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

Søren Højsgaard
Aarhus University, Denmark

March 31, 2011

## 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.

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 model objects created by these functions are of the respective classes `dModel`, `cModel`and `mModel`. All models are also of the class `iModel`. To the extent possible methods are developed to act on `iModel` objects that the same method will do something sensible on both discrete and continuous models.

If the `Rgraphviz` package is installed then `iModel`s can be plotted.

## 2 Introductory examples

The arguments to the model functions are:

```
> args(dmod)

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

> args(cmod)

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

> args(mmod)

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

Models are specified as generating classes. A generating class can be a list or a formula. In addition, various model specification shortcuts are available. Some of these are described in Section **??**. **FiXme**: *Ref!*

## 2.1 A First Example – A Discrete Model

```
> args(dmod)

function (formula, data = NULL, marginal = NULL, context, interactions = NULL,
    fit = TRUE)
NULL
```

Consider the *reinis* data from `gRbase`. 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:
  :"smoke" "systol"
  :"smoke" "mental" "phys"
Fit info:
 -2logL    :      9391.38432131 mdim  :    9
 ideviance :       730.47272159 idf   :    5
 deviance  :         3.80201740 df    :    6
is graphical=TRUE is decomposable=TRUE
```

The `ideviance` and `idf` gives the deviance and degrees of freedom between the model and the independence model for the same variables whereas `deviance` and `df` is the deviance and degrees of freedom between the model and the saturated model for the same variables. Finally `mdim` is the number of parameters in the model (no adjustments have been made for sparsity of data).

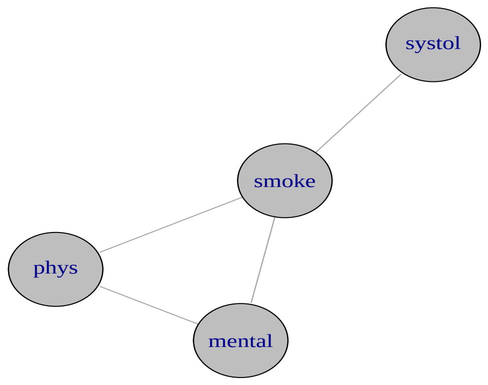
The internal representation of the model is through a generating class in the slot `glist`.

We have

```
> str(dm1$glist)


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

```
> iplot(dm1)
```



## 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.
- Attention can be restricted to a subset of the variables using the `margin` argument.
- The `context` argument can be used for specifying that only data for a specific slice of the table should be used.
- Variable names can be abbreviated.
- If we want, say, at most two–factor interactions in the model we can use the `interactions` argument.

The following model illustrates this:

```
> dm2 <- dmod(~.^2, margin=c("smo","men","phy","sys"),
+             data=reinis, context=list("fam", "y"))

Model:
  :"smoke" "mental"
  :"smoke" "phys"
  :"smoke" "systol"
  :"smoke" "protein"
  :"mental" "phys"
  :"mental" "systol"
  :"mental" "protein"
  :"phys" "systol"
  :"phys" "protein"
  :"systol" "protein"
Fit info:
 -2logL    :     10125.40198988 mdim  :   15
 ideviance :       696.45587241 idf   :   10
 deviance  :        17.46324669 df    :   16
is graphical=FALSE is decomposable=FALSE
```

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

Model:
  :"smoke" "systol"
  :"smoke" "mental"
  :"smoke" "phys"
  :"mental" "phys"
Fit info:
 -2logL    :      9397.37069914 mdim  :    8
 ideviance :       724.48634376 idf   :    4
 deviance  :         9.78839523 df    :    7
is graphical=FALSE is decomposable=FALSE
```

## 2.3 Plotting models

There are two main ways of plotting the dependence graph of a model: 1) `iplot()` creates an `igraph` object and plots this. 2) A `plot()` method is available, but this requires that the package `Rgraphviz` and the external program `Graphviz` is installed. The convention for both methods is that discrete variables are drawn as grey dots and continuous variables as white dots.

## 2.4 A First Example – A Continuous Model

```
> args(cmod)

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

For Gaussian models there is no such thing as a higher order interaction. Hence all Gaussian models are graphical and therefore the following specifications define the same model:

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

Model:
  :"Fat11" "Fat12" "Fat13"
Fit info:
 -2logL    :      4329.15579031 mdim  :    6
 ideviance :       886.10467639 idf   :    3
 deviance  :         0.00000000 df    :    0
 aic       :      4341.1558
 bic       :      4364.1996
is graphical=TRUE is decomposable=TRUE

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

Model:
  :"Fat11" "Fat12"
  :"Fat12" "Fat13"
  :"Fat11" "Fat13"
Fit info:
 -2logL    :      4329.15579031 mdim  :    6
 ideviance :       886.10467639 idf   :    3
 deviance  :         0.00000000 df    :    0
 aic       :      4341.1558
 bic       :      4364.1996
is graphical=TRUE is decomposable=FALSE
```

**FiXme**: *is decomposable er forkert i model nummer 2...*

Notice: Internally, `cmod()` works on the model as it is given by the user.

## 2.5 A First Example – A Mixed Model

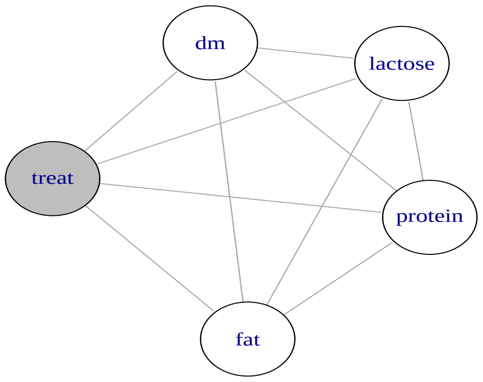
```
> args(mmod)

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

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

Mixed interaction model:
Model:
  :"treat" "fat" "protein" "dm" ...
Dimension:  40 df:   0 logL -237.957537 -2logL=475.915074
```

```
> iplot(mm1)
```



## 3 Methods for model objects

A `summary()` of a model:

```
> summary(dm1)

is graphical=TRUE; is decomposable=TRUE
generators (glist):
  :"smoke" "systol"
  :"smoke" "mental" "phys"
EXPERIMENTAL: components:  glist isGraphical isDecomposable cliques
```

**FiXme**: *Make summary() more informative*

The model can be plotted if the `Rgraphviz` package is installed (see Fig. 1):

```
> iplot(dm1)
```

Observed and fitted values are:

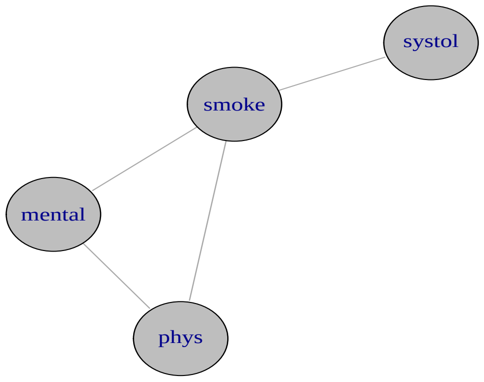


Figure 1: A marginal model for a slice of the `reinis` data.

```
> str(fitted(dm1))

 table [1:2, 1:2, 1:2, 1:2] 78.2 74.7 67.8 47.3 211.1 ...
 - attr(*, "dimnames")=List of 4
  ..$ smoke : chr [1:2] "y" "n"
  ..$ systol: chr [1:2] "y" "n"
  ..$ mental: chr [1:2] "y" "n"
  ..$ phys  : chr [1:2] "y" "n"

> str(dm1$data)

List of 1
 $ data: table [1:2, 1:2, 1:2, 1:2] 79 82 67 40 217 156 177 109 197 258 ...
  ..- attr(*, "dimnames")=List of 4
  .. ..$ smoke : chr [1:2] "y" "n"
  .. ..$ systol: chr [1:2] "y" "n"
  .. ..$ mental: chr [1:2] "y" "n"
  .. ..$ phys  : chr [1:2] "y" "n"
```

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`:

```
> str(dm2$glist)

List of 10
 $ : chr [1:2] "smoke" "mental"
 $ : chr [1:2] "smoke" "phys"
 $ : chr [1:2] "smoke" "systol"
 $ : chr [1:2] "smoke" "protein"
 $ : chr [1:2] "mental" "phys"
 $ : chr [1:2] "mental" "systol"
 $ : chr [1:2] "mental" "protein"
 $ : chr [1:2] "phys" "systol"
 $ : chr [1:2] "phys" "protein"
 $ : chr [1:2] "systol" "protein"
```

## 4 Model editing - `update()`

**FiXme**: *update() method needs to be described.*

## 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. 2:

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

Model:
  :"mental" "phys" "systol"
  :"mental" "systol" "family"
  :"phys" "systol" "smoke"
Fit info:
 -2logL    :     10888.82063844 mdim  :   15
 ideviance :       732.29408038 idf   :   10
 deviance  :        25.58766995 df    :   16
is graphical=TRUE is decomposable=TRUE
```

### 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:

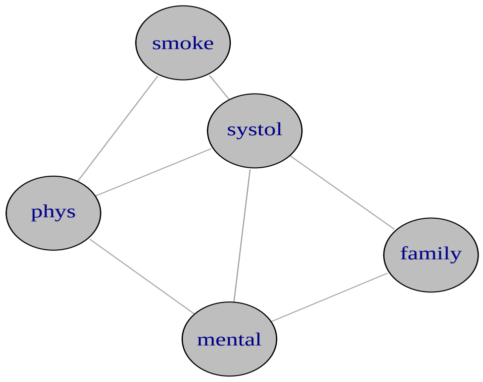


Figure 2: A decomposable graphical model for the `reinis` data.

```
> 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**: *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 Labeling several edges

```
> args(labelInEdges)

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

> args(labelOutEdges)

function (object, edgeMAT, 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:

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

   statistic df      p.value        aic     V1     V2 action
1 686.702943  2 0.000000e+00 671.666814   phys mental      -
2   4.692559  2 9.572463e-02 -10.343569 family mental      +
3  28.146937  2 7.726277e-07  13.110809  smoke   phys      -
4   1.084805  2 5.813498e-01 -13.951323 family systol      +
5  11.698229  2 2.882450e-03  -3.337899  smoke systol      +
```

**FiXme**: *labelInEdges()/labelOutEdges() kunne have default værdi for 2. argument.*

## 6 Stepwise Model Selection

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

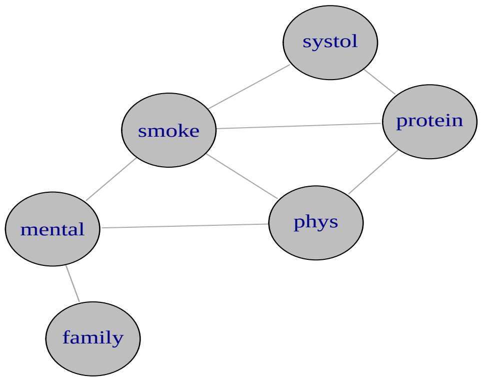
### 6.1 Backward search

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 decompsable=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.

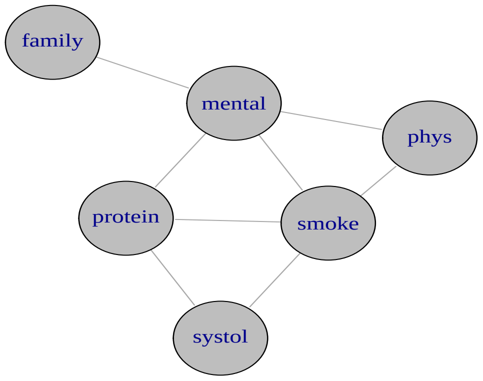
## 6.2 Forward search

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 decompsable=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 decompsable=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
```

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 decompsable=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   -1.9876 Edge added: smoke mental
  change.AIC  -16.4004 Edge added: smoke protein
  change.AIC  -12.5417 Edge added: smoke systol
  change.AIC   -1.0037 Edge added: protein family
  change.AIC   -0.1381 Edge added: family systol
```

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**: *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] 15
```

## Acknowledgements

## 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, context=list("family", "y"))
> names(dm3)

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

- The model (as a right hand sided formula) is

  ```
  > dm3$formula

  NULL
  ```

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

  ```
  > str(dm3[c("glist","glistNUM")])

  List of 2
   $ glist   :List of 2
    ..$ : chr [1:2] "smoke" "systol"
    ..$ : chr [1:3] "smoke" "mental" "phys"
   $ glistNUM:List of 2
    ..$ : int [1:2] 1 2
    ..$ : int [1:3] 1 3 4
  ```

- The dependency graph of the model is

```
> dm3$graph

NULL

> dm3$adjmat

NULL
```

- Information about the variables etc. is

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

List of 3
 $ varNames : chr [1:4] "smoke" "systol" "mental" "phys"
 $ conNames : chr "family"
 $ conLevels: chr "y"
```

- 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.