DSC 2001 Proceedings of the 2nd International Workshop on Distributed Statistical Computing
March 15-17, Vienna, Austria
http://www.ci.tuwien.ac.at/Conferences/DSC-2001
K. Hornik & F. Leisch (eds.) *ISSN 1609-395X*

GGobi meets R: an extensible environment for interactive dynamic data visualization

Duncan Temple Lang [*] **Deborah F. Swayne** [†]

Abstract

GGobi is a direct descendant of XGobi, designed so that it can be embedded in other software and controlled using an API (application programming interface). This design has been developed and tested in partnership with R. When GGobi is used with R, the result is a full marriage between GGobi's direct manipulation graphical environment and R's familiar extensible environment for statistical data analysis.

1 Introduction

This paper describes a partnership between two very different kinds of tools. Each tool can be described as "interactive statistical software," but "interactive" turns out to have a different meaning in each case.

Software with a direct manipulation graphical user interface allows its users to interact with displays of the data with a powerful sense of immediacy, querying and manipulating views using the mouse. Tools in this category, though, are usually not easy to alter or extend, even when the source code is made available. The intrepid user who wants to make changes has to be a competent programmer in the appropriate low-level language, and to be willing to explore the inner workings of code that was probably not written with other programmers in mind.

Tools with a command-line interface were dubbed "interactive" many years ago to distinguish them from programs that ran in "batch" mode, requiring users to walk

[*]Lucent Bell Laboratories
[†]AT&T Labs – Research

down the hall to some computer center or mail drop to retrieve their output. While aficionados of the direct manipulation interface sometimes think that the command-line interface has outlived its usefulness, the presence of a scripting language offers an open and flexible environment for modifying and extending existing functions, and for creating new ones. Direct manipulation tools may never be able to offer similar capabilities. On the other hand, the command-line tools have been slow to add interactive graphics (in the "direct manipulation" sense).

It does not seem reasonable given today's tools to expect the designers of graphics software to add scripting languages or full analytical functionality. It is more probable that designers of statistical languages and computing environments will enrich their direct manipulation graphics, but that's still a tall order. The approach described here is an alternative: it's a way to let two very different tools act as one.

Depending on your point of view, you might say that a statistical environment with a scripting language, R, has acquired a new interactive graphics device, or you might say that a direct manipulation program for exploratory data visualization, ggobi, has acquired a command-line interface, an extension language, and a statistical environment.

2 GGobi

The ggobi software is a statistical data visualization system with interactive and dynamic methods for viewing data and manipulating plots. It's a direct descendant of xgobi ([2, 3]), and they share many features, such as 1-D and 2-D views, 2-D grand tours, view scaling, brushing, and identification.

GGobi represents a big step in the evolution of XGobi, with multiple plotting windows, more flexible color management, XML file handling, and better portability to Windows. Its graphical user interface is more powerful and has a cleaner look and feel, because it is written in gtk+ (www.gtk.org), an evolving toolkit that's widely used by Linux programmers. XGobi was written using a toolkit that stopped changing in the early 1990's.

The new feature of ggobi that is likely to have the greatest impact, though, is the one described in this paper, and it is not visible in the graphical user interface: ggobi can be compiled into a library, embedded in other software and controlled using an API (application programming interface). This design has been developed and tested in partnership with R, but it also works with other scripting languages such as Perl and Python.

The designers of xgobi had always intended that it be used in conjunction with some analytical software. There exists an S (i.e. R or S-Plus) function, distributed with the xgobi software, that allows an S user to launch an xgobi process given S objects as arguments. That function is embarrassingly simple: the S objects are written out as ASCII files, and a system call executes xgobi with those files as arguments. An xgobi process launched in this way has very limited ability to create S objects directly: a vector of point colors, for example, or a rotation matrix. The S process has no ability to communicate further with xgobi.

The xgobi authors occasionally explored other approaches that would extend

this unsatisfactory relationship. As early as 1991, we used interprocess communication to maintain a live connection between xgobi and S. One of the applications would draw a clustering tree in S, allow the user to click on it to cut the tree and immediately set the point colors in xgobi to show the result [1]. It relied on a second program, also written in C, to gather input from the user and to manage the interprocess communication. It was necessary to assemble each command in the S language, ship it to S, read back the result and respond accordingly. To make this foolproof, it would have been necessary for xgobi to be fully able to parse S commands and handle errors. Because this was such a daunting task, work on this model was discontinued after the first prototype.

The relationship described here doesn't need to use interprocess communication, because there's only one process. There's no need to write a GUI to assemble each command in the S language, because the user will enter it in the customary way. Parsing of input and output between the two systems is also unnecessary as values are passed directly between them.

3 The ggobi API

The ggobi API allows the basic facilities of the stand-alone application to be incorporated into other applications, and to be made available in other programming languages such as R, Perl and Python. The routines in the API provide access to the entire ggobi computational model. It allows us to create new ggobi instances and manipulate both the data and the contents of the displays. The routines allow us to both query and set the state of multiple ggobi instances within the same process and also provide custom event handlers for certain high-level interactions.

The routines in the API allow us to create new ggobi instances at any time, specifying data either directly from a data frame or matrix, or from a file or URL. We can access the data from such an instance and replace individual cells. We can also add or remove variables from a dataset, edit the variable names, and so on.

We can retrieve and set the visual characteristics of points in the display – the color, glyph type and size of each point – and choose whether a point is displayed or not. We can query the current status of the active brushing region, including its position, color and even the points it encloses. While brushing and point identification are usually done interactively, ggobi's brushing region can be controlled programmatically. This can be appealing when the number of points becomes large, because the brush response begins to lag behind the user's hand. It's also convenient for providing directed animations or shows and also allows one to provide scripts which create a particular view that one wants a user to see.

As mentioned earlier, a ggobi process can have several plot displays open at the same time, and there are several types of displays, such as scatterplots, scatterplot matrices, and parallel coordinates plots. The API provides ways to manipulate the set of displays. For example, we can determine what displays are open, delete a particular display, opening a new one, and control which one is active.

One feature that is available through the API but not in the GUI is the ability to create mixed or composite displays. In other words, we can create a new display

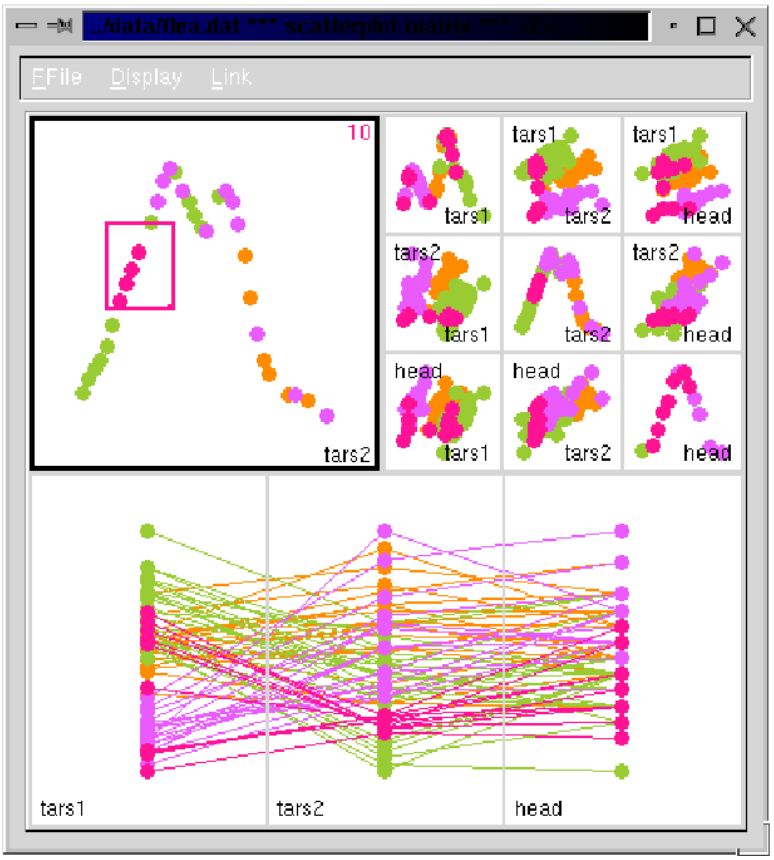

Figure 1: This figure shows a custom layout of ggobi plots created using **plotLayout()** in R.

that may contain any number of any kind of plots, and control how those plots are layed out within the window. In R, the function **plotLayout()** allows one to specify a collection of plot descriptions along with the cells in a grid that each should occupy to create a new display. This provides a way to arrange the plots in different configurations that best display the features of interest. Figure 1 shows an example.

Using the API, it's possible to launch and manage ggobi multiple instances within the same process, allowing custom linking algorithms to be implemented.

3.1 Embedding for tighter coupling

Most of the facilities in the ggobi API allow the user to both set and retrieve values. This style of embedding allows the two systems to directly share data, but essentially maintains a separation between them. A richer form of embedding

allows the two systems to interact in a more dynamic and symbiotic manner. Two examples illustrate this. First, ggobi can use functionality in the internal R API. For example, the random sampling of records provided by ggobi can make use of the random number generation facilities in the R math library. More interestingly, we can allow user-level S functions to be specified at any time within the lifetime of a ggobi session to implement particular tasks. For example, the user can specify a smoothing function that takes two variables (x and y) and a bandwidth and returns the smoothed predicted values of y for those x values. This allows a user to specify a smoother as an S function and integrate it into ggobi as a first class facility that is called each time the user changes the value on a graphical slider.

A second instance of the dynamic aspect of the embedding is that users can specify functions for handling a growing number of ggobi events. Let's consider ggobi's *identify* mode for interactively labelling points. As the user moves the pointer close to a point on the plot, the label for that case is displayed. The identification of a point corresponds to a high-level ggobi event and users can specify an S function which is to be called when such an event occurs. In this case, the function is given the index of the observation and can use this, for example, to highlight the corresponding row in a data editor or to display some characteristic of the observation in an R plot.

Another form of event handling allows one to associate an R function with one of the the number keys (i.e. 0, 1, ..., 9). When that key is pressed while viewing a ggobi plot, the function is called. This can be used for tasks like computing summary statistics or updating a plot based on the current state of the ggobi display.

This type of event-driven, dynamic feedback to the high-level programming language (e.g. R) makes it significantly easier to develop interactive graphics.

As a final note on embedding ggobi in other applications, we mention that one can use ggobi's own event loop or integrate the GUI events into the host application's own event loop.

As should be clear by now, integrating the ggobi API with a programming language such as R allows ggobi to become increasingly complex while its graphical interface remains simple. Some extensions do not lend themselves to GUI implementation: they may involve many steps, or they may only be interesting to a small number of users. This more complicated or rarely-used functionality can be added to ggobi's API and made available through functions in the different scripting languages, and each interface can continue be used for what it is best suited.

4 Example

The data used in the example is from an old AT&T marketing study in which nearly 2000 households responded to questions about their telecommunications needs, such as whether they needed certain classes of business tools. The data also includes a few demographic variables and the duration of the household's total incoming and outgoing calls. The goal was to use these variables for segmenting households into homogeneous groups.

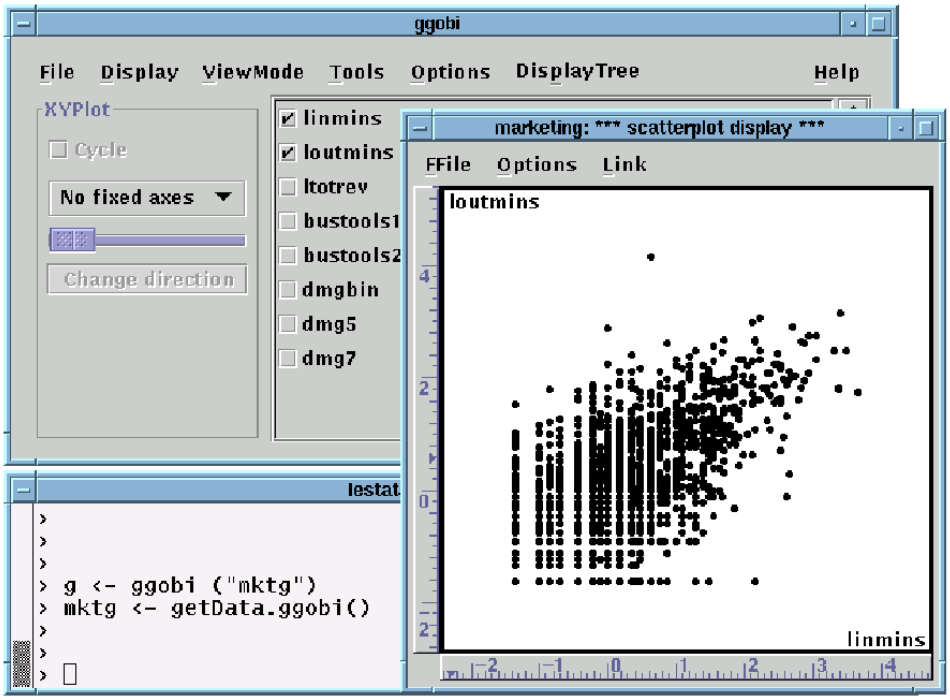

Figure 2: This figure shows two ggobi windows (the control panel at the top and a scatterplot at the right) and a window in which R is being run. R was used to launch ggobi, and then to grab the data from ggobi into R.

We'll analyze the data using a combination of ggobi and R. We begin by starting R, loading the ggobi library, and starting ggobi using the data from a file, *mktg*:

```
> library(Rggobi)
> ggobi ("mktg")
```

Figure 2 shows the result.

We can use ggobi to explore the data, to verify that the data looks reasonable, and to see that most of the variables are discrete. Some exploratory use of brushing shows us quickly that no natural clustering of the data will emerge this way, so we determine to use R to partition the data. We do this by first getting the data from ggobi into R and then standardizing the data, applying k-means clustering for four groups, and deriving a vector of cluster identifiers.

```
> mktg <- getData.ggobi()
> p <- dim(mktg)[2]
> mktg.scale <- apply(mktg, 2, scale)
> rn <- matrix(rnorm(p*4), ncol=p)
> mktg.kmeans <- kmeans(x = mktg.scale, centers = rn)
```

```
> mktg.clusters <- mktg.kmeans$cluster
```

Before we look at the results in ggobi, we derive the principal components of the standardized data, and add the first two principal components to the variables currently in use. We use the ggobi variable selection interface to select the plot of PC2 versus PC1.

```
> mktg.prcomp <- prcomp(mktg.scale)$x
> addVariable.ggobi(mktg.prcomp[,1], "PC1")
> addVariable.ggobi(mktg.prcomp[,2], "PC2")
```

Now we're ready to check the results of the partitioning. We set the colors in ggobi using the clustering vector. Since those aren't easy to distinguish in gray-scale, we also change the glyph type of two of the groups. Figure 3 shows the result.

```
> setColors.ggobi (mktg.kmeans$cluster)
> setGlyphs.ggobi ((1:1926)[mktg.kmeans$cluster==1], type="oc")
> setGlyphs.ggobi ((1:1926)[mktg.kmeans$cluster==4], type="plus")
```

4.1 Responding to ggobi key events

This partitioning obviously doesn't result in groups that are well separated in the data, so we might like to investigate the characteristics of each group. For the purpose of illustrating how this might be done, we write an R function called *clusterstats* that will display the mean and standard deviation of the first two variables in a table.

```
clusterstats <- function (key, plot=NULL, ggobi=NULL, ev=NULL) {
  mktg <- getData.ggobi()
  colorids <- getColors.ggobi()
  groups <- sort(unique(colorids))
  ngroups <- length(groups)

  mout <- matrix (0, ngroups, 4,
    dimnames=list(paste("Color", 1:ngroups),
      c("mean(incoming)", "sdev(incoming)",
        "mean(outgoing)", "sdev(outgoing)")))

  for (i in 1:length(groups)) {
    for (k in 1:2) {
      p <- (k-1)*2
      vec <- (mktg[,k])[colorids==groups[i]]
      mout[i,p+1] <- mean(vec)
      mout[i,p+2] <- sqrt(var(vec))
```

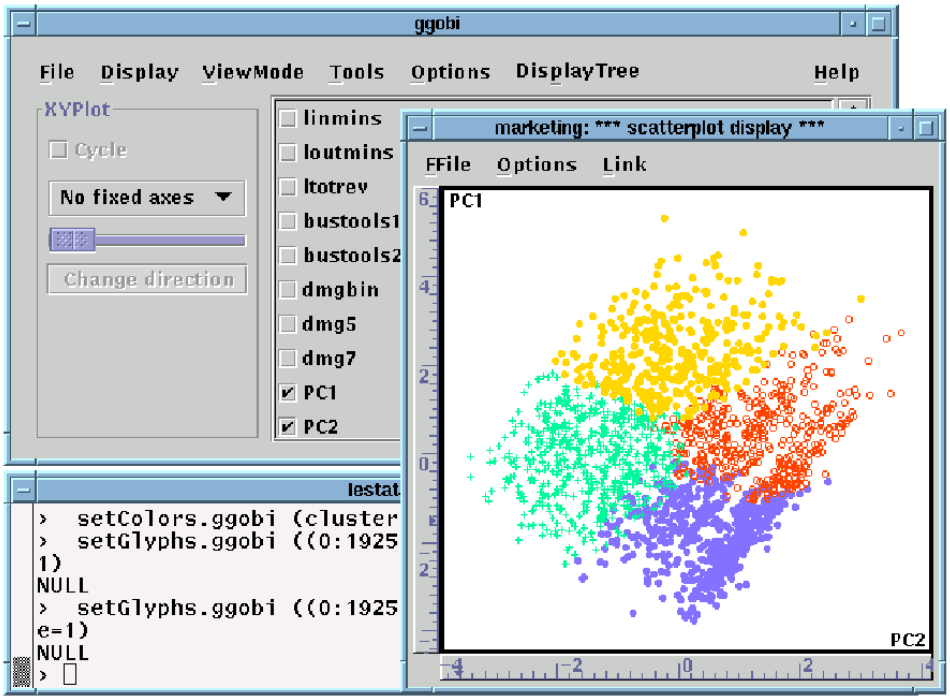

Figure 3: This figure shows the same configuration of windows. The ggobi scatterplot window now shows a plot of the first two principal components, and it has been brushed (using 3 different glyphs) to show the four segments found by the k-means algorithm.

```
    }
  }

  print (round(mout, digits=3))
  return
}
```

Now we attach that routine to the *1* key on the keyboard, so that **clusterstats()** will be executed each time we click on that key, printing out the data for the current clustering of the data.

```
> h <- NumberedKeyHandler.ggobi()
> registerNumberedKeyHandler.ggobi(h)
> h$addHandlers("1"=clusterstats)
```

This gives us an easy way to compare different ways of partitioning the data. We might try clustering the data into 3 or 5 groups instead of 4, or doing some manual brushing to adjust cluster membership, and use the output of **clusterstats()** as a diagnostic tool to help decide which segmentation is the easiest to interpret.

4.2 Animation

In order to seek an interpretation of the clusters in terms of the original variables, we might next want to look at a variety of plots. There is a great deal of overlap between these clusters in the space of the untransformed data, so it will be hard to evaluate them. In order to see the clusters more clearly, we can use R to animate the display, looking at one cluster at a time. We first define a function that will set the "hidden" attribute of each case: true if it's in cluster j, and false otherwise. If its argument is -1, all cases are shown.

```
showCluster <- function (j) {
  colorids <- getColors.ggobi()
  groups <- sort(unique(colorids))
  tmp <- logical( length(colorids) )
  if (j != -1) {
    tmp[colorids != groups[j]] <- T
  }
  setHiddenCases.ggobi(tmp)
}
```

We create a plot of the original variables. This can be a 1D plot, a scatterplot, or even an active tour, and we can create it using R or the graphical interface. Next we can loop (in R) through the clusters, showing them one at a time.

```
> for (i in 1:4) {
>  showCluster(i)
>  system("sleep 1")
> }
> showCluster (-1)
```

5 Conclusions

This smooth integration of R and ggobi is an example of a modern approach to software inter-operability in statistical computing. No single piece of software has to do everything, but any piece of software can be designed to work well with others. GGobi's design was greatly changed so that it could be compiled into a library and controlled via an API; R was changed so that it could accommodate ggobi's event loop along with its own.

This combination may hold special interest for the teachers of statistical computing courses. If R or S is already on the syllabus, this environment should make it easier to introduce interactive graphics, and it should also be a good platform for student projects.

References

[1] Deborah F. Swayne, Andreas Buja, and Nancy Hubbell. XGobi meets S: Integrating software for data analysis. In *Computing Science and Statistics: Pro-*

ceedings of the 23rd Symposium on the Interface, pages 430–434, Fairfax Station, VA, 1991. Interface Foundation of North America, Inc.

[2] Deborah F. Swayne, Dianne Cook, and Andreas Buja. XGobi: Interactive dynamic graphics in the X Window System with a link to S. In *American Statistical Association 1991 Proceedings of the Section on Statistical Graphics*, pages 1–8. American Statistical Association, 1992.

[3] Deborah F. Swayne, Dianne Cook, and Andreas Buja. XGobi: Interactive dynamic data visualization in the X Window System. *Journal of Computational and Graphical Statistics*, 7(1):113–130, 1998.