Journal of Statistical Software
April 2007, Volume 19, Code Snippet 1. http://www.jstatsoft.org/

Auxiliary Cartographic Functions in R: North Arrow, Scale Bar, and Label with a Leader Arrow

Susumu Tanimura
Nagasaki University

Chusi Kuroiwa
University of Tokyo

Tsutomu Mizota
Nagasaki University

Abstract

Auxiliary cartographic components such as a scale bar, a north arrow, and labels frequently play an important role in faciliating the use of maps. Since the current R environment does not fully support these tools, we developed a scale bar for a projected map, a north arrow, and interactive text labeling with a leader arrow. The functionalities of the developed codes and noteworthy points during the execution of these functions are discussed.

Keywords: cartography, thematic map, R, scale bar, north arrow, leader arrow.

1. Introduction

Cartographic expressions of spatial statistical data are important for imparting spatial concepts or ideas. Since using only shapes and colors of objects in a map is not always sufficient to lucidly represent them, we use supplementary components such as a legend, a scale bar, a north arrow, and text labels (Slocum, McMaster, Kessler, and Howard 2005).

Although these components play an important role for map users to understand the map, cartographic functions in R (R Development Core Team 2007) do not include scale bars for a projected map, north arrows, and interactive text labeling except the two functions `layout.north.arrow()` and `layout.scale.bar()` in package **sp**.

In this study, we develop the abovemetioned three auxiliary functions by using the R language. This paper explains the functionality of each tool and illustrates the result of the execution.

2. Labeling

It is troublesome to refer to a specific location on a map without using the label of the location, particularly when the location is indicated in a document.

The function `text()` in R can be used to place text on the graphic device. For example, we can use `text()` to automatically locate every name attribute of a polygon on the centroid calculated by `get.Pcent()` from **maptools** (Lewin-Koh and Bivand 2007). However, in many cases, the labels become congested and overprinted causing the text to be unsightly and unreadable. In such a case, in order to cull the essential labels referred to in the description, we manually reuse `text()` for each selected label. However, there arises another problem we must know the label content prior to its placement. Since dealing with a linkage between a location on a map and an attribute value is typically difficult, only users who are certain of both the location and attribute value can use `text()` for such labeling.

Another solution to avoid the overprint of labels is to move them to an outer place using a leader line or an arrow. However, this has the same disadvantage for the abovementioned labeling.

We develop an interactive labeling function `map.arrow.label()` using the R language, which enables the placement of a label on a polygon map with a leader arrow without the prior knowledge of its content. We internally use linkage between map polygons and attribute values in the application of `inout()` in the **splancs** package (Rowlingson, Diggle, and Bivand 2006). The `map.arrow.label()` code is as follows:

```
map.arrow.label <- function(map=NULL,label,n.label=1,length=.1,cex=.8,
                            box.margin=.1,bgcol="white",tcol="black",
                            adj=c(.5,.5),iconv.from=NULL,iconv.to=NULL,...)
{
  require(splancs)
  if (class(map)!="Map")
    stop("map is not Map object.")
  if (attr(map$Shapes, "shp.type") != "poly")
    stop("Type of map is not polygon.")
  if (missing(label))
    stop("label is missing.")
  ## count total polygons
  poly.n <- length(map$Shapes)
  j <- 0
  while (j < n.label) {
    ## get location of target and label
    cat("Click now the location of polygon and then label\n")
    loc <- unlist(.Internal(locator(2, type ="n")))[c(1,3,2,4)]
    ## draw the leader arrow
    arrows(loc[1],loc[2],loc[3],loc[4],length=length,code=1)
    ## find the content of the label
    loc1 <- matrix(c(loc[1],loc[2]),ncol=2)
    poly.inout <- vector(mode = "logical", length = poly.n)
    for (i in 1:poly.n) {
      poly.inout[i] <- inout(loc1,map$Shapes[[i]]$verts)
    }
    labels <- map$att.data[,which(names(map$att.data) == label)]
    thelabel <- as.character(labels[poly.inout])
    if(is.na(thelabel)) warning("The point may be out of range.")
    ## convert an encoding
    if(length(iconv.from)!=0 & length(iconv.to)!=0) {
      thelabel <- iconv(thelabel,iconv.from,iconv.to)
```

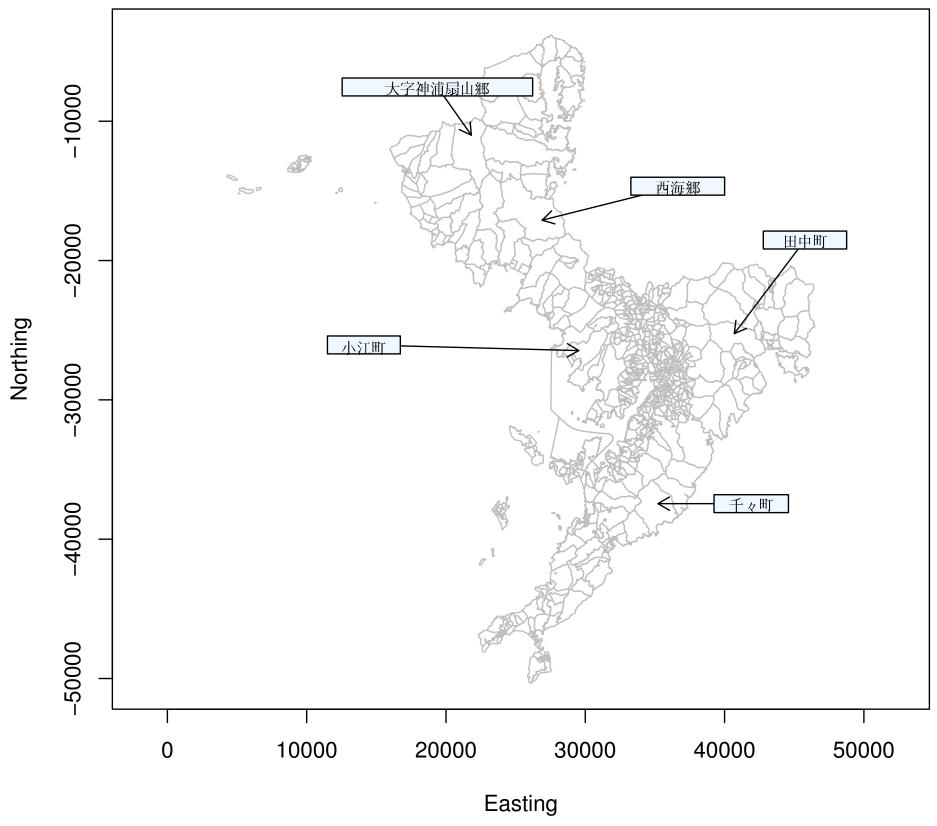

Figure 1: An example of labeling with a leader arrow using `map.arrow.label()`. The Japanese text in the labels was converted from sjis to euc-jp

```
    }
    ## draw a rectangle
    thelabel.size <- c(strwidth(thelabel), strheight(thelabel))
    rect(loc[3]-thelabel.size[1]/(2-box.margin),
         loc[4]-thelabel.size[2]/(2-box.margin),
         loc[3]+thelabel.size[1]/(2-box.margin),
         loc[4]+thelabel.size[2]/(2-box.margin),
         col=bgcol)
    ## place the text of label
    text(loc[3],loc[4],thelabel,adj=adj,cex=cex,col=tcol)
    j <- j+1
  }
}
```

For execution, users must specify the `map` as a base map to be labeled; a base map of this type should exclusively be a "Map object" in the **maptools** package for simplification. Users also must specify the name of variable for label text by `label` option. Note that it must be one of attributes in the Map object and specified in quotes. Users can repeatedly place labels without user intervention any number of times, which is specified by `n.label` option (the default value is one).

The label properties can be adjusted by using options such as `length` for the length of the edges of the arrow head (in inches), `cex` as a multiple proportion for the size of a label text, `box.margin` for the margin of the text in a box, `adj` for the adjustment of the text position, and `bgcol` and `tcol` for the background and text color in the label, respectively. A default label shows black letters against a white background in a rectangular box.

The usage of `map.arrow.label()` is simple and as follows: click on a target polygon with a mouse; then, click again on the location where the label is to be placed. The steps are repeated if the user specifies a value for `n.label`. An example output of `map.arrow.label()` is shown in Figure 1.

The code may face problems when a map has multi-polygons because it assumes a one-to-one matching of the data. A significant improvement is required for handling such data, and we intend focusing on it in the future.

Note that users who need to convert an encoding in the text of a label can use the options `iconv.from` and `iconv.to` in `map.arrow.label()`. This function is essential for users using a language that involves more than one encoding.

3. North arrow

Map users who attempt to find north at the top of a map might be confused due to the absence of an indication of direction such as a north arrow or a graticule on the map. Nevertheless, maps are occasionally rotated due to the shape of the map and/or the mapped space.

In order to provide the direction on the map for R users, we developed the north arrow function `northarrow()` as follows:

```
northarrow <- function(loc,size,bearing=0,cols,cex=1,...) {
  # checking arguments
  if(missing(loc))  stop("loc is missing")
  if(missing(size))  stop("size is missing")
  # default colors are white and black
  if(missing(cols)) cols <- rep(c("white","black"),8)
  # calculating coordinates of polygons
  radii <- rep(size/c(1,4,2,4),4)
  x <- radii[(0:15)+1]*cos((0:15)*pi/8+bearing)+loc[1]
  y <- radii[(0:15)+1]*sin((0:15)*pi/8+bearing)+loc[2]
  # drawing polygons
  for (i in 1:15) {
    x1 <- c(x[i],x[i+1],loc[1])
    y1 <- c(y[i],y[i+1],loc[2])
    polygon(x1,y1,col=cols[i])
  }
  # drawing the last polygon
  polygon(c(x[16],x[1],loc[1]),c(y[16],y[1],loc[2]),col=cols[16])
  # drawing letters
  b <- c("E","N","W","S")
  for (i in 0:3) text((size+par("cxy")[1])*cos(bearing+i*pi/2)+loc[1],
                      (size+par("cxy")[2])*sin(bearing+i*pi/2)+loc[2],b[i+1],
                      cex=cex)
}
```

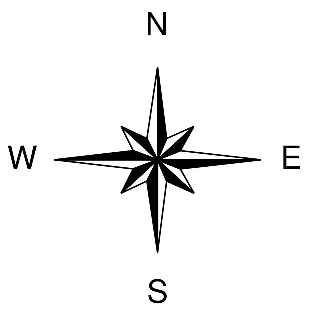

Figure 2: Output of `northarrow()`

The argument for this function consists of the location, size, bearing, colors, and any other graphic parameters. The size of the north arrow and the location of its center position cannot be omitted from the functional arguments, whereas the others can be omitted. The default north arrow is plotted in black against a white background, and it is positioned overhead (i.e., `bearing = 0`), as shown in Figure 2. Users can assign `bearing` for the angle (in degrees) of the north arrow in accordance with the orientation of the map. Alternatively, users can use a northarrow function `layout.north.arrow()` in **sp** package. The function includes two kinds of simple arrow, but no customizable option is available.

4. Scale bar

The scale is indispensable for map users to know the magnitude of reduction and to measure distances. Slocum *et al.* (2005) introduced three traditional types of scales: representative fraction (e.g., 1:24,000), verbal scale, and scale bar (or bar scale). Among these, the scale bar is most suited to a thematic map, particularly when distance information can enhance a user's understanding of the theme (Slocum *et al.* 2005, Chapter 11).

For an unprojected map in R, we can use the `map.scale()` function in the **maps** package. This is applicable to a map created not only by the **maps** package but also by other cartographic packages. However, in R, there is no scale bar applicable to a projected map.

We develop a scale bar function `scalebar()` for a projected map by using the R language. The arguments of this function are the location, length, unit, character size of text, and any other typical graphic parameters. Figure 3 shows a sample output of `scalebar()`. The design of the scale bar is fixed by this initial code. If users require other refined designs, they can be incorporated. However, we have been careful in this regard in order to maintain the clarity of the scale (see discussion). The code for `scalebar()` is as follows:

```
scalebar <- function(loc,length,unit="km",division.cex=.8,...) {
  if(missing(loc)) stop("loc is missing")
  if(missing(length)) stop("length is missing")
  x <- c(0,length/c(4,2,4/3,1),length*1.1)+loc[1]
  y <- c(0,length/(10*3:1))+loc[2]
  cols <- rep(c("black","white"),2)
  for (i in 1:4) rect(x[i],y[1],x[i+1],y[2],col=cols[i])
  for (i in 1:5) segments(x[i],y[2],x[i],y[3])
  labels <- x[c(1,3)]-loc[1]
```

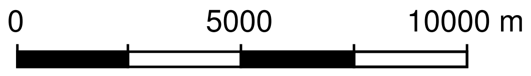

Figure 3: Output of `scalebar()` with `length = 10000` and `unit = 'm'`

```
  labels <- append(labels,paste(x[5]-loc[1],unit))
  text(x[c(1,3,5)],y[4],labels=labels,adj=.5,cex=division.cex)
}
```

It should be noted that using `scalebar()` for a decimal degree or the entire world is not logical because the function cannot convert units and retains them in the unconverted form in the current graphic device. Therefore, if necessary, users must project and coordinate the map appropriately prior to the execution of `scalebar()`.

Users can also use a scalebar function `layout.scale.bar()` in the **sp** package. It only draws a rectangle without text, but works well with `spplot()`.

5. Discussion

Due to the limitations in the R graphic device, the characteristics of the label type with the exception of type size cannot be separately arranged. However, we believe that characteristic parameters for typography, such as fonts, letter and word spacing, kerning, and leading, that are competently adjusted in R are acceptable even for professional use.

The rectangular box of the label is basically opaque and masks underlying objects such as polygons or other labels in order to avoid overprint and ensure label clarity. However, if necessary, users can use a transparent rectangular box with `bgcol = "transparent"`.

In this paper, with regard to the "labeling" section, we propose to cull labels and use them with a leader arrow in order to avoid congestion and overprinting. However, a more sophisticated solution can still be expected because some systems developed specially for cartography, such as geographic information systems (GIS), have already achieved an automated optimal placement of labels. In the future, their algorithms can be imported to R.

The style of the north arrow should be simple so as to not attract attention. The line weights and type must be sufficiently fine; however, we have used the default parameters (e.g., `lwd`) in a graphic device. As a default setting, users draw a scale bar with solid lines that are as thick as lines in other parts of the graphics. Optionally, users who need lines of greater or lesser thickness can adjust the north arrow with the usual graphic parameters.

The letter representing north is fixed as "N" in `northarrow()`. Fewer users may deal with magnetic north abbreviated as "MN" instead of the geographical north "N"; this alternative option has not been currently provided.

Although users can freely decide the location and the size of the north arrow, it is recommended that it be relatively small in order to be inconspicuous, but yet be sufficiently large to find and use. Further, it is recommended that the label be placed in a remote location, preferably near the scale bar.

A graticule is a system of grid lines, normally representing the longitude and latitude. It is used in a very specific case in which the north direction is variable (Slocum *et al.* 2005). Since most thematic maps are projected using a coordinate system, the north arrow is sufficient to indicate the orientation. Consequently, the graticule is not currently available.

The design of the scale bar obtained by `scalebar()` is simple and precise. Bulky, complex, and sloppy designs must be avoided in order to ensure distinct thematic maps. The line weights and type should be sufficiently fine, similar to the abovementioned `northarrow()` function.

By using the options in `scalebar()`, users can also freely decide the location and length of a scale bar on a map. However, both the location and length must be carefully selected for ensuring that the thematic maps are clear. If possible, the scale bar should be placed below the mapped area because map users are familiar with such an arrangement. The scale bar must be sufficiently long to be useful but must not occupy too much space.

When users provide the value of the length to the `scalebar()` function, it should always be an integer and easily workable. For instance, when `length = 7676` is assigned instead of `length = 8000`, it is difficult to obtain the division 7676/4. On the contrary, the quarter division indicated in Figure 3 is definitely 2500 m. Therefore, if the maximum distance in the scale bar is a fractional number, the clarity of the scale and map decreases.

Some scale bars include an "extension scale," which expands the bar to the left of zero. The `scalebar()` function does not support such an extension because this could confuse many map users who expect numbers to the left of zero to be negative as in a number line.

6. Conclusion

In this study, auxiliary cartographic functions were developed in R. The labeling, north arrow, and scale bar are expected to complement the cartographic functions in R, which will help map users to understand the map thoroughly.

References

Lewin-Koh NJ, Bivand R (2007). **maptools***: Tools for Reading and Handling Spatial Objects.* R package version 0.6-8, URL `http://CRAN.R-project.org/`.

R Development Core Team (2007). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL `http://www.R-project.org/`.

Rowlingson B, Diggle P, Bivand R (2006). **splancs***: Spatial and Space–Time Point Pattern Analysis.* R package version 2.01-19, URL `http://www.maths.lancs.ac.uk/~rowlings/Splancs/`.

Slocum TA, McMaster RB, Kessler FC, Howard HH (2005). *Thematic Cartography and Geographic Visualization.* Pearson Education, Inc., Upper Saddle River, NJ, 2nd edition. ISBN 0-13-035123-7.

Affiliation:

Susumu Tanimura
Department of Socio-environmental Medicine
Institute of Tropical Medicine
Nagasaki University
1-12-4 Sakamoto, Nagasaki, Japan 852-8523
E-mail: stanimura-ngs@umin.ac.jp

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 19, Code Snippet 1 *Submitted:* 2005-12-27
April 2007 *Accepted:* 2007-04-10