July 24, 2008

dmgBayes: Software for Bayesian Inference in Mixed Graph Models and Structural Equation Models

Ricardo Silva
silva@statslab.cam.ac.uk

Abstract

This document briefly describes the `dmgBayes` software for doing Bayesian inference with mixed graph models. The current main reference for this software is

> Silva, R. and Ghahramani, Z. (2008). "The hidden life of latent variables: Bayesian learning with mixed graph models".

Current version: 1.0

dmgBayes

Ricardo Silva
silva@statslab.cam.ac.uk

1 Introduction

Directed mixed graph (DMG) models are graphical models with directed and bi-directed edges. In particular, Gaussian models that factorize according to a DMG are commonly known as Structural Equation Models (SEMs). Five basic references are:

- K. Bollen (1989). *Structural Equations with Latent Variables*. Wiley & Sons;
- T. Richardson and P. Spirtes (2002). "Ancestral graph Markov models". Annals of Statistics, Volume 30, Number 4, pp. 962-1030;
- T. Richardson (2003). "Markov properties for acyclic directed mixed graphs". Scandinavian Journal of Statistics, Volume 30, Number 1, pp. 145-157;
- M. Drton and T. Richardson (2004). "Iterative conditional fitting for Gaussian ancestral graph models". Proceedings of UAI'04;
- R. Silva and Z. Ghahramani (2008). "The hidden life of latent variables: Bayesian learning with mixed graph models".

`dmgBayes` is a software that implements some of the algorithms described by Silva and Ghahramani (2008). The purpose of this document is to provide a simple description on how to use `dmgBayes` for performing Bayesian inference.

Currently, `dmgBayes` provides functionality for the following families of models:

- Gaussian mixed graph models: the software provides a Markov Chain Monte Carlo (MCMC) sampler for generating samples from the posterior. The output of the software is an estimated covariance matrix defined as the expected posterior covariance matrix given the data and prior. It also computes the predictive log-likelihood of the model in an optional test set. Moreover, it generates an output file with all MCMC samples, which can be easily accessed to generate plots and other MCMC estimates of functionals of the posterior;
- zero-mean Gaussian models of marginal independence (i.e., bi-directed models). On top of the above features, the software also allows for the computation of the marginal likelihood of such models.

Section 3 details the initial steps to run `dmgBayes` using the Java interpreter or the statistical software `R`. Sections 4 and 5 describe the basic functionality of the software. Two simple examples are referred to in Section 6. Finally, Section 7 lists a few extensions of `dmgBayes` that might be implemented in a near future. My recommendation is to first look at the example files distributed with this software.

We first start with a brief description of the Gaussian mixed graph model and the priors used in our formulation.

2 A short note on the models

Given that we use particular choices of priors and parameterizations to define these models, and that the given references might be unaccessible/too long, we first provide a brief summary of the models. Readers already familiar with such models should skip this section. We assume knowledge of standard graph terminology (nodes, parents, edges, etc.). Section 1 of Silva and Ghahramani (2008) should contain all the necessary information for the majority of the readers.

2.1 Parameters

Consider a Gaussian model with a set of variables $\mathbf{Y} = \{Y_1, Y_2, \ldots, Y_m\}$. For each variable Y_i with a (possibly empty) parent set $\{Y_{i1}, ..., Y_{ik}\}$, one provides a "structural equation"

$$Y_i = \mu_i + b_{i1}Y_{i1} + b_{i2}Y_{i2} + \cdots + b_{ik}Y_{ik} + \epsilon_i \tag{1}$$

where ϵ_i is a Gaussian random variable with zero mean and variance v_{ii}.

Unlike in standard regression models, "error term" ϵ_i is not necessarily constructed to be independent of the other variables in $\mathbf{Y}$. Dependencies arise when error terms of different equations are not independent. Independence is asserted by the graphical structure: ϵ_i and ϵ_j (the error term for some Y_j) are marginally independent if Y_i and Y_j are not connected by a bi-directed edge, but are otherwise free to be dependent.

By this parameterization, each directed edge $Y_i \leftarrow Y_j$ in the graph corresponds to a parameter b_{ij}. Each bi-directed edge $Y_i \leftrightarrow Y_j$ in the graph is associated with a covariance parameter v_{ij}, the covariance of ϵ_i and ϵ_j. Each node Y_j in the graph is associated with variance parameter v_{jj}, the variance of ϵ_j. Algebraically, let $\mathbf{B}$ be a $m \times m$ matrix, m being the number of observed variables. This matrix is such that $\mathbf{B}_{ij} = b_{ij}$ if $Y_i \leftarrow Y_j$ exists in the graph, and 0 otherwise. Let $\mathbf{V}$ be a $m \times m$ matrix, where $\mathbf{V}_{ij} = v_{ij}$ if $i = j$ or if $Y_i \leftrightarrow Y_j$ is in the graph, and 0 otherwise. Let $\mathbf{Y}$ be the column vector of observed variables, μ the column vector of intercept parameters, and ϵ be the corresponding vector of error terms. The set of structural equations can be given in matrix form as

$$\begin{aligned} & \mathbf{Y} = \mathbf{B}\mathbf{Y} + \mu + \epsilon \Rightarrow \mathbf{Y} = (\mathbf{I} - \mathbf{B})^{-1}(\epsilon + \mu) \\ \Rightarrow \quad & \Sigma(\Theta) = (\mathbf{I} - \mathbf{B})^{-1}\mathbf{V}(\mathbf{I} - \mathbf{B})^{-T} \end{aligned} \tag{2}$$

where $\mathbf{A}^{-T}$ is the transpose of the inverse of matrix $\mathbf{A}$ and $\Sigma(\Theta)$ is the *implied covariance matrix* of the model, $\Theta = \{\mathbf{B}, \mathbf{V}, \mu\}$.

2.2 Priors

We need priors for Θ in order to compute the posterior distribution of the parameters given the data. In our software, we assume that each coefficient b_{ij} and intercept μ_i is independent of all other parameters a priori. Its prior distribution is defined to be a Gaussian where individual means and variances can be specified.

The prior for the covariance matrix of the error terms is non-standard. We need a distribution for covariance matrices with zero entries for all pairs $\{Y_i, Y_j\}$ not connected by a bi-directed edge. Given the bi-directed component $\mathcal{G}$ of our DMG, we define the prior over the covariance matrix $\mathbf{V}$ of the error terms to be:

$$p(\mathbf{V}) = \frac{1}{I_{\mathcal{G}}(\delta, \mathbf{U})} |\Sigma|^{-(\delta+2m)/2} \exp\left\{-\frac{1}{2} tr(\mathbf{V}^{-1}\mathbf{U})\right\}, \mathbf{V} \in M^{+}(\mathcal{G}), \quad (3)$$

$p(\cdot)$ being the density function, $tr(\cdot)$ the trace function, and m the number of variables (nodes) in our model. Space $M^{+}(\mathcal{G})$ is the space of all positive definite matrices with entry $v_{ij} = 0$ if nodes Y_i and Y_j are not adjacent in $\mathcal{G}$. It is assumed that $\delta > 0$ and $\mathbf{U}$ is positive definite. In the software, one has to specify δ and $\mathbf{U}$.

There are no analytical expressions for the normalizing constant $I_{\mathcal{G}}(\delta, \mathbf{U})$. Since this constant is vital for Bayesian model selection of bi-directed graphs, the software also provides a routine to estimate this constant using Monte Carlo methods.

3 Preliminaries

You will need a Java Virtual Machine to run the software. Google "java runtime" in case you do not have it installed already.

You will also need the Tetrad library, developed at Carnegie Mellon University. Go to `http://www.phil.cmu.edu/projects/tetrad_download/download` (or google for "tetrad project" in case the link has changed) and download the latest JAR file. (Notice that the website also has versions of this software for Java Web Start. Download the jar instead). The current version of `dmgBayes` has been tested with `tetrad-4.3.8-6.jar`.

The `dmgBayes` software comes into two different versions:

- the `R` version: we provide `R` code that wraps the Java implementation of `dmgBayes`. Everything is handled within `R`, including the data structures that need to be passed to the sampler. Section 4 details how to use `dmgBayes` in this case;
- the stand-alone version: it is possible to run `dmgBayes` entirely from the command line. All the information used by the sampler should be passed through text files. Section 5 details how to format the input files and how to call the software from the command line.

4 Using `dmgBayes` in `R`

When calling `dmgBayes` within `R`, you will first have to install the `rJava` package. This package is available in CRAN and can be installed by the usual way.

The `R` version of `dmgBayes` can be configured in two ways. The first is by installing a package. After the package is loaded, nothing else is necessary. The second way is by using the source code directly (only one file is needed: `dmgsample.R`). In this case, we have to initialize the library as follows. Start `R`, make sure `dmgsample.R` is accessible within `R` (i.e., in the path) and enter the following:

```
> library(rJava)
> .jinit()
> .jaddClassPath(<enter here the path to the jar file dmgbayes1.0.jar>)
> .jaddClassPath(<enter here the path to the tetrad jar file>)
> source("dmgsample.R")
```

Now that R is ready to run dmgBayes, it is time to generate the information required by the Bayesian procedure: the data file(s), the graphical structure (which defines which parameters exist, as explained in Section 2), and the priors.

The current version (1.0) provides two applications in the R file dmgsample.R.

4.1 dmgbayes.gaussiansample

This application does Bayesian inference for Gaussian mixed graph models using a MCMC algorithm. The basic functionality consists on generating estimates of the covariance matrix (defined as the posterior mean) and predictive log-likelihood. Much more can be done by using the generated samples, which are stored in a output file. Marginal likelihood computation will be added in the future.

Call dmgbayes.gaussiansample with the following arguments:

- var.names: an array of strings providing the name of the variables;
- directed: a matrix indicating which directed edges exist;
- bidirected: a matrix indicating which bi-directed edges exist;
- train: the data;
- islatent: an array indicating which variables are latent and which are recorded in train;
- prior.V: the matrix hyperparameter for the prior distribution of the error covariance matrix $\mathbf{V}$;
- dmg.df: the degrees of freedom hyperparameter for the prior distribution of $\mathbf{V}$;
- b.priormean: an array containing the mean hyperparameters for the Gaussian prior of each coefficient/intercept;
- b.priorvar: an array containing the variance hyperparameters for the Gaussian prior of each coefficient/intercept;
- b.fixed: an array indicating which coefficient/intercept are *fixed to constants*. The prior mean is used as the pre-defined constant value;
- mcmc.num.samples: the total number of MCMC steps used in the procedure;
- mcmc.burn.in: number of initial steps from the Markov chain that will be thrown out;
- mcmc.step: if desired, set this to any integer value greater than 1 if one wants to use only some of the sampled points for the estimates. Setting step size to n means that, given that point i was kept, the next point to be stored will be point $i + n$. The first point to be kept is the first one after the burn-in period.
- output.filename: since this process can be memory intensive, we store all samples from the chain in an external text file with name given by output.filename;
- mcmc.options: an array of integers specifying a couple of options. Entry mcmc.options[1] specifies the level of verbosity of the output, from no output (1) to printing a message at each sample (4). Set mcmc.options[2] to 1 if you want to generate a file with samples of the latent variables, named <output.filename>.latents.mcmc. WARNING: this file can get very large quickly, since it will generate a sample with as many datapoints as train for each iteration of the Markov chain;

- `test`: an optional parameter providing another dataset. If provided, the posterior samples (using the burn in and step constraints) will be used to compute the predictive log-likelihood of `test`;

Information about how to structure the matrices and arrays described above is given in Section 4.3. A log containing all sampled points will be generated as `<output.filename>.mcmc`. This allows the user to estimate other posterior functionals and to generate plots. This file

Besides generating file `<output.filename>.mcmc` (and `<output.filename>.latents.mcmc`, if `mcmc.options[2]` is set to 1), this function returns an array of two entries. The first entry is equal to 1 if the procedure was concluded without any errors and 0 if some error happened (e.g., invalid graph). The second entry returns the average predictive log-likelihood of the `test` set if provided.

The current version of `dmgBayes` does not allow for models with directed cycles, i.e., no model where there is a path $X \to \cdots \to X$.

4.1.1 Reading the output

At the end of the run of the Markov chain, the corresponding `.mcmc` file is created. This file is structured such that it can be loaded within `R` as a data frame using the `read.table` command. The header in this file is encoded as follows. There are three types of variable names used:

- `x.D.y`: here, `x` and `y` are integers. The integers are indices for the entries in `var.names`. This column of the MCMC data frame contains the samples of the coefficient corresponding to the directed edge from the xth variable into the yth variable. For instance, if the xth variable is `AGE` and the yth variable is `INCOME`, then the column `x.D.y` corresponds to samples of the coefficient associated with `AGE` $\to$ `INCOME`;
- `.D.y`: as before, `y` is an integer indicating that the corresponding variable is the yth entry of `var.names`. Columns of this type contain samples of the intercepts of the corresponding structural equations. Continuing the previous example, since `AGE` corresponds to the yth variable, this column would contain samples for the intercept of the structural equation of `AGE`;
- `x.B.y`: as before, but corresponding to the bi-directed edges. In our example, this would correspond to the bi-directed edge `AGE` $\leftrightarrow$ `INCOME`. That is, samples of the covariance of the error terms of the structural equations for `AGE` and `INCOME` (notice that all samples are positive for `x.B.x`, which correspond to variances of the error terms).

4.2 `dmgbayes.gaussian.marginal.bidirected`

Simpler than the previous procedure, this assumes that the model is a Gaussian model that factorizes according to a DMG containing only bi-directed edges and no latent variables. The data is assumed to be centered at its empirical mean. This class computes the log-marginal likelihood of the given graph. Call this function with the following arguments:

- `var.names`: an array of strings providing the name of the variables;
- `bidirected`: a matrix indicating which bi-directed edges exist;
- `train`: the data;
- `prior.V`: the matrix hyperparameter for the prior distribution over the error covariance matrix $\mathbf{V}$;
- `dmg.df`: the degrees of freedom hyperparameter for the prior distribution of $\mathbf{V}$;
- `mc.num.samples`: the total number of Monte Carlo steps used in the procedure;
- `mc.options`: an array of integers specifying a couple of options. Entry `mcmc.options[1]` specifies the level of verbosity of the output, from no output (1) to printing a message at each sample (4).

This function returns an array of two entries. The first entry is equal to 1 if the procedure was concluded without any errors and 0 if some error happened (e.g., invalid graph). The second entry returns the desired marginal log-likelihood estimate.

4.3 Formatting the input

Different kinds of `R` objects are used to represent input information. Some are straightforward: datasets `train` and `test` are `R` data frames. Still, some care needs to be taken while providing this information. Also, refer to the examples discussed at the end of this document as an easy way of visualizing the formatting requirements.

4.3.1 Data frames and name arrays

The name information for all variables is passed through the standard array of strings `var.names`. The reason why we do not use the header information of the data frames is due to the possibility of latent variables. In this case, the array `var.names` should include all variables, observed or latent.

When providing `train` and `test`, two main points have to be observed:

- currently, we do not support missing data. No `NA` values are admitted;
- the *order* of the columns in the data frame should correspond to the order of the observed variables in `var.names`. For instance, if `var.names` is the vector ("$X1$","$X2$","H","$X3$","$X4$"), where H is the only latent variable, then the columns of the datasets should be correspond to the order ("$X1$","$X2$","$X3$","$X4$"),

The ordering within the integer array `islatent` should also correspond to the ordering in `var.names`.

4.3.2 Graph matrices

The graph matrices `directed` and `bidirected` are provided as integer matrices of two columns and as many rows as there are edges. Each row will correspond to an edge, and the numbers entered in the matrices correspond to their position in `var.names`.

For instance, suppose we have a graph with the directed edges $X \to Y \to Z$ and the bi-directed edge $Y \leftrightarrow Z$. Suppose also that `var.names` is ("X", "Y", "Z"). In the edge matrices, we refer to X, Y and Z as 1, 2 and 3, respectively.

The first row in `directed` corresponds to some arbitrary directed edge of this graph. Say, $X \rightarrow Y$. We encode this row as [2 1], corresponding to the convention that first we specify the node at the arrowhead ending of the edge. The `directed` matrix in our example becomes

$$\begin{bmatrix} 2 & 1 \\ 3 & 2 \end{bmatrix} \tag{4}$$

The 2-column matrix `bidirected` is specified in exactly the same way, except that the order of the elements in each row is not relevant.

4.3.3 Prior matrices

The matrix hyperparameter `prior.V` for the prior over the error covariance matrix is a regular `R` matrix. The number of rows and columns should correspond to the total number of variables (observed and latent). Again, for consistency it is required that the row/column order should correspond to the entries in `var.names`.

Arrays `b.priormean` and `bprior.var` are standard numerical arrays, but extra care should be taken while constructing these arrays.

Consider first `b.priormean`. Once again we follow the order defined by `var.names`. Let Y_i correspond to the ith variable according to this order. We add to `b.priormean` the prior means to all the edges into Y_i before considering any edges into $Y_j, j > i$. Within the edges into Y_i, we also order them according to the index of its parents.

For instance, suppose we are adding the edges into Y_5, which has parents Y_2 and Y_4. We add to `b.priormean` the mean hyperparameter for $Y_2 \rightarrow Y_5$ before $Y_4 \rightarrow Y_5$, since Y_2 precedes Y_4.

We also have to add the prior mean for the intercept term. We add this prior after adding the mean hyperparameters for all parents of Y_i. In our example, we add the prior mean for the intercept of Y_5 after the one corresponding to $Y_4 \rightarrow Y_5$.

The ordering within `b.priorvar` is completely analogous.

Finally, the integer array `b.fixed` indicates which coefficients are fixed to constants. The entries in `b.fixed` correspond to the entries in `b.priormean`, so the same ordering requirements apply. Use `1` to indicate that a particular coefficient is fixed, `0` otherwise. The fixed value will be the corresponding mean hyperparameter passed in `b.priormean`.

5 Using the stand-alone software

When calling `dmgBayes` directly from the command line (i.e., bypassing `R`), you will need to add the `tetrad*.jar` file to you classpath. This can be done directly from your command line using the option `-cp`. For instance:

```
> java -cp .:tetrad-4.3.8-6.jar rbas.dmg.app.RunGibbsGaussian
```

The current version (1.0) provides two applications:

5.1 `rbas.dmg.app.RunGibbsGaussian`

This application does Bayesian inference for Gaussian mixed graph models using a MCMC algorithm. The basic functionality consists on generating estimates of the covariance matrix (defined as the posterior mean) and predictive log-likelihood. Much more

can be done by using the generated samples, which are stored in a output file. Marginal likelihood computation will be added in the future.

Call the Java interpreter with the following arguments:

```
> RunGibbsGaussian <domain name> <number of samples> <burn in> <step size>
```

The meaning of the parameters is as follows:

- `domain name`: this indicates which graph, priors and datasets to use. More on that later;
- `number of samples`: number of MCMC steps;
- `burn in`: number of initial steps from the chain that will be thrown out;
- `step size`: if desired, set this to any integer value greater than 1 if one wants to use only some of the sampled points for the estimates. Setting step size to n means that, given that point i was kept, the next point to be stored will be point $i + n$. The first point to be kept is the first one after the burn-in period.

`RunGibbsGaussian` will access the following files:

- `<domain name>.train`, the dataset that we will condition on to generate the posterior distribution over parameters;
- `<domain name>.test`, an optional data file. If present, `RunGibbsGaussian` will calculate the log-likelihood of this dataset given the training data and the graph[1];
- `<domain name>.graph`, the directed mixed graph;
- `<domain name>.priors`, the prior for the parameters;

Information about the file formats is given in Section 5.3. A log containing all sampled points will be generated as `<domain name>.mcmc`. This allows the user to estimate other posterior functionals and to generate plots.

The current version of `dmgBayes` does not allow for models with directed cycles, i.e., no model where there is a path $X \to \cdots \to X$.

5.2 `rbas.dmg.app.RunGaussianBidirectedScoring`

Simpler than `rbas.dmg.app.RunGibbsGaussian`, this assumes that the model is a Gaussian model that factorizes according to a DMG containing only bi-directed edges. The data will be centered at its empirical mean automatically. This class computes the log-marginal likelihood of the given graph. Call the Java interpreter with the following arguments

```
> RunGaussianBidirectedScoring <domain name> <number of samples>
```

which will sample `<number of samples>` points using the Monte Carlo method of Silva and Ghahramani (2008), and use it to calculate the log-marginal likelihood of the model. As before, it assumes that the following files are on the path:

[1]The predictive log-likelihood is calculated individually for each test point. The computed log-likelihood of the test set is the sum over each individual log-likelihood.

- `<domain name>.train`, the dataset whose probability will be calculated given the graph;
- `<domain name>.graph`, the bi-directed graph;
- `<domain name>.priors`, the prior for the covariance matrix;

5.3 File Formats

There are three different types of files: data, graphs and priors. All of them are in plain text format.

5.3.1 Data files

The data file should contain the name of the variables in the first line, separated by a tab or space character. The data points themselves should start in the second line. Each data point should be in a single individual line, with variable values separated by a tab or space character.

5.3.2 Graph files

Each line should contain an edge (directed or bi-directed). The name of each node should be exactly the name of the corresponding variable in the data file. Nodes with names that do not correspond to any variable in the data file will be assumed to be latent variables automatically.

The syntax for a directed edge is as follows:

```
<variable 1> -> <variable 2>
```

where a valid name for a variable can be anything that does not include any space or tab character.

The syntax for a bi-directed edge is as follows:

```
<variable 1> <-> <variable 2>
```

5.3.3 Prior files

Each parameter corresponds to a particular edge, making the prior distribution file similar to the graph file: each parameter appears in an individual line, starting by the description of the parameter through its edge, followed by the hyperparameters.

For a coefficient associated with edge $X \rightarrow Y$, the corresponding line in the file should be

```
X -> Y <mean> <variance> [true]
```

This encodes that the coefficient associated with edge $X \rightarrow Y$ has a Gaussian prior distribution with the respective mean and variance. Optionally, adding the token `true` to the end of the line means that this parameter is actually fixed at its mean value and is not allowed to vary. For instance,

```
X -> Y 0 10
```

means that the parameter for $X \rightarrow Y$ has as a prior distribution a Gaussian of zero mean and variance 10, while

```
X -> Y 1 10 true
```

means that the parameter for $X \rightarrow Y$ has the fixed value of 1 (the variance value of 10 is ignored).

One of the parameters that defines the distribution of a variable Y given its parents is the intercept term in the respective linear equation (namely, the mean of Y conditioning on its parents is given by a linear combination of its parents plus the intercept). To indicate the prior for the intercept term, use the keyword `_INT`. For example,

```
_INT -> Y 0 10
```

gives a Gaussian prior of zero mean and variance 10 as the prior for the intercept term of Y.

If no prior is given for $X \rightarrow Y$, it takes a standard Gaussian prior as default.

Priors for variances and covariances of error terms are similar. For example, use

```
Y <-> Y 2
```

to indicate that the respective variance entry in the matrix hyperparameter of the G-Inverse Wishart is 2. Use

```
X <-> Y 1
```

to indicate a (X, Y) entry of 1 in the matrix hyperparameter of the error terms.

If no variance and covariance hyperparameters are given, variance hyperparameters will have a default of 1. Covariance hyperparameters will have a default of zero.

Finally, the degrees of freedom for the G-Inverse Wishart prior should be set in the first line of the priors file. Use

```
df <number>
```

to set the degrees of freedom accordingly. As explained in Silva and Ghahramani (2008), this parameterization already takes into the account the dimensionality of the dataset (some inverse Wishart parameterizations need the minimum number of degrees of freedom to be bounded by the dimensionality).

6 Examples

Included with the source code and compiled classes is a small example, analogous to the democratization/industrialization domain discussed in Silva and Ghahramani (2008) and Bollen (1989). It includes both directed and bi-directed edges, as well as latent variables.

The data is simulated: in order to obtain the real data used in the given references, please contact Kenneth Bollen from University of North Carolina.

In this example, we fix the coefficients $L_1 \to X_1$, $L_2 \to X_4$ and $L_3 \to X_8$ to 1, since the scale and sign of the latents is arbitrary. Fixing such coefficients will anchor the sign and scale of the latents to the observed variables.

Another example with synthetic data is included: a simple demonstration of the method for computing marginal likelihoods (i.e., the G-Inverse Wishart normalizing constant).

7 TO DO List

These are several extensions of this software that are planned for the future:

- probit models for binary and ordinal variables, and mixtures of binary/ordinal/Gaussian variables;
- variational approximations for Gaussian and probit models;
- structure learning: searching for graphical structures with high posterior value;
- better interface: generate more detailed log files, "real-time" traces of Markov chains, etc.;
- Gaussian cyclic models;
- nonparametric models;

Useful extensions and modifications that are not my priority:

- code optimization: use of sparse matrix inversion methods, message-passing schemes, etc. Virtually no use of special patterns (triangular, banded, etc.) in the error covariance matrix has been implemented (we did implement the *district* factorization − see Silva and Ghahramani, 2008 − that takes care of block-diagonal structures). The code can be much faster if it explores the graphical structure better;
- integration with the Tetrad project at Carnegie Mellon University;
- treatment of missing data;
- parallel implementation;

DISCLAIMER

This software is provided "as is" and free of charge. Feel free to modify and reuse the code in any way you want. There is certainly *a lot* of room for code optimization, which was not my main goal. Comments are welcome.

ACKNOWLEDGEMENTS

Funded in part by the Gatsby Charitable Foundation and the EPSRC grant #EP/D065704/1.