An Introduction to rggobi

Hadley Wickham, Michael Lawrence,
Duncan Temple Lang, Deborah F Swayne

Introduction

The rggobi (Temple Lang and Swayne, 2001) package provides a command-line interface to GGobi, an interactive and dynamic graphics package. Rggobi complements GGobi's graphical user interface by enabling fluid transitions between analysis and exploration and by automating common tasks. It builds on the first version of rggobi to provide a more robust and user-friendly interface. In this article, we show how to use ggobi and offer some examples of the insights that can be gained by using a combination of analysis and visualisation.

This article assumes some familiarity with GGobi. A great deal of documentation, both introductory and advanced, is available on the GGobi web site, `http://www.ggobi.org`; newcomers to GGobi might find the demos at `ggobi.org/docs` especially helpful. The software is there as well, both source and executables for several platforms. Once you have installed GGobi, you can install rggobi and its dependencies using `install.packages("rggobi", dep=T)`.

This article introduces the three main components of rggobi, with examples of their use in common tasks:

- getting data into and out of GGobi;
- modifying observation-level attributes ("automatic brushing");
- basic plot control.

We will also discuss some advanced techniques such as creating animations with GGobi, the use of edges, and analysing longitudinal data. Finally, a case study shows how to use rggobi to create a visualisation for a statistical algorithm: manova.

Data

Getting data from R into GGobi is easy: `g <- ggobi(mtcars)`. This creates a `GGobi` object called `g`. Getting data out isn't much harder: Just index that `GGobi` object by position (`g[[1]]`) or by name (`g[["mtcars"]]` or `g$mtcars`). These return `GGobiData` objects which are linked to the data in GGobi. They act just like regular data frames, except that changes are synchronised with the data in the corresponding GGobi. You can get a static copy of the data using `as.data.frame`.

Once you have your data in GGobi, it's easy to do something that was hard before: find multivariate outliers. It is customary to look at uni- or bivariate plots to look for uni- or bivariate outliers, but higher-dimensional outliers may go unnoticed. Looking for these outliers is easy to do with the tour. Open your data with GGobi, change to the tour view, and select all the variables. Watch the tour and look for points that are far away or move differently from the others—these are outliers.

Adding more data sets to an open GGobi is also easy: `g$mtcars2 <- mtcars` will add another data set named "mtcars2". You can load any file type that GGobi recognises by passing the path to that file. In conjunction with `ggobi_find_file`, which locates files in the GGobi installation directory, this makes it easy to load GGobi sample data. This example loads the olive oils data set included with GGobi:

```
library(rggobi)
ggobi(ggobi_find_file("data", "olive.csv"))
```

Modifying observation-level attributes, or "automatic brushing"

Brushing is typically thought of as an operation performed in a graphical user interface. In GGobi, it refers to changing the colour and symbol (glyph type and size) of points. It is typically performed in a linked environment, in which changes propagate to every plot in which the brushed observations are displayed. in GGobi, brushing includes shadowing, where points sit in the background and have less visual impact, and exclusion, where points are completed excluded from the plot. Using rggobi, brushing can be performed from the command line; we think of this as "automatic brushing." The following functions are available:

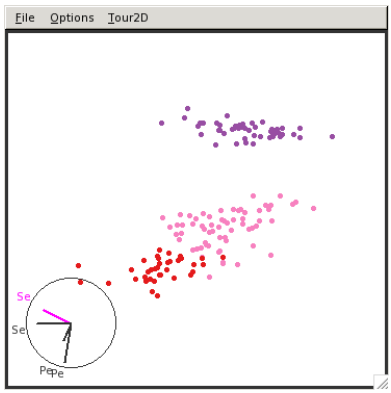
- change glyph colour with `glyph_colour` (or `glyph_color`)
- change glyph size with `glyph_size`
- change glyph type with `glyph_type`
- shadow and unshadow points with `shadowed`
- exclude and include points with `excluded`

Each of these functions can be used to get or set the current values for the specified `GGobiData`. The "getters" are useful for retrieving information that you have created while brushing in GGobi, and the "setters" can be used to change the appearance of points based on model information, or to create animations. They can also be used to store, and then later recreate, the results of a complicated sequence of brushing steps.

This example demonstrates the use of `glyph_colour` to show the results of clustering the

infamous Iris data using hierachical clustering. Using GGobi allows us to investigate the clustering in the original dimensions of the data. The graphic shows a single projection from the grand tour.

```
g <- ggobi(iris)
clustering <- hclust(dist(iris[,1:4]),
  method="average")
glyph_colour(g[1]) <- cuttree(clustering, 3)
```



Another function, `selected`, returns a logical vector indicating whether each point is currently enclosed by the brush. This could be used to further explore interesting or unusual points.

Displays

A `GGobiDisplay` represents a window containing one or more related plots. With rggobi you can create new displays, change the projection of an existing plot, set the mode which determines the interactions available in a display, or select a different set of variables to plot.

To retrieve a list of displays, use the `displays` function. To create a new display, use the `display` method of a `GGobiData` object. You can specify the plot type (the default is a bivariate scatterplot, called "XY Plot") and variables to include. For example:

```
g <- ggobi(mtcars)
display(g[1], vars=list(X=4, Y=5))
display(g[1], vars=list(X="drat", Y="hp"))
display(g[1], "Parallel Coordinates Display")
display(g[1], "2D Tour")
```

The following display types are available in GGobi (all are described in the manual, available from `ggobi.org/docs`):

Name	Variables
1D Plot	1 X
XY Plot	1 X, 1 Y
1D Tour	*n* X
Rotation	1 X, 1 Y, 1 Z
2D Tour	*n* X
2x1D Tour	*n* X, *n* Y
Scatterplot Matrix	*n* X
Parallel Coordinates Display	*n* X
Time Series	1 X, *n* Y
Barchart	1 X

After creating a plot you can get and set the displayed variables using the `variable` and `variable<-` methods. Because of the range of plot types in GGobi, variables should be specified as a list of one or more named vectors. All displays require an `X` vector, and some require `Y` and even `Z` vectors, as specified in the above table.

```
g <- ggobi(mtcars)
d <- display(g[1],
  "Parallel Coordinates Display")
variables(d)
variables(d) <- list(X=8:6)
variables(d) <- list(X=8:1)
variables(d)
```

A function which saves the contents of a GGobi display to a file on disk, is called `ggobi_display_save_picture`. This is what we used to create the images in this document. This creates an exact (raster) copy of the GGobi display. If you want to create publication quality graphics from GGobi, have a look at the `DescribeDisplay` plugin and package at `http://www.ggobi.org/describe-display`. These create R versions of GGobi plots.

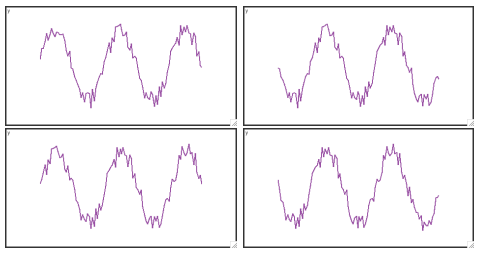
To support the construction of custom interactive graphics applications, rggobi enables the embedding of GGobi displays in graphical user interfaces (GUIs) based on the RGtk2 package. If `embed = TRUE` is passed to the `display` method, the display is not immediately shown on the screen but is returned to R as a `GtkWidget` object suitable for use with RGtk2. Multiple displays of different types may be combined with other widgets to form a cohesive GUI designed for a particular data analysis task.

Animation

Any changes that you make to the `GGobiData` objects are updated in GGobi immediately, so you can easily create animations. This example scrolls through a long time series:

```
df <- data.frame(
  x=1:2000,
  y=sin(1:2000 * pi/20) + runif(2000, max=0.5)
)
```

```
g <- ggobi_longitudinal(df[1:100, ])

df_g <- g[1]
for(i in 1:1901) {
  df_g[, 2] <- df[i:(i + 99), 2]
}
```



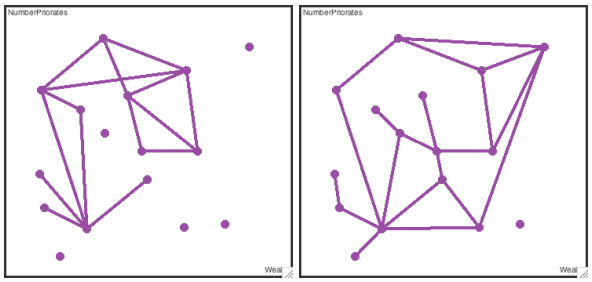
Edge data

In GGobi, an edge data set is treated as a special type of dataset in which a record describes an edge – which may still be associated with an n-tuple of data. They can be used to represent many differentent types of data, such as distances between observations, social relationships, or biological pathways.

In this example we explore marital and business relationships between Florentine families in the 15th century. The data comes from the `ergm` (social networking analysis) package (Handcock et al., 2003), in the format provided by the `network` package (Butts et al., July 26, 2008).

```
install.packages(c("network", "ergm"))
library(ergm)
data(florentine)

flo <- data.frame(
  priorates = get.vertex.attribute(
    flobusiness, "priorates"
  ),
  wealth = get.vertex.attribute(
    flobusiness, "wealth"
  )
)

families <- network.vertex.names(flobusiness)
rownames(flo) <- families

edge_data <- function(net) {
  edges <- as.matrix.network(
    net,
    matrix.type="edgelist"
  )
  matrix(families[edges], ncol=2)
}

g <- ggobi(flo)
edges(g) <- edge_data(flomarriage)
edges(g) <- edge_data(flobusiness)
```

This example has two sets of edges because some pairs of families have marital relationships but not business relationships, and vice versa. We can use the edges menu in GGobi to change between the two edge sets and compare the relationship patterns they reveal.



How is this stored in GGobi? An edge dataset records the names of the source and destination observations for each edge. You can convert a regular dataset into an edge dataset with the `edges` function. This takes a matrix with two columns, source and destination names, with a row for each edge observation. Typically, you will need to add a new data frame with number of rows equal to the number of edges you want to add.

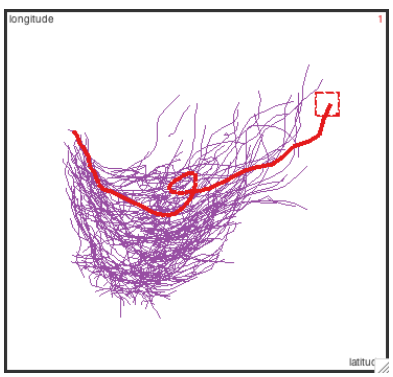
Longitudinal data

A special case of data with edges is time series or longitudinal data, in which observations adjacent in time are connected with a line. Rggobi provides a convenient function for creating edge sets for longitudinal data, `ggobi_longitudinal`, that links observations in sequential time order.

This example uses the `stormtracks` data included in rggobi. The first argument is the dataset, the second is the variable specifying the time component, and the third is the variable that distinguishes the observations.

```
ggobi_longitudinal(stormtracks, seasday, id)
```

For regular time series data (already in order, with no grouping variables), just use `ggobi_longitudinal` with no other arguments.



Case study

This case study explores using rggobi to add model information to data; here will add confidence ellipsoids around the means so we can perform a graphical manova.

The first (and most complicated) step is to generate the confidence ellipsoids. The `ellipse` function does this. First we generate random points on the surface of sphere by drawing *npoints* from a random normal distribution and standardising each dimension. This sphere is then skewed to match the desired variance-covariance matrix, and its is size adjusted to give the appropriate *cl*-level confidence ellipsoid. Finally, the ellipsoid is translated to match the column locations.

```
conf.ellipse <- function(data, npoints=1000,
  cl=0.95, mean=colMeans(data), cov=var(data),
  n=nrow(data)
) {
  norm.vec  <-  function(x) x / sqrt(sum(x^2))

  p <- length(mean)
  ev <- eigen(cov)

  normsamp <- matrix(rnorm(npoints*p), ncol=p)
  sphere <- t(apply(normsamp, 1, norm.vec))

  ellipse <- sphere %*%
    diag(sqrt(ev$values)) %*% t(ev$vectors)
  conf.region <- ellipse * sqrt(p * (n-1) *
    qf(cl, p, n-p) / (n * (n-p)))
  if (!missing(data))
    colnames(ellipse) <- colnames(data)

  conf.region + rep(mean, each=npoints)
}
```

This function can be called with a data matrix, or with the sufficient statistics (mean, covariance matrix, and number of points). We can look at the output with ggobi:

```
ggobi(conf.ellipse(
  mean=c(0,0), cov=diag(2), n = 100))

cv <- matrix(c(1,0.15,0.25,1),
  ncol=2, n = 100)
ggobi(conf.ellipse(
  mean=c(1,2), cov=cv, n = 100))

mean <- c(0,0,1,2)
ggobi(conf.ellipse(
  mean=mean, cov=diag(4), n = 100))
```

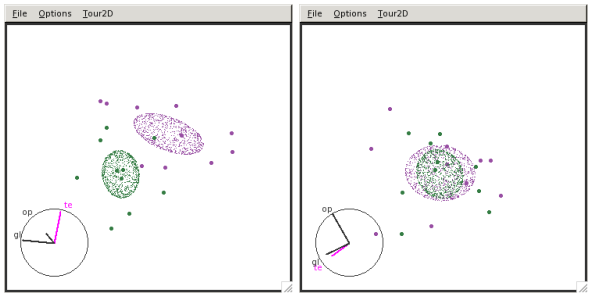
In the next step, we will need to take the original data and supplement it with the generated ellipsoid:

```
manovaci <- function(data, cl=0.95) {
  dm <- data.matrix(data)
  ellipse <- as.data.frame(
    conf.ellipse(dm, n=1000, cl=cl)
  )

  both <- rbind(data, ellipse)
  both$SIM <- factor(
    rep(c(FALSE, TRUE), c(nrow(data), 1000))
  )

  both
}
ggobi(manovaci(matrix(rnorm(30), ncol=3)))
```

Finally, we create a method to break a dataset into groups based on a categorical variable and compute the mean confidence ellipsoid for each group. We then use the automatic brushing functions to make the ellipsoid distinct and to paint each group a different colour. Here we use 68% confidence ellipsoids so that non-overlapping ellipsoids are likely to have significantly different means.

```
ggobi_manova <- function(data, catvar,
    cl=0.68) {
  each <- split(data, catvar)
  cis <- lapply(each, manovaci, cl=cl)

  df <- as.data.frame(do.call(rbind, cis))
  df$var <- factor(rep(
    names(cis), sapply(cis, nrow)
  ))

  g <- ggobi(df)
  glyph_type(g[1]) <- c(6,1)[df$SIM]
  glyph_colour(g[1]) <- df$var
  invisible(g)
}
```

These images show a graphical manova. You can see that in some projections the means overlap, but in others they do not.



Conclusion

GGobi is designed for data exploration, and its integration with R through rggobi allows a seamless workflow between analysis and exploration. Much of the potential of rggobi has yet to be realized,

but some ideas are demonstrated in the `classifly` package (Wickham, 2007), which visualises high-dimensional classification boundaries. We are also keen to hear about your work—if you develop a package using rggobi please let us know so we can highlight your work on the GGobi homepage.

We are currently working on the infrastructure behind GGobi and rggobi to allow greater control from within R. The next generation of rggobi will offer a direct low-level binding to the public interface of every GGobi module. This will coexist with the high-level interface presented in this paper. We are also working on consistently generating events in GGobi so that you will be able respond to events of interest from your R code. Together with the RGtk2 package (Lawrence and Temple Lang, 2007), this should allow the development of custom interactive graphics applications for specific tasks, written purely with high-level R code.

Acknowledgements

This work was supported by National Science Foundation grant DMS0706949. The ellipse example is taken, with permission, from Professor Di Cook's notes for multivariate analysis.

Bibliography

C. T. Butts, M. S. Handcock, and D. R. Hunter. *network: Classes for Relational Data*. Irvine, CA, July 26, 2008. URL `http://statnet.org/`. R package version 1.4-1.

M. S. Handcock, D. R. Hunter, C. T. Butts, S. M. Goodreau, and M. Morris. *ergm: A Package to Fit, Simulate and Diagnose Exponential-Family Models for Networks*. Seattle, WA, 2003. URL `http://CRAN.R-project.org/package=ergm`. Version 2.1 . Project home page at http://statnetproject.org.

M. Lawrence and D. Temple Lang. *RGtk2: R bindings for Gtk 2.8.0 and above*, 2007. URL `http://www.ggobi.org/rgtk2,http://www.omegahat.org`. R package version 2.12.1.

D. Temple Lang and D. F. Swayne. GGobi meets R: an extensible environment for interactive dynamic data visualization. In *Proceedings of the 2nd International Workshop on Distributed Statistical Computing*, 2001.

H. Wickham. *classifly: Explore classification models in high dimensions*, 2007. URL `http://had.co.nz/classifly`. R package version 0.2.3.