DSC 2003 Working Papers
(Draft Versions)
http://www.ci.tuwien.ac.at/Conferences/DSC-2003/

`Rasp` : A Package For Spatial Statistics

Barry Rowlingson[*],
Adrian Baddeley[†],
Rolf Turner[‡],
Peter Diggle[§]

Abstract

This paper describes the development of a new `R` package for spatial data and statistics. We review existing packages that deal with spatial information and comment on how they handle the basic data. Our goal is then to construct a unifying spatial data structure that can form a base for any spatial statistics libraries.

We build on our experience with earlier packages – `spatstat` [1] and `splancs` [2] – to produce a library with extensive data-manipulation and statistical functionality. An object-oriented design is used throughout to abstract functionality from any implementation specifics, but we take care to present the user with simple functions that have sensible defaults. The paper is as much a technical document on package design as it is a story of how such a package evolves from the ideas of the authors.

The basic functionality deals with coordinates and associated attributes of each coordinate. The coordinates may be Cartesian spatial locations in up to three dimensions, together with an optional temporal location to handle space-time data. Data structures and methods for polygonal regions are also implemented.

Currently we have functions for spatial point-pattern analysis derived from `spatstat` and `splancs`. The geostatistical functionality of the `geoR` library can be easily integrated, as well as other libraries currently in the `CRAN`.

[*]Lancaster University, Lancaster, UK
[†]UWA, Perth, Australia
[‡]University of New Brunswick, Canada
[§]Lancaster University, Lancaster, UK

Spatial Statistics

Spatial Statistics covers any branch of data analysis that uses locations. Simple data could be locations of trees in a forest together with their species and height, populations of different regions within a country, measurements of soil pH sampled over a field, or the positions of cell nuclei observed under a microscope. More complex data could involve a number of coordinates for each entity, such as tracking data from a number of individually identified animals. In this example we could also have the time of each tracking point, giving an example of a space-time dataset.

Cressie's book *Statistics for Spatial Data* [3] is a good reference on spatial statistics. It discusses most of what is included in our code, except for some of the functions that are on the leading edge of statistical methodology. Other spatial statistics books are available.

R/S Spatial Statistics libraries

There are many libraries for R/S that deal with spatial data. Some are full packages with data manipulation, display, and analysis routines. Others are single-purpose codes, often a simple R/S wrapper round some C or Fortran subroutines. This seems to be because most R and S development occurs in the academic field where leading-edge analytical routines are of interest, and so we end up with small packages that do technical things with little common basic functionality. One of the original goals of `splancs` was to provide those basic routines. Now with new functionality in R we feel a new spatial base is required.

Spatial statistics for R has been well-featured in the R-newsletter, for example by Ripley [4] and Bivand [5], where many packages that handle spatial data are reviewed.

`splancs` and `spatstat`

The `splancs` package was probably one of the first spatial data packages written for the S language. It was initially developed under Splus 2.3, and has received only minor upgrades and extensions since then. The port to R was done by Roger Bivand.

`spatstat` was written for the display and analysis of point-pattern data. It deals with points and the polygonal regions in which they are found. It was written for a more recent version of the S language than `splancs`, and so uses some of the more modern features, such as objects and formula notation.

It stores points in `ppp` objects. Since the polygonal area, or 'window', is such an intrinsic part of a realisation of a point pattern, there must be a specification of the region in the `ppp` object. The implementation also allows a single 'mark' variable to be associated with each point.

Spatial data in `splancs` is restricted to 2-dimensional spatial data, although there are a few functions that deal with space-time data. The package uses 2-column matrices for storing points, and any temporal data should be stored in a separate vector. More complex data structures were not implemented in Splus at the time.

Analysis routines in `splancs` concentrated on applications in environmental epidemiology, and included nearest-neighbour and second-order point pattern statistics, as well as model-fitting for "raised incidence" around a specified point. The `spatstat` analytical focus appears to be more toward development of cutting-edge point-process modelling, but includes many of the simpler descriptive tools that make the package much more rounded.

As an illustration we will examine one of the nearest-neighbour calculations as a comparison between packages. This will compute the distribution of nearest-neighbour distances from one set of points to another.

`spatstat`

First we generate a `ppp` object using the `spatstat` function for a uniform Poisson process. Then we assign marks of 0 and 1 to the data at random. After computing the distribution of the distances from type-0 points to the nearest type-1 point we plot the value for the uncorrected, 'raw' estimate:

```
> pp <- runifpoispp(100)
> pp$marks <- factor(sample(0:1, pp$n, replace=TRUE))
> Gx <- Gcross(pp,0,1)
> plot(Gx$r,Gx$raw)
```

Several things are happening here that are invisible to the user at first. `spatstat` tries hard to give default values to arguments wherever possible, simplifying the calling sequence. The `Gcross()` call needs a window in which to do edge corrections for some of the estimators, and since we don't supply one it uses a default of a unit square centred on $(0.5, 0.5)$. Also it has computed the distribution using a range of distances to span the data. This distance range can be supplied but a default is computed otherwise.

The returned object is a data frame with the distance values as the `$r` component and several other columns including different estimators of the distribution. It also includes the theoretical value for Poisson processes with the same intensity.

`splancs`

To illustrate the same calculation in `splancs` we will use the same data by extracting it from the `ppp` object. We will construct two matrices for the two different sets of points, and use the distance values returned by `Gcross()` as the domain of the calculation of the distribution.

```
> xy0 <- cbind(pp$x,pp$y)[pp$marks==0,]
> xy1 <- cbind(pp$x,pp$y)[pp$marks==1,]
> F10 <- Fhat(xy1,xy0,Gx$r)
> lines(Gx$r,F10)
```

Note that the `splancs` function is called `Fhat()`, and the ordering of the definition of the points is reversed compared to `spatstat`. The distance scale must be specified since `splancs` will not supply a default. The `Fhat()` function only returns the 'raw' estimate of the distribution, and so does not require a window for edge corrections. The return value is a vector of the same length as `Gx$r`.

`Rasp` Design Thoughts

Initial development of `Rasp` took place in Perth during August 2002 by Adrian Baddeley, Rolf Turner and Barry Rowlingson. It soon became clear that there was a fundamental difference between what the authors were trying to achieve. Adrian and Rolf were still thinking of a package for spatial point-pattern analysis, whereas Barry had in mind an initial lower-level codebase on which more spatial statistical functionality could be built. Certainly the `spatstat ppp` data structure was too closely tied to the point-pattern methodology to be of much use in other areas, for example in geostatistics, where there may be no meaningful polygonal region. There was also considerable discussion over terminology since the information associated to a point in point-pattern analysis is known as a "mark", whereas such information in a modelling context could accurately be called a covariate. Similarly the region in which a point-pattern exists is known as the 'window', but this is too specific a term for a general polygonal region in space.

After some considerable discussion and much mutual education into spatial statistics and software engineering, a short design for how the code should work was drawn up.

Data for Spatial Statistics

There seem to be as many ways of storing spatial point data as there are packages for manipulating them. What structure should we choose for `Rasp`? The simple two-column vector? The data frame with `$x` and `$y` components? Instead we chose none of these. We do not enforce any structure on the data fed to `Rasp` routines – instead we make the condition that the object must implement a method, `sp.coords`, that returns the coordinates. It is then left up to the object to return the coordinates as it sees fit.

To illustrate this we define a spatial data frame, and show how `sp.coords` works in this context. Suppose we have some quantities sampled at a number of locations. This information is read into a data frame in the usual way. We then create a spatial data frame by giving it coordinates using the `sp.coords<-` method:

```
> water <- read.table("water.dat")
> sp.coords(water) <- cbind(water$x,water$y)
```

This has modified the object's class. It now displays with the coordinates added to the data frame:

```
> class(water)
[1] "spatial.data.frame" "data.frame"
> water
               coords        x         y  z1          z2         z3
1  (25.7307, 1.20804) 25.73072  1.208041 -13  0.25666607  0.1330937
2  (90.0111, 33.6031) 90.01106 33.603051  18 -1.89270100  1.4347811
3  (96.8213, 66.4864) 96.82130 66.486440  12 -1.59967112 -2.4597535
4   (91.881, 24.9425) 91.88104 24.942455  -5  1.73899873  2.6009416
5   (95.8498, 18.917) 95.84980 18.916989 -12 -0.92256835 -0.6764757
6  (16.0386, 31.5343) 16.03858 31.534267  -5  1.07900015 -1.1096188
```

```
7   (81.864, 84.2214) 81.86399 84.221355   3 -3.21402785  2.5181521
8  (17.2057, 79.7311) 17.20568 79.731139 -18  0.59890535  4.5298231
9  (71.3799, 94.1818) 71.37992 94.181840  -2  0.13862134  0.5545065
10 (23.2602, 52.8181) 23.26015 52.818109  -3  0.01097744  1.7003302
```

Now that we have created this object, its behaviour will be slightly different from a normal data frame. For example the `plot` method will display a dot-map of the locations. To get the usual plot method for the data frame requires `plot(as.data.frame(water))`. Subscripting rows and columns from the object returns another spatial data frame with changed dimensions. Other features of data frames, such as `dim()` or `names()` work as expected, and you can even use it in a `glm` function with the `data=` parameter and put the names in the formula.

But what have we gained from this? Firstly, the coordinates are freed from the restriction of being called `x` or `y`, and these names are now free to be used in the data frame for other purposes. Also, you may have data that has several coordinates associated with each data point (such as a birth location and a death location), and you can assign each one to `sp.coords` when you want to work with that as the location. But perhaps the biggest win with this scheme is that you can feed this object to a `Rasp` function and it will get the coordinates.

This mechanism can be extended to almost any class of object. Suppose you have data that forms a ragged list, with different numbers of values for each data point. We could implement a spatial list object, and as long as an `sp.coords` method is implemented for that class then all the `Rasp` functionality will be available to it. There is no need to coerce it to a spatial point type.

A default method for `sp.coords` will mop up cases where an object with no class is passed, such as two-column matrices or lists with no other geographic data apart from a `$x` and `$y` component. This way the user can either adopt the simple route of keeping spatial data in simple vectors and matrices, or build other objects such as data frames and store the spatial data for each item within it.

Point-pattern Analysis Routines

To illustrate the thinking behind the `Rasp` point-pattern analysis routines we use the same nearest-neighbour calculation as before.

Firstly we demonstrate a function that enables the computation to be done in a manner similar to the `splancs` way of doing things. This takes two objects from which coordinates are extracted (using `sp.coords()`) and also needs a window and distance scale. We take the distance scale from the earlier example.

```
> Gxy2 <- G.biv2(xy0,xy1,rectangle(),r=Gx$r)
```

Although we have re-used the `xy0` and `xy1` objects from the `splancs` example, these could be any objects that return meaningful values with the `sp.coords()` function, such as spatial data frames.

If all the locations are stored in a single spatial data frame with mark variables, it is more convenient to use the `G.biv()` function, and use the

marks to specify the two classes of points. Here we construct a spatial data frame from the old `ppp` object and use a formula to specify from which points we are computing the nearest-neighbours:

```
> pps <- spatial.data.frame(data.frame(Type=pp$marks),
        coords=cbind(pp$x,pp$y))
> Gsx <-  G.biv(pps,~Type==0,r=Gx$r,window=rectangle())
```

Note that this computes nearest-neighbour distances for type-0 points to all non-type-0 points (in this case the type-1 points). If the data has more complex markings, then two formulae can be given to specify the partition of the points. The following example is exactly equivalent to the previous one if there are only type-0 and type-1 events in the data:

```
> Gsx <-  G.biv(pps,~Type==0,~Type==1,r=Gx$r,window=rectangle())
```

In fact these two arguments can be almost any expression that evaluates to a logical vector of the same length as the number of points in the first argument. Here we generate a random partitioning of the points into type-0 and type-1 points and compute the nearest-neighbour distribution again:

```
> npart <- sample(0:1,npts(pps),replace=T)
> Gp1 <- G.biv(pps,npart==0,r=Gx$r,window=rectangle())
```

We have tried in `Rasp` to make the analysis routines as flexible as possible. Thanks to the object-oriented way of getting coordinates from objects they can work on simple matrices and lists or more complex data structures.

Higher Dimensions and Space-time Data

We can extend our 2-dimensional framework to accommodate further spatial dimensions and a temporal dimension. We do this by adding an extra argument to the `sp.coords()` function to specify which axes we are dealing with. For example we can set up an object to have 2-dimensional spatial coordinates and a single time coordinate like so:

```
> sp.coords(foo,'xyt') <- cbind(x,y,t)
```

When a function wishes to get the spatial coordinates for this object it can do `sp.coords(foo)`, and to get the temporal locations it can call the function as `sp.coords(foo,'t')`. For three-dimensional data we use 'xyz', and can set or get individual components:

```
> sp.coords(foo,'xyz') <- cbind(x,y,z)
> hist(sp.coords(foo,'z'))
```

This scheme means that functions can get the dimensionality of data that they require from the supplied objects without having to worry about the internal arrangement of the objects.

Functionality - Current

`Rasp` currently has support for the following functions – most of which have been taken from `spatstat` and `splancs`:

- **Spatial Data** – Create spatial data frames, point-pattern objects, translations, rotations, shifts etc. Create spatial windows as rectangles, polygons, binary raster grid masks.
- **Graphics** – Display points with correct aspect ratio. Display spatial windows.
- **Point-pattern analysis** – F-, G-, and J- nearest-neighbour analyses. Ripley's K-function, inhomogeneous K-function. Edge corrections.
- **Point-pattern model-fitting** – maximum pseudo-likelihood fitting for a wide range of models. Include trends and edge corrections.

Functionality - Near Future

Some functions from `spatstat` and `splancs` should be easily transportable to the new framework. Most of the code is ready, and it just needs converting to use the new object-oriented methods.

- **Space-Time Clustering** – Tests and displays for space-time cluster detection. Monte-Carlo and analytical tests. Diagnostic plots.
- **Point-pattern generation** – assorted process models available including Poisson, Matérn inhibition, Neyman-Scott. Metropolis-Hastings algorithm for Strauss processes.
- **Raised Incidence Modelling** – isotropic and directional modelling of cases and controls of disease as a function of distance from a specified point or points

Functionality - Not-so-near Future: Spatial GLMs and Geostatistics

Whereas point-pattern analysis treats the locations of the points as being of intrinsic interest, geostatistics treats them as fixed and is interested in modelling measurements taken at the points, perhaps for interpolating them over a region.

A great deal of code for this is already available in the `geoR` [6] and `geoRglm` [7] packages. These use their own data structures for spatial data where the coordinates, measurements, and covariates are stored in a list structure. There appears to be no clean interface to access to the various parts of this structure.

Using spatial data frames, it should be possible to provide a formula-oriented interface to these libraries. This will make their functions more analogous to the standard `lm()` or `glm()` functions, but with extra arguments to specify the covariance structure or other model parameters.

Conclusion

Hopefully we have started work on code that can form the base of any spatial data analysis package.

The library is released under the GPL and is available from the Sourceforge web site via http://www.sourceforge.net/projects/r-asp and we encourage other developers of spatial packages to download and help develop the code.

References

[1] Adrian Baddeley and Rolf Turner. spatstat: An Splus/R library for spatial statistics. http://www.maths.uwa.edu.au/~adrian/spatstat.html.

[2] Barry Rowlingson and Peter Diggle. Splancs: spatial point-pattern analysis code in splus. http://www.maths.lancs.ac.uk/Software/Splancs.

[3] N. A. C. Cressie. *Statistics for spatial data*. Wiley, 1993.

[4] Brian D. Ripley. Spatial statistics in R. *R News*, 1(2):14–15, June 2001.

[5] Roger Bivand. More on spatial data. *R News*, 1(3):13–17, September 2001.

[6] Paulo Ribeiro and Peter Diggle. geoR, a package for geostatistical data analysis using the R software. http://www.est.ufpr.br/geoR/.

[7] Ole Christensen and Paolo Ribeiro. geoRglm: Software for generalised linear spatial models using R. http://www.maths.lancs.ac.uk/~christen/geoRglm/.

Acknowledgements

Barry Rowlingson would like to thank Adrian Baddeley for providing the funding to visit Perth to work on `Rasp` in August 2002.