# `imset.R` - a suite of imset functions for R

Jiří Vomlel and Milan Studený

September 5, 2007

## 1 Using the imset suite within R

R is a language and environment for statistical computing and graphics. It provides a wide variety of statistical and graphical techniques, and is highly extensible. R is available as Free Software under the terms of the Free Software Foundation's GNU General Public License in source code form. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. See `http://www.r-project.org/`.

`imset.R` is a suite of functions in R for integer valued multisets (imsets) of graphical models. It uses two packages:

- `graph` - a package that implements some simple graph handling capabilities that is part of the Bioconductor project and
- `ggm` - a package from CRAN containing functions for fitting Gaussian Markov models.

To install a package start R and use the Package menu. For both of the two required packages repeat the following procedure. First, select the repository - CRAN for ggm and Bioconductor for graph, respectively. Second, select Install package(s) item in the Package menu and select the required package. It will be installed in your system automatically.

The `imset.R` suite is available as a single file `imset.R` at

`http://staff.utia.cas.cz/vomlel/imset/`.

Dowload the file and save it into a working directory of your machine. To use the imset suite within R, first, start the R console, change directory to the directory, where the file `imset.R` is located, and type `source("imset.R")`. Now, you are ready to use the functions from the `imset.R` suite.

Next we will provide a brief description of functions provided in the `imset.R` suite. For a detailed exposition to the theory of imsets, see the book by Milan Studený [2]. For a description of the conversion functions, see the paper by Milan Studený and Jiří Vomlel [1], which is also available on line at `http://staff.utia.cas.cz/vomlel/msjv04ex.ps`. The implmented learning algorithms are described in [4], which is also available on-line at `http://staff.utia.cas.cz/vomlel/cze-jap-2007.pdf`.

## 2 Functions for graphical models

### `DAG(seq)`

This is a function of the `ggm` package. For a detailed description see the manual of this package: `http://cran.r-project.org/doc/packages/ggm.pdf`. Symbol `seq` stands for a sequence of model formulas. We give an example of such a formula for an acyclic directed graph (DAG).

**Example 1.** `D=DAG(b~a+c+d,a~c+d)`
This command defines a DAG consisting of four nodes - $a, b, c, d$, where node $b$ has three parents - $a$, $c$, and $d$ and node $a$ has two parents - $c$ and $d$. Nodes $c$ and $d$ have no parents.

The output graph is represented by its adjacency matrix. The columns of adjaceny matrix correspond to nodes and rows to their parents, i.e., since node $b$ (with index 1) has node $a$ (with index 2) as its parent therefore `D[[2,1]]==1`, but `D[[1,2]]!=1`.

## drawGraph(amat,coor=NULL)

This is a function of the `ggm` package. For a detailed description see the manual of this package: `http://cran.r-project.org/doc/packages/ggm.pdf`. Citing from the description in the ggm manual: "The function plots the graph with a initial positioning of the nodes, as specified by coor and remains in a waiting state. The position of each node can be shifted by pointing and clicking (with the first mouse button) close to the node. When the mouse button is pressed the node which is closer to the selected point is moved to that position. Thus, one must be careful to click closer to the selected node than to any other node. The nodes can be moved to any position by repeating the previous operation. The adjustment process is terminated by pressing any mouse button other than the first. At the end of the process, the function returns invisibly the coordinates of the nodes. The coordinates may be used later to redisplay the graph."

**Example 2.** `newCoor=drawGraph(D)`
This function plots graph D at default positions. The nodes can be moved by clicking the mouse. Their new coordinates are returned in newCoor.

## essentialGraph(amat)

It creates the esential graph (represented by its adjacency matrix) of a DAG represented by an adjacency matrix `amat`.

**Example 3.** `E=essentialGraph(D)`
This command provides the essential graph `E` of graph `D`. The esential graph `E` has four directed edges: $c \to a$,$d \to a$,$c \to b$,$d \to b$, and one undirected edge $a - b$. For each undirected edge, in this example it is the edge between node $a$ (with index 2) and node $b$ (with index 1), it holds that both entries in the adjacency matrix of `E` are one, i.e., `D[[1,2]]==1` and `D[[2,1]]==1`.

## egToDAG(amat)

This creates a DAG from the equivalence class of DAGs represented by the essential graph (represented by its adjacency matrix `amat`).

**Example 4.** `F=egToDAG(E)`
This command provides a DAG `F` from the equivalence class of DAGs represented by the essential graph `E`. One can observe that `F = D` but note that, generally, this graph need not equal to the original graph that generated `E`. The graph with edge $a \to b$ reversed would be also a correct result. However, the original graph and the result always belong to the same equivalence class.

`chainComps(amat)`

It constructs chain components of a chain graph (e.g. an essential graph) represented by its adjacency matrix. The output is an integer vector representing a partition of the set of nodes.

**Example 5.** `CC=chainComps(E)`
For the essential graph of directed graph from Example 1 it returns vector $1, 1, 2, 3$.

`chainCompList(amat)`

It returns the list of chain components of a chain graph represented by its adjacency matrix.

**Example 6.** `CL=chainCompList(E)`
For the essential graph of directed graph from Example 1 it returns list of chain components $\{a, b\}, \{c\}, \{d\}$.

`closChainCompI(i,amat)`

It creates a closure graph of a chain component $i$ of an essential graph. Graphs are represented by their adjacency matrices.

**Example 7.** `CL=closChainCompI(1,E)`
For the essential graph of directed graph from Example 1 it returns the closure graph of chain component $\{a, b\}$, which is the complete graph over nodes $a, b, c, d$.

`closChainComps(amat)`

It creates the list of closure graphs of all chain components of an essential graph. Graphs are represented by their adjacency matrices.

# 3 Imsets

In this section we describe functions for imsets. We use the hash function available in R for addressing variables in an environment, which makes the retrieval of imset values more efficient. What we actually do is that for each imset we create a new environment (with the hash option being true):

```
cacheImset=new.env(hash=TRUE)
```

This environment is used for storing values of the corresponding imset. Each set has associated a unique natural number (an index), which represents the name of corresponding variable in the environment.

`getIndex(vars,subset)`

This function computes index of a given set, i.e. a subset of variables `vars`, where `vars` are all variables of the imset.

`getSet(vars,ind)`

This function finds for a given index `ind` the corresponding set, `vars` are all variables of the imset.

### imsIndep(cacheImset,vars,Q,R,Z,sign)

This function adds (if sign is +1) to or subtratcts (if sign is -1) from the imset `cacheImset` the imset for the conditional independence statement $< Q, R|Z >$

**Example 8.** `imsIndep(cacheImset,c("a","b","c","d"),"a","b",c("c","d"),+1)`
This adds the elementary imset of the conditional independence statement $< \{a\}, \{b\}|\{c, d\} >$, which is `ims` $= \delta_{\{a,b,c,d\}} + \delta_{\{c,d\}} - \delta_{\{a,c,d\}} - \delta_{\{b,c,d\}}$.

### printIms=function(cacheImset,vars)

This function prints imset `cacheImset` in a compact form.

**Example 9.** Assume `cacheImset` given in Example 8.
`printImset(cacheImset,c("a","b","c","d"))` gives:

```
{ a b c d } = 1
{ c d } = 1
{ a c d } = -1
{ b c d } = -1
```

On the left hand side of each equation there is a set of the imset and on the right hand side the corresponding value.

### setIms(vars,set,v,cacheImset)

This function sets value of the set in imset `cacheImset` to v, if the set already has a non-zero value then it adds v to it. `vars` are all variables of the imset.

**Example 10.** Assume `cacheImset` given in Example 8.
`setIms(c("a","b","c","d"),c("c","d"),-1,cacheImset)`
adds $-1$ to the value for set $\{c, d\}$ since the value was already non-zero. It means that the resulting imset is $\delta_{\{a,b,c,d\}} - \delta_{\{a,c,d\}} - \delta_{\{b,c,d\}}$.

### getIms(vars,set,cacheImset)

This function provides value of the set in imset `cacheImset`, `vars` are all variables of the imset.

**Example 11.** Assume `cacheImset` given in Example 8.
`getIms(c("a","b","c","d"),c("c","d"),cacheImset)`
provides the value for the set $\{c, d\}$, which is 1.

### copyIms(cacheImset,vars,cacheImset2)

This function makes a copy of imset `cacheImset` into `cacheImset2`, which means that we actually create a new environment with the same variables and values as the original one. `vars` are all variables of the imset.

### getImsVars(cacheImset,vars)

This function provides the union of sets for which the imset is non-zero. The input list of variables (`vars`) provides the reference for addressing imset sets. Of course, they need not be equivalent, but the `vars` is the superset of (or equal to) the output.

`getDataIms(imsSet,vars,data,vers,cacheDataImset)`

This function reads the value of an imset that represents a data set - we call the imset *data imset*. The imset is stored in `cacheDataImset`. If the value is not available it computes it and stores it there. For the computation the data set `data` is used and for the computation the criteria `vers` is used. The criteria can be either the Bayesian Information Criteria (`vers="BIC"`), Akaike's Information Criteria (`vers="AIC"`), or the log-likelihood (`vers="LL"`). The `data` are assumed to be created by

```
data=read.table(fileName,header=TRUE,colClasses="factor",sep=",")
```

where `fileName` is the name of the file where the data set is stored in the comma separated format with the first line consisting of the names of the variables.

`dotProductIms(cacheImset,cacheDataImset,nodes,data,vers)`

This function computes the scalar product of a standard imset `cacheImset` and a data imset `cacheDataImset`. This function is extensively used during the model search.

# 4 Conversion of graphical models to imsets

`ims(G,cacheImset,vars)`

This function provides in `cacheImset` the standard imset of an essential graph or of an acyclic directed graph `G`, `vars` are all nodes of the graph (they become the variables of the imsets).

**Example 12.** Assume `D` to be the directed graph from Example 1.
Then `ims(D,cacheImset,rownames(D))` computes the imset $\delta_{\emptyset}+\delta_{\{c,d\}}-\delta_{\{c\}}-\delta_{\{d\}}$.
Note that it would give the same result for the essential graph `E` of DAG `D`.

# 5 Conversion of imsets to graphical models

`reconstrComps(cacheImset,vars)`

This function constructs from an imset `cacheImset` (on variables `vars`) an ordered list of subsets of nodes (a kind of a hierarchical junction tree), which is a representant of an equivalence class of DAGs. This function performs the first step for the reconstruction of the essential graph from its imset.

**Example 13.** Assume `ims3` from Example 12. `comps=reconstrComps(ims)` gives the list $\{\{a,b,c,d\},\{c\},\{d\}\}$.

`reconstrEssentialGraph(comps)`

This function constructs from an ordered list of subsets of nodes an essential graph. It performs the second step for the reconstruction of the essential graph from its imset.

**Example 14.** Assume `comps` from Example 13. `EG=reconstrEssentialGraph(comps)` gives the essential graph from example 3.

# 6 Tests

`isStandard(cacheImset,vars)`

This function tests whether imset `cacheImset` is a standard imset.

`isImsEmpty(cacheImset)`

This function tests whether imset `cacheImset` is empty.

# 7 Learning essential graphs of Bayesian networks

`bestModel(data,initialGraph=emptyGraph(names(data)), firstGoDown=TRUE,vers="BIC",useExtremalOnly=FALSE, coordinates=c(),printImsets=FALSE)`

This function returns a best essential graph for `data` found be greedy equivalence search (using imsets). Data can be read using function `read.table`, e.g.
`d=read.table(fileName,header=TRUE,colClasses="factor",sep=",")`
The search starts with the initial graph (the default is the empty graph) and either it first goes down in the lattice of imsets, i.e. it first deletes independencies (it adds edges) in the model (which is the default). When it reaches maximum in the downwards direction it performs the search in the upwards direction. If `firstGoDown==FALSE` the search is first performed upwards and then downwards.

For the search in the inclusion neighbourhood of an essential graph the method uses the description of the neigbourhood by a tuft [3]. This representation implies that each essential graph in the neigbourhood is visited only once. For the evaluation of the respective change of the criteria the imset representation is used, more specifically the scalar product of the respective differential imset and data imset is computed.

**Example 15.**
```
data=read.table("abcdef.dat",header=TRUE,colClasses="factor",sep=",")
eg=bestModel(data)
```
gives the best model represented by its essential graph for the data set in `abcdef.dat` file.

## 7.1 Example of an R session

```
# load the imset.R suite of functions
source("imset.R")
# the file with the data set - a data set generated
# from the well known Chest Clinic Example
fileName="asia-long-names.dat"
data=read.table(fileName,header=TRUE,colClasses="factor",sep=",")
# variables of the model - in the alphabetic order
vars=names(data)
o1=order(vars)
vars=vars[o1]
# coordinates for the nodes in the graph
dataCoor=c(71.55299, 68.20930,
           59.32657, 38.38371,
           40.83020, 70.40697,
           11.51814, 39.16860,
```

```
           57.75908, 91.44186,
           11.20465, 71.34883,
           24.37156, 57.06395,
           11.04790, 90.34302)
# names of the coordinates
dnames=list(vars,c("x","y"))
# matrix of coordinates
coor=matrix(dataCoor,nrow=8,ncol=2,byrow=TRUE,dimnames=dnames)
# The learning algorithm uses our coordinates
# (otherwise it would place all variables in a cricle)
# and prints the imset for each step of the algorithm
# (default is no prints).
# The output is a best essential graph for the criteria
# (default is BIC)
print("learning the model from data",quote=FALSE)
eg=bestModel(data,coordinates=coor,printImsets=TRUE)
```

## Acknowledgements

This work has been supported by the grant GAČR nr. 201/04/0393 and by the grant nr. 1M0572 of the Ministry of Education of the Czech Republic.

## References

[1] M. Studený and J. Vomlel, Transition between graphical and algebraic representatives of Bayesian network models, In *Proceeding of the 2nd European Workshop on Probabilistic Graphical Models (PGM'04)*, Leiden, the Netherlands. `http://staff.utia.cas.cz/vomlel/msjv04ex.ps`

[2] M. Studený: *Probabilistic Conditional Independence Structures*, Springer-Verlag 2005.

[3] M. Studený: *Characterization of inclusion neighbourhood in terms of the essential graphs.* International Journal of Approximate Reasoning 38 (2005), n. 3, pp. 283-309. `ftp://ftp.utia.cas.cz/pub/staff/studeny/char-inc-es.ps`

[4] J. Vomlel and M. Studený. Using imsets for learning Bayesian networks. In Proceedings of the 10th Czech-Japan Seminar on Data Analysis and Decision Making under Uncertainty, Liblice, Czech Republic, 2007.