On the Usage of the `gRim` Package
** WORKING DOCUMENT **

Søren Højsgaard
Aarhus University, Denmark

June 4, 2012

Contents

List of Corrections

1 Introduction

The `gRim` package is an `R` package for *gRaphical interaction models* (hence the name). `gRim` implements 1) graphical log–linear models for discrete data, that is for contingency tables and 2) Gaussian graphical models for continuous data (multivariate normal data) and 3) mixed homogeneous interaction models for mixed data (data consisiting of both discrete and continuous variables).

The package is at an early stage of development and so is this document.

2 Introductory examples

The main functions for creating models of the various types are:

- Discrete data: The *dmod()* function creates a hierarchical log–linear model.
- Continuous data: The *cmod()* function creates a Gaussian graphical model.
- Mixed data: The *mmod()* function creates a mixed interaction model.

The arguments to the model functions are:

```
 args(dmod)

function (formula, data, marginal = NULL, interactions = NULL,
    fit = TRUE, details = 0)
NULL

 args(cmod)

function (formula, data, marginal = NULL, fit = TRUE, details = 0)
NULL

 args(mmod)

function (formula, data, marginal = NULL, fit = TRUE, details = 0)
NULL
```

The model objects created by these functions are of the respective classes `dModel`, `cModel` and `mModel`. All models are also of the class `iModel`. We focus the presentation on models for discrete data, but most of the topics we discuss apply to all types of models.

2.1 A Discrete Model

The *reinis* data from `gRbase` is a 2^6 contingency table.

```
 data(reinis)
 str(reinis)

 table [1:2, 1:2, 1:2, 1:2, 1:2, 1:2] 44 40 112 67 129 145 12 23 35 12 ...
 - attr(*, "dimnames")=List of 6
  ..$ smoke  : chr [1:2] "y" "n"
  ..$ mental : chr [1:2] "y" "n"
  ..$ phys   : chr [1:2] "y" "n"
  ..$ systol : chr [1:2] "y" "n"
  ..$ protein: chr [1:2] "y" "n"
  ..$ family : chr [1:2] "y" "n"
```

Models are specified as generating classes. A generating class can be a list or a right–hand–sided formula. In addition, various model specification shortcuts are available. Some of these are described in Section 2.2.

The following two specifications of a log–linear model are equivalent:

```
 data(reinis)
 dm1<-dmod(list(c("smoke","systol"),c("smoke","mental","phys")), data=reinis)
 dm1<-dmod(~smoke:systol + smoke:mental:phys, data=reinis)
 dm1

Model: A dModel with 4 variables
 graphical :  TRUE  decomposable :  TRUE
 -2logL    :        9391.38 mdim :    9 aic :      9409.38
 ideviance :         730.47 idf  :    5 bic :      9459.05
 deviance  :           3.80 df   :    6
```

The output reads as follows: `-2logL` is minus twice the maximized log–likelihood and `mdim` is the number of parameters in the model (no adjustments have been made for sparsity of data). The `ideviance` and `idf` gives the deviance and degrees of freedom between the model and the independence model for the same variables. `deviance` and `df` is the deviance and degrees of freedom between the model and the saturated model for the same variables.

Section 8.1 describes model objects in more detail. Here we just notice that the generating class of the model is contained in the slot `glist`:

Notice that the generating class does not appear directly. However the generating class can be retrieved using *formula()* and *terms()*:

```
 formula(dm1)
 str(terms(dm1))

List of 2
 $ : chr [1:2] "smoke" "systol"
 $ : chr [1:3] "smoke" "mental" "phys"
```

A summary of a model is provided by the *summary()* function:

```
 summary(dm1)

is graphical=TRUE; is decomposable=TRUE
generators (glist):
  :"smoke" "systol"
  :"smoke" "mental" "phys"
```

FiXme Note: The summary() method leaves a bit to be desired...

2.2 Model specification shortcuts

Below we illustrate various other ways of specifying log–linear models.

- A saturated model can be specified using `~.^.` whereas `~.^2` specifies the model with all–two–factor interactions. Using `~.^1` specifies the independence model.
- If we want, say, at most two–factor interactions in the model we can use the `interactions` argument.
- Attention can be restricted to a subset of the variables using the `marginal` argument.
- Variable names can be abbreviated.

The following models illustrate these abbreviations:

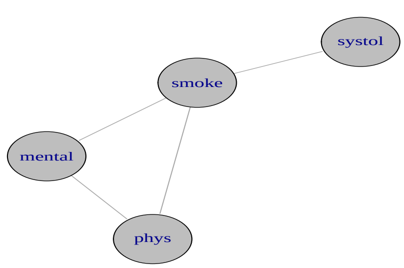
```
dm2 <- dmod(~.^2, margin=c("smo","men","phy","sys"),
              data=reinis)
formula(dm2)
```

```
dm3 <- dmod(list(c("smoke", "systol"), c("smoke", "mental", "phys")),
              data=reinis, interactions=2)
formula(dm3)
```

2.3 Plotting models

There are two methods for plotting the dependence graph of a model: Using *iplot()* and *plot()*. The convention for both methods is that discrete variables are drawn as grey dots and continuous variables as white dots. 1) *iplot()* creates an `igraph` object and plots this. 2) 2) *plot()* creates a `graphNEL` object and plots this. However, to plot a `graphNEL` object, the package `Rgraphviz` and the external program `Graphviz` must be installed installed.

```
iplot(dm1)
```



2.4 A Continuous Model

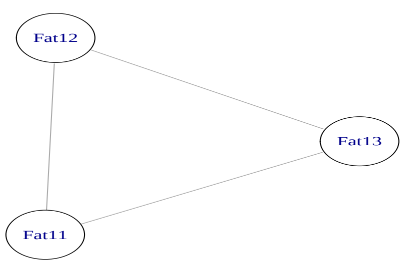
For Gaussian models there are at most second order interactions. Hence we may specify the saturated model in different ways:

```
data(carcass)
cm1 <- cmod(~Fat11:Fat12:Fat13, data=carcass)
cm1 <- cmod(~Fat11:Fat12 + Fat12:Fat13 + Fat11:Fat13, data=carcass)
cm1

Model: A cModel with 3 variables
 graphical :  TRUE  decomposable :  TRUE
 -2logL    :        4329.16 mdim :    6 aic :      4341.16
 ideviance :         886.10 idf  :    3 bic :      4364.20
 deviance  :           0.00 df   :    0
```

FiXme Note: Harmonize cmod() output with that of dmod()

```
iplot(cm1)
```



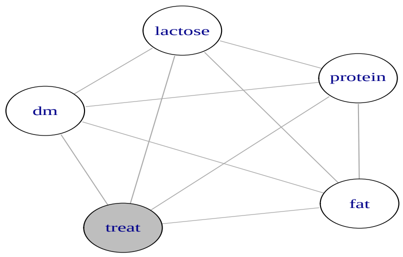
2.5 A Mixed Model

```
data(milkcomp1)
mm1 <- mmod(~.^., data=milkcomp1)
mm1

Model: A mModel with 5 variables
 graphical :  TRUE  decomposable :  TRUE
 -2logL    :         475.92 mdim :   44 aic :       563.92
 ideviance :         101.57 idf  :   30 bic :       652.24
 deviance  :           0.00 df   :    0
```

FiXme Note: Harmonize mmod() output with that of dmod()

```
iplot(mm1)
```



3 Model editing - update()

The *update()* function enables `dModel` objects to be modified by the addition or deletion of interaction terms or edges, using the arguments `aterm`, `dterm`, `aedge` or `dedge`. Some examples follow:

- Set a marginal saturated model:

```
ms <- dmod(~.^., marginal=c("phys","mental","systol","family"), data=reinis)
formula(ms)
```

- Delete one edge:

```
ms1 <- update(ms, list(dedge=~phys:mental))
formula(ms1)
```

- Delete two edges:

```
ms2<- update(ms, list(dedge=~phys:mental+systol:family))
formula(ms2)
```

- Delete all edges in a set:

```
ms3 <- update(ms, list(dedge=~phys:mental:systol))
formula(ms3)
```

- Delete an interaction term

```
ms4 <- update(ms, list(dterm=~phys:mental:systol) )
formula(ms4)
```

- Add three interaction terms:

```
ms5 <- update(ms, list(aterm=~phys:mental+phys:systol+mental:systol) )
formula(ms5)
```

- Add two edges:

```
ms6 <- update(ms, list(aedge=~phys:mental+systol:family))
formula(ms6)
```

A brief explanation of these operations may be helpful. To obtain a hierarchical model when we delete a term from a model, we must delete any higher-order relatives to the term. Similarly, when we add an interaction term we must also add all lower-order relatives that were not already present. Deletion of an edge is equivalent to deleting the corresponding two-factor term. Let $m - e$ be the result of deleting edge e from a model m. Then the result of adding e is defined as the maximal model m^* for which $m^* - e = m$.

4 Testing for conditional independence

Tests of general conditional independence hypotheses of the form $u \perp\!\!\!\perp v \,|\, W$ can be performed using the *ciTest()* function.

```
 cit <- ciTest(reinis, set=c("systol","smoke","family","phys"))

Testing systol _|_ smoke | family phys
Statistic (DEV):   13.045 df: 4 p-value: 0.0111 method: CHISQ
```

The general syntax of the `set` argument is of the form (u, v, W) where u and v are variables and W is a set of variables. The `set` argument can also be given as a right–hand sided formula.

In model terms, the test performed by *ciTest()* corresponds to the test for removing the edge $\{u, v\}$ from the saturated model with variables $\{u, v\} \cup W$. If we (conceptually) form a factor S by crossing the factors in W, we see that the test can be formulated as a test of the conditional independence $u \perp\!\!\!\perp v \,|\, S$ in a three way table. The deviance decomposes into independent contributions from each stratum:

$$D = 2 \sum_{ijs} n_{ijs} \log \frac{n_{ijs}}{\hat{m}_{ijs}} = \sum_s 2 \sum_{ij} n_{ijs} \log \frac{n_{ijs}}{\hat{m}_{ijs}} = \sum_s D_s$$

where the contribution D_s from the sth slice is the deviance for the independence model of u and v in that slice. For example,

```
 cit$slice

  statistic  p.value df family phys
1  4.734420 0.029565  1      y    y
2  0.003456 0.953121  1      n    y
3  7.314160 0.006841  1      y    n
4  0.993337 0.318928  1      n    n
```

The sth slice is a $\texttt{u} \times \texttt{v}$ table $\{n_{ijs}\}_{i=1\ldots\texttt{u},j=1\ldots\texttt{v}}$. The number of degrees of freedom corresponding to the test for independence in this slice is

$$df_s = (\#\{i : n_{i\cdot s} > 0\} - 1)(\#\{j : n_{\cdot js} > 0\} - 1)$$

where $n_{i\cdot s}$ and $n_{\cdot js}$ are the marginal totals.

An alternative to the asymptotic χ^2 test is to determine the reference distribution using Monte Carlo methods. The marginal totals are sufficient statistics under the null hypothesis, and in a conditional test the test statistic is evaluated in the conditional distribution given the sufficient statistics. Hence one can generate all possible tables with those given margins, calculate the desired test statistic for each of these tables and then see how extreme the observed test statistic is relative to those of the calculated tables. A Monte Carlo approximation to this procedure is to randomly generate large number of tables with the given margins, evaluate the statistic for each simulated table and then see how extreme the observed test statistic is in this distribution. This is called a *Monte Carlo exact test* and it provides a *Monte Carlo p–value*:

```
 ciTest(reinis, set=c("systol","smoke","family","phys"), method='MC')

Testing systol _|_ smoke | family phys
Statistic (DEV):   13.045 df: NA p-value: 0.0175 method: MC
```

FiXme Note: Missing ciTest for continuous and mixed data...

5 Fundamental methods for inference

This section describes some fundamental methods for inference in `gRim`. As basis for the description consider the following model shown in Fig. 1:

```
 dm5 <- dmod(~ment:phys:systol+ment:systol:family+phys:systol:smoke,
             data=reinis)

Model: A dModel with 5 variables
 graphical :  TRUE  decomposable :  TRUE
 -2logL    :       10888.82 mdim :   15 aic :     10918.82
 ideviance :         732.29 idf  :   10 bic :     11001.59
 deviance  :          25.59 df   :   16
```

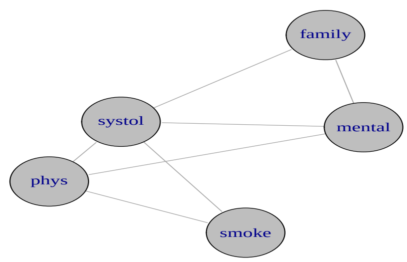


Figure 1: A decomposable graphical model for the `reinis` data.

5.1 Testing for addition and deletion of edges

Let $\mathcal{M}_0$ be a model and let $e = \{u, v\}$ be an edge in $\mathcal{M}_0$. The candidate model formed by deleting e from $\mathcal{M}_0$ is $\mathcal{M}_1$. The `testdelete()` function can be used to test for deletion of an edge from a model:

```
 testdelete(dm5, ~smoke:systol)

dev:   11.698 df:  2 p.value: 0.00288 AIC(k=2.0):    7.7 edge: smoke:systol
host:  systol phys smoke
Notice: Test performed in saturated marginal model

 testdelete(dm5, ~family:systol)

dev:    1.085 df:  2 p.value: 0.58135 AIC(k=2.0):   -2.9 edge: family:systol
host:  systol family mental
Notice: Test performed in saturated marginal model
```

In the first case the p–value suggests that the edge can not be deleted. In the second case the p–value suggests that the edge can be deleted. The reported AIC value is the difference in AIC between the candidate model and the original model. A negative value of AIC suggest that the candidate model is to be preferred.

Next, let $\mathcal{M}_0$ be a model and let $e = \{u, v\}$ be an edge not in $\mathcal{M}_0$. The candidate model formed by adding e to $\mathcal{M}_0$ is denoted $\mathcal{M}_1$. The `testadd()` function can be used to test for deletion of an edge from a model:

```
 testadd(dm5, ~smoke:mental)

dev:    7.797 df:  4 p.value: 0.09930 AIC(k=2.0):    0.2 edge: smoke:mental
host:  mental systol phys smoke
Notice: Test performed in saturated marginal model
```

The p–value suggests that no significant improvedment of the model is obtained by adding the edge. The reported AIC value is the difference in AIC between the candidate model and the original model. A negative value of AIC would have suggested that the candidate model is to be preferred.

FiXme Note: A function for testing addition / deletion of more general terms is needed.

5.2 Finding edges

The `getInEdges()` function will return a list of all the edges in the dependency graph $\mathcal{G}$ defined by the model. If we set `type='decomposable'` then the edges returned are as follows: An edge $e = \{u, v\}$ is returned if $\mathcal{G}$ minus the edge e is decomposable. In connection with model selection this is convenient because it is thereby possibly to restrict the search to decomposable models.

```
 ed.in <- getInEdges(ugList(dm5$glist), type="decomposable")

     [,1]     [,2]
[1,] "phys"   "mental"
[2,] "family" "mental"
[3,] "smoke"  "phys"
[4,] "family" "systol"
[5,] "smoke"  "systol"
```

The `getOutEdges()` function will return a list of all the edges which are not in the dependency graph $\mathcal{G}$ defined by the model. If we set `type='decomposable'` then the edges returned are as follows: An edge $e = \{u, v\}$ is returned if $\mathcal{G}$ plus the edge e is decomposable. In connection with model selection this is convenient because it is thereby possibly to restrict the search to decomposable models.

```
 ed.out <- getOutEdges(ugList(dm5$glist), type="decomposable")

     [,1]     [,2]
[1,] "smoke"  "mental"
[2,] "family" "phys"
```

5.3 Testing several edges

```
 args(testInEdges)

function (object, edgeMAT = NULL, criterion = "aic", k = 2, alpha = NULL,
    headlong = FALSE, details = 1, ...)
NULL

 args(testOutEdges)

function (object, edgeMAT = NULL, criterion = "aic", k = 2, alpha = NULL,
    headlong = FALSE, details = 1, ...)
NULL
```

The functions `labelInEdges()` and `labelOutEdges()` will test for deletion of edges and addition of edges. The default is to use AIC for evaluating each edge. It is possible to specify the penalty parameter for AIC to being other values than 2 and it is possible to base the evaluation on significance tests instead of AIC. Setting `headlong=TRUE` causes the function to exit once an improvement is found. For example:

```
 testInEdges(dm5, getInEdges(ugList(dm5$glist), type="decomposable"),
             k=log(sum(reinis)))

  statistic df   p.value     aic     V1     V2 action
1   686.703  2 0.000e+00 671.667   phys mental      -
2     4.693  2 9.572e-02 -10.344 family mental      +
3    28.147  2 7.726e-07  13.111  smoke   phys      -
4     1.085  2 5.813e-01 -13.951 family systol      +
5    11.698  2 2.882e-03  -3.338  smoke systol      +
```

6 Stepwise Model Selection

Two functions are currently available for model selection: `backward()` and `forward()`. These functions employ the functions in Section 5.3)

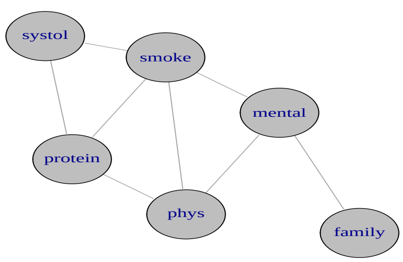
6.1 Backward search

For example, we start with the saturated model and do a backward search.

```
dm.sat <- dmod(~.^., data=reinis)
dm.back <- backward(dm.sat)

. BACKWARD: type=decomposable search=all, criterion=aic(2.00), alpha=0.00
. Initial model: is graphical=TRUE is decomposable=TRUE
  change.AIC  -19.7744 Edge deleted: systol mental
  change.AIC   -8.8511 Edge deleted: systol phys
  change.AIC   -4.6363 Edge deleted: protein mental
  change.AIC   -1.6324 Edge deleted: family systol
  change.AIC   -3.4233 Edge deleted: protein family
  change.AIC   -0.9819 Edge deleted: family phys
  change.AIC   -1.3419 Edge deleted: family smoke

iplot(dm.back)
```



Default is to search among decomposable models if the initial model is decomposable. Default is also to label all edges (with AIC values); however setting `search='headlong'` will cause the labelling to stop once an improvement has been found.

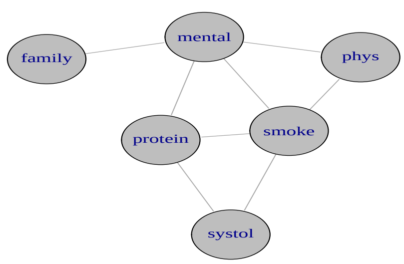
6.2 Forward search

Forward search works similarly; for example we start from the independence model:

```
dm.i   <- dmod(~.^1, data=reinis)
dm.forw <- forward(dm.i)

. FORWARD: type=decomposable search=all, criterion=aic(2.00), alpha=0.00
. Initial model: is graphical=TRUE is decomposable=TRUE
  change.AIC -683.9717 Edge added: phys mental
  change.AIC  -25.4810 Edge added: phys smoke
  change.AIC  -15.9293 Edge added: mental protein
  change.AIC  -10.8092 Edge added: protein systol
  change.AIC   -2.7316 Edge added: mental family
  change.AIC   -1.9876 Edge added: smoke mental
  change.AIC  -16.4004 Edge added: smoke protein
  change.AIC  -12.5417 Edge added: smoke systol

iplot(dm.forw)
```



6.3 Fixing edges/terms in model as part of model selection

The stepwise model selection can be controlled by fixing specific edges. For example we can specify edges which are not to be considered in a bacward selection:

```
fix <- list(c("smoke","phys","systol"), c("systol","protein"))
fix <- do.call(rbind, unlist(lapply(fix, names2pairs),recursive=FALSE))
fix

     [,1]      [,2]
[1,] "phys"    "smoke"
[2,] "smoke"   "systol"
[3,] "phys"    "systol"
[4,] "protein" "systol"

dm.s3 <- backward(dm.sat, fixin=fix, details=1)

. BACKWARD: type=decomposable search=all, criterion=aic(2.00), alpha=0.00
. Initial model: is graphical=TRUE is decomposable=TRUE
  change.AIC  -19.7744 Edge deleted: systol mental
  change.AIC   -4.6982 Edge deleted: systol family
  change.AIC   -6.8301 Edge deleted: family protein
  change.AIC   -1.2294 Edge deleted: mental protein
  change.AIC   -0.9819 Edge deleted: family phys
  change.AIC   -1.3419 Edge deleted: family smoke
```

There is an important detail here: The matrix `fix` specifies a set of edges. Submitting these in a call to *backward* does not mean that these edges are forced to be in the model. It means that those edges in `fixin` which are in the model will not be removed.

Likewise in forward selection:

```
dm.i3 <- forward(dm.i, fixout=fix, details=1)

. FORWARD: type=decomposable search=all, criterion=aic(2.00), alpha=0.00
. Initial model: is graphical=TRUE is decomposable=TRUE
  change.AIC -683.9717 Edge added: phys mental
  change.AIC  -15.9293 Edge added: mental protein
  change.AIC  -15.4003 Edge added: protein smoke
  change.AIC   -8.6638 Edge added: smoke mental
  change.AIC   -2.7316 Edge added: mental family
  change.AIC   -1.1727 Edge added: protein phys
```

Edges in `fix` will not be added to the model but if they are in the starting model already, they will remain in the final model.

7 Further topics on models for contingency tables

7.1 Adjusting for sparsity

FiXme Note: Comment on adjustment for sparsity in testadd() and testdelete()

7.2 Dimension of a log–linear model

The `loglinDim()` is a general function for finding the dimension of a log–linear model. It works on the generating class of a model being represented as a list:

```
loglinGenDim(dm2$glist, reinis)

[1] 10
```

8 Miscellaneous

8.1 The Model Object

It is worth looking at the information in the model object:

```
dm3 <- dmod(list(c("smoke", "systol"), c("smoke", "mental", "phys")),
            data=reinis)
names(dm3)

[1] "glist"          "varNames"       "datainfo"       "fitinfo"
[5] "isFitted"       "glistNUM"       "isDecomposable" "isGraphical"
```

- The model, represented as a list of generators, is

  ```
  str(dm3$glist)

  List of 2
   $ : chr [1:2] "smoke" "systol"
   $ : chr [1:3] "smoke" "mental" "phys"
  ```

  ```
  str(dm3$glistNUM)

  List of 2
   $ : int [1:2] 1 2
   $ : int [1:3] 1 3 4
  ```

The numeric representation of the generators refers back to

```
dm3$varNames

[1] "smoke"  "systol" "mental" "phys"
```

Notice the model object does not contain a graph object. Graph objects are generated on the fly when needed.

- Information about the variables etc. is

```
str(dm3[c("varNames","conNames","conLevels")])

List of 3
 $ varNames: chr [1:4] "smoke" "systol" "mental" "phys"
 $ NA      : NULL
 $ NA      : NULL
```

- Finally `isFitted` is a logical for whether the model is fitted; `data` is the data (as a table) and `fitinfo` consists of fitted values, logL, df etc.