Addendum to the paper

The `mimR` Package for Graphical Modelling in `R` DRAFT – comments are welcome!

Søren Højsgaard
Danish Institute of Agricultural Sciences

February 20, 2007

Contents

Introduction to the addendum

The `mimR` package for graphical modelling in `R` was described by Højsgaard (2004). A major revision of the package has implied some changes in the functionality related to the description in Højsgaard (2004). Therefore, this addendum is the relevant document to use in connection with practical use of `mimR`.

The major changes relative to Højsgaard (2004) are:

- Models are fitted at the time of specification (unless one explicitly wants to avoid this).
- Models can be displayed graphically if the `Rgraphviz` package is installed.
- Facilities for reading data in various formats are available.

The addendum is organised differently from (Højsgaard 2004) but covers otherwise the same material.

1 Introduction and background

The `mimR` package is a package which provides facilities for graphical modelling in the statistical program `R` (R Development Core Team 2006). `mimR` is part of the gR–initiative (Lauritzen 2002) which aims to make graphical models available in `R`.

The statistical background for `mimR` is (M)ixed (I)nteraction (M)odels which is a general class of statistical models for mixed, discrete and continuous variables, where focus is on modelling conditional independence restrictions.

Statistical inference in mixed interaction models can be made with the program `MIM`, (Edwards 2000). The core of `mimR` is an interface from `R` to `MIM`.

This paper does not describe the statistical theory; instead the reader is referred to Edwards (2000). For a comprehensive account of graphical models we refer to Lauritzen (1996). Other important references are Edwards (1990) and Lauritzen and Wermuth (1989).

2 Preliminaries

2.1 Availability, information and installation

The mimR package uses the MIM program as inference engine. MIM is only available on Windows platforms and hence so is mimR. The MIM program itself (available from http://www.hypergraph.dk) must be installed on the computer. The communication between R and MIM is based on the rcom package which is automatically installed when mimR is installed. The mimR package has a homepage, http://gbi.agrsci.dk/~sorenh/mimR.

In addition to the documentation in the mimR package, the MIM program itself contains a comprehensive help function which the user of mimR is encouraged to make use of. To access the help function in MIM either type helpmim() in R or switch to the MIM program window and press F1.

2.2 Limitations

The maximum number of variables in models in mimR is 52. This is because the internal representation of variables in MIM is as letters (MIM is case sensitive in this respect).

2.3 Known problems

MIM is automatically started by mimR if MIM is not already running. Sometimes (but not always) this causes a window to pop up with a text like "Access violation at address 00541FDD in module 'mim3206.exe'. Read of address 00EAE238." We do not know why this happens, but the problem can be avoided by simply starting up MIM manually before invoking mimR.

When a dataframe is sent to MIM this is done by writing a file in the tmpdir of the current R session. This file is afterwards read into MIM. (This turns out to be the fastest way of getting larger amounts of data from R to MIM). MIM can not read such files if the tmpdir contains a hyphen ("-"). For example, if the tmpdir is c:/my-tmp-dir/ then mimR will not work.

3 Specifying and displaying models

In this section we show how to specify and display models in mimR for data arranged in a dataframe (where each row represent a case) or in a table as cumulated counts (for discrete variables). It is also possible to work with data arranged in other forms. Details are given in Section 9.

3.1 Discrete models

The discrete models are hierarchical log–linear models for contingency tables. For example, the contingency table HairEyeColor (which comes with R) contains a cross classification of persons with respect to gender, hair colour and eye colour:

```
> HairEyeColor
```

```
, , Sex = Male

      Eye
```

```
Hair    Brown Blue Hazel Green
  Black    32   11    10     3
  Brown    38   50    25    15
  Red      10   10     7     7
  Blond     3   30     5     8

, , Sex = Female

       Eye
Hair    Brown Blue Hazel Green
  Black    36    9     5     2
  Brown    81   34    29    14
  Red      16    7     7     7
  Blond     4   64     5     8
```

The model with generating class `"Eye:Hair+Sex"` satisfies that $(Eye, Hair)$ are independent of Sex and is specified with:

```
> hec1 <- mim("Eye:Hair+Sex//", data = HairEyeColor)
> hec1
```

```
Formula: Eye:Hair+Sex//
-2logL: 3643.191 DF: 15
```

If the `Rgraphviz` package is installed, the model can be displayed graphically as in Figure 1 by:

```
> display(hec1)
```

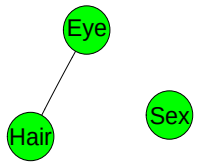

Figure 1: A graphical (log–linear) model for discrete data.

3.2 Continuous models

The following data set (taken from Mardia *et al.* (1979), see also Edwards (2000)) contains the examination marks for 88 students in 5 different subjects. Data is contained the data set `math`. A stepwise backward model selection yields the "butterfly" model shown in Figure 2 see also Whittaker (1990), p. 4.

This model can be specified as

```
> data(math)
> math2 <- mim("//me:ve:al+al:an:st", data = math)
```

3.3 Mixed models

Mixed models, or conditional Gaussian models (CG–models), arise by combining log–linear models and graphical Gaussian models. The `rats` dataset is from a

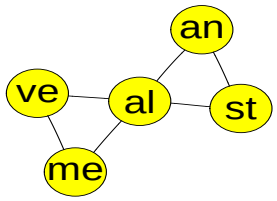

Figure 2: The selected graphical Gaussian "butterfly" model for the mathmarks data.

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. See Edwards (2000) for more details. The first rows of the data are:

```
> data(rats)
> rats[1:5, ]
```

```
  Sex Drug W1 W2
1   M   D1  5  6
2   M   D1  7  6
3   M   D1  9  9
4   M   D1  5  4
5   M   D2  9 12
```

For example, the model in Figure 3 is obtained with

```
> m1 <- mim("Sex:Drug/Sex:Drug:W2 + Drug:W1/W1:W2", data = rats)
> m1
```

```
Formula: Sex:Drug/Sex:Drug:W2 + Drug:W1/W1:W2
-2logL: 273.89 DF: 18
```

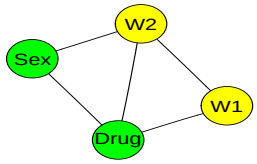

Figure 3: The model with generating class "Sex:Drug/Sex:Drug:W2 + Drug:W1/W1:W2"

4 Models in mimR

Only undirected models are available in `mimR`. That is, models in which all variables are treated on equal footing as response variables. Models where a possible response structure has to be accounted for can not be dealt with in `mimR`.

An undirected model is created using the `mim` function (which returns a `mim` object). Default is that the model is fitted to data, but fitting can be avoided by setting `fit=FALSE`. To explicitly fit a model, use the `fit()` function which is described in Section 11.

4.1 Model formulae

The general form of a model formula in `mimR` is

$$d_1 + d_2 + \cdots + d_r / l_1 + l_2 + \cdots + l_s / q_1 + q_2 + \cdots + q_t$$

where d_j, l_j and q_j are the respectively discrete, linear and quadratic generators.

A formula in `mimR` must be given as a string, i.e. in quotes (`"..."`). It is not possible to specify models using the conventional `R` syntax, i.e. with `~...`. The engine for specifying and fitting models is the `mim` function.

For example:

```
> mRats <- mim("Sex:Drug/Sex:Drug:W1+Sex:Drug:W2/W1:W2", data = rats)
```

4.2 Specification of special models

It is possible to specify certain specific models (possibly for only a subset of the variables) in short form. These are 1) the main effects model (as `"."`), 2) the saturated model (as `".."`) and 3) the homogeneous saturated model as (as `"..h"`). For example:

```
> mim(".", data = rats, marginal = c("Sex", "Drug", "W1"))
> mim("..", data = rats, marginal = c("Sex", "Drug", "W1"))
> mim("..h", data = rats, marginal = c("Sex", "Drug", "W1"))
```

4.3 Model summary and model properties

A summary and a description of certain model properties of a `mim` model can be achieved using the `summary()` and `properties()` functions:

```
> summary(mRats)
```

```
Formula: Sex:Drug/Sex:Drug:W1+Sex:Drug:W2/W1:W2
Variables in model  :  Sex Drug W1 W2
deviance: 27.807 DF: 15 likelihood: 273.705
```

Some properties of the model can be obtained with:

```
> properties(mRats)
```

```
Model properties:
 Variables in model  :  Sex Drug W1 W2
 Cliques: [1] "Sex:Drug:W1:W2"
 Is graphical        :  TRUE   Is decomposable: TRUE
 Is mean linear      :  TRUE   Is homogeneous : TRUE   Is delta-collapsible:  TRUE
```

The model summary reads as follows: 1) The model is fitted to data. 2) The model is graphical (such that there is a 1–1 correspondence between the model and its interaction graph). 3) The model is decomposable meaning that the maximum likelihood estimate exists in closed form (i.e. no iteration is needed). 4) The model is mean linear meaning that the regressions of each continuous variable on the discrete variables all have the same structural form. 5) The model is homogeneous meaning that the variance of the continuous variables does not vary with the levels of the discrete variables. 6) Finally, the model is Δ–collapsible which means that the model can be collapsed onto the discrete variables.

A more general function is `modelInfo()` which provides various model information as a list. The function can be given an additional argument to take out a specific slot in the list. For example, to take out the linear generators do:

```
> modelInfo(mRats, "mimGamma")
```

```
[1] "W1" "W2"
```

4.4 Fitted values (parameter estimates)

The fitted values (parameters estimates) can be obtained using the `fitted()` function:

```
> fitted(mRats)
```

```
  Drug Sex Freq    W1    W2 W1:W1 W1:W2 W2:W1 W2:W2
1    1   1    4  7.50  8.25 3.938 3.187 3.187  4.75
2    2   1    4  7.75  8.75 3.938 3.187 3.187  4.75
3    3   1    4 13.50  8.50 3.938 3.187 3.187  4.75
4    1   2    4  6.50  6.25 3.938 3.187 3.187  4.75
5    2   2    4  7.25  8.25 3.938 3.187 3.187  4.75
6    3   2    4 16.00 12.00 3.938 3.187 3.187  4.75
```

The data frame contains for each configuration of the discrete variables 1) the number of cases with that configuration and 2) the estimated mean vector and covariance matrix.

5 Model editing

Models can be edited using the `editmim()` function by which one can 1) delete edges, 2) add edges, 3) homogeneously add edges, 4) delete terms (interactions) and 5) add terms. We refer to Edwards (2000) for the precise definitions of these terms. It should be noted that operations are conducted in the order specified above. For example:

```
> m1 <- mim(".", data = rats)
> m2 <- editmim(m1, addEdge = c("Sex:Drug", "Sex:W2"))
```

Some properties of this model are

```
> properties(m2)
```

```
Model properties:
 Variables in model  :  Sex Drug W1 W2
 Cliques: [1] "Sex:Drug" "Sex:W2"   "W1"
 Is graphical        :  TRUE   Is decomposable: TRUE
 Is mean linear      :  TRUE   Is homogeneous : FALSE   Is delta-collapsible:  TRUE
```

The model specified this way is heterogeneous because the variance of `W2` depends on `Sex`). To add homogeneous terms, the `haddEdge` keyword can be used as in:

```
> m3 <- editmim(m1, addEdge = "Sex:Drug", haddEdge = "Drug:W1:W2")
> properties(m3)
```

```
Model properties:
 Variables in model  :  Sex Drug W2 W1
 Cliques: [1] "Sex:Drug"   "Drug:W1:W2"
 Is graphical        :  TRUE   Is decomposable: TRUE
 Is mean linear      :  TRUE   Is homogeneous : TRUE   Is delta-collapsible:  TRUE
```

Note the difference between deleting edges and terms:

```
> h1 <- mim("..", data = HairEyeColor)
> editmim(h1, deleteEdge = "Hair:Eye:Sex")
```

```
Formula: Sex + Eye + Hair//
-2logL: 3789.635 DF: 24
```

```
> editmim(h1, deleteTerm = "Hair:Eye:Sex")
```

```
Formula: Eye:Sex + Hair:Sex + Hair:Eye//
-2logL: 3622.028 DF: 9
```

Note that if the starting model is (un)fitted, then so are all subsequent models derived using the `editmim()` function. To explictly fit a model, use the `fit()` function, see Section 11.

6 Testing for deletion of an edge

Consider again the saturated model for the `HairEyeColor` data:

```
> h1 <- mim("Hair:Eye:Sex//", data = HairEyeColor)
```

We can test for deletion of edges from the model using the `testdelete()` function:

```
> testdelete("Hair:Eye", h1)
```

```
test: Chi-squared method: asymptotic
stat: 162.208 df: 18 P: 0
```

```
> testdelete("Hair:Sex", h1)
```

```
test: Chi-squared method: asymptotic
stat: 22.032 df: 12 P: 0.037
```

The `testdelete()` function also applies in a natural way if the model is hierarchical, for example with the all two–factor model:

```
> h2 <- mim("Hair:Eye+Hair:Sex+Eye:Sex//", data = HairEyeColor)
> testdelete("Hair:Eye", h2)
```

```
test: Chi-squared method: asymptotic
stat: 154.021 df: 9 P: 0
```

```
> testdelete("Hair:Sex", h2)
```

```
test: Chi-squared method: asymptotic
stat: 13.845 df: 3 P: 0.003
```

Rather than applying the asymptotic likelihood ratio test we may calculate Monte Carlo p–values with:

```
> testdelete("Hair:Sex", h2, arg = "m")
```

```
test: Chi-squared method: asymptotic
stat: 13.845 df: 3 P: 0.003
```

Additional examples on the use of `testdelete()` are given in Section 10.

7 Model comparison

Consider the models

```
> h1 <- mim("Hair:Eye:Sex//", data = HairEyeColor)
> h2 <- mim("Hair:Eye+Sex//", data = HairEyeColor)
```

Model `h2` can be tested under `h1` with the `modelTest` function:

```
> modelTest(h2, h1)
```

```
Test of H0 :  Hair:Eye:Sex//
Against    :  Hair:Eye+Sex//

test: Chi-squared method: asymptotic
stat: 29.35 df: 15 P: 0.014
```

8 Model selection

The `stepwise()` function performs stepwise model selection. This function takes as additional arguments all arguments that the `STEPWISE` command in `MIM` does. The `stepwise()` function returns a new `mim` object.

We consider the pig carcass data `carcass` and start with the independence model:

```
> data(carcass)
> mainCarc <- mim(".", data = carcass)
```

A forward stepwise selection using significance testing as selection criterion with 0.001 as critical level is obtained with:

```
> carcForw <- stepwise(mainCarc, arg = "f", critlevel = 0.001)
```

The resulting model

```
> carcForw
```

```
Formula: //F11:F12:F13 + F11:F12:LMP + F11:M12:M13 + F11:M13:LMP + M11:M12:M13
-2logL: 11438.84 DF: 10
```

is shown in Figure 4.

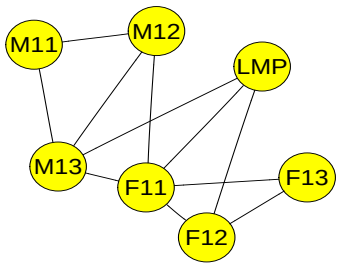

Figure 4: The covariance selection model obtained after a forward selection for the carcass data.

Alternatively we can make a backward stepwise selection using BIC as selection criterion, make an unrestricted search (as opposed to searching among decomposable models, which is the default) and make a non–coherent search (which means that the same edge can be tested several times during the models search):

```
> satCarc <- mim("..", data = carcass)
> carcBack <- stepwise(satCarc, arg = "snu", critlevel = 0.001)
```

The resulting model is:

```
> carcBack
```

```
Formula: //F11:F12:M12:M13 + F11:F12:F13:M13 + F11:F12:F13:LMP + M11:M12:M13 + M11:F13:M13 + M11:F13:LMP
-2logL: 11375.99 DF: 5
```

9 Graphical meta data – gmData

The internal representation of data in mimR is by gmData which is short for "graphical meta data". A gmData object contains information about variables, their labels, their levels (for discrete variables) etc. A gmData object will typically also contain data, but need not do so. The idea behind separating the specification of the variables from data is that some properties of a model, for example decomposability and collapsibility, can be investigated without any reference to data.

Data represented as a dataframe or table (as in Section 3) are automatically converted to gmData in the mim function. Therefore we can (as we have done above) simply specify data to the mim function directly as a dataframe or a table.

Data in certain other can also be used in mimR. However, for such data, one needs to create a gmData object. The most typical cases are described below; additional options are given in Section A.

The generic function for creating gmData objects is the as.gmData function.

9.1 Making a gmData object from a dataframe or a table

To create a gmData object with from a dataframe do:

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

```
  name letter factor levels
1  Sex      a   TRUE      2
2 Drug      b   TRUE      3
3   W1      c  FALSE     NA
4   W2      d  FALSE     NA
Data origin :      data.frame
```

To each variable, there is associated a letter. This letter is used in connection with the internal representation of models and variables in MIM and the user should not be concerned with this. The procedure is the same for data arranged in a table. Observations in their original form can be extracted with the observations function. To extract the first 5 rows of data do:

```
> observations(gmdRats)[1:5, ]
```

```
  Sex Drug W1 W2
1   M   D1  5  6
2   M   D1  7  6
3   M   D1  9  9
4   M   D1  5  4
5   M   D2  9 12
```

To see the labels of the discrete variables, do:

```
> vallabels(gmdRats)
```

```
$Sex
[1] "F" "M"

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

9.2 Creating a gmData object without data

A gmData object (without data) can be created by the gmData() function:

```
> gmData(c("Sex", "Drug", "W1", "W2"), factor = c(2, 3, FALSE,
     FALSE), vallabels = list(Sex = c("M", "F"), Drug = c("D1",
     "D2", "D3")))
```

If no vallabels are given, default values are imposed. With such a specification, one can afterwards specify models and have mimR to find important properties of these models, e.g. whether a given model is decomposable.

9.3 Discrete data arranged as cumulated cell counts in dataframe

Sometimes discrete data are arranged as cumulated cell counts, for example

```
> library(MASS)
> housing[1:5, ]
```

```
     Sat   Infl  Type Cont Freq
1    Low    Low Tower  Low   21
2 Medium    Low Tower  Low   21
3   High    Low Tower  Low   28
4    Low Medium Tower  Low   34
5 Medium Medium Tower  Low   22
```

Here Freq contains the counts. To use these data in mimR, first turn the dataframe into a table, and then turn the table into a gmData object, i.e.

```
> housingTab <- xtabs(Freq ~ Sat + Infl + Type + Cont, data = housing)
> ht <- as.gmData(housingTab)
> ht
```

10 Models with ordinal variables

Consider the housing data (represented as the gmData object ht in Section 9). The variables Sat and Infl are ordinal. This is declared as:

```
> ordinal(ht) <- c("Sat", "Infl")
> ht
```

```
  name letter factor levels
1  Sat      a   TRUE      3
2 Infl      b   TRUE      3
3 Type      c   TRUE      4
4 Cont      d   TRUE      2
Ordinal     :      Sat Infl
Data origin :      table
```

Declaring variables to be ordinal has an impact on the tests for edge removal/addition if the option w is given. For example we can test the significance of all edges in the saturated model with:

```
> msat <- mim("Sat:Infl:Cont:Type//", data = ht)
> stepwise(msat, arg = "ow")
```

```
Coherent Backward Single-step Selection.
Fixed edges: none.
Critical value:   0.0500
Decomposable mode, Chi-squared tests.
DFs adjusted for sparsity.
Model: abcd
Deviance:   0.0000 DF:   0 P:  1.0000
    Edge        Test
Excluded   Statistic DF          P
    [ab]  91192.5000 32      0.0000 +
    [ca]     87.3486 18      0.0000 +
    [da]  60178.0000 24      0.0000 +
    [cb]     23.0770 18      0.1876
    [db]  70812.0000 24      0.0000 +
    [cd]     64.3488 27      0.0001 +
Formula: Sat:Infl:Type:Cont//
-2logL: 13544.49 DF: 0
```

196 Compare this with the results when factors are not declared as being ordinal:

```
> ht2 <- ht
> nominal(ht2) <- c("Sat", "Infl")
> msat2 <- mim("Sat:Infl:Cont:Type//", data = ht2)
> stepwise(msat2, arg = "ow")
```

```
Coherent Backward Single-step Selection.
Fixed edges: none.
Critical value:   0.0500
Decomposable mode, Chi-squared tests.
DFs adjusted for sparsity.
Model: abcd
Deviance:   0.0000 DF:   0 P:  1.0000
    Edge        Test
Excluded   Statistic DF          P
    [ab]    135.6898 32      0.0000 +
    [ac]     99.0937 36      0.0000 +
    [ad]     32.8715 24      0.1068
    [bc]     43.7552 36      0.1754
    [bd]     33.7206 24      0.0898
    [cd]     64.3488 27      0.0001 +
Formula: Sat:Infl:Type:Cont//
-2logL: 13544.49 DF: 0
```

197 When one or more factors are declared as ordinal, different tests are available
198 for testing for edge deletion:

```
> testdelete("Sat:Infl", msat)
```

```
test: Chi-squared method: asymptotic
stat: 135.69 df: 32 P: 0
```

```
> testdelete("Sat:Infl", msat, arg = "k")
```

```
test: Kruskal-Wallis method: asymptotic
stat: 112.919 df: 16 P: 0
```

11 Model fitting

11.1 Direct maximum likelihood estimation

The function for fitting models via direct maximum likelihood estimation is `fit`:

```
> m1 <- mim("..", data = rats, marginal = c("Sex", "Drug", "W1"),
     fit = FALSE)
> fit(m1)
```

```
Formula: Sex:Drug/Sex:Drug:W1/Sex:Drug:W1
-2logL: 178.873 DF: 0
```

11.2 EM algorithm

For data given as a dataframe, the EM algorithm (Dempster *et al.* 1977) is available to handle incomplete observations. For example

```
> r2 <- rats
> r2[1:2, 3] <- r2[3:4, 4] <- NA
> r2[1:5, ]
```

```
  Sex Drug W1 W2
1   M   D1 NA  6
2   M   D1 NA  6
3   M   D1  9 NA
4   M   D1  5 NA
5   M   D2  9 12
```

The EM algorithm is switched on by `fit="e"`:

```
> mim("..", data = r2, fit = "e")
```

```
Formula: Sex:Drug/Sex:Drug:W1 + Sex:Drug:W2/Sex:Drug:W1:W2
-2logL: 168.46 DF: 0
```

If the argument `fit="e"` is not given, then `fit` will try to use the EM algorithm if direct maximum likelihood estimation fails:

```
> m2 <- mim("..", data = r2)
```

```
Seems that there are incomplete observations - trying EMfit
```

The EM algorithm starts by substititing random starting values for missing data.

12 Latent variables

12.1 Fitting a model with a discrete latent variable

First we consider a latent variable model: We suppose that there is a latent binary variable `A` such that the manifest variables are all conditionally independent given `A`.

First we add a binary factor `A` (with missing values) to the `math` dataset:

```
> data(math)
> math$A <- factor(NA, levels = 1:2)
> gmdMath <- as.gmData(math)
```

Next, we make explicit in the gmData object that A is indeed a latent variable using the latent() function (in Section 12.2 it is explained why it must be specified explicitely that A is a latent variable):

```
> latent(gmdMath) <- "A"
> gmdMath
```

```
  name letter factor levels
1   me      a  FALSE     NA
2   ve      b  FALSE     NA
3   al      c  FALSE     NA
4   an      d  FALSE     NA
5   st      e  FALSE     NA
6    A      f   TRUE      2
Data origin :      data.frame
Latent variables: A
```

The model can be specified as

```
> m1 <- mim("A/st:A+an:A+al:A+ve:A+me:A/st:A+an:A+al:A+ve:A+me:A",
     data = gmdMath)
```

```
Model has latent variable - trying EM algorithm
```

The model is shown in Figure 5.

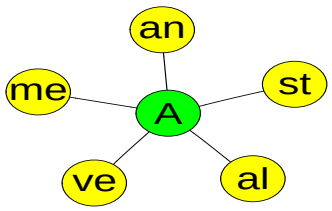

Figure 5: Latent variable model for math data.

Predicted values for the latent variable under the model can be imputed in MIM using

```
> imputeMissing()
```

To get the data (including the imputed values) from MIM to R do:

```
> d.imp <- retrieveData()
> d.imp[1:5, ]
```

```
  me ve al an st A
1 77 82 67 67 81 2
2 63 78 80 70 81 2
3 75 73 71 66 81 2
4 55 72 63 70 68 2
5 63 63 65 70 63 2
```

and so we see that the first 5 cases are assignes `A` to have level 1.

Next, we plot the predicted value of `A` against the observation number:

```
> plot(as.numeric(d.imp$A))
```

The plot is shown in Figure 6. The grouping of the values of `A` suggests that data have been processed somehow prior to presentation. (Edwards 2000), p. 181, conclude: "Certainly they (the data) have been mistreated in some way, doubtless by a statistician."

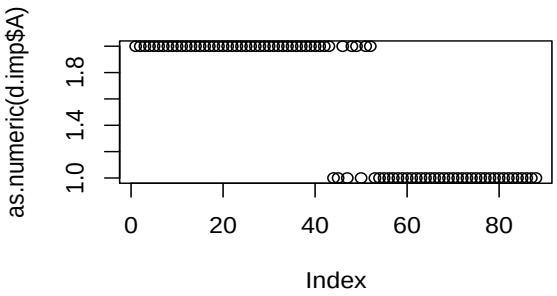

Figure 6: An index plot of the discrete latent variable `A`.

12.2 Controlling the EM algorithm

The EM algorithm needs a set of initial values for the unobserved values to start from when calculating the parameter estimates in the first iteration. The final estimate of the EM algorithm may depend on the initial values and that (especially in the case of latent variables) the likelihood may have multiple maxima. Default is that random starting values are imputed and that was actually the case above, where the factor `A` was given `NA` values.

An alternative is to specify starting values for the latent variables in the dataframe, e.g. as

```
> data(math)
> math$A <- factor(1:2, levels = 1:2)
> latent(gmdMath) <- "A"
> m1 <- mim("A/st:A+an:A+al:A+ve:A+me:A/st:A+an:A+al:A+ve:A+me:A",
     data = gmdMath, fit = "es")
> m1
```

```
Formula: A/st:A+an:A+al:A+ve:A+me:A/st:A+an:A+al:A+ve:A+me:A
Latent variables in model: A
-2logL: 3534.869 DF: 20
```

The specification `fit='es'` means that the model should be fitted with the EM algorithm and that the given values of the latent variables should be used as starting values for the EM algorithm. Setting `fit='er'` means that random starting values will be used for the EM algorithm.

For this reason latent variables must be declared explicitely in a gmData object. By this approach the sensitivity of the EM algorithm on starting values can be investigated.

12.3 Fitting a model with a continuous latent variable

To illustrate controlling of the EM algorithm, we make an alternative analysis, where A is regarded as a continuous variable. To speed up the convergence of the EM algorithm, we do a factor analysis to get good starting values:

```
> data(math)
> fa <- factanal(math, factors = 1, scores = "regression")
> math$A <- fa$scores
```

Then we create a gmData object with this new augmented data set and declares that A is to be regarded as a latent variable:

```
> gmdMath <- as.gmData(math)
> latent(gmdMath) <- "A"
> m1 <- mim("//st:A+an:A+al:A+ve:A+me:A", data = gmdMath)
```

```
Model has latent variable - trying EM algorithm
```

As before we impute the missing values, retrieve the data to R and plot the imputed values for the latent variable:

```
> imputeMissing()
> d.imp <- retrieveData()
> plot(d.imp$A)
```

The plot of the imputed values for the latent variables are shown in Figure 7 and this also suggests that the data do not emerge in random order.

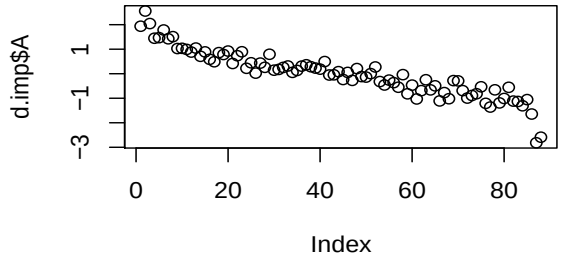

Figure 7: An index plot of the continuous latent variable A.

13 Discussion

In this manual we have illustrated some aspects of the `mimR` package for graphical modelling in `R`. It is the hope that `mimR` will be obsolete in a not too distant future – not because of lack of relevance of being able to work with graphical models in `R`. Rather, it is the hope that a more proper package with with at least the functionality of `mimR` will be created. That is one of the aims of the gR–project, which has lead to the minimal package `gRbase`, (Dethlefsen and Højsgaard 2005), which is available on CRAN. The fucntionality of `gRbase` is however very limited and as such `mimR` is a relevant package to use for graphical modelling in `R`.

14 Acknowledgements

David Edwards (the creator of `MIM`) is greatly acknowledged for his support in the creation of `mimR`. Claus Dethlefsen has made valuable comments to this addendum. The members of the gR initiative are acknowledged for their inspiration.

A Additional ways of getting data into `mimR`

A.1 Creating gmData from sufficient statistics

For mixed interaction models, 1) a list of cell counts for the discrete variables, 2) a mean vector for the continuous variables for each cell, and 3) and a covariance matrix for each cell are a set of sufficient statistics. Data represented in this form (as moment statistics) can be used in `mimR` as will be illustrated below.

A.1.1 Mixed data

For mixed data there are two options, both to be illustrated for the `rats` data.

Option 1 Specify a list with as many elements as there are cells in the table. Each element of the list must consist of three items: 1) The covariance matrix, 2) the mean vector, and 3) the number of observations in the cell (in that order). The covariances must be the estimate obtained by dividing the sum of products of residuals by the number of observations n per group, not $n - 1$.

For the `rats` data we can extract first splitting data by the levels of the discrete variables using the `doBy` package, (Højsgaard 2006):

```
> r <- splitBy(~Sex + Drug, data = rats)
```

The necessary list can be obtained by:

```
> cmc <- lapply(r, function(x) cov.wt(x[, c("W1", "W2")], method = "ML"))
```

```
> x <- momentstats(factor = c("Sex", "Drug"), level = c(2, 3),
     continuous = c("W1", "W2"), cmc = cmc)
> as.gmData(x)
```

```
  name letter factor levels
1  Sex      a   TRUE      2
2 Drug      b   TRUE      3
3   W1      c  FALSE     NA
4   W2      d  FALSE     NA
Data origin :      momentstats
```

Option 2 Specify 1) a list of covariances matrices, 2) a list of mean vectors, and 3) a list of cell counts:

```
> covmats <- lapply(r, function(x) cov.wt(x[, c("W1", "W2")], method = "ML")$cov)
> meanvecs <- lapply(r, function(x) mean(x[, c("W1", "W2")]))
> counts <- lapply(r, function(x) nrow(x))
> x <- momentstats(factor = c("Sex", "Drug"), level = c(2, 3),
     continuous = c("W1", "W2"), covariances = covmats, means = meanvecs,
     counts = counts)
> as.gmData(x)
```

```
  name letter factor levels
1  Sex      a   TRUE      2
2 Drug      b   TRUE      3
3   W1      c  FALSE     NA
4   W2      d  FALSE     NA
Data origin :      momentstats
```

It is wise to check that data have been entered correctly by:

```
> toMIM(x)
> mim.cmd("print s")
```

A.1.2 Continuous data

For continuous data the same two options as for mixed data are available. For example for the `math` data we can do:

```
> cmc <- cov.wt(math, method = "ML")
> x <- momentstats(continuous = names(math), cmc = cmc)
> as.gmData(x)
```

```
  name letter factor levels
1   me      a  FALSE     NA
2   ve      b  FALSE     NA
3   al      c  FALSE     NA
4   an      d  FALSE     NA
5   st      e  FALSE     NA
6    A      f  FALSE     NA
Data origin :      momentstats
```

or:

```
> x <- momentstats(continuous = names(math), counts = nrow(math),
     means = mean(math), covariances = cov.wt(math, method = "ML")$cov)
> as.gmData(x)
```

```
  name letter factor levels
1   me      a  FALSE     NA
2   ve      b  FALSE     NA
3   al      c  FALSE     NA
4   an      d  FALSE     NA
5   st      e  FALSE     NA
6    A      f  FALSE     NA
Data origin :      momentstats
```

A.1.3 Discrete data

Schoener (1968) describes data concerning the perching behaviour of two species of lizards, see also Edwards (2000). Data is a three–way contingency. Data, represented as a list of counts, can be turned into a gmData object with:

```
> x <- momentstats(factor = c("species", "diameter", "height"),
     level = c(2, 2, 2), counts = c(32, 86, 11, 35, 61, 73, 41,
         71), vallabels = list(species = c("anoli", "disticus"),
         diameter = c("<=4", ">4"), height = c(">4.75", "<=4.75")))
> z <- as.gmData(x)
> vallabels(z)
```

```
$species
[1] "anoli"    "disticus"

$diameter
[1] "<=4" ">4"

$height
[1] ">4.75"  "<=4.75"
```

The order of the cells are $(1,1,1),(1,1,2),(1,2,1),(1,2,2),\ldots,(2,2,1),(2,2,2)$, i.e. the last index varies fastest.

B Low level access to MIM from R

B.1 Primitive use of MIM from R – the mim.cmd() function

The core of mimR is the mim.cmd function. The arguments to mim.cmd are simply MIM commands (given as strings). For example:

```
>mim.cmd("fact a2 b2; statread ab; 25 2 17 8 !")
>mim.cmd("mod a,b; fit; print; print f")
```

The mim.cmd function returns the result of the commands submitted to MIM. The result of the last call of mim.cmd above is:

```
Deviance:       5.3111 DF: 1
The current model is: a,b.
Fitted counts, means and covariances.
 a b   Count
 1 1  21.808
 1 2   5.192
 2 1  20.192
 2 2   4.808
```

B.2 Using MIM directly from mimR– the mcm() function

The mcm function (short for "MIM command mode") provides a direct interface to MIM, i.e. the possibility to write MIM commands directly. The mcm function returns no value to R, and is intended only as an easy way to submit MIM commands without the overhead of wrapping them into the mim.cmd function (or submitting the commands directly to MIM). Hence, using mcm, the session above would be:

```
> mcm()
Enter MIM commands here. Type quit to return to R
MIM->fact a2 b2; statread ab
MIM->25 2 17 8 !
Reading completed.
MIM->mod a,b; fit
Deviance:       5.3111 DF: 1
MIM->print; print f
The current model is: a,b.
Fitted counts, means and covariances.
 a b   Count
 1 1  21.808
 1 2   5.192
 2 1  20.192
 2 2   4.808
MIM->quit
>
```

To return to R from the mcm function type 'quit', 'exit', 'end', 'q' or 'e' (i.e. the commands one would use to terminate MIM). These commands, however, do not terminate MIM – they only return control to R.

References

Dempster, A. P., Laird, N., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm (with discussion). *Journal of the Royal Statistical Society, Series B*, **39**, 1–38.

Dethlefsen, C. and Højsgaard, S. (2005). A common platform for graphical models in r: The grbase package. *Journal of Statistical Software*, **14**, 1–12.

Edwards, D. (1990). Hierarchical interaction models. *Journal of the Royal Statistical Society, Series B*, **52**, (1), 3–20.

Edwards, D. (2000). *Introduction to graphical modelling*, (2nd edition edn). Springer Verlag, New York.

Højsgaard, S. (2004). The mimR package for graphical modelling in R. *Journal of Statistical Software*, **11**, (6).

Højsgaard, S. (2006). *doBy: Groupwise computatations*. R package version 0.7.

Lauritzen, S. L. (1996). *Graphical models*. Oxford University Press.

Lauritzen, S. L. (2002). gRaphical models in R: A new initiative within the R project. *Rnews*, **2**, 39.

Lauritzen, S. L. and Wermuth, N. (1989). Graphical models for associations between variables, some of which are qualitative and some quantitative. *Annals of Statistics*, **17**, 31–57.

Mardia, K. V., Kent, J. T., and Bibby, J. M. (1979). *Multivariate analysis*. Academic Press.

R Development Core Team (2006). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-00-3.

Schoener, T. W. (1968). The anolis lizards of bimini: Resource partitioning in a complex fauna. *Ecology*, **49**, 704–26.

Whittaker, J. (1990). *Graphical models in applied multivariate statistics*. Wiley.