Journal of Statistical Software

December 2005, Volume 14, Issue 17. *http://www.jstatsoft.org/*

A Common Platform for Graphical Models in R: The gRbase Package

Claus Dethlefsen
Aalborg Hospital, Aarhus University Hospital

Søren Højsgaard
Danish Institute of Agricultural Sciences

Abstract

The **gRbase** package is intended to set the framework for computer packages for data analysis using graphical models. The **gRbase** package is developed for the open source language, R, and is available for several platforms. The package is intended to be widely extendible and flexible so that package developers may implement further types of graphical models using the available methods.

The **gRbase** package consists of a set of S version 3 classes and associated methods for representing data and models. The package is linked to the **dynamicGraph** package (Badsberg 2005), an interactive graphical user interface for manipulating graphs.

In this paper, we show how these building blocks can be combined and integrated with inference engines in the special cases of hierarchical log-linear models. We also illustrate how to extend the package to deal with other types of graphical models, in this case the graphical Gaussian models.

Keywords: graphical models, hierarchical log-linear models, graphical Gaussian models.

1. Introduction

Graphical models in their modern form have been around for nearly a quarter of a century. In the present context, a graphical model is a class of statistical models that can be represented by a graph which can be used to identify conditional independence properties. For terminology and theoretical aspects of graphical models, see e.g. Lauritzen (1996). Some common examples of graphical models are Bayesian networks (directed graphical models), log-linear models (undirected models), block-recursive graphical models, and models defined in the BUGS language, see Thomas (1994).

Various computer programs for inference in graphical models exist today. Some examples of free software programs are BUGS (Thomas 1994), **CoCo** (Badsberg 2001), **Digram** (Klein, Keiding, and Kreiner 1995), **MIM** (Edwards 2000), and **Tetrad** (Glymour, Scheines, Spirtes,

and Kelley 1987). Most such packages for graphical models have common characteristics: 1) They are tailor-made to analyse a particular class of models, 2) they have their own command language. Recently the source code of BUGS and **MIM** were released as open source, making it possible to extend the functionality by integrating these packages into more general tools.

It is of interest to make some of these programs and ideas underlying them go into a general purpose statistical package. The gR initiative (Lauritzen 2002) is a project launched in 2002 for making facilities in R (R Development Core Team 2005) for graphical modelling. The facilities should be easy to use, and they should be easily extendable. R is free software, open source, and runs on various platforms. This facilitates extensions in the form of R packages which may rely on the whole R system.

Recently, some of the existing graphical modelling programs have been made available in R. One example is the **mimR** package (Højsgaard 2004) which integrates the functionality of the stand-alone program **MIM** into R. Also, the **CoCo** program is now available in R. Other packages for graphical modelling available in R are **ggm** (Marchetti and Drton 2005), **deal** (Bøttcher and Dethlefsen 2003), and **SIN** (Drton and Perlman 2004).

The work in the gR initiative is organized in three levels: A *core group* works with defining data structures and standard methods, in particular developing the packages **gRbase**, **dynamicGraph** (see Section 6), and **giRaph**. The latter package, which is not described in this paper provides methods for representing and manipulating graphs efficiently. The *package developers* use the work from the core group to adapt or develop new packages for R that use a common user interface and data structures. Finally, the *group of end users* use the developed packages in their work with data or model analysis.

In this paper we describe the elements of **gRbase** (available from `http://CRAN.R-project.org/`) and illustrate how to combine them to create facilities for analysis of hierarchical log-linear models for discrete variables (undirected models). We also illustrate how to extend **gRbase** for other classes of models, here graphical Gaussian models.

The kernel of **gRbase** are the `gmData` and `gModel` classes described below.

`gmData` objects: A fundamental element of **gRbase** is a common class for representing data. No matter the actual representation of data, the important characteristics are contained in a graphical metadata (`gmData`) object. It contains the abstraction of data into a meta data object including variable names and types etc. However, the actual data might not be present or may be represented by a reference to data, such as a database file. This enables modelling although data are unavailable at the time of modelling, or if the data-amount is huge or if the data changes dynamically. Also, it may be possible to work without data, which may be valuable if the point of interest is in the model alone. Separating the specification of the variables from data has the benefit, that some properties of a model can be investigated without any reference to data, for example decomposability and collapsibility. The `gmData` class is described in Section 3.

`gModel` objects: A `gModel` object links a model to a `gmData` object. The model may be specified by either a formula or using an interactive graphical user interface. When defining a `gModel` object, no fitting is done. This is an important difference between model in **gRbase** and e.g., linear models in the function `lm` in R. The idea is, as mentioned before, that a model

may be interesting to analyse without any data attached. The `gModel` class is described in Section 4.

Depending on the type of the particular `gModel`, one may choose to fit the model, i.e., combine the model and data using a specified *engine*. There may be several engines available, depending on the methodology. For example models may be analysed from either a Bayesian or frequentist perspective. The result of the fitting process is an object that can be post-processed and provide the results from the analysis. Inference is described in Section 5.

Some features of **gRbase** will be illustrated in the present paper on the basis of the `rats` dataset in the **gRbase** package. The `rats` dataset is from a hypothetical drug trial, where the weight losses of male and female rats under three different drug treatments have been measured after one and two weeks. The dataset is provided in the **gRbase** package, and is further described in Edwards (2000). We will also refer to the dataset `HairEyeColor` (Snee 1974), included in R.

2. A small sample session

Before describing the core elements of **gRbase**, we present a sample session intended to give the reader a feel for how an end user will use **gRbase**.

Creating a gmData object first, data are created as a `gmData` object from an existing `table` object.

```
> library(gRbase)
> data(HairEyeColor)
> gmdHec <- as.gmData(HairEyeColor)
> gmdHec

  varNames shortNames varTypes nLevels
1     Hair          H Discrete       4
2      Eye          E Discrete       4
3      Sex          S Discrete       2
To see the values of the factors use the 'valueLabels' function
To see the data use the 'observations' function
```

Then, the model with sex independent of hair- and eye-color is defined, fitted (with the `loglm`-engine) and finally the output is analysed using the `anova` procedure to test the model against the saturated model.

```
> hecM1 <- hllm(~Hair * Eye + Sex, gmdHec)
> hecM1 <- fit(hecM1, engine = "loglm")
> anova(getFit(hecM1))

Call:
loglm(formula = loglm.formula, data = rawdata)

Statistics:
```

```
                     X^2 df   P(> X^2)
Likelihood Ratio 29.34982 15 0.01449443
Pearson          28.99286 15 0.01611871
```

The end user would likely display the models of interest using **dynamicGraph** (see Section 6), and could interact with the graph to investigate properties of the model or change the model. Also, the end user could change *engine* and for example use the same model, but analysed with a Gibbs sampling algorithm, providing another type of output.

3. The gmData class

A gmData object contains, by default, information about variable names, variable types, their labels, their levels (for discrete variables), and whether the variables are latent or not. Unique abbreviations (short names) of the variable names are created for ease of use when specifying model formulas. Besides, a gmData object may contain data or a reference to data, but need not do so.

3.1. Creating a gmData object from a data frame or a table

Typically one will create a gmData object (with data) from a data frame as follows:

```
> data(rats)
> gmdRats <- as.gmData(rats)
> gmdRats
```

```
  varNames shortNames   varTypes nLevels
1      Sex          S   Discrete       2
2     Drug          D   Discrete       3
3       W1         W1 Continuous      NA
4       W2         W2 Continuous      NA
To see the values of the factors use the 'valueLabels' function
To see the data use the 'observations' function
```

Also, data from a table can be converted into a gmData object:

```
> data(HairEyeColor)
> gmd.hec <- as.gmData(HairEyeColor)
> gmd.hec
```

```
  varNames shortNames varTypes nLevels
1     Hair          H Discrete       4
2      Eye          E Discrete       4
3      Sex          S Discrete       2
To see the values of the factors use the 'valueLabels' function
To see the data use the 'observations' function
```

Observe, that when an object is printed, only the summary of the variables are printed. Data and value labels are not displayed, but may be accessed separately.

It is also possible to write conversion methods for other data types, if needed.

3.2. Creating a gmData object manually

A gmData object may be created by the initialize method using the newgmData command.

```
> gmdRatsNodata <- newgmData(varNames = c("Sex", "Drug", "W1",
+     "W2"), varTypes = c("Discrete", "Discrete", "Continuous",
+     "Continuous"), nLevels = c(2, 3, NA, NA), valueLabels = list(Sex = c("M",
+     "F"), Drug = c("D1", "D2", "D3")))
> gmdRatsNodata

  varNames shortNames   varTypes nLevels
1      Sex          S   Discrete       2
2     Drug          D   Discrete       3
3       W1         W1 Continuous      NA
4       W2         W2 Continuous      NA
To see the values of the factors use the 'valueLabels' function

> observations(gmdRatsNodata)

NULL

> valueLabels(gmdRatsNodata)

$Sex
[1] "M" "F"

$Drug
[1] "D1" "D2" "D3"
```

The variable types must be from a vector of predefined types which may be inspected by the command validVarTypes(). The available types may be extended by the package developers as demonstrated below.

```
> oldtypes <- validVarTypes()
> validVartypes <- function() c(oldtypes, "MyVarType")
> validVartypes()

[1] "Discrete"   "Ordinal"    "Continuous" "MyVarType"
```

The types of the variables are important for the way they are displayed using the package **dynamicGraph**. The type is also important when the models are fitted to data.

3.3. Editing gmData objects

The information contained in a `gmData` object may be accessed or modified by the methods: `varTypes`, `varNames`, `nLevels`, `latent`, `valueLabels`, and `observations`. For example, to redefine the levels of the variable `Sex`, we can do:

```
> observations(gmdRatsNodata) <- rats
> valueLabels(gmdRatsNodata)$Sex <- c("Male", "Female")
> valueLabels(gmdRatsNodata)

$Sex
[1] "Male"   "Female"

$Drug
[1] "D1" "D2" "D3"
```

4. The gModel class

The general class `gModel` contains a formula object and a `gmData` object. Implementations of different specific graphical model classes can inherit from this class and provide methods for parsing the formula. Here, we illustrate by implementation of a class for hierarchical log-linear models, `hllm`.

For a hierarchical log-linear model, we use the following formula language. The right hand side of the formula is a list of the generators separated by '+'. A generator is specified by variable names with separated by '*'. Commonly used models have short hand notations: saturated model (`~.^.`), main effects (`~.^1`), all kth order interactions (`~.^k`). By an optional argument, `marginal`, it is possible to specify a subset of the variables from the `gmData` object.

The saturated model

```
> m1 <- hllm(~.^., gmdHec)
> formula(m1)

~Hair * Eye * Sex
```

The model where sex is independent of hair- and eye-color

```
> m2 <- hllm(~Hair * Eye + Sex, gmdHec)
```

The model with all main effects

```
> m3 <- hllm(~.^1, gmdHec)
> formula(m3)

~Hair + Eye + Sex
```

The saturated model in the hair-eye marginal

```
> m4 <- hllm(~.^., gmdHec, marginal = c("Hair", "Eye"))
> formula(m4)

~Hair * Eye
```

Also, the `gModel` class will have associated methods for making inference, which will be treated in Section 5.

4.1. Model editing

One important aspect of graphical modelling is the ability to interact with the model. Editing the model means e.g., that edges are added or removed and the resulting model is further investigated. The package developer needs to provide the methods `addEdge` and `dropEdge` for his model class.

```
> m5 <- addEdge(m2, "Hair", "Sex")
> formula(m5)

~Hair * Sex + Hair * Eye

> m6 <- dropEdge(m5, "Hair", "Eye")
> formula(m6)

~Eye + Hair * Sex
```

In addition, variables may be added or deleted from the model by the methods `dropVertex` and `addVertex`, which should also be provided by the package developer.

It is up to the package developer to define the body of these methods. The output should be an object similar to the input object. If for example the input object is a fitted object, the returned object should also be fitted with the same engine.

5. Inference

In **gRbase** we intend to exploit already existing software by letting these packages do the actual calculations, much like the approach taken in **mimR** which uses the the **MIM** stand alone program as an "inference engine". Hierarchical log-linear models can (when taking a frequentist perspective) be fitted by e.g., `loglm`, **CoCo**, and **MIM**. The default inference engine is `loglm` which is a part of R (in the **MASS** package). In the future, we envision changing the default inference engine to **CoCo**, because **CoCo** facilitates working with discrete and continuous variables in a frequentist setting. However, graphical models can also be analyzed in a Bayesian setting, and in this connection one can envision to use BUGS or JAGS as inference engine.

5.1. Model fitting

The `fit` procedure combines a model with an engine and produces an output object that may be further post-processed to yield the results used for inference. As the output objects may be very different, we have set the class of the fitted object to be multiple inheritance from the model class and the engine.

The default engine for fitting objects of class `hllm` is the `loglm` procedure contained in the **MASS** package, which uses the function `loglin`.

```
> m2.f <- fit(m2, engine = "loglm")
> m2.f
```

```
Model information (gRbase)
 Class:    gRfit <- loglm <- hllm <- gModel
 Formula:  ~Hair * Eye + Sex
Fit information (gRbase)
   logL 29.34982 df 15
```

6. Display and interaction with models

Using **dynamicGraph**, it is possible to edit the model using a graphical user interface.

The package **dynamicGraph** (Badsberg 2005) provides an implementation of an interactive graphical user interface for manipulating graphs, using `tcl/tk` (included in R). The end user should be provided with a homogeneous user interface no matter what type of graphical model he is analysing. The package developer can provide methods that creates the interface with **dynamicGraph**. As an example, the graph for the log-linear model `m2`, may be displayed (Figure 1) by the following command.

```
> dynamic.Graph(m2)
```

The end user can then interactively insert/delete edges in the graph. The user menu and context menus for edges and vertices provide access to the functionality for the current type of models and it is, for example, possible to fit the model. Also, clicking on edge labels will display the p-value for the test between the current model and the model with the edge removed.

The package developer can link procedures to items in the user menu, thus providing functionality for easy access to frequently used procedures. The following piece of code will add an item to the User Menu with the label "Stepwise". When selected, the procedure `stepwise` will be called with the current object as argument.

```
> UserMenus <- list(MainUser = list(label = "Stepwise", command = function(object,
+     ...) step(object, ...)))
> Z <- dynamic.Graph(m2, UserMenus = UserMenus)
```

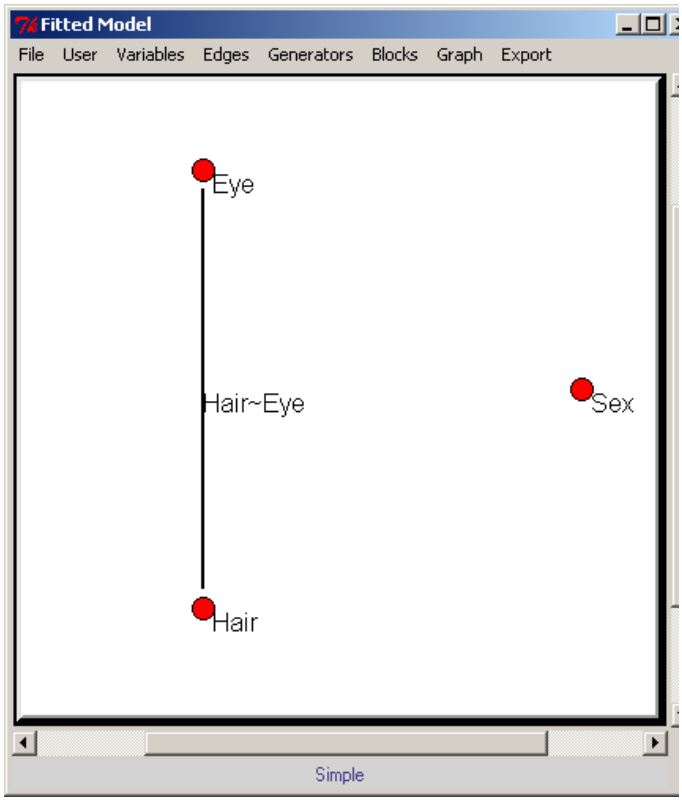

Figure 1: A display of the hierarchical log-linear model using **dynamicGraph**. The end user can manipulate the graph using a point and click interface.

Other packages exist for displaying graphs, e.g.,**Rgraphviz** and **mathgraph**.

7. Extending gRbase

The first step needed to extend **gRbase** to handle graphical Gaussian models consists of a constructor for the class, called `ggm` inheriting from `gModel`. We have chosen to use the same language for formulas as for `hllm` objects. The formulas are passed using the function `processFormula`. The definition of `ggm` is

```
> ggm <- function(formula = ~.^1, gmData, marginal) {
+     value <- processFormula(formula, gmData, marginal, "Continuous")
+     value$gmData <- gmData
+     class(value) <- c("ggm", "gModel")
+     return(value)
+ }
```

For fitting these types of models, we have implemented the function `fit.ggm` which uses the IPS algorithm to fit the model. After providing functions for `fit.ggm`, it is possible to work with this class of models in the same manner as `hllm`s.

```
> data(carcass)
> car <- as.gmData(carcass)
> m1 <- ggm(~.^., gmData = car)
> m1 <- fit(m1)
> dynamic.Graph(m1)
```

8. Discussion

In this paper we have described **gRbase**, a package that provides tools for creating new packages for graphical models in R. We believe that creating a common package for graphical modelling in R will not only help package developers in mutual support, but will also benefit the group of end users. It is the intention that **gRbase** can serve as a common platform for future developments of software for graphical modelling in R, such that new packages will have a common interface, will use the `gmData` objects as basic data objects etc. Packages should have similar user interfaces, both the command interface and graphical user interface and it should not be too difficult to switch between the different model classes.

The **dynamicGraph** package is a standalone package that enables an end user to manipulate graphs using a point and click interface. The package developers also have the benefit that it is possible to attach a graphical user interface to existing or new packages. The **gRbase** package provides the structure for a common representation of data, no matter the data source. This way facilitates working with models without/before data collection. Also, **gRbase** defines how models should be structured in classes and which methods should be associated by default. The **giRaph** package is planned to include methods for graph computations, which will work as a library for package developers.

Acknowledgements

The members of the gR project are acknowledged for their inspiration.

References

Badsberg JH (2001). "A Guide to **CoCo**." *Journal of statistical software*, **6**(4).

Badsberg JH (2005). **dynamicGraph***: Interactive Graphical Tool for Manipulating Graphs*. R package version 0.2.0.1, URL http://www.agrsci.org/.

Bøttcher S, Dethlefsen C (2003). "**deal**: A Package for Learning Bayesian Networks." *Journal of Statistical Software*, **8**(20).

Drton M, Perlman MD (2004). "A SINful Approach to Gaussian Graphical Model Selection." *Technical Report 457*, Department of Statistics, University of Washington.

Edwards D (2000). *Introduction to Graphical Modelling*. Springer-Verlag, New York, 2nd edition.

Glymour C, Scheines R, Spirtes P, Kelley K (1987). *Discovering Causal Structure*. Academic Press, San Diego.

Højsgaard S (2004). "The **mimR** Package for Graphical Modelling in R." *Journal of Statistical Software*, **11**(6).

Klein JP, Keiding N, Kreiner S (1995). "Graphical Models for Panel Studies, Illustrated on Data from the Framingham Heart Study." *Statistics in Medicine*, **14**, 1265–1290.

Lauritzen SL (1996). *Graphical Models*. Oxford University Press, Oxford.

Lauritzen SL (2002). "gRaphical models in R." *R News*, **2**(2), 39.

Marchetti GM, Drton M (2005). **ggm***: Graphical Gaussian Models*. R package version 1.0.

R Development Core Team (2005). *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/.

Snee RD (1974). "Graphical Display of Two-way Contingency Tables." *The American Statistician*, **28**, 9–12.

Thomas A (1994). "BUGS: A Statistical Modelling Package." RTA/BCS Modular Languages Newsletter.

Affiliation:

Claus Dethlefsen
Center for Cardiovascular Research
Aalborg Hospital, Aarhus University Hospital
9000 Aalborg, Denmark
E-mail: aas.claus.dethlefsen@nja.dk

Søren Højsgaard
Department of Genetics and Bioinformatics
Danish Institute of Agricultural Sciences
Research Center Foulum
8830 Tjele, Denmark
E-mail: Soren.Hojsgaard@agrsci.dk

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 14, Issue 17 *Submitted:* 2005-08-10
December 2005 *Accepted:* 2005-12-03