```
  attr(ff, "source") <- c(deparse(mm),
                          deparse(expr))
  ## return the 'macro'
  ff
}
```

The kernel of `defmacro()` is the call

```
ff <- eval(substitute(
    function(){
      tmp <- substitute(body)
      eval(tmp, parent.frame())
    },
    list(body = expr)))
```

In the `setNA` example this creates a function

```
function(){
  tmp <- substitute(
        df$var[df$var %in% values] <- NA)
  eval(tmp, parent.frame())
}
```

that performs the macro expansion and then evaluates the expanded expression in the calling environment. At this point the function has no formal argument list and most of `defmacro()` is devoted to creating the correct formal argument list.

Finally, as printing of functions in R actually uses the `source` attribute rather deparsing the function, we can make this print in a more user-friendly way. The last lines of `defmacro()` tell the function that its source code should be displayed as

```
macro(df, var, values){
  df$var[df$var %in% values] <- NA
}
```

To see the real source code, strip off the `source` attribute:

```
attr(setNA, "source") <- NULL
```

It is interesting to note that because `substitute` works on the parsed expression, not on a text string, `defmacro` avoids some of the problems with C preprocessor macros. In

```
mul <- defmacro(a, b, expr={a*b})
```

a C programmer might expect `mul(i, j + k)` to expand (incorrectly) to `i*j + k`. In fact it expands correctly, to the equivalent of `i*(j + k)`.

Conclusion

While `defmacro()` has many (ok, one or two) practical uses, its main purpose is to show off the powers of `substitute()`. Manipulating expressions directly with `substitute()` can often let you avoid messing around with pasting and parsing strings, assigning into strange places with `<<-` or using other functions too evil to mention. To make `defmacro` really useful would require local macro variables. Adding these is left as a challenge for the interested reader.

Thomas Lumley
University of Washington, Seattle
tlumley@u.washington.edu

More on Spatial Data Analysis

by Roger Bivand

Introduction

The second issue of *R News* contained presentations of two packages within spatial statistics and an overview of the area; yet another article used a fisheries example with spatial data. The issue also showed that there is still plenty to do before spatial data is as well accommodated as date-time classes are now. This note will add an introduction to the **splancs** package for analysing point patterns, mention briefly work on packages for spatial autocorrelation, and touch on some of the issues raised in handling spatial data when interfacing with geographical information systems (GIS).

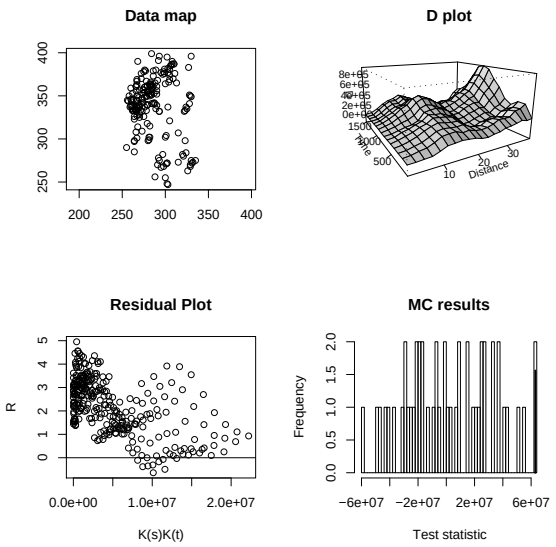

Figure 1: Burkitt's lymphoma — `stdiagn()` output.

The splancs package

The **splancs** package in R is based on a package of that name written in S and FORTRAN in the early 1990's by Barry Rowlingson and Peter Diggle to provide a tool for display and analysis of spatial point pattern data. The functions provided by the package are described in detail in Rowlingson and Diggle (1993), and additional space-time and raised incidence functions introduced in version 2 are described in file 'Sp2doc.ps', available from Barry Rowlingson's web pages[1]. Some of the functionality duplicates that already in the **spatial** package (Venables and Ripley, 1999, Chapter 14) in the **VR** bundle, a recommended package, see Ripley (1981) and Diggle (1983). It is worth noting that the **splancs** functions use an arbitrary polygon to define the study region in the computation of edge effects. The name of the package perhaps plays on the continuing strength of Lancaster University in spatial statistics.

The examples and exercises in Bailey and Gatrell's 1995 spatial data analysis textbook Bailey and Gatrell (1995) add a lot to teaching from it. They are part of software called INFO-MAP packaged with the book and running under DOS. Replicating the functionality needed to study the point pattern examples under R has been important in porting **splancs** to R, especially as the book actually reproduces output from **splancs**. Consequently, the topics covered best by the port are those that carry most weight in Bailey and Gatrell: kernel estimation, nearest neighbour distances, the K function, tests for nearest neighbours and the K function based on complete spatial randomness, and thanks to a contribution by Giovanni Petris, also on the Poisson cluster process. These functions cover Chapter 3; with permission from the authors, the package includes data sets taken from the book.

Topics treated in Chapter 4 include space-time clustering, correcting for variations in the population at risk, and testing for clustering around a specific point source. The **splancs** functions for the first two groups are now fully documented and have datasets that allow the user to re-create data visualizations as shown in print; these are usually the source of the graphics for `example()` for the function concerned. Running `example(stdiagn)` generates the output shown in Figure 1, corresponding to graphics on page 125 in Bailey and Gatrell — here to examine space-time clustering in cases of Burkitt's lymphoma in the West Nile District of Uganda, 1961–75.

Because porting **splancs** has been influenced by the textbook used for teaching, some parts of the original material have been omitted — in particular the `uk()` function to draw a map of England, Scotland and Wales (now partly available in package **blighty**). An alternative point in polygon algorithm has been added for cases where the result should not be arbitrary for points on the polygon boundary (thanks to Barry Rowlingson and Rainer Hurling). In conclusion, learning about point pattern analysis ought not to start by trying out software without access to the underlying references or textbooks, because of the large number of disparate methods available, and the relatively small volume of analyses conducted using them.

[1] http://www.maths.lancs.ac.uk/~rowlings/Splancs/

Spatial autocorrelation

As pointed out in Brian Ripley's overview in the previous issue, the availability of the commercial **S+SpatialStats** module for S-PLUS does make the duplication of implementations less interesting than trying out newer ideas. A topic of some obscurity is that of areal or lattice data, for which the available data are usually aggregations within often arbitrary tessellations or zones, like counties. They differ from continuous data where attributes may be observed at any location, as in geostatistics, often because the attributes are aggregates, like counts of votes cast in electoral districts. While this is perhaps not of mainstream interest, there is active research going on, for example Bavaud (1998), Tiefelsdorf et al. (1999) and Tiefelsdorf (2000).

This activity provides some reason to start coding functions for spatial weights, used to describe the relationships between the spatial zones, and then to go on to implement measures of spatial autocorrelation. So far, a package **spweights** has been released to handle the construction of weights matrices of various kinds, together with associated functions. Comments and suggestions from Elena Moltchanova, Nicholas Lewin-Koh and Stephane Dray have helped, allowing lists of neighbours to be created for distance thresholds and bands between zone centroids, by triangulation, by graph defined neighbourhoods, by k-nearest neighbours, by finding shared polygon boundaries, and reading legacy format sparse representations. Two classes have been established, `nb` for a list of vectors of neighbour indices, and `listw` for a list with an `nb` member and a corresponding list of weights for the chosen weighting scheme. Such a sparse representation permits the handling of $n \times n$ weights matrices when n is large.

An active area of research is the construction of of tessellations of the plane from points, or other objects in a minimum amount of time, since without limiting the search area computations can quickly exceed $O(n^2)$. Research in this area is active in computational geometry, machine learning, pattern recognition, operations research and geography. Reduction of computation in these searching operations requires data structures that facilitate fast range searching and query. This is still an area where R is defi-

cient in relation to packages like S-PLUS and Matlab which both support quadtrees for accelerated neighbourhood searching.

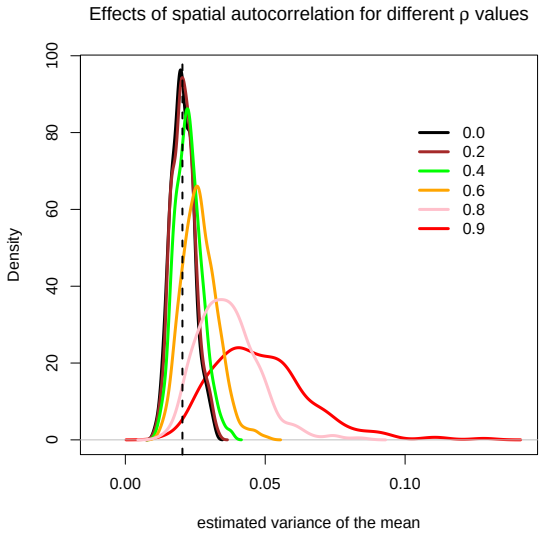

Figure 2: Simulation of the effects of simultaneous autocorrelation on estimates of the variance of the mean $\sigma^2_{\bar{x}}$

Autocorrelation in a sample is important to consider, since the presence of autocorrelation can severely bias the estimation of the variance of the sample moments. Figure 2 is the output of `example(invIrM)`, illustrating the effect of increasing the simultaneous autocorrelation parameter ρ on estimates of the variance $\sigma^2_{\bar{x}}$ of the mean. The simulation used 500 samples of ϵ, a random normal variate with zero mean and unit variance on a 7×7 lattice on a torus (a square grid mapped onto a circular tube to remove edge effects). Autocorrelation is introduced into x by $\mathbf{x} = \sigma(\mathbf{I} - \rho\mathbf{W})^{-1}\boldsymbol{\epsilon}$, where $w_{ij} > 0$ when i, j are neighbours, under certain conditions on ρ (Cliff and Ord, 1981, p. 152). (Sparse representations are not used because the inverse matrix is dense.) The vertical line indicates the estimator assuming independence of observations, for known $\sigma^2 = 1$. Since strong positive autocorrelation erodes the effective number of degrees of freedom markedly, assuming independence with spatial data may be brave.

Tests for autocorrelation using these matrices are implemented in **sptests** — so far Moran's I, Geary's C and for factors, the same-colour join count test. Using the `USArrests` and `state` data sets, and dropping Alaska and Hawaii, we can examine estimates of Moran's I:

$$I = \frac{n}{\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}} \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}(x_i - \bar{x})(x_j - \bar{x})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

for weights matrices using row-standardized k-nearest neighbours schemes for $k = 1, \ldots, 5$, for $w_{ij} = 1/k$ (Cliff and Ord, 1981, p. 17). Moran's I for assault arrests (per 100,000) for 48 US continental states in 1975 for increasing k is shown in the following table: the expectation of I is known ($-1/(n-1)$) and the variance is calculated under randomisation. 'Rank' is the rank of the observed statistic when added to the values from 499 permutations.

k	Moran's I	Variance	Std. deviate	Rank
1	0.405	0.0264	2.63	497
2	0.428	0.0161	3.53	500
3	0.306	0.0109	3.14	500
4	0.294	0.0083	3.46	498
5	0.282	0.0066	3.74	500

As can be seen, it does seem likely that observed rates of assault arrests of k-nearest neighbour states are positively autocorrelated with each other. Using these packages, this may be run for $k = 4$ by:

```
Centers48 <-
  subset(data.frame(x=state.center$x,
                    y=state.center$y),
         !state.name %in% c("Alaska", "Hawaii"))
Arrests48 <-
  subset(USArrests, !rownames(USArrests) %in%
    c("Alaska", "Hawaii"))
k4.48 <- knn2nb(knearneigh(as.matrix(Centers48),
  k=4))
moran.test(x=Arrests48$Assault,
  listw=nb2listw(k4.48))
moran.mc(x=Arrests48$Assault,
  listw=nb2listw(k4.48), nsim=499)
```

where `knearneigh`, `knn2nb` and `nb2listw` are in **spweights** and `moran.test` and `moran.mc` in **sptests**. The exact distribution of Moran's I has been solved as a ratio of quadratic forms Tiefelsdorf and Boots (1995) but is not yet implemented in the package. The MC solution is however more general since it can be applied to any instance of the General Cross Product statistic Hubert et al. (1981).

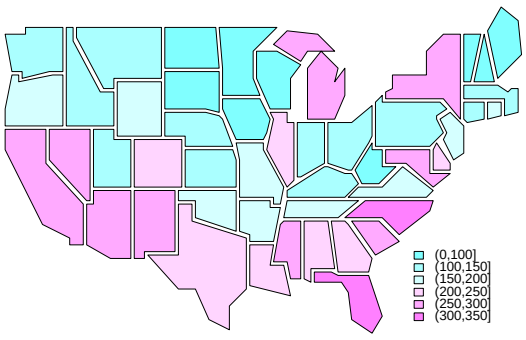

Figure 3: Assault arrests (per 100,000) for 48 US continental states in 1975.

Spatial locations

The `state.center` data used in the above example are documented as being in geographical coordinates, eastings and northings measured in degrees.

In finding near neighbours, distances were measured as if the points representing the states' location were on the plane, which they are not. Metadata about the projection and measurement units of spatial data are of importance in the same way that adequate handling of date and time objects may matter. There may be systematic regularities within the data series itself that are obscured by a lack of registration, and such a lack may make it impossible to combine the data at hand with other data with positional accuracy. In this case, the point locations may be projected onto a plane, here using the Universal Transverse Mercator projection, a standard ellipsoid and datum, for zone 15 (centring the plane East-West on Iowa) and measuring in km, using PROJ.4 software[2].

```
write.table(Centers48, file="ll48.txt",
  row.names=FALSE, col.names=FALSE)
system(paste("proj -m 1:1000 +proj=utm",
  "+zone=15 ll48.txt > utm48.txt"))
Centers48.utm15a <- read.table("utm48.txt")
k3.48utm15 <- knn2nb(knearneigh(as.matrix(
  Centers48.utm15), k=3))
summary(diffnb(k3.48utm15, k3.48, verbose=FALSE))
```

Comparing the neighbour lists for $k = 3$ nearest neighbours for the two sets of coordinates shows that, of the 48 states, 29 had the same 3 nearest neighbours, 18 changed one nearest neighbour, and Idaho changed 2. Despite this, the results of testing for spatial autocorrelation were unchanged, confirming the strong impression of spatial structure visualized in Figure 3.

k	Moran's I	Variance	Std. deviate	Rank
2	0.484	0.0164	3.94	500
3	0.327	0.0108	3.36	499
4	0.315	0.0084	3.67	499

While in this case there is no change in the conclusion drawn, it seems to a geographer to be worth being as careful with spatial metadata as we now can be with temporal metadata. One approach implemented in the **GRASS** package for interfacing R with the GPL'ed geographical information system GRASS[3] is to store the metadata in a separate object recording the current settings of the GIS: region of interest, projection, measurement units, and raster cell resolution. This secures the use of the same metadata on both sides of the interface for a given work session, but separate data objects, such as sets of point coordinates, do not have their own embedded metadata. An alternative approach is used in the new package for importing and exporting portable anymap graphics files (**pixmap**). Here metadata are attached to data objects through attribute values, as `ts` does with time series objects.

In the same way that it has taken time for dates and times to find forms that are both powerful and sufficiently general, spatial data will find a class structure probably with metadata attributes—even date/time metadata attributes. It is obvious that duplicating GIS functionality in R is not a good solution, but much spatial data analysis needs a blend of positional registration, visualization and analytical tools that are not available within the GIS either. This means that the GIS-style metadata need to accompany the data from the GIS to the analysis environment and back again. Connections functions now allow us to move data very freely, but having to rely on intervention by the analyst to make sure that metadata follows is not a good solution.

Prospects

There is now a range of packages for spatial statistics in R. They all have different object structures for positional data, and metadata is handled differently. R still does not have a `map` function on CRAN, but sorting out how to interface with spatial data should help with this. More efficient mechanisms for exchanging data with GIS will add both to access to modern statistical tools by GIS users, and to more appropriate treatment of spatial metadata in spatial statistics. Happily, GPL'ed software like that used for projection above is under active development, and standards for spatial data and spatial reference systems are gelling. These can be given R package wrappers, but there is, though, plenty to do!

Bibliography

T. C. Bailey and A. C. Gatrell. *Interactive spatial data analysis*. Longman, Harlow, 1995. 14

F. Bavaud. Models for spatial weights: a systematic look. *Geographical Analysis*, 30:152–171, 1998. 14

A. D. Cliff and J. K. Ord. *Spatial processes — models and applications*. Pion, London, 1981. 15

P. J. Diggle. *Statistical analysis of spatial point patterns*. Academic Press, London, 1983. 14

L. J. Hubert, R. G. Golledge, and C. M. Costanzo. Generalized procedures for evaluating spatial autocorrelation. *Geographical Analysis*, 13:224–233, 1981. 15

B. D. Ripley. *Spatial statistics*. Wiley, New York, 1981. 14

B. Rowlingson and P. J. Diggle. Splancs: spatial point pattern analysis code in S-PLUS. *Computers and Geosciences*, 19:627–655, 1993. 14

[2] http://www.remotesensing.org/proj/
[3] http://grass.itc.it/

M. Tiefelsdorf. *Modelling spatial processes*, volume 87 of *Lecture notes in earth sciences*. Springer, Berlin, 2000. 14

M. Tiefelsdorf and B. Boots. The exact distribution of Moran's I. *Environment and Planning A*, 27:985–999, 1995. 15

M. Tiefelsdorf, D. A. Griffith, and B. Boots. A variance-stabilizing coding scheme for spatial link matrices. *Environment and Planning A*, 31:165–180, 1999. 14

W. N. Venables and B. D. Ripley. *Modern applied statistics with S-PLUS*. Springer, New York, 1999. 14

Roger Bivand
Economic Geography Section, Department of Economics, Norwegian School of Economics and Business Administration, Bergen, Norway
Roger.Bivand@nhh.no

Object-Oriented Programming in R

by John M. Chambers & Duncan Temple Lang

Although the term *object-oriented programming* (OOP) is sometimes loosely applied to the use of methods in the S language, for the computing community it usually means something quite different, the style of programming associated with Java, C++, and similar languages. OOP in that sense uses a different basic computing model from that in R, specifically supporting mutable objects or references. Special applications in R can benefit from it, in particular for inter-system interfaces to OOP-based languages and event handling. The **OOP** module in the Omegahat software implements the OOP model for computing in R.

S language philosophy and style

When you write software in R, the computations are a mixture of calls to functions and assignments. Although programmers aren't usually consciously thinking about the underlying philosophy or style, there is one, and it affects how we use the language.

One important part of the S language philosophy is that functions ordinarily don't have side effects on objects. A function does some computation, perhaps displays some results, and returns a value. Nothing in the environment from which the function was called will have been changed behind the scenes.

This contrasts with languages which have the notion of a pointer or *reference* to an object. Passing a reference to an object as an argument to a function or routine in the language allows the called function to alter the object referred to, in essentially arbitrary ways. When the function call is complete, any changes to that object persist and are visible to the caller.

In general, S functions don't deal with references, but with objects, and function calls return objects, rather than modifying them. However, the language does include assignment operations as an explicit means of creating and modifying objects in the local frame. Reading the S language source, one can immediately see where any changes in an object can take place: only in the assignment operations for that specific object.[1]

Occasionally, users ask for the addition of references to R. Providing unrestricted references would radically break the style of the language. The "raw pointer" style of programming used in C, for example, would be a bad temptation and could cause chaos for R users, in our opinion.

A more interesting and potentially useful alternative model, however, comes from the languages that support OOP in the usual sense of the term. In these languages, the model for programming is frequently centered around the definition of a class of objects, and of methods defined for that class. The model does support object references, and the methods can alter an object remotely. In this sense, the model is still sharply different from ordinary R programming, and we do not propose it as a replacement.

However, there are a number of applications that can benefit from using the OOP model. One class of examples is inter-system interfaces to languages that use the OOP model, such as Java, Python, and Perl. Being able to mimic in R the class/method structure of OOP software allows us to create a better and more natural interface to that software. R objects built in the OOP style can be used as regular objects in those languages, and any changes made to their state persist. The R code can work directly in terms of the methods in the foreign language, and much of the interface software can be created automatically, using the ability to get back the metadata defining classes (what's called *reflectance* in Java).

Mutable objects (i.e., object references) are also particularly useful when dealing with asynchronous

[1] Assuming that the function doesn't cheat. Almost anything is possible in the S language, in that the evaluator itself is available in the language. For special needs, such as creating programming tools, cheating this way is admirable; otherwise, it is unwise and strongly deprecated.