Users Guide to the **mimR** package for Graphical Modelling in **R**

Søren Højsgaard

May 12, 2005

Contents

1 Introduction and background

The **mimR**[1] package provides facilities for graphical modelling in the statistical program **R** (R Development Core Team, 2004) and is a part of the gR–project[2] which is a project to make graphical models available in **R**. **mimR** is available as an **R**–package on the Comprehensive R Archive Network (CRAN)[3].

The statistical foundation for **mimR** is Mixed Interaction Models, a very general class of statistical models for mixed discrete and continuous variables. Statistical inference in mixed interaction models can be made with the stand–alone program **MIM**[4], and the core of **mimR** is an interface from **R** to the **MIM** program. Edwards (2000) describes the models and the **MIM** program in a very clear way. For a comprehensive account of graphical models we refer to Lauritzen (1996). Other important references are Edwards (1990) and Lauritzen and Wermuth (1989). In this paper, the statistical theory behind mixed interaction models will not be outlined; instead the reader is referred to Edwards (2000).

2 Preliminaries

2.1 Availability:

mimR is available only on Windows platforms because **MIM** only runs on Windows platforms.

[1] `http://genetics.agrsci.dk/~sorenh/mimR`
[2] `http://www.R-project.org/gR`
[3] `http://www.R-project.org`
[4] `http://www.hypergraph.dk`

2.2 MIM as inference engine:

From the users perspective, the **MIM** stand–alone program can be regarded as an "inference engine" which the user (at least in principle) needs not be concerned with. However, in practice it is worth keeping in mind that **MIM** is indeed a stand–alone program rather than integrated with **mimR**: In **MIM** there can at any time be only 1) one specification of a data set, 2) one data set and 3) one model. This means that when fitting a new model, a lot of information may have to be conveyed to **MIM**, and this may take some time (if the data set is large). Likewise, returning fitted values etc. from **MIM** to **R** may take some time too.

2.3 Installation:

Because **mimR** uses **MIM** as inference engine, the **MIM** program must be installed on the computer. The communication between **R** and **MIM** is based on the `RDCOMClient` package, which will be automatically installed when **mimR** is installed. After completing these installation steps, **mimR** is loaded by the command:

```
> library(mimR)
```

2.4 Known problems

When **mimR** invokes **MIM** it sometimes happens that a window with the text `Access violation at address...` pops up. We have not yet been able to track down the source of this error message, but it turns out that **mimR** (and **MIM**) work despite this message. One can often avoid this problem by starting **MIM** manually before starting to use **mimR**.

2.5 Getting help:

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()
```

or go to the **MIM** program and press F1.

3 The `rats` dataset

Some features of **mimR** will be illustrated in the present paper on the basis of the `rats` dataset in the **mimR** 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. See Edwards (2000) for more details. To load the data, type

```
> data(rats)
```

The first 8 rows of the data are:

```
> rats[1:8, ]
```

```
  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
6   M   D2  7  7
7   M   D2  7  6
8   M   D2  6  8
```

4 Objects in mimR

The core of **mimR** are the `gmData` and `mim` objects. Informally, a `gmData` object can be regarded as the **mimR** analogue to a dataframe in **R**, whereas a `mim` object can be regarded as the **mimR** analogue to model objects in **R**, e.g. `lm` and `glm` objects. However, there are important exceptions to these analogies:

`gmData` objects A `gmData` object contains information about variables, their labels, their levels (for discrete variables) etc. A `gmData` object may also contain data, but need not do so. The name "`gmData`" is short for "graphical meta data". The idea behind separating the specification of the variables from data is that some properties of a model can be investigated without any reference to data, for example decomposability and collapsibility. See Section 5 for details.

`mim` objects A `mim` object links a model formula to a `gmData` object. Since a `gmData` object need not contain any data, fitting a `mim` model is separate process. (This is an important difference between models in **mimR** and e.g. the `lm()` function **R**, which also fits the model to data.) When the model has been fitted (provided that there are data in the `gmData` object), the `mim` object also contains the fitted values, parameter estimates etc. See Section 6 for details.

5 gmData objects – graphical meta data

5.1 Making a gmData object from a data frame or a table

Typically one will create a gmData object (with data) from a data frame (or a table) as using the as.gmData() function as follows:

```
> data(rats)
> 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 see the values of the factors use the 'vallabels' function
To see the data use the 'observations' function

> data(HairEyeColor)
> gmdHec <- as.gmData(HairEyeColor)
> gmdHec

  name letter factor levels
1 Hair      a   TRUE      4
2  Eye      b   TRUE      4
3  Sex      c   TRUE      2
Data origin:      table
To see the values of the factors use the 'vallabels' function
To see the data use the 'observations' function
```

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. It is possible use the letters in specifying models but it is not recommended.

5.2 Creating a gmData from a list of sufficient statistics

For purely continuous and purely discrete data it is possible to read in the sufficient statistics directly. This is done by putting data in the form of a list. For continuous data, the format of the list is as follows:

```
> suffc <- list(means = apply(mathmark, 2, mean),
     stddev = apply(mathmark, 2, sd), corr = cor(mathmark),
     n = nrow(mathmark))
> as.gmData(suffc)

        name letter factor levels
1  mechanics      a  FALSE     NA
2    vectors      b  FALSE     NA
3    algebra      c  FALSE     NA
4   analysis      d  FALSE     NA
5 statistics      e  FALSE     NA
Data origin:      contSuffStats
To see the values of the factors use the 'vallabels' function
To see the data use the 'observations' function
```

For discrete data the format is:

```
> suffd <- list(names = c("foo", "bar"), levels = c(2,
     2), counts = c(1, 2, 3, 4), vallabels = list(foo = c("f1",
     "f2"), bar = c("b1", "b2")))
> as.gmData(suffd)

  name letter factor levels
1  foo      a   TRUE      2
2  bar      b   TRUE      2
Data origin:      discSuffStats
To see the values of the factors use the 'vallabels' function
To see the data use the 'observations' function
```

The slot `vallabels` can be omitted.

5.3 Creating a gmData object without data

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

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

  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:      table
To see the values of the factors use the 'vallabels' function
To see the data use the 'observations' function
```

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.

6 Models in mimR

Currently, only undirected models are available in **mimR**. That is, models in which all variables are treated on equal footing as response variables. (Thus 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):

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

Observe that models in **mimR** are specified as a string i.e. in quotes ("..."). It is NOT possible to specify models using the conventional **R** syntax, i.e. with `~....`

It is possible to specify 1) the main effects, 2) the saturated and 3) the homogeneous saturated models (possibly for only a subset of the variables) in short form:

```
> mainRats <- mim(".", data = gmdRats, marginal = c("Sex",
      "Drug", "W1"))
> satRats <- mim("..", data = gmdRats, marginal = c("Sex",
      "Drug", "W1"))
> hsatRats <- mim("..h", data = gmdRats, marginal = c("Sex",
      "Drug", "W1"))
```

7 Model fitting

Model fitting is separated from model specification so the models created above are not fitted to data. For model fitting two functions are available: `fit()` and `emfit()` (`emfit()` will be discussed in Section 12).

```
> mRatsFit <- fit(mRats)
```

```
Deviance:      27.8073 DF: 15
Printing format: 12,8
```

```
> mRatsFit
```

```
Formula: Sex:Drug/Sex:Drug:W1+Sex:Drug:W2/W1:W2
likelihood: 273.705 DF: 15
```

8 Fitted values (parameter estimates)

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

```
> fitted(mRatsFit)
```

```
  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. The function `modelInfo()` provides fitted values (along with other information) in a different form.

9 Model summary

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

```
> summary(mRatsFit)
```

```
Formula: Sex:Drug/Sex:Drug:W1+Sex:Drug:W2/W1:W2
Formula(letter): ab/abc,abd/cd
Variable type: mixed
deviance: 27.8073 DF: 15 likelihood: 273.705
 Cliques: [1] "Sex:Drug:W1:W2"
For model properties, use        : 'properties()'
For fitting information etc. use : 'modelInfo()'
```

The formula as letters show the internal representation of the models in **MIM**. The variable type being `mixed` shows that the model contains both discrete and continuous variables.

```
> properties(mRatsFit)

 Variables in model  :  Sex Drug W1 W2
 Is fitted           :  TRUE
 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.

10 Model editing and model selection

10.1 Editing models directly

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:

```
> mainRats <- mim(".", data = gmdRats)
> m2Rats <- editmim(mainRats, addEdge = c("Sex:Drug",
     "Sex:W2"))

Printing format: 12,8
```

The model specified this way is heterogeneous as the variance of `W2` depends on `Sex`. Some properties of this model are

```
> summary(m2Rats)
```

```
Formula: Sex:Drug/W1+Sex:W2/W1+Sex:W2
Formula(letter): ab/c,ad/c,ad
Variable type: mixed
 Cliques: [1] "Sex:Drug" "Sex:W2"   "W1"
For model properties, use        : 'properties()'
For fitting information etc. use : 'modelInfo()'
```

```
> properties(m2Rats)
```

```
 Variables in model  :  Sex Drug W1 W2
 Is fitted           :  FALSE
 Is graphical        :  TRUE   Is decomposable: TRUE
 Is mean linear      :  TRUE   Is homogeneous : FALSE
 Is delta-collapsible:  TRUE
```

We see that `m2Rats` is not homogeneous (because the variance of `W2` depends on `Sex`). To add homogeneous terms, the `haddEdge` keyword can be used as in

```
> m3Rats <- editmim(mainRats, addEdge = "Sex:Drug",
     haddEdge = "Drug:W1:W2")
```

```
Printing format: 12,8
```

```
> summary(m3Rats)
```

```
Formula: Sex:Drug/Drug:W2+Drug:W1/W1:W2
Formula(letter): ab/bd,bc/cd
Variable type: mixed
 Cliques: [1] "Sex:Drug"   "Drug:W1:W2"
For model properties, use        : 'properties()'
For fitting information etc. use : 'modelInfo()'
```

```
> properties(m3Rats)
```

```
 Variables in model  :  Sex Drug W2 W1
 Is fitted           :  FALSE
 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:

```
> msatHec <- mim("..", data = gmdHec)
> msatHec
```

```
Formula: Hair:Eye:Sex//

> editmim(msatHec, deleteEdge = "Hair:Eye:Sex")

Printing format: 12,8
Formula: Sex+Eye+Hair//

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

Printing format: 12,8
Formula: Eye:Sex+Hair:Sex+Hair:Eye//
```

10.2 Stepwise model selection

To a `mim` object the function `stepwise()` applies and this function takes as additional arguments all arguments that the `STEPWISE` command in **MIM** does. The `stepwise()` function returns a new `mim` object.

```
> data(carcass)
> gmdCarc <- as.gmData(carcass)
> mainCarc <- mim(".", data = gmdCarc)
> satCarc <- mim("..", data = gmdCarc)
> carcForw <- stepwise(mainCarc, arg = "f")
> carcBack <- stepwise(satCarc, arg = "s")
```

The `arg="f"` specifies forward selection (default is backward) and `arg="s"` requests exact tests. The selected models are:

```
> carcForw

Formula: //F11:F12:M12:F13+F11:F12:M12:M13+F11:F12:M13:LMP+M11:M12:M13
likelihood: 11405.13 DF: 7

> carcBack

Formula: //F11:M11:F12:M12:M13+F11:M11:F12:F13:LMP+F11:M11:F12:M13:LMP
likelihood: 11370.74 DF: 3
```

11 Missing values and/or latent variables

To fit a model with to incomplete data or to fit a latent variable model, the `emfit()` function can be used. This function be illustrated in connection with a latent variable model.

12 Example – Mathematics marks

This dataset (taken from Mardia, Kent, and Bibby (1979)) contains the examination marks for 88 students in 5 different subjects. Data is contained the data set `mathmark` in the **mimR** package. Edwards (2000) also investigates these data.

12.1 Selecting a graphical model

We start out by specifying the saturated model and do a backward elimination:

```
> data(mathmark)
> gmdMath <- as.gmData(mathmark)
> satMath <- mim("..", data = gmdMath)
> m2Math <- stepwise(satMath)

> m2Math

Formula: //mechanics:vectors:algebra+algebra:analysis:statistics
likelihood: 3391.021 DF: 4
```

The model `math2` is shown in Figure 1.

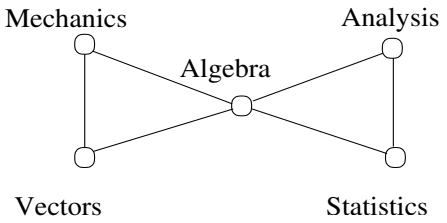

Figure 1: The "butterfly" model selected for the mathmarks data.

12.2 Fitting a model with a discrete latent variable

Next 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`. We fit such a model by:

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

With the specification above, `A` is a binary variable consisting of `NA` values. Next, we make explicit in the `gmData` object that `A` is indeed a latent variable using the `latent()` function:

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

In Section 12.4 it will become clear why it must be specified explicitely that `A` is indeed a latent variable.

One consequence of this last specification is that the model can not be fitted using the `fit()` function. Instead, the `emfit()` function which uses the EM algorithm Dempster, Laird, and Rubin (1977), must be used:

```
> satMath <- mim("..", data = gmdMath)
> mathNames <- names(mathmark)
> mathNames

[1] "mechanics"  "vectors"    "algebra"    "analysis"
[5] "statistics"

> delEdges <- paste(mathNames, collapse = ":")
> delEdges

[1] "mechanics:vectors:algebra:analysis:statistics"

> m2Math <- editmim(satMath, deleteEdge = delEdges)

Printing format: 12,8

> m2MathFit <- emfit(m2Math, plot = TRUE)

Printing format: 12,8
```

The argument `plot=TRUE` in `emfit()` creates the plot in Figure 2.

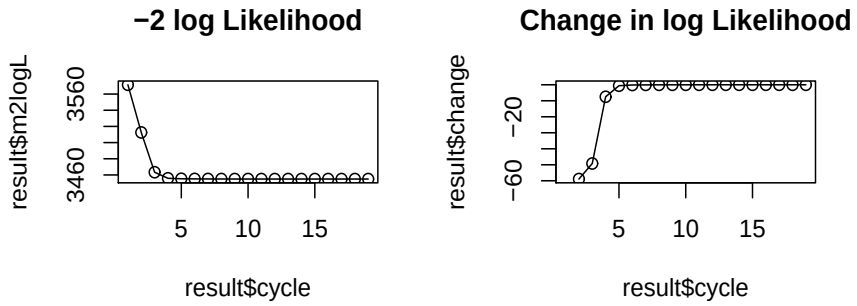

Figure 2: Convergence of the EM algorithm.

On the basis of the fitted model, **mimR** can impute the latent variables. For discrete latent variables this amounts for each case to assign the value with

the highest probability. For a continuous latent variable, the imputed values is the conditional mean of the latent variable given the observed variables. To impute the missing values we use the `imputeMissing()` function (which takes no arguments) as:

```
> imputeMissing()
```

To get the data (including the imputed values) we use the `retrieveData()` function (which takes no arguments):

```
> d.imp <- retrieveData()

Printing format: 12,8

> d.imp[1:5, ]

  mechanics vectors algebra analysis statistics 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(d.imp$A)
```

The plot is shown in Figure 3. 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."

12.3 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. It is well known, that 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 in the call of `emfit()` is that **MIM** substitutes random values for the missing values. It is, however, possible to control the starting values of the EM algorithm as follows: The user can 1) specify the values of `A` and 2) subsequently declare `A` to be latent. In the call of `emfit()`, the argument `S` causes the EM algorithm to take the specified values as starting point for the EM algorithm.

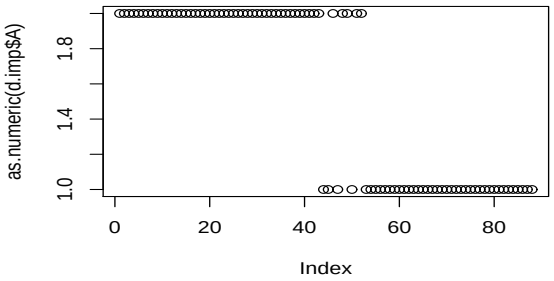

Figure 3: An index plot of the discrete latent variable A.

12.4 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:

```
> fa <- factanal(mathmark, 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"
> satMath <- mim("..", data = gmdMath)
> m2Math <- editmim(satMath, deleteEdge = delEdges)

Printing format: 12,8

> m2MathFit <- emfit(m2Math, arg = "S")

Printing format: 12,8
```

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()

Printing format: 12,8

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

The plot of the imputed values for the latent variables are shown in Figure 4 and this also suggests that the data do not emerge in random order.

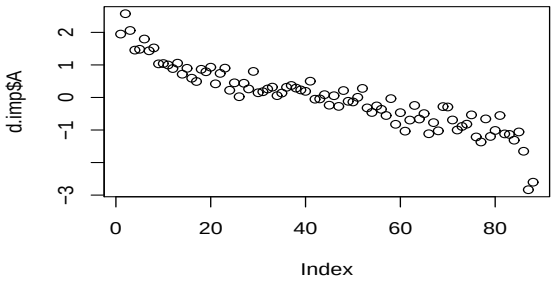

Figure 4: An index of the predicted value of the continuous latent variable `A`.

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` 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**. Also the members of the `gR` project are acknowledged for their inspiration.

References

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

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

David Edwards. *Introduction to Graphical Modelling*. Springer Verlag, New York, 2nd edition edition, 2000.

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

Steffen L. Lauritzen. *Graphical Models*. Oxford University Press, 1996.

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

R Development Core Team. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, 2004. URL `http://www.R-project.org`. ISBN 3-900051-00-3.

A Low level access to MIM from R

A.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
```

A.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**.