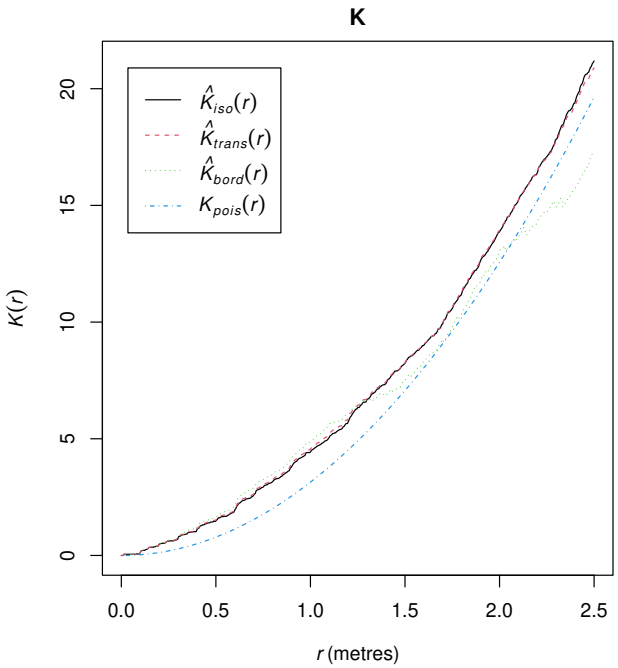
An object of class `"fv"` ('function value table') is a convenient way of storing several different estimates of the same function.

It is common practice to summarise a spatial point pattern dataset using a summary function, such as Ripley's K-function $K(r)$, rather than a single numerical summary value. Typically, an empirical estimate of the function, obtained from the data, will be compared with the 'theoretical' version of the function that would be expected if the point pattern was completely random. There may be several different empirical estimates of the function, based on different estimation techniques, and we also want to compare these estimates with one another.

The `spatstat` family of packages makes it very easy to compute and handle multiple versions of a summary function. Taking the Finnish Pines data `finpines` as an example, we can compute and plot estimates of Ripley's K-function by typing

```
> K <- Kest(finpines)

> plot(K)
```



The plot shows several curves, which represent the different empirical estimates of the K-function (namely the isotropic correction $\widehat{K}_{\mathrm{iso}}(r)$, translation correction $\widehat{K}_{\mathrm{trans}}(r)$, and border correction $\widehat{K}_{\mathrm{bord}}(r)$) and also the theoretical value $K_{\mathrm{pois}}(r)$ that would be expected if the point pattern was completely random. All these functions are plotted against the distance argument r.

The object `K` belongs to class `"fv"` ("function value table"). It is a data frame (that is, it also belongs to the class `"data.frame"`) with attributes giving extra information such as the recommended way of plotting the function. One column of the data frame contains evenly spaced values of the distance argument r, while the other columns contain estimates of the value of the function, or the theoretical value of the function under CSR, corresponding to these distance values.

More information is given by the print method `print.fv`, which can be invoked just by typing the name of the object:

```
>   K

Function value object (class 'fv')
for the function r -> K(r)
.....................................................................
       Math.label       Description
r      r                distance argument r
theo   K[pois](r)       theoretical Poisson K(r)
border hat(K)[bord](r)  border-corrected estimate of K(r)
trans  hat(K)[trans](r) translation-corrected estimate of K(r)
iso    hat(K)[iso](r)   isotropic-corrected estimate of K(r)
```

```
.....................................................
Default plot formula:  .~r
where "." stands for 'iso', 'trans', 'border', 'theo'
Recommended range of argument r: [0, 2.5]
Available range of argument r: [0, 2.5]
Unit of length: 1 metre
```

The output indicates that the columns in the data frame are named `r`, `theo`, `border`, `trans`, and `iso`, and explains their contents. For example, the column `iso` contains estimates of the K-function using the isotropic edge correction. This column is labelled in the plot by the R expression `hat(K)[iso](r)` which is rendered as the mathematical notation $\widehat{K}_{\rm iso}(r)$.

The function argument in an `"fv"` object is usually, but not always, called `r`. (Counterexamples include `transect.im` which returns an `"fv"` object with function argument `t`, and `roc` which returns an `"fv"` object with function argument `p`.)

The command `plot(K)` is dispatched to the method `plot.fv` to generate the graphic shown above. The plot method uses the auxiliary information contained in `K` to attach meaningful labels to the graphic.

Stripping off the auxiliary information we can inspect the data frame itself:

```
> head(as.data.frame(K))
```

```
            r         theo     border      trans        iso
1 0.000000000 0.000000e+00 0.00000000 0.00000000 0.00000000
2 0.004882812 7.490141e-05 0.00000000 0.00000000 0.00000000
3 0.009765625 2.996056e-04 0.00000000 0.00000000 0.00000000
4 0.014648438 6.741127e-04 0.05161956 0.05088267 0.05079365
5 0.019531250 1.198422e-03 0.05161956 0.05088267 0.05079365
6 0.024414062 1.872535e-03 0.05161956 0.05088267 0.05079365
```

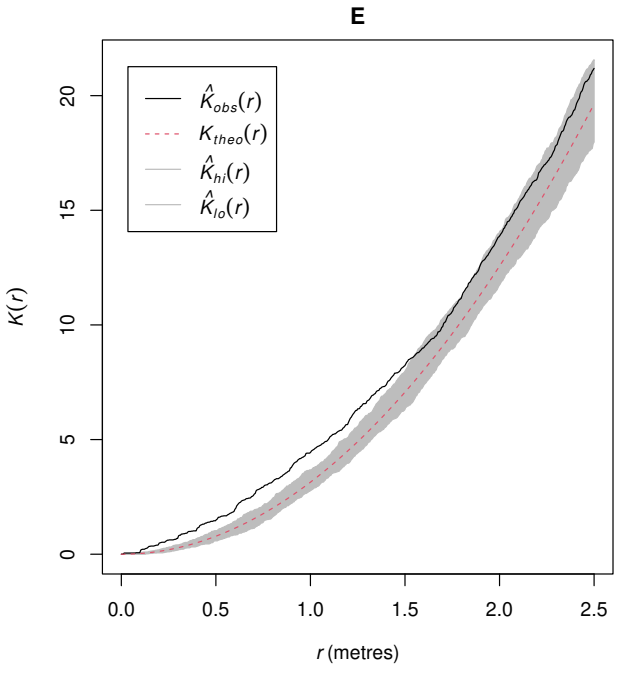
This vignette explains how to plot, manipulate and create objects of class `"fv"`.

1.2 Simulation envelopes

Simulation envelopes of summary functions are often used to assess statistical significance in early stages of analysis. The `spatstat` command `envelope` generates simulation envelopes of a summary function:

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

```
> plot(E)
```



In this example, the command `E <- envelope(finpines, Kest, nsim=39)` generates 39 simulated point patterns according to a completely random process, computes the estimated K-function for each simulated pattern, and finds the

simulation envelopes by identifying the pointwise minimum and maximum of the 39 simulated functions. The result `E` is again an object of class `"fv"`, but additionally belongs to the class `"envelope"`, and contains additional information about how the envelopes were computed.

In the resulting plot, generated by the method `plot.envelope`, the region between the upper and lower simulation envelopes is filled in grey shading. The solid black line is the estimated K-function for the original `finpines` dataset, and the dashed red line is the theoretical K-function for a completely random pattern.

There is a lot of auxiliary information, displayed by `print.envelope`:

```
> E

Pointwise critical envelopes for K(r)
and observed value for 'finpines'
Edge correction: "iso"
Obtained from 39 simulated realisations of CSR
Alternative: two.sided
Significance level of pointwise Monte Carlo test: 2/40 = 0.05
.....................................................................
     Math.label     Description
r    r              distance argument r
obs  hat(K)[obs](r) observed value of K(r) for data pattern
theo K[theo](r)     theoretical value of K(r) for CSR
lo   hat(K)[lo](r)  lower pointwise envelope of K(r) from simulations
hi   hat(K)[hi](r)  upper pointwise envelope of K(r) from simulations
.....................................................................
Default plot formula:  .~r
where "." stands for 'obs', 'theo', 'hi', 'lo'
Columns 'lo' and 'hi' will be plotted as shading (by default)
Recommended range of argument r: [0, 2.5]
Available range of argument r: [0, 2.5]
Unit of length: 1 metre
```

This vignette also explains how to plot, manipulate and create objects of class `"envelope"`. Since envelope objects also belong to class `"fv"`, the vignette first focuses on the capabilities of class `"fv"`.

1.3 Why bother?

Any self-respecting programmer would regard it as a trivial task to organise data in a data frame and plot each column of data as a curve in a graph. Although the task is trivial, it can be time-consuming, it is prone to error, and it can take many attempts to get it exactly right. The authors of `spatstat` developed the class `"fv"` to make this job easier.

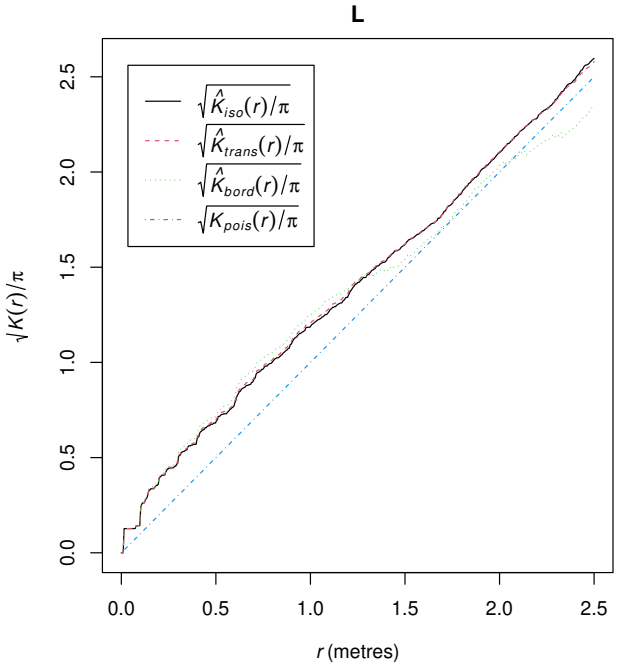
The class `"fv"` is designed to

- support *multiple versions of a function*, such as the different estimates of the K-function obtained using different edge corrections, the theoretical version of the K-function for a completely random process, the upper and lower simulation envelopes of the K-function, and so on.
- do the *"book-keeping"* about the different versions of the function, such as the names of the different columns.
- perform automatic *plotting* of the function, handling all the details of layout and labelling, including generating the mathematical labels for each curve.
- support *calculations* that will be applied automatically to all the versions of the function.
- support *conversion* to other data types in base R, such as data frames and functions.

For example, Besag's L function is defined as $L(r) = \sqrt{K(r)/\pi}$. Since we have already computed the K-function in the example above, we can compute and plot the L-function just by typing

```
> L <- sqrt(K/pi)

> plot(L)
```



Several kinds of magic have happened here:

- The expression `sqrt(K/pi)`, where `K` is an object of class `"fv"`, has been evaluated automatically by calculating $\sqrt{K(r)/\pi}$ for each of the versions of the function stored in `K`;
- The internal data in the object `K`, which provide mathematical labels for each version of the K-function, have been modified according to the algebraic operation that was just performed;
- The result has been saved as a new object of class `"fv"` named `L`;
- The `plot` method has correctly displayed each version of the modified function using the modified mathematical labels, both on the vertical axis and in the legend box;
- The `plot` method has **automatically computed the position of the legend box** to prevent it from overlapping the plotted curves;
- The unit of length for the function argument has been correctly saved in the object `L` and correctly reported on the horizontal axis label.

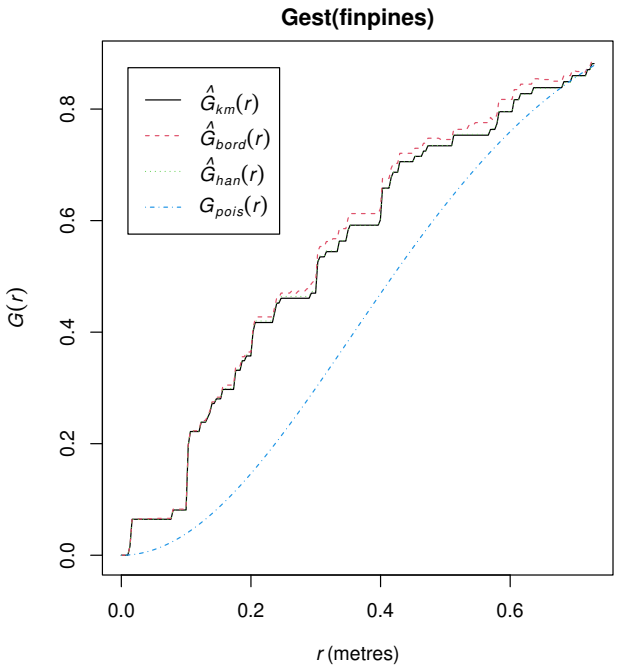
The class `"envelope"` extends the class `"fv"` to handle additional information about how the envelopes were computed. The code supporting the class `"envelope"` performs many of the "trivial" but error-prone calculations involving envelopes. An object of class `"envelope"` can also contain the simulated data (the point patterns and/or the summary functions) that were used to compute the envelopes, which makes it possible to re-use the simulated data to compute a different version of the envelope.

2 Plotting

2.1 Default plot

If `f` is an object of class `"fv"`, the command `plot(f)` is dispatched to the method `plot.fv`. The default behaviour of `plot(f)` is to generate a plot containing several curves, each representing a different version of the same target function, plotted against the distance argument r.

```
> plot(Gest(finpines))
```



Here `Gest` computes estimates of the nearest-neighbour distance distribution function $G(r)$. The plot shows three empirical estimates of $G(r)$ for the `finpines` dataset, together with the 'theoretical' curve $G_{\mathrm{pois}}(r)$ expected for a completely random pattern, all plotted against the distance argument r. The legend indicates the meaning of each curve. The main title identifies the object in R that was plotted.

The return value from `plot.fv` is a data frame containing more detailed information about the meaning of the curves. For the plot generated above, the return value is

```
> aa <- plot(Gest(finpines))
> aa
```

```
      lty col  key                    label                             meaning
km      1   1   km   italic(hat(G)[km](r))     Kaplan-Meier estimate of G(r)
rs      2   2   rs italic(hat(G)[bord](r)) border corrected estimate of G(r)
han     3   3  han  italic(hat(G)[han](r))          Hanisch estimate of G(r)
theo    4   4 theo      italic(G[pois](r))          theoretical Poisson G(r)
```

Here `lty` and `col` are the graphics parameters controlling the line type and line colour, and `label` is the mathematical notation for each edge-corrected estimate, in the syntax recognised by R graphics functions.

The plot generated by `plot.fv` uses the base R graphics system (not `lattice` or `ggplot`). and is affected by graphics parameters specified by `par()`.

2.2 Modifying parameters of the default plot

The default plot can easily be modified:

margin space: To change the amount of white space around the plot, use `par('mar')`.

main title: use `main=""` to suppress the main title.

legend: Set `legend=FALSE` to suppress the legend. Use the argument `legendargs` to modify the legend. The legend position is automatically computed to avoid overlap with the plotted curves, but this can be overridden by `legendpos`.

range of values: Use `xlim` and `ylim` to specify the ranges of values on the x and y axes. **See the note below about the "recommended range".** Use `ylim.covers` to specify a numerical value or values that must be covered by the y axis. For example, `ylim.covers=0` means that the y axis will always include the origin.

For further information, see `help(plot.fv)`.

2.3 Recommended range and recommended columns

The default plot of an `"fv"` object does not necessarily display all the data that is contained in the object:

shorter range of distances: the range of values of the distance argument r displayed in the default plot may be shorter than the range of values actually contained in the data frame.

not all columns of data: the plot may not display all the columns of data contained in the data frame.

This happens because an object of class `"fv"` contains "recommendations" about the range of distances that should be displayed, and about the columns of data that should be shown. These recommendations are based on standard statistical practice. The recommendations are followed when the default plot is generated, unless they are specifically overridden.

Consider this example:

```
> G <- Gest(finpines)
> G

Function value object (class 'fv')
for the function r -> G(r)
.....................................................................
        Math.label      Description
r       r               distance argument r
theo    G[pois](r)      theoretical Poisson G(r)
han     hat(G)[han](r)  Hanisch estimate of G(r)
rs      hat(G)[bord](r) border corrected estimate of G(r)
km      hat(G)[km](r)   Kaplan-Meier estimate of G(r)
hazard  hat(h)[km](r)   Kaplan-Meier estimate of hazard function h(r)
theohaz h[pois](r)      theoretical Poisson hazard function h(r)
.....................................................................
Default plot formula:  .~r
where "." stands for 'km', 'rs', 'han', 'theo'
Recommended range of argument r: [0, 0.72947]
Available range of argument r: [0, 1.7054]
Unit of length: 1 metre
```

The printout shows the range of values of `r` that are present in the table as the '`available range`'. It also gives a '`recommended range`' which is generally shorter than the available range. *The default plot of the object will only show the function values over the recommended range* and not over the full range of values available. This is done so that the interesting detail is clearly visible in the default plot. Values outside the recommended range may be unreliable due to increased variance or bias, depending on the edge correction. To prevent this behaviour and use the full range of function values available, set `clip.xlim=FALSE` in the plot command. Alternatively, specify the desired range of `r` values using the argument `xlim` in the plot command.

The printout also says that the default plot formula is `. ~ r` where "`.`" stands for `"km", "rs", "han", "theo"`. This means that the default plot will display only the columns named `"km", "rs", "han"` and `"theo"` and will **not** display the columns named `"hazard"` and `"theohaz"` which are mentioned in the printout. This is consistent with the graphic shown above.

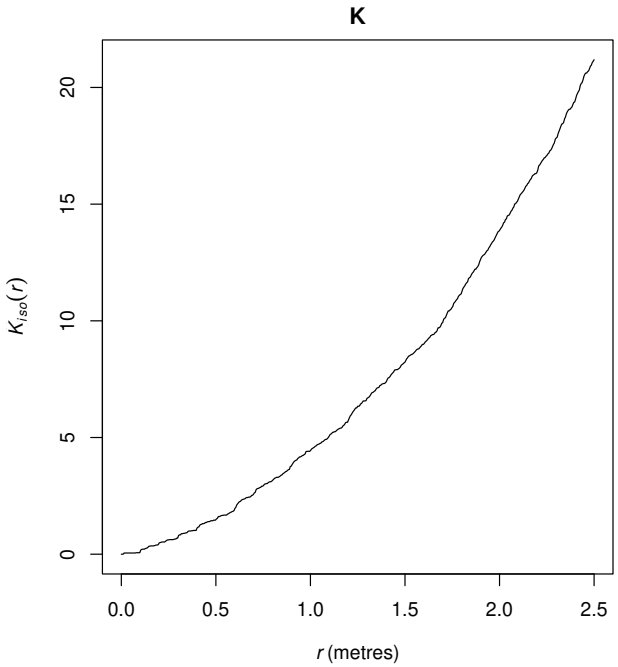
In this example, the column named `"hazard"` is an estimate of the *hazard rate* $h(r) = G'(r)/(1 - G(r))$ of the nearest neighbour distance function, rather than an estimate of $G(r)$ itself. The column named `"theohaz"` is the corresponding theoretical value of the hazard rate, expected if the point pattern is completely random. It makes sense that the hazard rate $h(r)$ and distribution function $G(r)$ should not normally be plotted together. Therefore when `Gest` is executed, it designates `"km", "rs", "han", "theo"` as the "recommended columns" that should be displayed by default, and it stores this information in the resulting object `G`. When `plot(G)` is executed, `plot.fv` uses this information to determine which columns are to be plotted.

2.4 Plot specified by a formula

Different kinds of plots can be specified using a `formula` as the second argument to `plot.fv`. The left side of the formula represents what variables will be plotted on the vertical (y) axis, and the right side determines the variable on the horizontal

(x) axis. For example, in the object `K <- Kest(finpines)`, the column named `iso` contains the values of the isotropic correction estimate. To plot the isotropic correction estimate against r, simply do

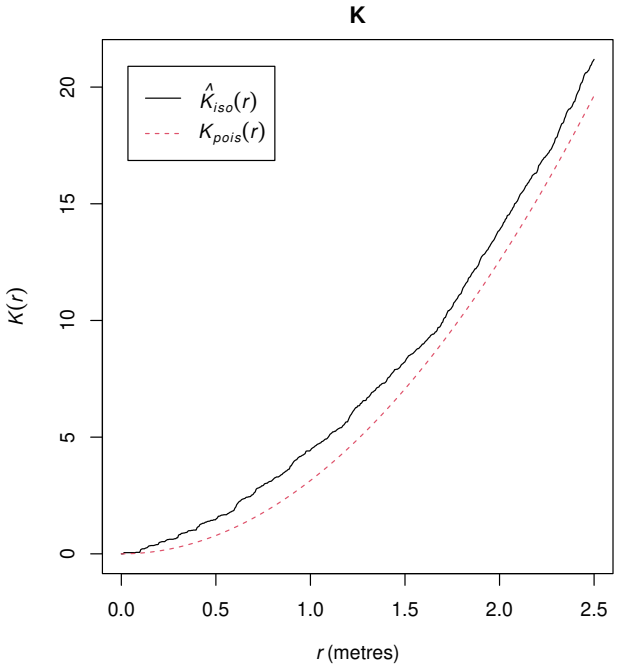
```
> plot(K,  iso ~ r)
```



In `plot.fv`, both sides of the plot formula are interpreted as mathematical expressions, so that operators like '`+`', '`-`', '`*`', '`/`' have their usual meaning in arithmetic. The right-hand side of the formula can be any expression that, when evaluated, yields a numeric vector, and the left-hand side is any expression that evaluates to a vector or matrix of compatible dimensions.

If the left-hand side of the formula, when evaluated, yields a matrix, then each column of that matrix is plotted against the specified x variable as a separate curve. In particular the left-hand side of the formula may invoke the function `cbind` to indicate that several different curves should be plotted. For example, to plot only the isotropic correction estimator and the theoretical curve:

```
> plot(K, cbind(iso, theo) ~ r)
```



Notice that, in this example, `plot.fv` is clever enough to recognise that `iso` and `theo` are both versions of the K-function $K(r)$, and to decide that the appropriate label for the vertical axis is just $K(r)$.

The plot formula may also involve the names of constants like `pi`, standard functions like `sqrt`, and some special abbreviations listed in Table 1.

`.x`	argument of function
`.y`	best estimate of function
`.`	all recommended estimates of function
`.a`	all columns of function values
`.s`	upper and lower limits of shading

Table 1: Recognised abbreviations for columns of an `"fv"` object.

The symbol `.x` represents the function argument, usually `"r"`. The symbol `.y` represents one of the columns of function values which has been designated as the 'best' estimate, for use by some other commands in `spatstat`. The symbol '`.`' represents the 'recommended' estimates. The default plotting formula is `. ~ .x` indicating that each of the recommended estimates will be plotted against the function argument. The formula `.y ~ .x` means that the best estimate of the function will be plotted against the function argument.

To expand these abbreviations for a particular `"fv"` object, use the function `fvnames`.

```
> fvnames(K, ".y")

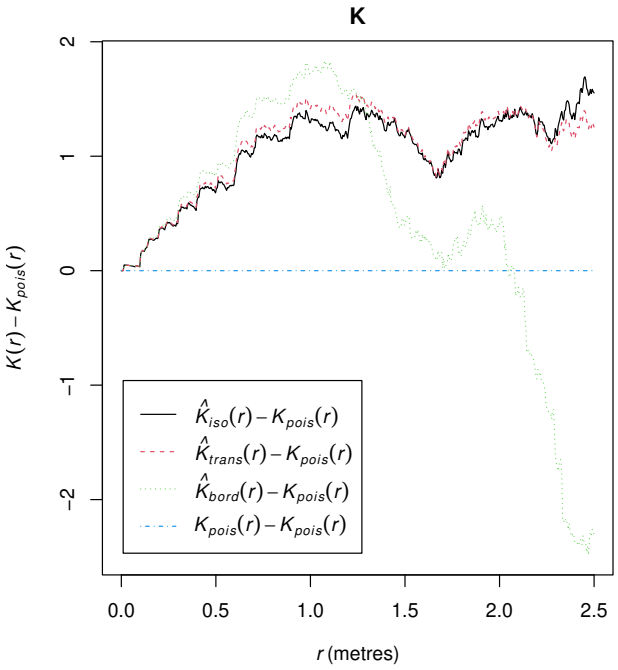
[1] "iso"

> fvnames(K, ".")

[1] "iso"    "trans"  "border" "theo"
```

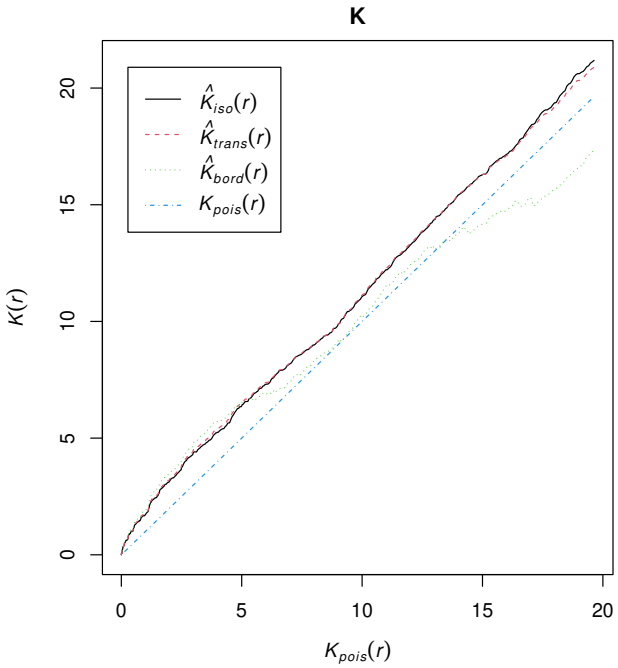
A plot formula can be used to specify a transformation that should be applied to the function values before they are displayed. For example, to subtract the theoretical Poisson value from each of the function estimates:

```
> plot(K, . - theo  ~ r)
```



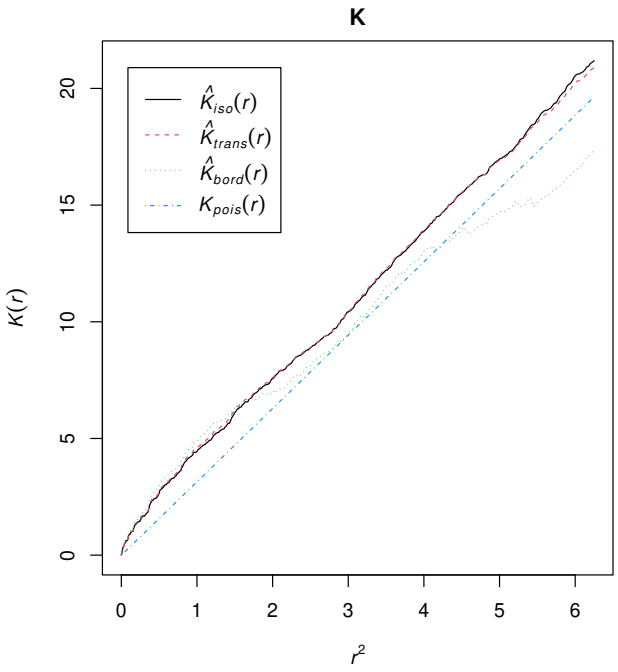
Alternatively one could plot the function estimates *against* the Poisson value:

```
> plot(K, . ~ theo)
```



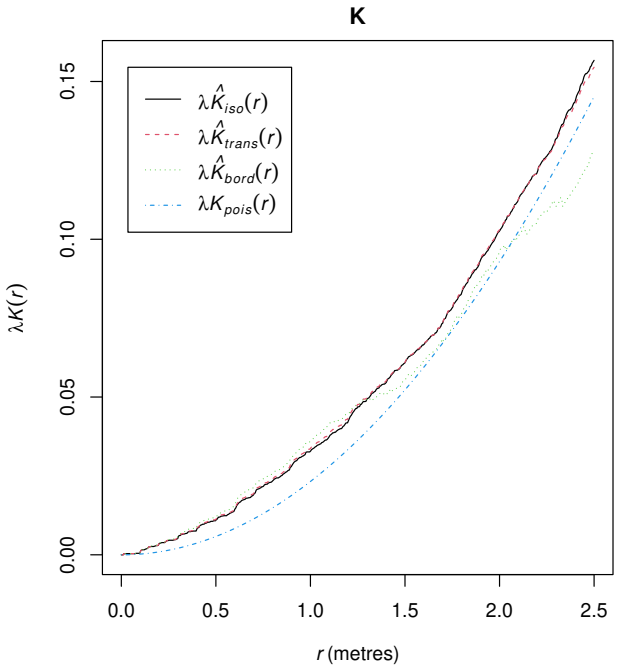
This plot has some theoretical support. In the discussion of Ripley's paper, Cox [3] proposed that $\widehat{K}(r)$ should be plotted against r^2, which is almost equivalent. We can follow Cox's recommendation exactly:

```
> plot(K, . ~ r^2)
```



The mathematical labels for the plot axes, and for the individual curves, are constructed automatically by `spatstat` from the plot formula. If the plot formula involves the names of external variables, these will be rendered in Greek where possible. For example, to plot the average number of trees surrounding a typical tree in the Swedish Pines data,

```
> lambda <- intensity(swedishpines)
> plot(K, lambda * . ~ r)
```



Here we use the name `lambda` so that it will be rendered as the Greek letter λ in the graphics: the y-axis will be labelled $\lambda K(r)$.

3 Calculating with an "fv" object

This section explains how to do calculations involving a single object of class "fv". The next section covers calculations involving several objects of class "fv".

3.1 Arithmetic and mathematical operators

Arithmetic and mathematical operations on an object of class "fv" can be performed by simply writing the arithmetic expression involving the name of the object. The following are valid:

```
> K <- Kest(cells)
> K/pi
> sqrt(K/pi)
```

These inline calculations are performed by the operators `Ops.fv` and `Math.fv`. The operation is applied to each column of *function values*; the function argument `r` will not be affected. The result is another object of class "fv" with the same number of columns, with the same column names, but with appropriately adjusted auxiliary information.

The expression can involve a command which returns an object of class "fv":

```
> sqrt(Kest(cells)/pi)
```

The auxiliary information contained in the resulting object will be slightly less elegant in this case.

These arithmetic and mathematical operations are applied only to the *recommended* columns of function values identified by `fvnames(, ".")`.

3.2 Other vectorised operations

Functions such as `pmax` and `cumsum` apply to vector data, but are not recognised as arithmetic or mathematical operators by the R parser, so they are not covered by `Ops.fv` and `Math.fv`.

For expressions involving `pmax` and `cumsum` (or indeed any algebraic expression whatsoever), use the command `eval.fv` to perform the calculation simultaneously for each column of function values:

```
> Kpos <- eval.fv(pmax(0, K))
```

The result `Kpos` is another object of class "fv" in which the function values are all non-negative.

The first argument of `eval.fv` should be an expression involving the **name** of the object of class "fv".

By default, the calculation is only applied to the *recommended* columns of function values identified by `fvnames(, ".")`. This may be overridden by setting `dotonly=FALSE` in the call to `eval.fv`.

The computations of `Ops.fv` and `Math.fv` are implemented using `eval.fv` but there may be slight differences in the handling of the auxiliary information.

3.3 Calculations involving specific columns

To manipulate or combine one or more columns of data in an "fv" object, it is typically easiest to use with.fv, a method for the generic with. This behaves in a very similar way to with.data.frame. For example:

```
> Kr <- Kest(redwood)
> z <- with(Kr, iso - theo)
> x <- with(Kr, r)
```

The results x and z are numeric vectors, where x contains the values of the distance argument r, and z contains the difference between the columns iso (isotropic correction estimate) and theo (theoretical value for CSR) for the K-function estimate of the redwood seedlings data. For this to work, we have to know that Kr contains columns named r, iso and theo. Printing the object will reveal this information, as would typing names(Kr) or colnames(Kr).

The general syntax is with(X, expr) where X is an "fv" object and expr can be any expression involving the names of columns of X. The expression can include functions, so long as they are capable of operating on numeric vectors. The expression can also involve the abbreviations listed in Table 1:

```
> Kcen <- with(Kr, . - theo)
```

subtracts the 'theoretical' value from all the available edge correction estimates. The result Kcen is another "fv" object.

You can also get a result which is a vector or single number:

```
> with(Kr, max(abs(iso-theo)))

[1] 0.04945199
```

3.4 Extracting data

An object of class "fv" is essentially a data frame with additional attributes. It contains the values of the desired function (such as $K(r)$) at a finely spaced grid of values of the function argument r.

The data frame can be extracted (and the additional attributes removed) using as.data.frame.fv:

```
> df <- as.data.frame(K)
```

A single column of values can be extracted using the $ operator in the usual way: K$iso would extract a vector containing the isotropic correction estimates of $K(r)$.

The subset extraction operator '[' has a method for "fv" objects. This always returns another "fv" object, so it will refuse to remove the column containing values of the function argument r, for example. To override this refusal, convert the object to a data frame using as.data.frame and then use '[': the result will be a data frame or a vector.

Commands designed for data frames often work for "fv" objects as well. The functions head and tail extract the top (first few rows) and bottom (last few rows) of a data frame. They also work on "fv" objects: the result is a new "fv" object containing the function values for a short interval of r values at the beginning or end of the range. The function subset selects designated subsets of a data frame using an elegant syntax and this also works on "fv" objects. To restrict K to the range $r \leq 0.1$ and remove the border correction,

```
> Ko <- subset(K, r < 0.1, select= -border)
```

3.5 Converting to a true function

An object of class "fv" is meant to represent a function, but it contains only sample values of the function at a grid of values of the function argument. The table of function values can also be converted to a true function in the R language using as.function. This makes it easy to evaluate the function at any desired distance r.

```
> Ks <- Kest(swedishpines)
> kfun <- as.function(Ks)
> kfun(9)

[1] 129.096
```

By default, the result kfun is a function in R, with a single argument r (or whatever the original function argument was called). The new function accepts numeric values or numeric vectors of distance values, and returns the values of the 'best' estimate of the function, interpolated linearly between entries in the table.

If one of the other function estimates is required, use the argument value to as.function to select it.

```
> kt <- as.function(Ks, value="trans")
> kt(9)
```

```
[1] 130.4998
```

To retain the option to select any one of the function estimates, type

```
> kf <- as.function(Ks, value=".")
> kf(9, "trans")
```

```
[1] 130.4998
```

3.6 Special operations

An "fv" object can be manipulated using the operations listed in Table 2.

`f`	print a description
`print(f)`	print a description
`plot(f)`	plot the function estimates
`as.data.frame(f)`	strip extra information (returns a data frame)
`f$iso`	extract column named `iso` (returns a numeric vector)
`f[i,j]`	extract subset (returns an "fv" object)
`subset(f, ...)`	extract subset (returns an "fv" object)
`with(f, expr)`	perform calculations with columns of data frame
`eval.fv(expr)`	perform calculations with several "fv" objects
`bind.fv(f, d)`	combine an "fv" object `f` and data frame `d`
`min(f)`, `max(f)`, `range(f)`	range of function values
`Smooth(f)`	apply smoothing to function values
`deriv(f)`	derivative of function
`stieltjes(g,f)`	compute Stieltjes integral with respect to `f`
`as.function(f)`	convert to a function

Table 2: Operations for manipulating an "fv" object `f`.

4 Calculating with several "fv" objects

4.1 Arithmetic and mathematical operators

Arithmetic and mathematical operations involving several objects of class "fv" can be performed by simply writing the arithmetic expression involving the objects:

```
> Kcel <- Kest(cells)
> Kred <- Kest(redwood)
> Kdif <- Kcel - Kred
```

These inline calculations are performed by the operators `Ops.fv` and `Math.fv`.

The operation is applied to each column of *function values*; the function argument `r` will not be affected. The result is another object of class "fv" with the same number of columns, with the same column names, but with appropriately adjusted auxiliary information.

The "fv" objects should be 'compatible' in the sense that they have the same column names, and the same vector of r values. However, `eval.fv` will attempt to reconcile incompatible objects. (The `spatstat` generic function `compatible` determines whether two or more objects are compatible, and the generic function `harmonise` makes them compatible, if possible.)

The expression can involve sub-expressions which return objects of class "fv":

```
> Kest(cells) - Kest(redwood)
```

The auxiliary information contained in the resulting object will be slightly less elegant in this case.

4.2 Other vectorised operations

For expressions involving `pmax` and `cumsum` (or indeed any algebraic expression whatsoever), use the command `eval.fv` to perform the calculation simultaneously for each column of function values.

```
> Kcel <- Kest(cells)
> Kred <- Kest(redwood)
> Kmax <- eval.fv(pmax(Kcel, Kred))
```

The result `Kmax` is another object of class `"fv"`.

The first argument of `eval.fv` should be an expression involving the **names** of the objects of class `"fv"`.

By default, the calculation is only applied to the *recommended* columns of function values identified by `fvnames(x, ".")` where `x` is the object of class `"fv"`. This may be overridden by setting `dotonly=FALSE` in the call to `eval.fv`.

The expression is not permitted to contain sub-expressions that evaluate to objects of class `"fv"`. However, you can use the argument `envir` to supply such sub-expressions:

```
>   Kmax <- eval.fm(pmax(Kcel, Kred),
+                   envir=list(Kcel=Kest(cells), Kred=Kest(redwood)))
```

The computations of `Ops.fv` and `Math.fv` are implemented using `eval.fv` but there may be slight differences in the handling of the auxiliary information.

4.3 Combining objects

Several `"fv"` objects can be combined using the operations listed in Table 3.

`eval.fv(expr)`	perform calculations with several `"fv"` objects
`cbind(f1, f2, ...)`	combine `"fv"` objects `f1, f2, ...`
`bind.fv(f, d)`	combine an `"fv"` object `f` and data frame `d`
`collapse.fv(f1, f2, ...)`	combine several redundant `"fv"` objects
`compatible(f1, f2, ...)`	check whether `"fv"` objects are compatible
`harmonise(f1, f2, ...)`	make `"fv"` objects compatible

Table 3: Operations for manipulating several `"fv"` objects `f1`, `f2`.

Use `cbind.fv` to combine several `"fv"` objects. Use `bind.fv` to glue additional columns onto an existing `"fv"` object.

5 Creating `"fv"` objects from raw data

This section explains how to create objects of class `"fv"` from raw numerical data. This would be useful if you are implementing a completely new kind of summary function.

Subsection 5.1 explains how to create an object of class `"fv"` by providing the numerical data and the required auxiliary information. Section 5.2 describes an easier way to convert a data frame (or similar object) to an object of class `"fv"` without specifying the auxiliary information, using default rules for the auxiliary information. Section 5.3 describes special tools `compileK, compilepcf, compileCDF` for creating an object of class `"fv"` from a numeric vector of distance values, using the rules that apply to the K-function, or the pair correlation function, or the nearest-neighbour distance distribution function.

5.1 The creator function `fv`

5.1.1 The creator function

The low-level function `fv` is used to create an object of class `"fv"` from raw numerical data. It has the following syntax:

```
fv(x, argu = "r", ylab = NULL, valu, fmla = NULL,
   alim = NULL, labl = names(x), desc = NULL,
   unitname = NULL, fname = NULL, yexp = ylab)
```

The arguments are as follows:

- `x` contains the numerical data. It should be a data frame, in which one column gives the values of the function argument for which the function has been evaluated, and at least one other column contains the corresponding values of the function.

  These other columns typically give the values of different versions or estimates of the same function, for example, different estimates of the K-function obtained using different edge corrections. However they may also contain the values of related functions such as the derivative or hazard rate.

- `argu` specifies the name of the column of `x` that contains the values of the function argument (typically `argu="r"` but this is not compulsory).

- `valu` specifies the name of another column that contains the 'recommended' estimate of the function. It will be used to provide function values in those situations where a single column of data is required. For example, `envelope` computes its simulation envelopes using the recommended value of the summary function.

- `fmla` specifies the default plotting behaviour. It should be a formula, or a string that can be converted to a formula. Variables in the formula are names of columns of `x`. See `plot.fv` for the interpretation of this formula.
- `alim` specifies the recommended range of the function argument. This is used in situations where statistical theory or statistical practice indicates that the computed estimates of the function are not trustworthy outside a certain range of values of the function argument. By default, `plot.fv` will restrict the plot to this range.
- `fname` is a character string (or a vector of 2 character strings) giving the name of the function itself. For example, the K-function would have `fname="K"`, while the inhomogeneous K-function has `fname=c("K", "inhom")`.
- `ylab` is a mathematical expression for the function value, used when labelling an axis of the plot, or when printing a description of the function. It should be an R language object. For example the K-function's mathematical name $K(r)$ is rendered by `ylab=quote(K(r))`.
- `yexp` is another mathematical expression for the function value. If `yexp` is present, then `ylab` will be used only for printing, and `yexp` will be used for annotating axes in a plot. (Otherwise `yexp` defaults to `ylab`).
- `labl` is a character vector specifying plot labels for each column of `x`. These labels will appear on the plot axes (in non-default plots), legends and printed output. Entries in `labl` may contain the string `"%s"` which will be replaced by `fname` when plotted or printed. For example the border-corrected estimate of the K-function has label `"%s[bord](r)"` which becomes `"K[bord](r)"` when it is used in `plot.fv` or `print.fv`.
- `desc` is a character vector containing intelligible explanations of each column of `x`. Entries in `desc` may contain the string `"%s"` which will be replaced by `ylab`. For example the border correction estimate of the K-function has description `"border correction estimate of %s"`. This will be replaced by `"border correction estimate of K(r)"` when it is used in `print.fv`.
- `unitname` is the name of the unit of length for the function argument. Typically the function argument `"r"` represents distance between points. The distance values are typically expressed in terms of a distance unit, such as metres or feet. This unit will be printed on the horizontal axis. The argument `unitname` is an object of class `"unitname"`, or `NULL` representing dimensionless values.

5.1.2 Syntax for `ylab` and `yexp`

Mathematical symbols and notation are supported in R base graphics. The labels on the axes of a graph, in the body of the graph, and in graph legends, can all include mathematical notation. The notation has to be encoded as an R language expression. The decoding is slightly idiosyncratic, and this affects the programming of the class `"fv"`.

The arguments `ylab` and `yexp` are mathematical expressions for the function value: `ylab` is used when printing a description of the function, and `yexp` is used when labelling an axis. Usually `ylab` and `yexp` are the same. For example the K-function's mathematical name $K(r)$ is rendered by `ylab=quote(K(r))` and `yexp=ylab`. An example where they are different is the multitype K-function $K_{1,2}(r)$ where we set `ylab=quote(Kcross[1,2](r))` and `yexp=quote(Kcross[list(1,2)](r))` to get the most satisfactory behaviour.

A useful programming tip is to use `substitute` instead of `quote` to insert values of variables into an expression, e.g. `substitute(Kcross[i,j](r), list(i=42,j=97))` yields the same as `quote(Kcross[42, 97](r))`.)

5.1.3 Syntax for `labl`

The argument `labl` is a character vector specifying plot labels for each column of `x`. These labels will appear on the plot axes (in non-default plots), legends and printed output.

Entries in `labl` may contain the string `"%s"` which will be replaced by `fname` when plotted or printed. For example the border-corrected estimate of the K-function has label `"%s[bord](r)"` which becomes `"K[bord](r)"` when it is used in `plot.fv` or `print.fv`. This mechanism allows the code to adjust the labels when the object is changed — for example, to produce the correct labels in `plot(sqrt(K/pi))` as shown in Section 1.3.

Things become more complicated if `fname` is a character vector of length 2. In that case the appropriate expression for the border-corrected estimate is `"{hat(%s)[%s]^{bord}}(r)"` which becomes `{hat(K)[inhom]^{bord}}(r)` when it is used in `plot.fv` or `print.fv`.

We strongly recommend using the function `makefvlabel` to create the appropriate labels. Its syntax is:

```
makefvlabel(op=NULL, accent=NULL, fname, sub=NULL, argname="r")
```

where the arguments are character strings:

op is a prefix or operator such as `"var"` (rarely used);

accent is an accent that should be applied to the main function symbol, usually `"hat"` for empirical estimates;

fname is the name of the function (usually a single letter or a character vector of length 2);

sub is an optional subscript, typically used to discriminate between different estimates of the function, such as different edge corrections;

argname is the name of the function argument.

Examples:

```
> makefvlabel(NULL, NULL, "K", "pois")
[1] "%s[pois](r)"
> makefvlabel(NULL, "hat", "K", "bord")
[1] "hat(%s)[bord](r)"
> makefvlabel(NULL, "hat", c("K", "inhom"), "bord")
[1] "{hat(%s)[%s]^{bord}}(r)"
> makefvlabel("var", "hat", c("K", "inhom"), "bord")
[1] "{bold(var)~hat(%s)[%s]^{bord}}(r)"
```

5.1.4 Syntax for `desc`

Each entry of `desc` is a single character string. It may contain a single instance of `"%s"`, which will be replaced by the function name when required.

5.2 Conversion function `as.fv`

The generic function `as.fv` converts other kinds of data to an object of class `"fv"`.

The methods `as.fv.matrix` and `as.fv.data.frame` provide a lazy way to convert a table of function data to an object of class `"fv"`. The auxiliary information is determined by applying default rules.

Other methods apply to classes of objects which intrinsically contain an object of class `"fv"`, and they simply extract the `"fv"` object. For example, a fitted model of class `"kppm"` contains the summary function (either the K function or the pair correlation function) that was used to fit the model; so the method `as.fun.kppm` simply extracts this summary function.

5.3 compileK, compilepcf, compileCDF

A shortcut is provided for programmers wishing to implement a summary function that is similar to Ripley's K function, the pair correlation function g, the empty space function F or the nearest-neighbour distance distribution function G.

5.3.1 K functions and pair correlation functions

Programmers who wish to implement a summary function similar to Ripley's K function or the pair correlation function can use the commands `compileK` or `compilepcf`.

These low-level functions construct estimates of the K function or pair correlation function, or any similar functions, given only the matrix of pairwise distances and optional weights associated with these distances.

These functions are useful for code development and for teaching, because they perform a common task, and do the housekeeping required to make an object of class `"fv"` that represents the estimated function. However, they are not very efficient.

The basic syntax of `compileK` and `compilepcf` is:

```
> compileK(D, r, weights = NULL, denom = 1, ...)
> compilepcf(D, r, weights = NULL, denom = 1, ...)
```

where

- `D` is a square matrix giving the distances between all pairs of points;
- `r` is a vector of distance values, equally spaced, at which the summary function should be calculated;
- `weights` is an optional matrix of numerical weights for the pairwise distances;

- `denom` is the denominator for the estimator. It may be a single number, or a numeric vector with the same length as `r`.

The command `compileK` calculates the weighted estimate of the K function,

$$K(r) = \frac{1}{v(r)} \sum_i \sum_{j \neq i} w_{i,j} \, 1\{d_{i,j} \leq r\}$$

and `compilepcf` calculates the weighted estimate of the pair correlation function,

$$g(r) = \frac{1}{v(r)} \sum_i \sum_{j \neq i} w_{i,j} \, \kappa(d_{i,j} - r)$$

where $d_{i,j}$ is the distance between spatial points i and j, with corresponding weight $w_{i,j}$, and $v(r)$ is the specified denominator. Here κ is a fixed-bandwidth smoothing kernel.

For a point pattern in two dimensions, the usual denominator $v(r)$ is constant for the K function, and proportional to r for the pair correlation function:

```
> X <- japanesepines
> D <- pairdist(X)
> Wt <- edge.Ripley(X, D)
> lambda <- intensity(X)
> a <- (npoints(X)-1) * lambda
> r <- seq(0, 0.25, by=0.01)
> K <- compileK(D=D, r=r, weights=Wt, denom=a)
> g <- compilepcf(D=D, r=r, weights=Wt, denom= a * 2 * pi * r)
```

The result of `compileK` or `compilepcf` can then be edited (as explained in the next section) to change the function name and other information as desired.

5.3.2 Cumulative distribution functions

Programmers wishing to implement a summary function which is a cumulative distribution function, similar to the functions `Gest` or `Fest`, can use the command `compileCDF`.

The basic syntax of `compileCDF` is:

```
> compileCDF(D, B, r, ..., han.denom = NULL)
```

where

- `D` is a numeric vector of observed distances (such as the distance from each data point to its nearest neighbour);
- `B` is a numeric vector of censoring distances (such as the distance from each data point to the boundary of the window);
- `r` is a vector of distance values, equally spaced, at which the summary function should be calculated;
- `han.denom` is the denominator for the Hanisch estimator. It is usually a numeric vector with the same length as `r`.

An example for the nearest-neighbour distance distribution function $G(r)$:

```
> X <- japanesepines
> D <- nndist(X)
> B <- bdist.points(X)
> r <- seq(0, 1, by=0.01)
> h <- eroded.areas(Window(X), r)
> G <- compileCDF(D=D, B=B, r=r, han.denom=h)
> ## give it a better name
> G <- rebadge.fv(G, new.fname="G", new.ylab=quote(G(r)))
```

6 Editing the auxiliary information in "fv" objects

The "auxiliary information" in an object of class "fv" consists of the function name, a mathematical expression for the function, mathematical expressions for each version of the function contained in a column of data, the choice of which columns will be plotted by default, and other information.

A programmer will often wish to create an "fv" object first, perhaps using some existing code, and then edit the auxiliary information.

The safe way to edit the auxiliary information is to **use the internal functions in spatstat** which support the "fv" class:

- rebadge.fv is a low-level function which allows the user to change any of the entries in the auxiliary information as desired.
- rebadge.as.crossfun and rebadge.as.dotfun are wrappers for rebadge.fv which change the auxiliary information into the form expected for a cross-type or dot-type summary function.
- fvlabels extracts the mathematical code for each version of the summary function, and fvlabels<- changes the codes.
- makefvlabel creates suitable mathematical code for a version of the summary function.
- The functions fvnames and fvnames<- manage the definition of the abbreviations listed in Table 1.
- The methods formula.fv and formula<-.fv manage the default plotting formula.
- names<-.fv changes the names of the columns in the "fv" object, and adjusts the internal data accordingly.
- tweak.fv.entry is a very low-level function that changes the auxiliary information about one of the columns of data.
- prefixfv is another wrapper for rebadge.fv that adds a prefix to the name of the function.

6.1 Low-level editing

rebadge.fv is a low-level function which allows the user to change any of the entries in the auxiliary information as desired. It has syntax

```
> rebadge.fv(x, new.ylab, new.fname,
+            tags, new.desc, new.labl,
+            new.yexp=new.ylab, new.dotnames,
+            new.preferred, new.formula, new.tags)
```

where x is the object of class "fv".

The arguments new.fname, new.ylab and new.yexp (if present) specify new values for the function name fname and the mathematical expressions for the function, ylab and yexp, described in Section 5.1.

The argument new.dotnames specifies a new value for the selection of columns that are plotted by default. This is a character vector of column names of x and is associated with the abbreviation "." in Table 1.

The argument new.preferred specifies a new value for the choice of column that is designated the "preferred" column and is used in calculations which require a single column of data, such as simulation envelopes. This is a single character string which must be a column name of x and is associated with the abbreviation ".y" in Table 1.

The argument new.formula specifies a new default plotting formula for the summary function.

The argument new.desc specifies new values for the string descriptions of the individual columns, replacing the argument desc described in Section 5.1. It should be a character vector with one entry for every column of x (or see below).

The argument new.labl specifies new values for the mathematical labels of the individual columns, replacing the argument desc described in Section 5.1. It should be a character vector with one entry for every column of x (or see below).

The argument tags can be used to select some of the columns of data so that only the auxiliary data for the selected columns will be changed. It should be a character vector with entries which match the names of columns of x. In that case, new.desc and new.labl should have the same length as tags, and they will be taken as replacement values for the selected columns only.

The optional argument new.tags changes the names of the columns of x (or the columns selected by tags) to new values.

6.2 Changing information about one column

The method `names<-.fv` changes the names of the columns in the `"fv"` object, and adjusts the internal data accordingly.

The function `tweak.fv.entry` is a very low-level function that changes the auxiliary information about one of the columns of data. It has syntax

```
> tweak.fv.entry(x, current.tag, new.labl=NULL, new.desc=NULL, new.tag=NULL)
```

where `current.tag` is the current name of the column for which the information should be changed, `new.labl` is the new mathematical label for the column, `new.desc` is the new text description of the column, and `new.tag` is the new name of the column. All these arguments are single strings or `NULL`.

6.3 Special idioms

A few functions are available for performing special idioms.

The function `prefixfv` is a wrapper for `rebadge.fv` that adds a prefix to the name of the function, and to all the relevant auxiliary information. It has syntax

```
> prefixfv(x, tagprefix="", descprefix="", lablprefix=tagprefix,
+          whichtags=fvnames(x, "*"))
```

where `tagprefix`, `descprefix` and `lablprefix` are strings that should be added to the beginning of the column name, the text description, and the mathematical expression for each column of data. The argument `whichtags` specifies which columns of data should be changed; the default is to change all columns.

The function `rebadge.as.crossfun` changes the auxiliary information into the form expected for a bivariate, cross-type summary function, analogous to the bivariate K-function $K_{i,j}(r)$ between two types of points labelled i and j that is computed by the `spatstat` function `Kcross`. It has syntax

```
> rebadge.as.crossfun(x, main, sub=NULL, i, j)
```

where `main` is the main part of the function name, `sub` is the subscript part of the function name, and `i` and `j` are the type labels. For example

```
> rebadge.as.crossfun(x, "L", i="A", j="B")
```

would create a function $L_{A,B}(r)$, and

```
> rebadge.as.crossfun(x, "L", "inhom", "A", "B")
```

would create a function $L_{\mathrm{inhom},A,B}(r)$.

Similarly the function `rebadge.as.dotfun` changes the auxiliary information into the form expected for a "one type-to-any type" summary function, analogous to the function $K_{i\bullet}(r)$ that is computed by the `spatstat` function `Kdot`. It has syntax

```
> rebadge.as.dotfun(x, main, sub=NULL, i)
```

6.4 Handling mathematical labels

The auxiliary information in an `"fv"` object includes mathematical labels for the different versions of the function, which are displayed by `plot.fv`.

The function `fvlabels` extracts the mathematical code for each version of the summary function from the `"fv"` object, and `fvlabels<-` changes the codes.

The mathematical codes are strings which must be recognisable to the `plotmath` code in the R base graphics system which is somewhat idiosyncratic. The strings may also (and usually do) include the substring `%s` (appearing once or twice) which will be replaced by the function name. For example, if the function name is `"K"` and the label is `"hat(%s)[iso](r)"` this will be parsed as `hat(K)[iso](r)` which is rendered as $\widehat{K}_{\mathrm{iso}}(r)$.

The function `makefvlabel` creates suitable mathematical code for a version of the summary function. Programmers are strongly advised to use `makefvlabel`.

6.5 Changing default behaviour

The default behaviour for plotting an `"fv"` object depends on its default `plot formula` and typically on its `dot names`.

The default plot formula is printed when the object is printed, and can be extracted using `formula.fv`.

```
> K <- Kest(cells)
> formula(K)
```

```
[1] ".~r"
```

The interpretation of the plot formula is explained in Section 2.4. In the example above, the left hand side of the formula uses the abbreviation "." which stands for "the default list of columns to be plotted". This abbreviation can be expanded using `fvnames`:

```
> fvnames(K, ".")

[1] "iso"    "trans"  "border" "theo"
```

which indicates that the columns named `"iso"`, `"trans"`, `"border"` and `"theo"` will be plotted.

The choice of "dot names" can be changed using `fvnames<-`:

```
> fvnames(K, ".") <- c("iso", "theo")
```

In general the functions `fvnames` and `fvnames<-` manage the definition of all the abbreviations listed in Table 1.

7 Pooling several function estimates

7.1 Pooling

"Pooling" or combining several datasets into a single dataset is a common statistical procedure. If we are only interested in a summary statistic of the data, then in some special circumstances, the summary statistic of the pooled dataset can be calculated from the summary statistics of the original, separate datasets. For example, if we have a set of n_1 observations with sample mean m_1, and another set of n_2 observations with sample mean m_2, then the sample mean of the pooled set of $n_1 + n_2$ observations has sample mean $(n_1 m_1 + n_2 m_2)/(n_1 + n_2)$, a weighted average of the sample means of the original datasets. This procedure is loosely called "pooling" the sample mean.

If we have two point pattern datasets, observed in different windows, we can "pool" the patterns by simply treating them as a single point pattern observed in the combined window. If we pool two point pattern datasets, the estimated K-function of the pooled pattern can be calculated from the estimated K-functions $K_1(r)$ and $K_2(r)$ of the original point patterns, if we know the number of points in each of the two original patterns. That is, Ripley's K-function can be "pooled".

The summary functions used in spatial statistics can be pooled, provided they are able to be expressed as a ratio $f(r) = A(r)/B$ or $f(r) = A(r)/B(r)$ where $A(r)$ is the "numerator" and B or $B(r)$ is the "denominator". The pooled estimate is the ratio of the sum of numerators divided by the sum of denominators. For details, see section 16.8.1 of [1].

7.2 Pooling summary functions

The generic function `pool` performs pooling of summary statistics (including summary functions like the K-function). In order for this to work correctly, we must know the numerator and denominator for each of the individual summary statistics or summary functions. For this purpose there is a special class `"rat"` (for "ratio object"). An object of class `"rat"` contains two attributes named `"numerator"` and `"denominator"` which contain the numerator and denominator of the ratio.

For many of the summary functions provided in `spatstat`, if we set the argument `ratio=TRUE`, the numerator and denominator will be calculated separately and saved in the resulting object, which will belong to the class `"rat"` ("ratio object") as well as `"fv"`.

```
> class(Kest(cells))

[1] "fv"         "data.frame"

> class(Kest(cells, ratio=TRUE))

[1] "rat"        "fv"         "data.frame"
```

This capability is currently available for the functions `compileK`, `compilepcf`, `Finhom`, `Gcross.inhom`, `Gdot.inhom`, `Ginhom`, `GmultiInhom`, `Jcross.inhom`, `Jdot.inhom`, `Jinhom`, `Jmulti.inhom`, `K3est`, `Kcross`, `Kdot`, `Kest`, `Kinhom`, `Kmulti`, `Ksector`, `linearKinhom`, `linearK`, `linearpcfinhom`, `linearpcf`, `nnorient`, `pairorient`, `pcfcross`, `pcfdot`, `pcfmulti`, `pcf.ppp` and `Tstat`.

The method `pool.rat` will pool several objects which all belong to the classes `"fv"` and `"rat"`:

```
> X1 <- runifpoint(50)
> X2 <- runifpoint(50)
> K1 <- Kest(X1, ratio=TRUE)
> K2 <- Kest(X2, ratio=TRUE)
> K <- pool(K1, K2)
```

```
> Xlist <- runifpoint(50, nsim=6)
> Klist <- lapply(Xlist, Kest, ratio=TRUE)
> K <- do.call(pool, Klist)
```

There is also a fallback method `pool.fv` which is used when some of the objects do not contain ratio information. This method effectively pretends that all the objects have the same denominator.

7.3 Low level utilities

Programmers wishing to implement a summary function with ratio information can use the following low-level utilities:

- `ratfv` is the creator function, analogous to `fv`. Its syntax is

  ```
  > ratfv(df, numer, denom, ..., ratio=TRUE)
  ```

  where `df` is a data frame, `numer` and `denom` are objects of class `"fv"`, and additional arguments `...` are passed to `fv`. It is sufficient to specify either `df` or `numer`, in addition to `denom`.

- `bind.ratfv` glues extra columns onto an existing object of class `"fv"`and `"rat"`.

- `conform.ratfv` forces the auxiliary information in the numerator and denominator of an object of class `"fv"`and `"rat"` to agree with the auxiliary information of the main object.

8 Structure of objects of class `"fv"`

This section explains the information contained in objects of class `"fv"`.

8.1 Advice

We strongly discourage the user from unpacking the internal contents of objects of class `"fv"` and manipulating the contents directly. Instead, we recommend using the functions that are available in `spatstat` for handling these objects.

Although it is easy to extract the internal data contained in an object in R, the structure of objects of class `"fv"` is idiosyncratic, and the internal format is variable. Looking at one example of an object of class `"fv"` will not tell you how it all works. This is because there are many cases to handle, and many quirks in the formatting of algebraic expressions in R.

Using the functions provided in `spatstat` is also more efficient than extracting data yourself, because it avoids creating multiple copies of the data.

Most of all, **do not change the internal contents of objects of class `"fv"`**. This can easily violate the internal format and cause errors. Use the functions supplied for handling these objects.

8.2 Objects of class `"fv"`

Objects of class `"fv"` are returned by many commands in the `spatstat` packages. Usually these objects are obtained by analysing a spatial point pattern dataset. There are also functions to create such objects from raw data.

An object of class `"fv"` is essentially a data frame with additional attributes containing auxiliary information.

Data frame structure

The first column of the data frame contains values of the function argument. These values are arranged in increasing order, are usually evenly-spaced, and usually start from zero. The first column usually (but not always) has the column name `r`.

Subsequent columns of the data frame contain the values of different versions of the summary function, corresponding to the values of the function argument. These columns may have any column names. These versions of the function may be referred to by their column names when plotting and manipulating the object.

```
> G <- Gest(finpines)
> df <- as.data.frame(G)
> head(df)

           r         theo        han         rs         km    hazard    theohaz
1 0.000000000 0.000000e+00 0.00000000 0.00000000 0.00000000  0.000000 0.00000000
2 0.003330908 4.391736e-05 0.00000000 0.00000000 0.00000000  0.000000 0.02637018
3 0.006661817 1.756579e-04 0.00000000 0.00000000 0.00000000  0.000000 0.05274036
4 0.009992725 3.951868e-04 0.00000000 0.00000000 0.00000000  0.000000 0.07911054
```

```
5 0.013323633 7.024464e-04 0.01607756 0.01612903 0.01612903  4.881708 0.10548072
6 0.016654541 1.097356e-03 0.06437475 0.06451613 0.06451613 15.140271 0.13185090
```

In this example, the object `G` contains estimates of the nearest-neighbour distance distribution function $G(r)$ for the `finpines` dataset. For the distance value $r = 0.016654541$ metres, the estimate of $G(r)$ using the `han` method is 0.06437475.

Columns of data can be extracted using the data frame structure. To extract the sequence of `r` values, use `df$r` or `G$r` or `df[, "r"]`. To extract the corresponding values of `han`, use `df$han` or `G$han` or `df[, "han"]`.

Auxiliary information

In the example above, to find out what the column `han` means, we need the auxiliary information stored in the object `G`. This can be printed out directly in readable form:

```
> G

Function value object (class 'fv')
for the function r -> G(r)
.....................................................................
        Math.label      Description
r       r               distance argument r
theo    G[pois](r)      theoretical Poisson G(r)
han     hat(G)[han](r)  Hanisch estimate of G(r)
rs      hat(G)[bord](r) border corrected estimate of G(r)
km      hat(G)[km](r)   Kaplan-Meier estimate of G(r)
hazard  hat(h)[km](r)   Kaplan-Meier estimate of hazard function h(r)
theohaz h[pois](r)      theoretical Poisson hazard function h(r)
.....................................................................
Default plot formula:  .~r
where "." stands for 'km', 'rs', 'han', 'theo'
Recommended range of argument r: [0, 0.72947]
Available range of argument r: [0, 1.7054]
Unit of length: 1 metre
```

Thus, `han` refers to the estimate of $G(r)$ using Hanisch's method.

The auxiliary information is stored in attributes of the object. The full list of attributes is as follows:

`argu`	character(1)	Name of function argument (usually `"r"`)
`valu`	character(1)	Name of preferred function value
`ylab`	language	Mathematical expression for function (for vertical axis of plot)
`yexp`	language	Mathematical expression for function (in algebra)
`fmla`	character(1)	Default plotting formula
`alim`	numeric(2)	Recommended range of function argument
`labl`	character(m)	Mathematical labels for each column
`desc`	character(m)	Text descriptions of each column
`units`	unitname	Unit of length (for function argument)
`fname`	character(1 or 2)	Symbol for function only
`dotnames`	character($k \leq m$)	Column names of all recommended versions
`shade`	character(0 or 2)	Column names of limits of grey shading

`argu` is the name of the column of the data frame that contains the values of the function argument (typically `argu="r"` but this is not compulsory).

`valu` specifies the name of another column that contains the 'recommended' estimate of the function. It will be used to provide function values in those situations where a single column of data is required. For example, `envelope` computes its simulation envelopes using the recommended value of the summary function.

`fmla` specifies the default plotting behaviour, as explained in Section 2. It is a character string that can be converted to a `formula` in the R language.

`alim` specifies the recommended range of the function argument. It is a numeric vector of length 2. This is used in situations where statistical theory or statistical practice indicates that the computed estimates of the function are not trustworthy outside a certain range of values of the function argument. By default, `plot.fv` will restrict the plot to this range.

`fname` gives the name of the function itself. For example, the K-function would have `fname="K"`. It is either a character string, or a vector of two character strings, where the second element is interpreted as a subscript. For example, the inhomogeneous K-function computed by `Kinhom` has `fname=c("K", "inhom")`.

`ylab` is a mathematical expression for the function value, used when printing a description of the function. It is an R language object. For example the K-function's mathematical name $K(r)$ is rendered by `ylab=quote(K(r))`.

`yexp` is another mathematical expression for the function value, used for annotating axes in a plot.

`labl` is a character vector specifying plot labels for each column of the data frame. These labels will appear on the plot axes (in non-default plots), legends and printed output. Entries in `labl` may contain the string `"%s"` which will be replaced by `fname`.

`desc` is a character vector containing intelligible explanations of each column of the data frame. Entries in `desc` may contain the string `"%s"` which will be replaced by `ylab`.

9 Structure of objects of class "envelope"

This section explains the information contained in objects of class `"envelope"`.

9.1 The envelope command

The `spatstat` function `envelope` performs the calculations required for envelopes. It computes the summary function for a point pattern dataset, generates simulated point patterns, computes the summary functions for the simulated patterns, and computes the envelopes of these summary functions.

```
> E <- envelope(swp, Kest, nsim=39, fix.n=TRUE)
```

The result is an object of class `"envelope"` and `"fv"` which can be printed and plotted and manipulated using the tools for `"fv"` objects, and by additional tools provided for `"envelope"` objects.

The print method gives a lot of detail:

```
> E

Pointwise critical envelopes for K(r)
and observed value for 'finpines'
Edge correction: "iso"
Obtained from 39 simulated realisations of CSR
Alternative: two.sided
Significance level of pointwise Monte Carlo test: 2/40 = 0.05
.....................................................................
     Math.label     Description
r    r              distance argument r
obs  hat(K)[obs](r) observed value of K(r) for data pattern
theo K[theo](r)     theoretical value of K(r) for CSR
lo   hat(K)[lo](r)  lower pointwise envelope of K(r) from simulations
hi   hat(K)[hi](r)  upper pointwise envelope of K(r) from simulations
.....................................................................
Default plot formula:  .~r
where "." stands for 'obs', 'theo', 'hi', 'lo'
Columns 'lo' and 'hi' will be plotted as shading (by default)
Recommended range of argument r: [0, 2.5]
Available range of argument r: [0, 2.5]
Unit of length: 1 metre
```

9.2 Re-using envelope data

The method `envelope.envelope` allows new `envelope` commands to be applied to a previously computed `"envelope"` object, provided it contains the necessary data.

In the original call to `envelope`, if the argument `savepatterns=TRUE` was given, the resulting `"envelope"` object contains all the simulated point patterns. Alternatively if the argument `savefuns=TRUE` was given, the resulting object contains the individual summary functions for each of the simulated patterns. This information is not saved, by default, for efficiency's sake.

Envelopes created with `savepatterns=TRUE` allow any kind of new envelopes to be computed using the same simulated point patterns:

```
> E1 <- envelope(redwood, Kest, savepatterns=TRUE)
> E2 <- envelope(E1, Gest, global=TRUE,
+                transform=expression(fisher(.)))
```

Envelopes created with `savefuns=TRUE` allow the user to switch between pointwise and global envelopes of the same summary function, to apply different transformations of the summary function, and to change some parameters:

```
> A1 <- envelope(redwood, Kest, nsim=39, savefuns=TRUE)
> A2 <- envelope(A1, global=TRUE, nsim=19,
+                transform=expression(sqrt(./pi)))
```

9.3 Pooling several envelopes

It is also possible to combine the simulation data from several envelope objects and to compute envelopes based on the combined data. This is done using `pool.envelope`, a method for the `spatstat` generic `pool`. The envelopes must be compatible, in that they are envelopes for the same function, and were computed using the same options. The individual summary functions must have been saved.

```
> E1 <- envelope(cells, Kest, nsim=10, savefuns=TRUE)
> E2 <- envelope(cells, Kest, nsim=20, savefuns=TRUE)
> E <- pool(E1, E2)
```

9.4 Structure of envelope objects

An object of class `"envelope"` is an object of class `"fv"` with additional auxiliary information:

- the names of two of the columns of function values, designated as the upper and lower simulation envelopes of the function, saved in `attr(, "shade")` and retrievable as `fvnames(, .s)`
- details of how the envelopes were computed, saved in `attr(, "einfo")`
- optionally, the simulated point patterns used to compute the envelopes, saved in `attr(, "simpatterns")`
- optionally, the simulated summary functions (the summary functions computed for the simulated point patterns) used to compute the envelopes, saved in `attr(, "simfuns")`

Objects of class `"envelope"` inherit the class `"fv"`, so they can be manipulated using methods for class `"fv"`, but there are extra methods for the special class `"envelope"`.

9.5 The `einfo` list

Additional attribute `einfo` is a list of:

`call`	character(1)	original function call
`Yname`	character(1)	name of original dataset
`valname`	character(1)	column name of function values used
`csr`	logical(1)	`TRUE` if simulations based on CSR
`csr.theo`	logical (1)	see below
`use.theory`	logical (1)	see below
`pois`	logical(1)	`TRUE` if simulations are Poisson process
`simtype`	character(1)	Type of simulation (see below)
`constraints`	character(1)	Additional information (see below)
`nrank`	integer(1)	Rank of envelopes
`nsim`	integer(1)	Number of simulations for envelope
`Nsim`	integer(1)	Total number of simulations
`global`	logical(1)	`TRUE` if global envelopes
`ginterval`	numeric(0 or 2)	Domain of function argument for global envelopes
`dual`	logical(1)	`TRUE` if two sets of simulations performed
`nsim2`	integer(1)	Number of simulations in second set
`VARIANCE`	logical(1)	`TRUE` if limits are based on standard deviation
`nSD`	numeric(1)	Number of standard deviations defining limits
`alternative`	character(1)	`two.sided`, `less` or `greater`
`scale`	`NULL` or function	Scaling function for function argument
`clamp`	logical(1)	`TRUE` if one-sided deviations must be positive
`use.weights`	logical(1)	`TRUE` if sample mean is weighted
`do.pwrong`	logical(1)	`TRUE` if "wrong p-value" should be calculated
`gaveup`	logical(1)	`TRUE` if simulations terminated early

References

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

[2] J.E. Besag. Contribution to the discussion of the paper by Ripley (1977). *Journal of the Royal Statistical Society, Series B* **39** (1977) 193–195.

[3] D.R. Cox. Contribution to the discussion of the paper by Ripley (1977). *Journal of the Royal Statistical Society, Series B* **39** (1977) 206.