R, lattice and RgoogleMaps: A practical framework for the development of new geovisualizations

Karl Ropkins[1*], David C. Carslaw[2], Said Munir[1,3]
Haibo Chen[1]
[1]Institute for Transport Studies, University of Leeds, LS2 9JT, UK
[2]Science Policy Group, Environmental Research Group, King's College London, SE1 9NH, UK
[3]Umm Al Qura University, Makkah 21955, Saudi Arabia
*Corresponding author (k.ropkins@its.leeds.ac.uk)

Abstract

The R package *lattice* provides an elegant and highly powerful implementation of the Trellis plotting structure described by Cleveland and colleagues. Therefore, the combination of another R package *RgoogleMaps* and *lattice* is an obvious starting point for the development of map based data visualization and analysis tools. While 'stand alone' plot functions developed using this combination are of obvious value, the code framework, which allows simple-to-use and flexible means of merging Google Maps and multiple *lattice* plot panels, is of possibly wider interest because it provides a highly practical template for the rapid third-party development of novel geovisualization functions. To support this approach, this paper presents results from on-going work to integrate *lattice* and *RgoogleMaps*. Firstly, the basic *lattice* style code is used to demonstrate panel layering and plot development methods, and their use to quickly generate a wide range of geovisualizations. Then, a 'higher level' version of the code, the *GoogleMapsPlot* function in the R package *openair*, is used as an example to illustrate its incorporation and use in 'third party' code.

Key Words: Geovisualization; R; R package *lattice*; R package *openair*; R package *RgoogleMap*

1. Introduction

In additional to the standard (or base) installation graphics packages, graphics and grid, there are also a number of higher level graphics packages available for use in R (R Development Core Team, 2012). lattice (Sarkar, 2008) is one of those packages. It is an elegant and highly powerful implementation of the Trellis1 plotting structure described by Cleveland and colleagues (Cleveland, 1993; Becker et al. 1996). Other popular alternatives include the vcd package developed by Meyer et al (2006) and based on methods described by Friendly (2000) and Hadley Wickham's ggplot2 package based on methods described by Wilkinson (2005). Each of these has different advantages and disadvantages, and we make no claim that any one of these is the better data visualization tool. However, lattice has three features which are perhaps worth recognising when developing data visualization tools in R.

Firstly, lattice uses a formula structure to shape plots. Data is typically presented in forms such as z ~ x * y | cond, where x and y are the x- and y-axis components, z is

[1] The "Trellis" name derives from the appearance of the plots which are often made up of rows and columns of multilayer panels that are sometimes compared to garden trelliswork.

any data to be used as a perpendicular z axis component or data surface and cond is any additional information that can be used to subsample (condition or bank) the x, y and z data by if multiple-panel plots are required. While this might seem an arbitrary choice, it means that plots can be very quickly and intuitively rearranged and, perhaps, even more importantly that the formula used to generate the plot is often also the formula that would also be used to model any observed plot features in subsequent analyses.

Secondly, the lattice Trellis structure is particularly well suited to the visualization and interpretation of multidimensional datasets because additional data can be easily incorporated by expansion of the formula argument in the call. The most commonly coded/used example is expansion by multiple conditioning (z ~ x * y | cond1 + cond2 …), but other options are also possible for some lattice plot functions (e.g. z1 + z2 … ~ x * y | cond, z ~ x1 +x2 … * y | cond, z ~ x * y1 + y2 … | cond, etc ; Figure 1).

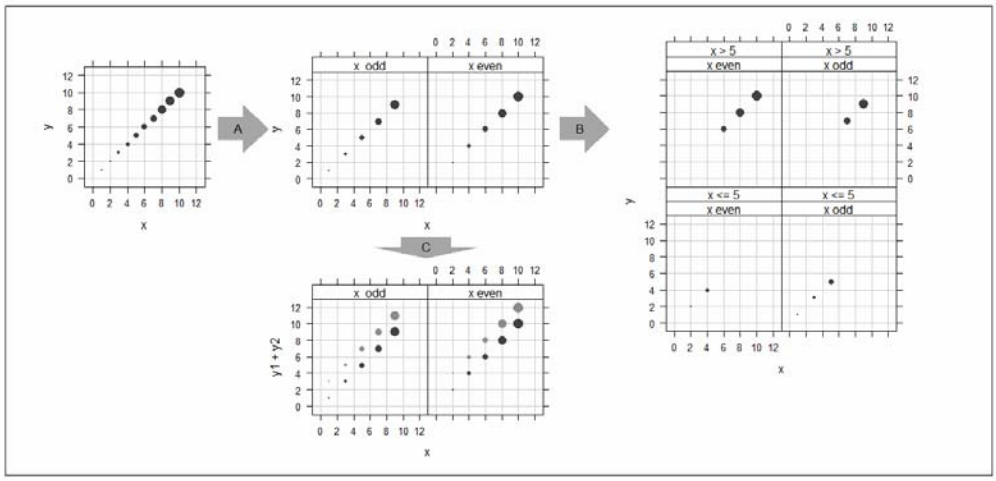

Figure 1: Example expansions of a lattice (Trellis) plot. Starting from the general formula z ~ x * y and z = y = x = 1:10 with z linked to the plot symbol size argument cex, then applying: (A) Conditioning using the term x is odd/even (z ~ x * y | cond); (B) Multiple conditioning using the terms x is odd/even and x <=/> 5 (z ~ x * y | cond1 + cond2); (C) Conditioning using the term x is odd/even and y grouping, y1 = y; y2 = y + 2; (z ~ y1 + y2 * x | cond).

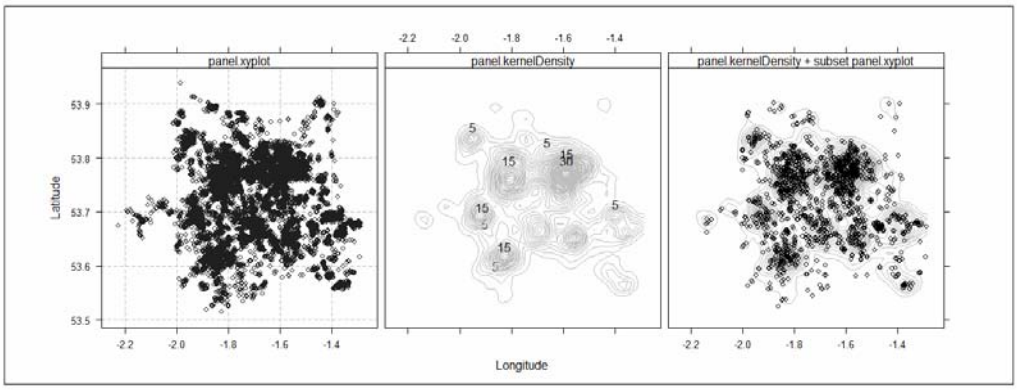

Figure 2: Example plot build using panels using West Yorkshire crime data for June 2011 (source: http://policeapi2.rkh.co.uk): (Left) all data scatter-plotted with panel.xyplot; (Middle) kernel density estimation for all data with panel.kernelDensity; (Right) vehicle crime subset scatter-plotted over the all data kernel density.

Finally, *lattice* includes a large number of panel functions which are used to generate different plots. A number of the panels are dedicated functions, e.g. the *panel.xyplot* function which is used to generate that standard xy scatter plot panel content associated with the *lattice* function *xyplot*. However, others generate more generic panel elements, e.g. lines, arrows, polygons and text, and any of these can be layered to generate novel plot outputs (as in Figure 2). This makes *lattice* a particularly useful 'blocking block' when developing novel data visualizations.

Markus Loecher's *RgoogleMaps* package (Loecher & Sense Networks, 2011) is a highly convenient interface for R and the Google Static Maps server (https://developers.google.com/maps/documentation/staticmaps/). It handles multiple tasks: it allows users to generate Google Map server queries; recovers and imports standard Google Maps outputs into R; and, provides proper coordinate scaling for the routine use of these as backgrounds images for overlay plots. Therefore, the combination of *lattice* and *RgoogleMaps* would appear to an obvious starting point for the development of map based data visualization and analysis tools.

Geovisualization is a particularly powerful class of data visualisation. The combination of statistical graphics and geographical reference is something that holds a strong fascination for many of us, and works on multiple (emotive, intuitive and informative) levels. See e.g. related discussion in Butler (2006), de Adana et al (2009) or Batty et al (2010). Even relatively simple geovisualizations can be highly effective 'scene setters'. However, a wide range of coloration and glyph strategies have been extensively used in both traditional cartographic and more recent computer-based geovisualizations to further enhance the visualisation for more complex multivariate datasets, e.g. c.f. Kraaf & Ormeling (1996) and Dykes et al (2006).

2. Software

When making any new software, developers face a number of design questions. Often the tendency is to favour highly structured functions that enforce their own ideas of how data should be handled and visualised. This type of approach generates 'stand alone' functions which are perhaps best suited to and preferred by non-expert audiences. However, many users are typically experts in one or more area associated with the visualization process, e.g. the spatial handling of data, the data types or sources, or visualisation options, and for these users such rigid approaches can often actually hinder efforts to best represent their data. Here, what users typically require is more direct access to the underlying code that controls the different aspects of the main plot functions. Furthermore, many of the coding steps associated with visualisation are relatively generic and therefore transferrable, e.g. the latitude and longitude handling when locally scaling data is basically the same regardless of any subsequent data handling. Therefore, an approach analogous to the *panel…* approach adopted in the *lattice* package could provide a highly effective framework for the rapid development of novel geovisualization functions. To support this approach, this paper presents results from on-going work to integrate *lattice* and *RgoogleMaps*, both as a direct geovisualization tool and an aid for third parties wishing to develop novel geovisualisations of their own. In keeping with the spirit of R and open software development practices, associated geovisualization code and the extension of this approach to other (non-geo) data visualizations are provided in the *loa* package (Ropkins, 2012).

R source: http://www.R-project.org
loa source: https://sites.google.com/site/karlropkins/rpackages/loa/

3. The *loa* Function *GoogleMap*

GoogleMap, the main geovisualization function in *loa*, takes the default form:

GoogleMap(*x*, *data* = NULL, *map* = NULL,
map.panel = *panel.GoogleMapsRaster*,
panel = *panel.xyplot*, *recolor.map* = FALSE, …)

x is a formula of the general form *z* ~ *latitude* * *longitude* | *cond*, with required elements *latitude* and *longitude* (WGS-84 ellipsoid) and optional elements *z* and *cond*. The default source for *x* elements is the R workspace unless an alternative is supplied as *data* as a specific source. The map layer of the visualisation is obtained using the *loa* function *makeMapArg* unless an alternative is supplied as *map*. *makeMapArg* uses other call arguments (e.g. data *latitude* and *longitude* ranges and *xlim* and *ylim* ranges) to set required maps dimensions and returns a slightly modified form of the Google Maps static map output[2] generated by the *RgoogleMaps* function *GetMap*[3]. *map.panel* and *panel* are the functions used to generate the map and data layers of the geovisualization, respectively. *recolor.map* is an additional argument that allows users to further modify the appearance of the map layer. All other supplied arguments are passed on to other functions, some with minor local modification. This mechanism provides access to a wide range of common plot arguments, e.g. *col*, *pch* and *cex* to control the color, type and size of symbols plotted using *panel*. A trivial example (output shown as Figure 3) of the *loa GoogleMap* function usage is:

```
latitude <- c(32.893426, 32.770893, 32.799973, 32.731624),
longitude <- c(-117.250214, -117.251501, -117.256393, -117.146015),
area <- c("Torrey Pines", "Mission Beach", "Mission Beach", "Town")
GoogleMap(~ latitude * longitude | area, col = "blue")
```

Figure 3: loa GoogleMap plot generated using above trivial example. Google Map data subject to copyright © 2012 Google/GeoBais-DE/BKG/Tele Atlas.

[2] The Google Map API returns the smallest panel it can generate that contains the supplied latitude/longitude range. So, the returned map range may not be exactly the requested map range.
[3] The *loa* function *getMapArg* allows users to recover the map component from previous *GoogleMap* plots. Google limits free access to the Google Maps API to one thousand requests per day. So, this is a useful option for, e.g., animations or multiple 'same map' plots.

4. The Basic Plot Structure *quickMap*

The plot code can be accessed using the R function *edit* both in its full form in *GoogleMap* and in simplified form in the trivial function *quickMap* also supplied as part of the *loa* package. Latitudes and longitudes are transformed to a local scale by *RgoogleMaps* function *LatLon2XY.centered* in order to provide the conventional cartographic 2-dimensional projection of the Earth's (3-dimensional) curved surface. Therefore, in addition to *makeMapArg* and *panel.googleMapsRaster* both plot functions also make use of several axis management functions. The main function amongst these is *axis.components.googleMaps*, a convenient wrapper for x (longitude) and y (latitude) axis handlers that in turn align supplied (latitude/longitude) and local (x/y) scales using *LatLon2XY.centered* and its inverse form *XY2LatLon*. Similarly, any additional data placed on the map (or map rescaling, e.g. using the *lattice* arguments *xlim* and *ylim* arguments) needs to be locally scaled before plotting, again using *LatLon2XY.centered.*

5. Routine *lattice* Plot Handlers

If a code author develops move than one plot function using a package, it is highly likely that these functions will share a number of common elements, e.g. plot naming and argument handling methods. This is an obvious convenience for the plot users who can much more intuitively work with multiple plots that share common elements. However, it is also a convenience for the author who can reuse the transferable subroutines within their code, and, by extension, for third-party plot developers if those subroutines are packaged as workhorse functions.

Therefore, a number of common plot element handlers are included in the *loa* package as documented functions. Perhaps the most amenable of these are the workhorses written for the visible elements of plots, e.g. *colHandler* and *cexHandler* functions for the management of plot point color and size, and *localScaleHandler* for generation of alternative plot axes. However, there are also a number of workhorses for the routine handling of some of the less obvious generic structure within *lattice*, most notably *panelPal*. *lattice* makes extensive use of subscripts to appropriately handle different data series in conditioned multiple panel plots. *panelPal* is a wrapper for the panel argument of a *lattice* plot function which routinely manages some aspects of subscripting, as well as providing an alternative mechanism for handling data grouping within *lattice* plots.

6. The Data Layer *map.panel*

By default, the map layer of *loa GoogleMap* plots is a Google Map API output[4] imported into R using the *RgoogleMaps GetMap* function and modified for use with *lattice* plots. Map layers can very easily dominate geovisualizations, especially if they are overly colourful or text-dense. In order to balance the appearance of map and data layers it is

[4] The *GoogleMap* function was developed specifically for *lattice* (Trellis-style) Google Map *geovisualizations*. However, one of the longer term intentions is extend the range of map layers available by, e.g., developing sister functions using other map sources, e.g. the Natural Earth spatial polygons (http://www.naturalearthdata.com/downloads). The intention here is to build similar plot structure about different *makeMapArg* and *panel.…Map* functions to provide an intuitive and readily interchangeable range of geovisualization tools.

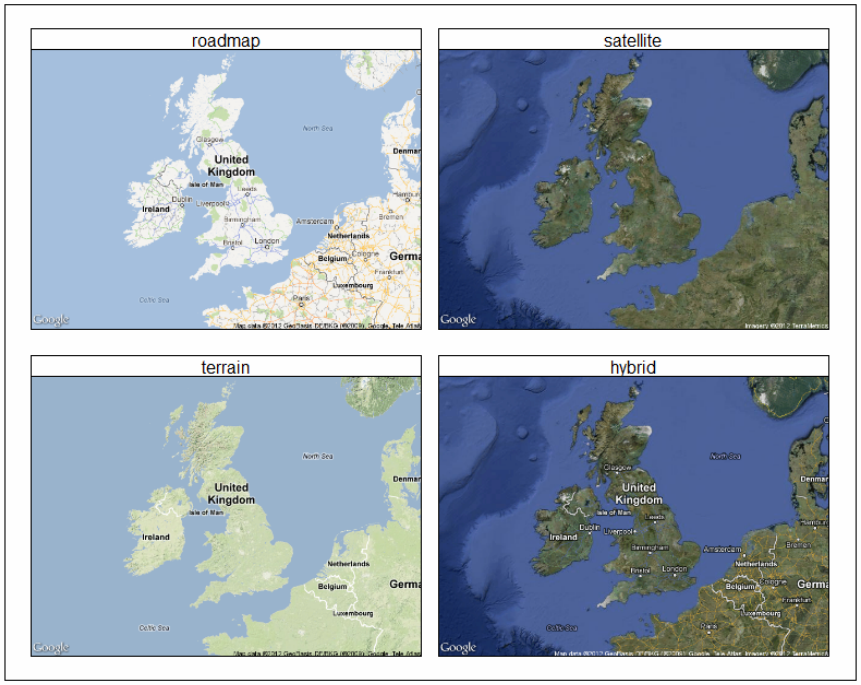

Figure 4: loa GoogleMap map layers generated using R calls in the form GoogleMap(..., maptype = "[option]") and options roadmap, satellite, terrain and hybrid. Google Map data subject to copyright © 2012 Google/Terrametrics/GeoBais-DE/BKG/Tele Atlas.

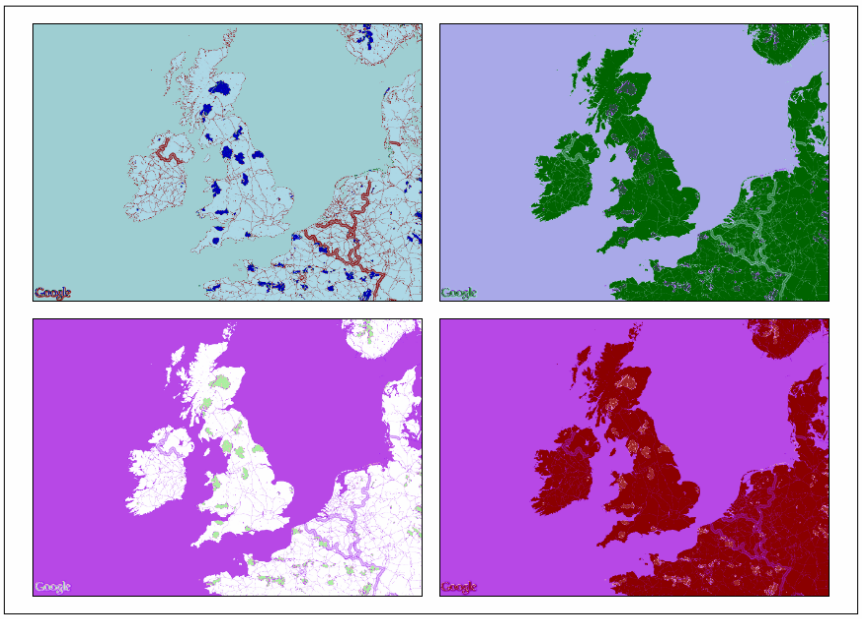

Figure 5: loa GoogleMap map layers (map.panel) generated using R calls in the form GoogleMap(..., maptype = "roadmap", recolor.map = [option], path = [option]) and different recolor.map and path settings. All plots are reworkings of the top left panel in Figure 4. Note in particular that the use of element control via path allows the user to remove map text from the (Google) map layer. Google Map data subject to copyright © 2012 Google/GeoBais-DE/BKG/Tele Atlas.

therefore very important that users have fine control of both. So, *GoogleMap* allows further map appearance modification via two mechanisms. Firstly, it allows the direct access to *GetMap* arguments such as *maptype* and *path*. *maptype* sets the type of map construct requested from the Google Map API, using options including "satellite", "terrain", "hybrid", and "mobile" (Figure 4). *path* allows users to side-step both *loa* and *RgoogleMaps* and add text to the Google Maps API call directly. This provides access to further map modifiers, including options to reset map element colors individually, as described in https://developers.google.com/maps/documentation/staticmaps/. Secondly, the loa function itself *GoogleMap* includes an additional argument, *recolor.map*, which allows users to redefine maps color scales as part of the import step (Figure 5).

7. The Data Layer *panel*

By default the *GoogleMap* data layer is the standard *lattice* scatter plot data handler *panel.xyplot*. Also by default, plot point size and color are linked to the *z* element of the plot call, although both options can be reset using the common R data point size and color arguments, *cex* and *col*. Other aspects of the default plot appearance can also be managed using common R arguments, e.g. *xlab*, *ylab* and *main* for adding x- and y-axis and title captions, making the routine use of the default form of *GoogleMap* function highly intuitive for anyone already familiar with R graphical outputs. However, more importantly other *panel…* functions can be supplied as part of the plot call using the argument *panel* to generate alternative data layer visualizations. Furthermore, these can be existing panel functions, e.g. those in *lattice* or dependent packages, novel functions written by the plot developer themselves or built-up using combinations of panels and code. This means even users with relatively limited experience of code development can very quickly start putting together unique data visualisations. To illustrate this four very different geovisualizations are presented in Figures 6 to 9.

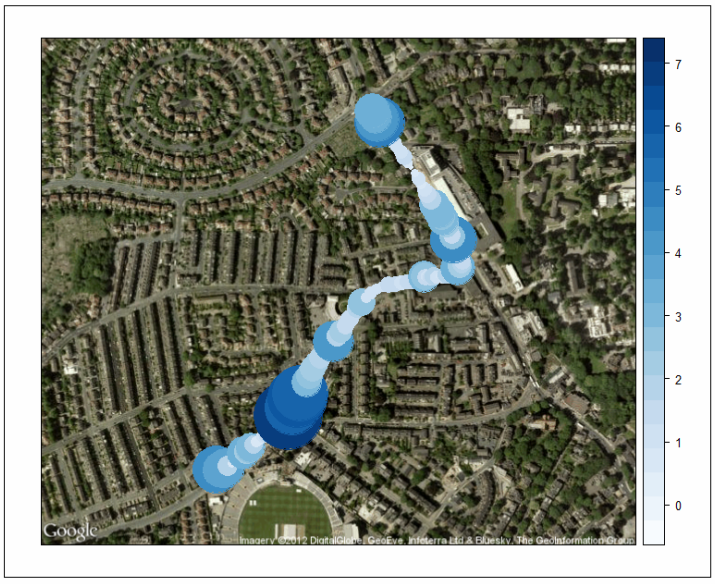

Figure 6: loa GoogleMap bubble plot visualization of vehicle CO_2 emissions. Mobile data was collected at 1-second resolution using a standard (EURO II petrol) passenger car and a Horiba OBS-1300 PEMS/GPS, and emissions reported in grams. Google Map data subject to copyright © 2012 Google/Terrametrics/GeoBais-DE/BKG/Tele Atlas.

Figure 6 uses mobile data collected using a passenger car fitted with a Portable Emission Measurement System (PEMS) and Global Positioning System (GPS). The figure visualizes vehicle exhaust CO_2 emissions in the form *emissions ~ latitude * longitude* over a satellite image map layer (*maptype = "satellite"*).

By default the *loa GoogleMap* function used plot handlers (*colHandlers* and *cexHandler*) to color-code and size plotted points/symbols according to value of the *z* term in the plot formula. But either color or size can be fixed or forced to an alternative scale using the *col* and *cex* call arguments, respectively. Therefore, in this, its' simplest form, *GoogleMap* generates an output analogous to the bubble plots geovisualizations widely used elsewhere, e.g. in Bivand et al (2008), Gesmann & de Castillo (2011), South (2011), Kilibarda & Bajat (2012) and SAS (2012). Here, the visualization illustrates some very important points associated with vehicle emissions: Emissions are not uniformly distributed over a vehicle journey, and, more importantly some of the very highest contributions are associated with quite discrete events, e.g. turns at junctions and stop/starts at signals. Both data conditioning, e.g. by speed or acceleration rate, and map analysis, e.g. using Google Maps, traffic network shape files or Streetview images, can then be used to extend to analysis and discussion of findings.

Figure 7 uses West Yorkshire Police crime records, including logged event locations and crime types, from July 2011 (source: http://policeapi2.rkh.co.uk). Data is visualized in the form *~ latitude * longitude | crime.type* but uses a kernel density estimator data panel, *panel.kernelDensity*, in place of the standard scatter plot data handler, *panel.xyplot*. Both the map and data layers are user-recolored.

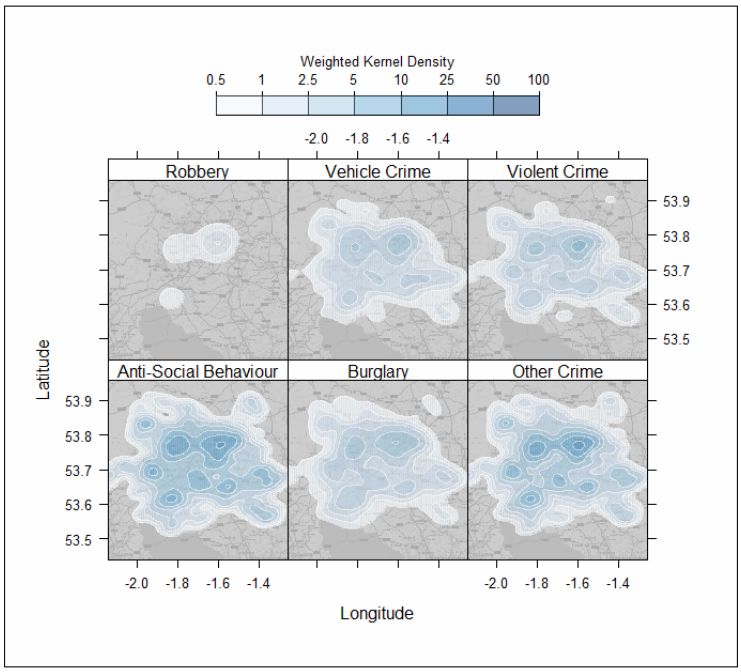

Figure 7: loa GoogleMap kernel density visualization of crime levels in West Yorkshire. The plot is generated in the form ~ latitude * longitude | crime.type and uses dedicated panel function, panel.kernelDensity. The crime data (also shown in Figure 2) is from http://policeapi2.rkh.co.uk. Google Map data subject to copyright © 2012 Google/GeoBais-DE/BKG/Tele Atlas.

Here, the visualization illustrates several important points associated with crime: Most notably, many of the types of crimes that we fear the most, e.g. violent crime and burglary, are relatively uncommon while the vast majority of crimes are linked with anti-social behaviour and other (minor unclassified) criminal activity. But also there are very clearly high and low crime areas and for the most part the distributions of crimes are quite similar regardless of the nature of the crimes. Furthermore, in a statistical environment like R, the crime densities these plots generate can also be easily weighted to study the influence of factors such as local population size and income or geographically modelled to uncover hidden information, that can help us to identify the areas where less typical criminal activity patterns predominate. However, the panel does not have to be a density estimator. It can be any form of data handler, e.g. any data binning, cluster analysis or feature extraction routine, and incorporate *latitude*, *longitude* and any amount of *z* information. The panel function can be supplied by a third-party (or taken from available open source archives), developed around available code or developed entirely by the user, or built up of multiple panels, allowing the user a high level of fine control, e.g.:

```
GoogleMap(…, panel = [my.panel.function])
GoogleMap(…, panel = function(…, arg) [my.panel.function](…, arg))
GoogleMap(…, panel = function(x, …){
                        x2 <- [my.modification.of.x]
                       [my.first.panel.function(x = x2)]
                      [my.second.panel.function(x = x, …)}) #etc.
```

The use of this type of layered panel function is illustrated by Figure 8, which combines hex-binning and arrow layers to visualize UK wind farm data (source: www.bwea.com).

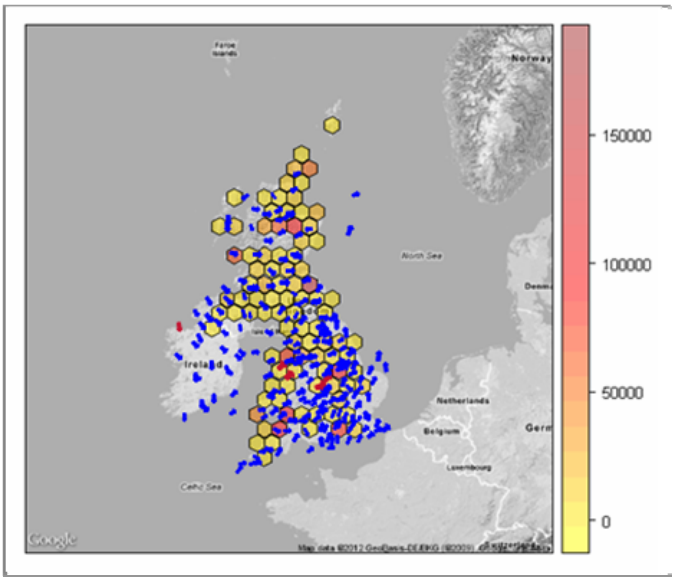

Figure 8: *loa GoogleMap* hex-bin and arrow data layer visualization of UK wind farm performance. The base plot is generated in the form Annual Household Equivalents ~ Latitude * Longitude. Annual Household Equivalents, a wind farm electrical power production metric, is binned and summed using a modified form of *the hexbin* function *panel.hexbinplot*. Additional data (wind direction, average speed and variability) is supplied separately and handled by an in-development arrows panel. Wind farm data is from www.bwea.com. Google Map data subject to copyright © 2012 Google/GeoBais-DE/BKG/Tele Atlas.

The hex-bin layer is generated with a modified version of the *lattice hexbin panel*, *panel.hexbinplot* (from the R package *hexbin*; Carr et al, 2011), and shows the sum of Annual Household Equivalents, an electrical power production potential metric, for all wind turbines in each hex-binned region. The arrows layer is an in-development variation of *panel.arrows* intended for the visualization of data fields like air and water flow that can have multiple forms of associated information (direction, speed, speed variability, species loading, temperature, etc.). Here, the hexbins show the relative electricity production potential of a given area, and the arrow directions and sizes indicate the local wind direction and average speed. Obviously, areas with higher wind speeds and larger electricity production potentials, indicated by a big arrow in/near a dark hexbin, would be expected to produce more electricity. So, this type of approach can be used as benchmark or normalization step when comparing the performance of different wind farms. However, it can also be used to highlight issues. For example, here relatively stable and more variable wind speeds are indicated by blue and red arrows, respectively. Wind farms cannot be safely operated in very variable wind conditions, so high but variable speed speeds and associated potential wind farm down-time, indicated by red arrows in/near dark hexbins, need to be factored into such analyses.

Figure 9 uses Polycyclic Aromatic Hydrocarbon (PAH) data from the UK Department of the Environment Food and Rural Affairs (DEFRA) Air Quality Archive (source: http://uk-air.defra.gov.uk/data/pah-data) to demonstrate glyph-style geovisualisation.

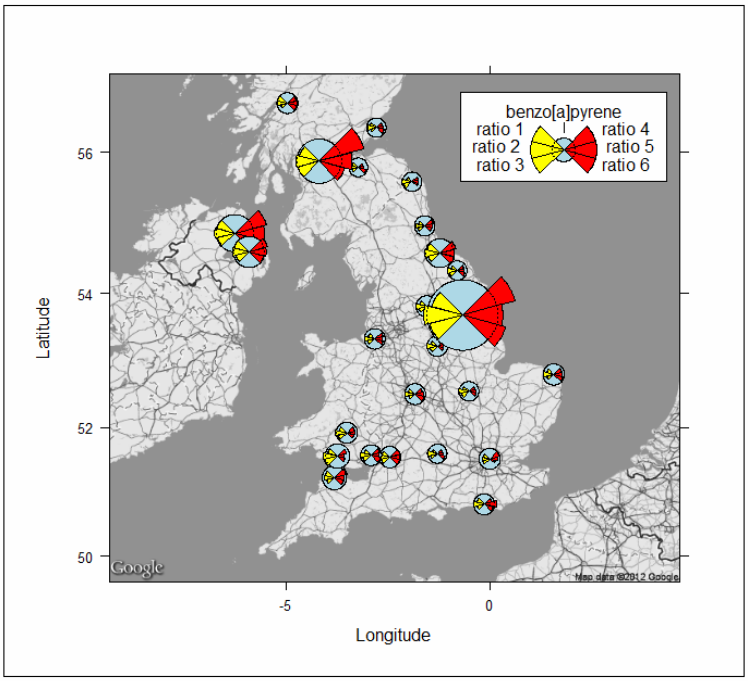

Figure 9: loa GoogleMap glyph-based visualization of 2008 UK PAH data. The plot is generated in the form z0 + z1 + z2 + z3 + z4 + z5 + z6 ~ latitude * longitude. The glyph itself depicts the concentration of one PAH, benzo[a]pyrene, (supplied as z0) using the pale blue circle scale, and six source indicator PAH ratios (supplied as z1 to z6) as superimposed segments. PAH data is from http://uk-air.defra.gov.uk/data/pah-data. The data layer is generated with a dedicated glyph panel function. The Google Map data is subject to copyright © 2012 Google.

The glyph structure is built up using a similar approach to the previous plot, except here all glyph inputs are supplied as separate *z* terms (in this case *z0* to *z6*) and the glyph is produced using a dedicated panel function that is in turn comprised of a series of subpanels, each one generating a separate element of the glyph. In this case, the first (back most) element of each glyph is a pale blue circle, which is scaled according to the concentration of benzo[a]pyrene (BaP), a PAH of particular concern. This input is supplied as *z0* and provides a measure of the environmental loading of BaP. Elsewhere (e.g. del Rosario Sienra et al, 2005; Sofowote et al 2010; Dvorská et al, 2011; Tobiszewski & Namiesnik, 2012), the ratios of certain PAHs have been identified as potential indicators for different source types[5]. So, Figure 9 uses superimposed circle segments, supplied as *z1* to *z6* and scaled relative to *z0*, to characterize possible source contributions for BaP. Ratios 1 to 3, (*z1* to *z3* shown in yellow to the left of the glyph centre) are associated with urban sources including road traffic (gasoline/diesel mixes) and domestic activity, while ratios 4 to 6 (*z4* to *z6* shown in red to the right of the glyph centre) are associated with coal, coke and diesel use. Here, none of glyph structures are completely dominated by a single source indicator ratio, suggesting that BaP is unlikely to be derived from a discrete source at any of the monitoring locations. This is consistent with current source inventory estimates and illustrates the complex nature of environmental PAH loadings. However, what is also apparent from the visualization is that the locations where BaP concentrations are highest are also the locations with the highest ratio 4, 5 and 6 contributions, suggesting that coal and coke burning and diesel-rich emissions may be important contributors to elevated levels of BaP.

A similar approach can also be used to add in static elements to aid visualization or simplify plot interpretation. *lattice* includes a number of 'building block' panels and elements (e.g. *lpolygon, llines* and *ltext*) that can be easily combined to generate novel plot components. For example, the glyph key in Figure 9 is generated using the glyph panel function, a 'dummy' set of plot inputs and additional box, line and text elements.

8. Plot Interaction

There are already a number of R packages that have been written specifically to provide very highly refined levels of plot interaction. These include *iplots* (Urbanek & Wichtrey, 2011), the ggobi interface *rggobi* (Cook & Swayne, 2007; Temple Lang et al 2012) and the Google API interface *googleVis* (Gesmann & de Castillo, 2011, 2012), amongst others. *loa* is not one of these. Rather, it is a plot development package that includes some limited plot interaction functions intended to simplify plot development and routine work with the *loa* and *lattice* outputs. The main plot-interactive function in *loa* is *getXY*. This is wrapper for the existing *lattice*/*grid* function *grid.locator* that behaves more like *locator*, the equivalent function used for recovering coordinate locations from previously generated *plot* outputs. So, although by default *getXY* selections are not automatically marked, adding common plot parameters to the function call overrides this, e.g. to add red symbols and lines, and the limit the number of points collected from a plot:

```
#after making a lattice or loa plot
ans <- getXY(col = "red", pch = 4, type = "b", n = 4)
```

[5] Here, it should perhaps be emphasized that while these ratios are commonly described as diagnostics in the literature, they are more sensibly regarded as indicators. The ratios have a degree of spatial and temporal variability that hinders more rigorous interpretation/source apportionment; see e.g. comments of Dvorská et al (2011) and Katsoyiannis et al (2012).

The function also provides a mechanism for handling locally scaled axes like the latitude/longitude axes of *GoogleMap*, which is implemented by the *getXY* wrapper *getLatLon*. These functions are intended to be used in three ways: (1) To recover coordinate information from existing plots; (2) In combination with *trellis.focus* and *panel…* style functions, to manually add items to plots; and, (3) As the 'front end' of functions used to analysis elements of *loa* outputs like *GoogleMap* plots. Figure 10 presents a relative simple example of this latter application, the measurement of the turning angle between arms on a roundabout, coded as part of on-going work for a study of the relationship between the geometric properties of road features and accident rates. Associated functions combine *getLatLon* and existing methods/code (e.g. bearings and distances using modifications of methods and JavaScript codes posted by Chris Veness on http://www.movable-type.co.uk/) to allow users to quickly and manually recover such metrics from GoogleMap outputs that they can also quickly generate using location coordinates.

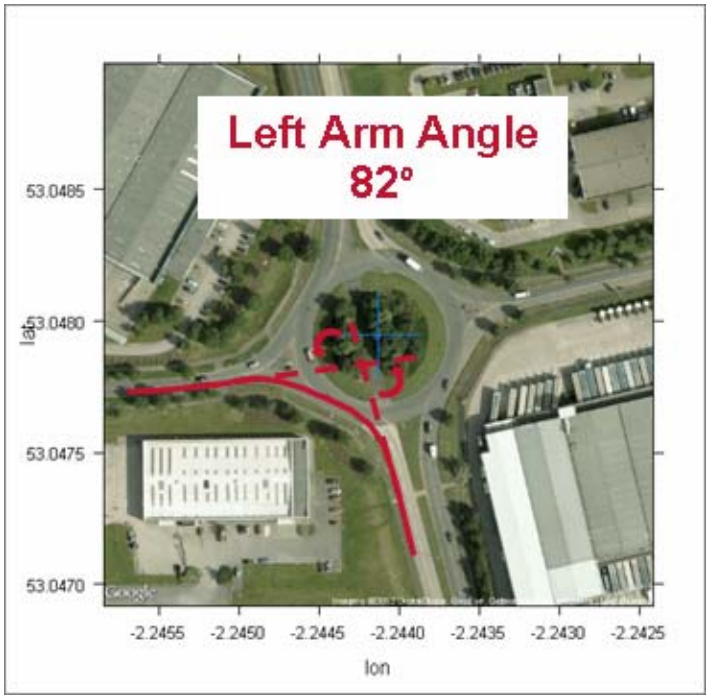

Figure 10: Roundabout turning angle measurement using loa GoogleMap, getLatLon and code based on methods published by Chris Veness at http://www.movable-type.co.uk/). The Google Map data subject to copyright © 2012 Google/Terrametrics/GeoBais-DE/BKG/Tele Atlas.

9. Example Third Party Use: *openair GoogleMapsPlot*

openair (Carslaw & Ropkins, 2012a) is an R package developed for the air quality community as part of the UK Natural Environment Research Council (NERC) Knowledge Exchange Programme (http://www.nerc.ac.uk/using/introduction/; NERC award NE/G001081/1). One of the key issues that had to be addressed within the associated project was that many *openair* users would be new to both R and *openair*, so potentially subject to a relatively steep learning curve. This is a particular concern for many potential users in Local Authority Air Quality Departments (one of the main target

audiences for *openair*) because many have high workloads and limited time available to learn to use new tools. With this in mind, the argument structure of *openair* functions is highly standardized to make function use more intuitive and all data is supplied and used in R data frames to simplify data handling, see e.g. related discussion in Carslaw & Ropkins (2012b) or Ropkins & Carslaw (2012). So, *openair* functions typically take the form *openair.function(data.frame, [openair arguments])*. In addition many of the common but sometimes frustrating activities associated with plot generation, e.g. figure labelling, label formatting and information key handling, are routinely handled by *openair*. The *openair* function *GoogleMapsPlot* was developed using modified forms of the same workhorse functions in the *loa* function *GoogleMap*. The basic structure of *GoogleMapsPlot* combines different components of the *loa quickMap* and *GoogleMap* structures and *openair* functions like *quickText*, *openColours* and *drawOpenKey* that handle text formatting, plot point colors and color keys, respectively, in *openair*. The function therefore provides a convenient example of how code from *loa* can be used as a 'building block' for the development of third-party functions. A typical *openair GoogleMapsPlot* output is shown in Figure 10.

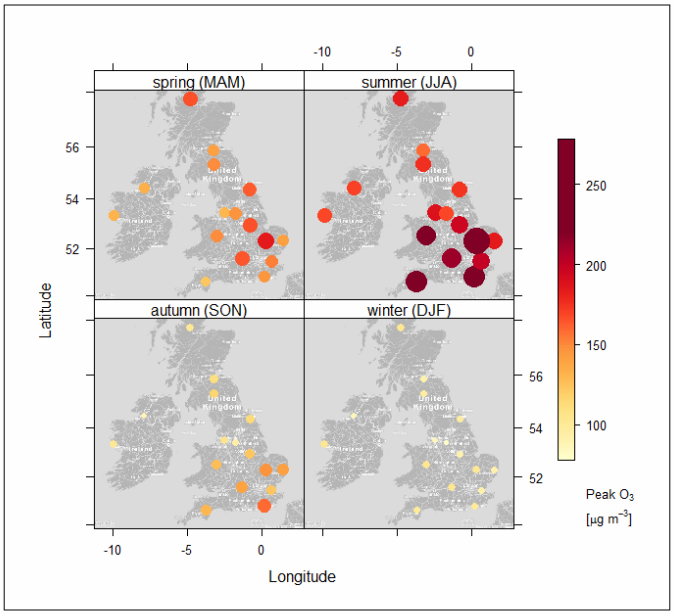

Figure 10: openair GoogleMapsPlot presented as an example of the integration of loa geovisualization functions and openair label, color and key handlers. Here, the data is supplied in as an R *data.frame*, 'by-season' conditioning is automatically handled by *openair cutData* function, accessed via the common *openair* plot argument *type*. Ozone (O_3) data is from http://www.erg.kcl.ac.uk/downloads/Policy_Reports/AQdata/. Google Map data subject to copyright © 2012 Google/GeoBais-DE/BKG/Tele Atlas.

10. Future Work

Like most R packages, *loa* is a 'work-in-progress'. Functions will continue to be developed and refined based on feedback from users. However, *loa* was envisioned as both a source of 'stand alone' plotting functions employing the *lattice* Trellis data handling framework and a tool for third-parties wishing to develop novel plots of their

own. In terms of geovisualization, the current 'next objectives' are to further develop the *map.recolor* argument handlers to provide more convenient control of the map layer appearance, add additional *makeMapArg* type functions to provide access to alternative map layer data sources such as the Natural Earth spatial polygons (http://www.naturalearthdata.com/downloads), and to continue to develop data panel handlers for spatial data in the form ~ *latitude* * *longitude*. In terms of general plot handling, the current 'next objectives' are to refine the *panelPal* function to further simplify group handling and develop *keyHandler* functions. These will most likely be introduced as minor updates. In terms of major developments (and major updates), the current 'next objectives' are to introduce *trianglePlot* and *binPlot* functions.

Acknowledgements

The first two authors (KR and DCC) gratefully acknowledge the funding and support of the UK Natural Environment Research Council (NERC) in association with the development of *openair*. The first author (KR) also gratefully acknowledges partial funding and support from the UK Engineering and Physical Sciences Research Council (ESPRC) towards the development of interactive functions for the measurement of geometric roundabout parameters. The authors also gratefully acknowledge the on-going work of the R Core Team and R community in the development and upkeep of R, and the always welcome feedback of users.

References

Batty, M., Hudson-Smith, A., Milton, R., Crooks, A. (2010) Map mashups, Web 2.0 and the GIS revolution. Annals of GIS, 16(1), p.1-13

Becker, R. A., Cleveland, W. S., Shyu, M. J. (1996) The Visual Design and Control of Trellis Display, Journal of Computational and Graphical Statistics, 5(2), 123–155.

Bivand, R.S., Pebesma, E.J., Gomez-Rubio, J. (2008) Applied spatial data analysis with R. The Use-R Series. Springer, NY. ISBN: 978-0-387-78170-9.

Butler, D. (2006) Mashups mix data into global service. Nature, 439, p.6–7.

Carr, D. (ported by Lewin-Koh, N., Maechler, M.) (2011) hexbin: Hexagonal Binning Routines. R package version 1.26.0. (http://CRAN.R-project.org/package=hexbin).

Carslaw, D.C., Ropkins, K (2012a) openair: Open-source tools for the analysis of air pollution data. R package version 0.5-21 (http://www.openair-project.org/).

Carslaw, D.C., Ropkins, K (2012b) openair - an R package for air quality data analysis. Environmental Modelling & Software, 27-28, p.52-61.

Cleveland, W.S. (1993) Visualizing Data, Hobart Press, Summit, New Jersey.

Cook, D., Swayne, D.F. (2007) Interactive and Dynamic Graphics for Data Analysis. Springer.

de Adana, F.S., Fernandez, F.J., Loranca, J.L., Kronberger, R. (2009) Covermap: Computer tool to calculate the propagation in open areas importing data from GoogleMaps. Antennas & Propagation Conference. Conference proceedings, p.229-232.

del Rosario Sienra, M., Rosazza, N. G. Préndez, M. (2005) Polycyclic aromatic hydrocarbons and their molecular diagnosticratios in urban atmospheric respirable particulate matter. Atmospheric Research, 75 (4), 267-281.

Dvorská, A., Lammela, G., Klánová, J. (2011) Use of diagnostic ratios for studying source apportionment and reactivity of ambient polycyclic aromatic hydrocarbons over Central Europe. Atmospheric Environment, 45(2), 420-427.

Dykes, J. MacEachren, A., Kraaf, M.-J. (2006) Exploring Geovisualization. Elsevier, London. ISBN 10: 0-08-044531-4.

Friendly, M. (2000) Visualizing Categorical Data. SAS Institute, Carey, NC. ISBN: 1-580025-660-0.

Gesmann, M., de Castillo, D. (2011) Using the Google Visualisation API with R. The R Journal, 3(2):40-44.

Gesmann, M., de Castillo, D. (2012) googleVis: Interface between R and the Google Visualisation API. (http://code.google.com/p/google-motion-charts-with-r/).

Katsoyiannis, A., Sweetman, A.J., Jones, K.C. (2012) PAH Molecular Diagnostic Ratios Applied to Atmospheric Sources: A Critical Evaluation Using Two Decades of Source Inventory and Air Concentration Data from the UK. Environmental Science and Technology, 45(20), 8897-8906.

Kilibarda, M., Bajat, B. (2012) plotGoogleMaps: The R-based web-mapping tool for thematic spatial data. Geomatica 66(1), p.37-49.

Kraaf, M.-J., Ormeling, F.L. (1996) Cartography – Visualization of Spatial Data. Longman, London.

Loecher, M., Sense Networks (2011) RgoogleMaps: Overlays on Google map tiles in R (http://cran.r-project.org/web/packages/RgoogleMaps/).

Meyer, D., Zeileis, A., Hornik, K. (2006) The Strucplot Framework: Visualizing multi-way contingency table with vcd. Journal of Statistical Software. 17(3), 1-48. (http://www.jstatsoft.org/v17/i03).

R Development Core Team (2012) R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, (http://www.R-project.org/).

Ropkins, K. (2012) loa: Various plots, options and add-ins for use with the lattice package. (https://sites.google.com/site/karlropkins/rpackages/loa/).

Ropkins, K., Carslaw, D.C. (2012). openair - Data Analysis Tools for the Air Quality Community. The R Journal, 4(1), p.20-29. (http://journal.r-project.org/archive/2012-1/RJournal_2012-1_Ropkins+Carslaw.pdf)

Sarkar, D. (2008) Lattice: Multivariate Data Visualization with R, Springer. ISBN: 978-0-387-75968-5. (http://lmdvr.r-forge.r-project.org/).

SAS (2012) SAS® Visual Analytics 5.1 User's Guide. SAS.

Sofowote, U.M., Allan, L.M., McCarry, B.E. (2010) Evaluation of PAH diagnostic ratios as source apportionment tools for air particulates collected in an urban-industrial environment. Journal of Environmental Monitoring, 12, 417-424.

South, A. (2011) rworldmap: A New R package for Mapping Global Data. The R Journal 3(1), p.35-43.

Temple Lang, D. Swayne, D.F., Wickham, H., Lawrence, M. (2012) rggobi: Interface between R and GGobi. (http://www.ggobi.org/rggobi).

Tobiszewski, M., Namiesnik, J. (2012) PAH diagnostic ratios for the identification of pollution emission sources. Environmental Pollution, 162, 110-119.

Urbanek, S., Wichtrey, T. (2011) iplots: iPlots - interactive graphics for R. (http://www.iPlots.org/).

Wickham, H. (2009) ggplot2: Elegant graphics for data analysis. Springer, New York. ISBN: 978-0-387-98140-6. (http://had.co.nz/ggplot2/book).

Wilkinson, L. (2005) The Grammar of Graphics (Second Edition). Springer, New York. ISBN 978-0-387-24544-7.