

GDAGsim: Sparse matrix algorithms for Bayesian computation

Darren J Wilkinson

Department of Statistics, University of Newcastle, UK

d.j.wilkinson@ncl.ac.uk

Abstract

GDAGsim is a software library which can be used to carry out conditional sampling of linear Gaussian directed acyclic graph models, and hence can be used for the implementation of efficient block MCMC samplers for such models. This paper examines the software library and its design, and how it can be applied to problems in Bayesian inference.

1 Introduction

GDAGsim (<http://www.staff.ncl.ac.uk/d.j.wilkinson/software/gdagsim/>) is a ‘C’ software library for analysis of conditionally specified linear models. In particular, it can be used to carry out conditional sampling of linear Gaussian Directed Acyclic Graph (GDAG) models. These techniques can be very valuable in the context of Bayesian computation, as they can be used for the implementation of efficient block MCMC samplers for such models.

Linear GDAG models represent a large class of linear latent process models encountered in complex statistical modelling. In particular, general linear models, a large range of hierarchical (multi-level) random-effects models (including one-way and two-way ANOVA), and a range of linear time series models (including lattice-Markov spatio-temporal STARMA models (Cressie 1993)) are all linear GDAG systems. The basic decomposition of the model structure is shown in Figure 1. There is a collection of “parameters” denoted by the vector θ . Conditional on these parameters the (linear Gaussian) latent process X is defined. Then conditional on both the parameters and the latent structure the (linear Gaussian) observables Y are defined. Until section 6 it will be assumed that the parameter vector θ is fixed and known, and the distributions $(X, Y|\theta)$ and $(X|\theta, y)$ will be studied. From section 6 onwards, Bayesian applications which study the posterior distribution $(\theta, X|y)$ will be considered. With the conditioning on θ implicit, an example of a DAG model is shown in Figure 2. In this DAG, X_1 depends on X_0 , X_2 depends on both X_0 and X_1 , and X_3 depends on X_1 and X_2 . That is, X_3 is conditionally (though not marginally) independent of X_0 given both X_1 and X_2 . Similarly, Y_0 depends on X_0 and X_2 , Y_1 depends on X_2 and X_3 , and Y_2 depends only on X_3 . For this paper, it is assumed that each DAG node is univariate, and that elements of Y are conditionally independent given X (and θ).

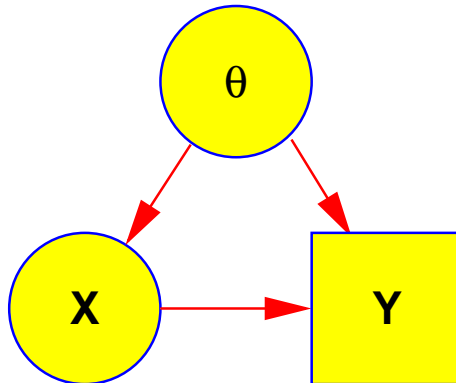


Figure 1: Basic structure for latent process models

2 Linear Gaussian DAG systems

Linear GDAG models are specified conditionally. Consider a DAG model for the n -dimensional multivariate Normal (MVN) random vector, $X = (X_0, X_1, \dots, X_{n-1})'$. Associated with the vector X is the graph $\mathcal{G} = (V, E)$, where $V = \{X_0, X_1, \dots, X_{n-1}\}$ denotes a set of vertices, and $E = \{(v, v') | v, v' \in V, v \rightarrow v'\}$ is the set of ordered pairs denoting the set of directed edges in the graph. The *parents* of $v \in V$ are denoted $\text{pa}(v) = \{v' \in V | (v', v) \in E\}$. The joint density of X factors according to the graph \mathcal{G} if

$$\pi(x) = \prod_{i=0}^{n-1} \pi(x_i | \text{pa}(x_i)).$$

For MVN vectors, X , each conditional distribution is normal, and the dependence on parents is linear, so that

$$\pi(x_i | \text{pa}(x_i)) = N(\alpha_i' x + b_i, 1/\tau_i),$$

where the n -dimensional vector, α_i has non-zero elements only in positions corresponding to the parents of X_i . Thus, a GDAG model can be constructed by specifying a precision, τ_i , a mean-shift, b_i , and the sparse-vector α_i , for each component, X_i .

3 Canonical parameterisation of the MVN and sparse matrix algorithms

In principle, computing with linear GDAG models is straightforward, as the joint distribution of X and Y is Multivariate normal (MVN), and MVN vectors are tractable. In practice, working with high-dimensional MVN vectors can be difficult because of the computational overhead associated with manipulating large variance matrices. In the context of DAG models, it is possible to overcome most of these difficulties by working directly with the canonical parameterisation of the MVN (rather than the usual moment parameterisation); see [Wilkinson and Yeung \(2002\)](#) and [Wilkinson and Yeung \(2001\)](#) for further details. Write $X \sim \mathcal{N}(h, K)$ for the random vector with density

$$\pi(x) \propto \exp\{x' K x - 2x' h\},$$

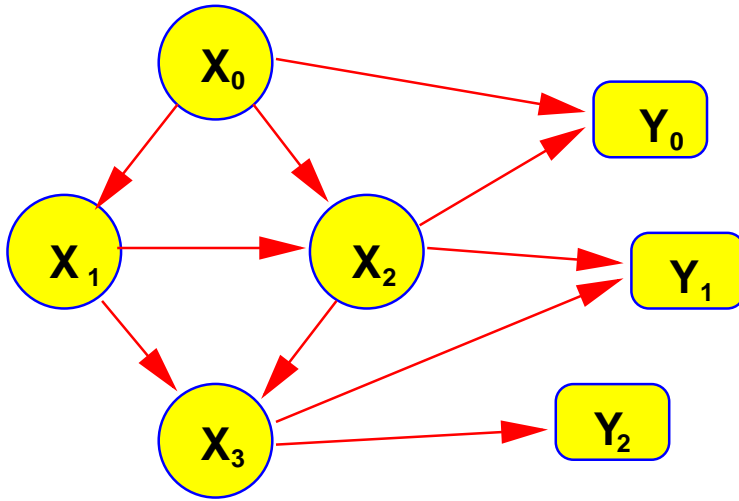


Figure 2: An example of a linear Gaussian DAG model

so that X is MVN with mean $K^{-1}h$ and variance K^{-1} . The primary advantage of this parameterisation is that densities are multiplied together simply by adding parameters. In the context of this paper, there is an additional (related) benefit of this parameterisation, due to the fact that the second canonical parameter, K (the “precision” matrix), is very “sparse” for DAG models, in the sense that there are relatively few non-zero elements in the matrix. In contrast, the variance matrix will typically have very few zero entries. Consequently, by working with the canonical parameterisation, only the non-zero elements of the precision matrix need to be stored (along with their positions), reducing storage requirements and computation time through the use of special sparse matrix algorithms.

4 Computations

The theory upon which `GDAGsim` is based is described in [Wilkinson and Yeung \(2002\)](#) and [Wilkinson and Yeung \(2001\)](#), and the software library itself is described in [Wilkinson \(2001\)](#). Here, a brief indication is given of the kinds of computational techniques that are used. Once the model has been specified, the dense vector h and sparse matrix K are stored, where $(X|\theta, y) \sim \mathcal{N}(h, K)$. There are many aspects of this distribution that may be of interest, but let us focus first on the computation of the mean vector $\mu = K^{-1}h$. The matrix K is sparse, but K^{-1} is not, and is extremely demanding to compute. However, μ may be calculated anyway using the sparse Cholesky factorisation, $K = LL'$, where L is sparse and lower triangular. This factorisation procedure is a standard sparse matrix algorithm, included as part of most sparse matrix libraries. Now using

$$\mu = K^{-1}h = L'^{-1}(L^{-1}h)$$

the sparse triangular linear system $Lv = h$ may be solved for v by forward substitution, and then $L'\mu = v$ may be solved for μ by backward substitution. Forward and backward substitution algorithms are very efficient for sparse matrices, and also included as part of standard sparse matrix libraries.

Another commonly required task is the generation of independent realisations from the conditional distribution $(X|\theta, y)$. Again, this may be accomplished in a straightforward manner using the sparse Cholesky decomposition. First generate $Z \sim N(0, I)$ (a collection

of iid $N(0,1)$ random quantities), and then put $X = \mu + U$, where U is a solution to $L'U = Z$. It can be seen by direct calculation that the X so generated has the correct distribution. Various other computational tasks may be carried out in a similar way, all exploiting the sparse Cholesky factorisation; these include calculations of log-likelihoods of samples, and the marginal log-likelihood of the data; see [Wilkinson and Yeung \(2001\)](#) for further details.

Sampling from non-DAG Gaussian models is also possible using a similar approach. For example, sampling from Gaussian Markov Random Field (GMRF) models is described in [Rue \(2001\)](#), and implemented in a software library, `GMRFSim` (<http://www.math.ntnu.no/~hrue/GMRFSim/>) which is available from the author.

5 GDAGsim

Despite being written in 'C', GDAGsim has an object-oriented (O-O) style, making for a very clean interface. In particular, the interface has been designed so that it may be “wrapped up” for calling from a very high level O-O language such as `Python` (<http://www.python.org/>) or `LISP-STAT` (<http://www.stat.umn.edu/~luke/xls/xlsinfo/xlsinfo.html>).

The GDAGsim software library works by building the (sparse) precision matrix for the latent (unobserved) variables, and then computing with it using “off-the-shelf” sparse matrix algorithms. The software interface has been designed so as to make user code completely independent of the underlying sparse matrix library, so that the implementation may be changed without requiring any changes to user code. The current implementation is practical for the MCMC analysis of problems involving up to 10,000 latent variables, but a parallel version is planned, which (it is hoped) will allow MCMC analysis of problems containing over 100,000 latent variables.

As well as relying on a sparse matrix library (currently, the sparse section of the [Meschach](ftp://ftpmaths.anu.edu.au/pub/meschach/meschach.html) (<ftp://ftpmaths.anu.edu.au/pub/meschach/meschach.html>) matrix library is used), the software also relies on [The GNU Scientific Library \(GSL\)](http://sources.redhat.com/gsl/) (<http://sources.redhat.com/gsl/>), for general numerical routines. The dependencies between the software components are illustrated in [Figure 3](#), which shows that users make calls only to GDAGsim and the GSL, and *not* to the sparse matrix library, which is only called indirectly via GDAGsim.

There are two fundamental “objects” in GDAGsim; one representing sparse vectors, and another representing a GDAG. Note that there are no objects representing sparse matrices that are visible to the user. The basic procedure for using the library is as follows:

- Initialise DAG object
- Add latent structure to DAG object (X)
- Add observations to DAG object ($y|X$)
- “Process” DAG object ($\rightarrow X|y$)
- Access features
 - conditional moments: $E(X|y)$, $\text{var}(X_i|y)$
 - samples: x , from $(X|y)$
 - likelihoods: $[x|y]$, $[y]$

The objects and associated function calls are now examined in more detail, but see the software documentation ([Wilkinson 2001](#)) for precise syntax.

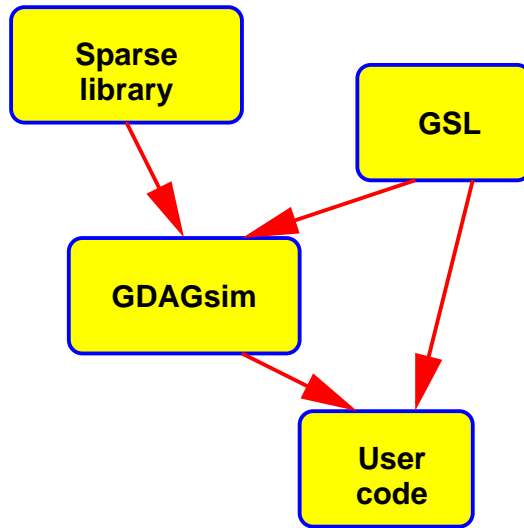


Figure 3: Dependencies (in terms of function calls) between the various software components relevant to `GDAGsim`.

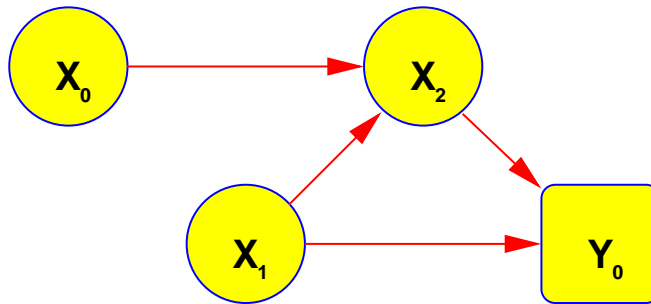


Figure 4: DAG for the simple example model

5.1 Model construction

The model construction process is best illustrated in the context of a very simple example. Here, a completely artificial example is chosen, involving 3 latent process variables and a single observable. The conditional independence structure underlying the model is given by the DAG shown in Figure 4. Precise specifications of the associated conditional distributions are given as they are required.

The two ‘C’ structs, `gdag` and `gsl_usv`, and the `GDAGsim` function templates are all given in the include file, `gdag.h`. All `GDAGsim` functions begin `gdag_`, minimising name-space pollution. If any sampling is to be done, the `GDAGsim` random number stream must be initialised with

```
gdag_rng_init();
```

5.1.1 Sparse vectors

Associated with each variable is a sparse vector α , relating the variable to its parents. Thus, the `GDAGsim` library has a struct and associated functions for building sparse vectors.

For example, to build the sparse vector $(0, 11, 0, 13, 14, 0, 0, \dots)'$, which has 3 non-zero elements, the commands

```
gsl_usv * alpha=gdag_usv_alloc(3);
gdag_usv_add(alpha,1,11.0);
gdag_usv_add(alpha,3,13.0);
gdag_usv_add(alpha,4,14.0);
```

can be used. Once the objects have been used, the memory associated with them can be released by making the call

```
gdag_usv_free(alpha);
```

5.1.2 GDAG structure

All specifications relating to a given DAG model are stored in a `gdag` struct. In `GDAGsim`, space requirements are determined by the number of *latent* variables; that is, the number of Gaussian variables for which observations are not available. Observations are added once all of the latent variables have been declared, and do not affect the size of the `gdag` struct required. For example, to declare and clear a 3-variable structure, the command:

```
gdag * d=gdag_calloc(3);
```

should be used. The objects can be freed with

```
gdag_free(d);
```

5.1.3 Building latent structure

There are two basic kinds of variables; root nodes (which have no parents), and other nodes (which do have parents). To build the model

$$X_0 \sim N(1, 1/2), \quad X_1 \sim N(3, 1/4), \quad (X_2 | X_0 = x_0, X_1 = x_1) \sim N(x_0 + 2x_1, 1),$$

use the commands

```
gdag_add_root(d,0,1.0,2.0);
gdag_add_root(d,1,3.0,4.0);
alpha=gdag_usv_alloc(2);
gdag_usv_add(alpha,0,1.0);
gdag_usv_add(alpha,1,2.0);
gdag_add_node(d,2,alpha,0,1);
gdag_usv_free(alpha);
```

5.1.4 Adding observations

Observed variables are added by specifying their conditional distributions, together with their observed values. Note that an index is not specified, since they are not added to the latent structure. For example, to add the observation $Y_0 = 20$, where

$$(Y_0 | X_1 = x_1, X_2 = x_2) \sim N(x_1 + x_2 + 5, 2),$$

use the commands

```
alpha=gdag_usv_alloc(2);
gdag_usv_add(alpha,1,1.0);
gdag_usv_add(alpha,2,1.0);
gdag_add_observation(d,alpha,5,0.5,20);
gdag_usv_free(alpha);
```

5.2 Structure analysis

Once all latent variables have been specified, and any observations have been added to the structure, analysis of the resulting MVN distribution is possible. First the “process” function is called (which, *inter alia*, forms the Cholesky decomposition of the precision matrix).

```
gdag_process(d);
```

Now various analyses are possible. If the mean is required, this can be obtained with

```
gsl_vector * mean=gdag_mean(d);
```

If samples from the distribution are required, these can be obtained with

```
gsl_vector * sample=gdag_sim(d);
```

and if the log-likelihood of the last sample generated is required, this can be obtained with

```
double ll=gdag_loglik(d);
```

If the variance of the random quantity $\alpha'X$ is required (conditional on the observations, but marginalised over other latent variables), this can be obtained with

```
double var=gdag_var(d,alpha);
```

For other possibilities, see [Wilkinson \(2001\)](#).

6 Application to Bayesian inference

`GDAGsim` allows the construction of the distribution $(X|\theta,y)$. It will then generate samples from this distribution, and evaluate associated log-likelihoods. Of course, in the context of Bayesian inference, interest usually lies in the distribution $(\theta,X|y)$. `GDAGsim` can be used to develop MCMC algorithms for exploring this distribution that are much more efficient than naive univariate Gibbs sampling algorithms. For example, the classic data augmentation algorithm ([Tanner and Wong 1987](#)) consists of alternately sampling from the distributions $(\theta|x,y)$ and $(X|\theta,y)$. This latter stage, which is usually the most difficult, can be accomplished using `GDAGsim`. The basic procedure is as follows.

- Initialise parameters (θ)
- Initialise DAG object
- Main MCMC loop
 - Reset DAG object
 - Add latent variables $(X|\theta)$
 - Add observations $(y|\theta,X)$
 - Sample x from $(X|\theta,y)$
 - Sample θ from $(\theta|x,y)$

Such a scheme will usually result in a Markov chain with better mixing properties than a simple Gibbs sampler. However, if there are strong correlations between θ and X , the sampler may still perform poorly. One possible strategy for improving the sampler is to “integrate out” the latent variables X from the MCMC state variable completely, so that analysis concerns only the marginal posterior $(\theta|y)$. Here, `GDAGsim` is used to compute the marginal likelihood $[y|\theta]$, which is needed as part of the Metropolis Hastings acceptance probability for a proposed update of θ . This “marginal updating” scheme has the following form.

- Initialise parameters (θ)
- Initialise DAG object
- Main MCMC loop
 - Reset DAG object
 - Propose new θ^* based on θ
 - Add latent variables ($X|\theta^*$)
 - Add observations ($y|\theta^*, X$)
 - Evaluate $[y|\theta^*]$
 - Accept/reject $\theta \rightarrow \theta^*$ update

`GDAGsim` has a function (`gdag_mloglik`) for evaluating the marginal likelihood required in the acceptance probability. See [Wilkinson and Yeung \(2001\)](#) for further details of these schemes and an illustration of their performance on an example problem.

7 Discussion

`GDAGsim` allows straightforward construction of efficient MCMC samplers for a large range of linear Gaussian systems. In particular, it allows a shift of focus from computational difficulties to substantive modelling issues. The software could be improved in a number of ways. First, it currently does not attempt to permute the precision matrix before forming the Cholesky decomposition. This is potentially problematic, since the Cholesky factors are not as sparse as the original matrix, and the proportion of additional non-zero elements (the “fill-in”) depends strongly on the ordering of the variables. Heuristic algorithms for computing “fill-reducing” permutations are available, and so automatic deployment of these should improve overall performance. A parallel version of `GDAGsim` is also planned, based around the [PSPASES](http://www-users.cs.umn.edu/~mjoshi/pspases/) (<http://www-users.cs.umn.edu/~mjoshi/pspases/>) parallel direct solver. This should provide performance improvements of more than an order of magnitude on relatively inexpensive parallel PC clusters, and potentially much greater improvements for specialist parallel super-computers. Since the `GDAGsim` interface will remain unchanged, this will provide opportunities for the development of high-performance algorithms by programmers unfamiliar with parallel programming techniques.

References

- Cressie, N. (1993). *Statistics for Spatial Data* (second ed.). New York: Wiley.
- Rue, H. (2001). Fast sampling of Gaussian Markov random fields. *Journal of the Royal Statistical Society B*:63(2), xx–xx.

- Tanner, M. A. and W. H. Wong (1987). The calculation of posterior distributions by data augmentation. *Journal of the American Statistical Association* 82(398), 528–540.
- Wilkinson, D. J. (2001). GDAGsim 0.2 user guide. Statistics Preprint STA01,1, University of Newcastle.
- Wilkinson, D. J. and S. K. H. Yeung (2001). A sparse matrix approach to Bayesian computation in large linear models. Statistics Preprint STA01,2, University of Newcastle.
- Wilkinson, D. J. and S. K. H. Yeung (2002). Conditional simulation from highly structured Gaussian systems, with application to blocking-MCMC for the Bayesian analysis of very large linear models. *Statistics and Computing* 12(x), xx–xx. To appear.