DSC 2003 Working Papers
(Draft Versions)
http://www.ci.tuwien.ac.at/Conferences/DSC-2003/

JAGS: A program for analysis of Bayesian graphical models using Gibbs sampling

Martyn Plummer *

Abstract

JAGS is a program for Bayesian Graphical modelling which aims for compatibility with Classic BUGS. The program could eventually be developed as an R package. This article explains the motivations for this program, briefly describes the architecture and then discusses some ideas for a vectorized form of the BUGS language.

1 BUGS

BUGS (Bayesian inference Using Gibbs Sampling) is a program for analyzing Bayesian graphical models via Markov Chain Monte Carlo (MCMC) simulation [1]. It is aimed at applied statisticians who have a problem that does not fit into a standard model class, but who do not wish to write their own custom software. BUGS fills an important role by making the analysis of problems with various sources of "complexity" such as random effects, measurement error and missing data accessible to a wide audience.

The BUGS project has promoted the Bayesian approach not only through the BUGS program, but also through the accompanying documentation, which includes a large collection of worked examples taken from the literature, and practical advice on model construction and criticism. The development team have also used BUGS as a platform for exploring new ideas in Bayesian modelling. Recently they proposed the "Deviance information criterion" [2], a Bayesian analogue of the Akaike Information Criterion, which has been implemented in WinBUGS and will most likely become very popular as a result. More speculative ideas, such as "cuts" in the graph, which allow only a one-way flow of information in the graph, have also been implemented.

1.1 History of BUGS

The original form of BUGS (now referred to as "classic BUGS") had a simple command line interface that could used interactively or from a script. Classic BUGS had a very limited capacity to analyze the output of the Markov chain. The customary usage was to dump selected output from the chain to a file and then analyze the contents with an output processor, such as the S-PLUS library CODA [3], which is also available as an R package.

In 1997 classic BUGS was superseded by WinBUGS. WinBUGS includes a powerful GUI which allows a model to be defined by literally drawing the graph on a "doodle". The GUI also includes a compound document interface which allows the model, the data and the output to

*International Agency for Research on Cancer, Lyon, France

be stored together in a single document. WinBUGS has built in capabilities for summarizing the output of the chain in graphical or tabular form, and these summaries can be incorporated into a compound document. WinBUGS is therefore a much more comprehensive package, and has a much reduced need for an output processor.

Two capabilities were lost in the transition from classic BUGS to WinBUGS. The first was the ability to run on platforms other than Windows. Classic BUGS was written in MODULA-2 and could be compiled on some versions of Unix using the Gardens Point Modula compiler (http://www.citi.qut.edu.au/projects/). WinBUGS is written in Component Pascal and depends on the Black Box component framework (http://www.oberon.ch) which is only available for Windows. A second capability that was lost in the transition to WinBUGS was the ability to run a model from a script. However, a new scripting language has been introduced in the latest version (WinBUGS 1.4) and solutions have been developed to run WinBUGS remotely from Splus, R, SAS, Matlab and Python (http://www.mrc-bsu.cam.ac.uk/bugs/winbugs/remote14.shtml).

2 JAGS: Just Another Gibbs Sampler

2.1 Motivation

There are two distinct motivations for wanting to clone BUGS. The first is to have the ability to modify the program. This is a typical motivation for free software. As an example, I have been exploring, with Vincenzo Bagnardi of the University of Milan, multi-state Markov models for analyzing panel data. Such models can be analyzed with the R package msm [4], but the attraction of the Bayesian graphical modelling approach is the ability to adapt the analysis to complex study designs. Bayesian analysis of multi-state Markov models has been considered, in an epidemiological context, by Sharples [5] and Guihenneuc-Jouyaux *et al* [6], and in the context of actuarial statistics by Bladt *et al* [7]. In order to represent such models in the BUGS language, two new distributions are required:

- A discrete distribution describing the probability of being in state j at time t, given that the subject was in state i at time 0.
- A survival distribution describing the time to an absorbing state (e.g. death), which is known exactly, or may be right censored.

Our aim is to incorporate these new distributions into JAGS and so avoid having to write a new program for each application.

A second motivation for JAGS is to create a platform for exploring new methodological ideas and, in particular, tools for criticism of graphical models. Currently, the ability to build complex graphical models has outstripped our ability to criticize them. I have developed some diagnostics for local sensitivity of reference priors [8] and some criteria for model choice [9]. In order to be useful, however, these developments need a reference implementation, and one of the purposes of JAGS is to provide a platform for such implementations. This need was brought home to me by the following extract from a rejection letter:

> There are also concerns about ... the computing time required to obtain the criteria ... You have an interesting and novel proposal but further research is required before your criteria will be used in practice.

2.2 Goals

The current goal of JAGS is to be able to analyze all of the examples provided with classic BUGS. There are several reasons for choosing classic BUGS, rather than WinBUGS, as a target. Firstly, classic BUGS has not been developed for 5 years. Consequently, it is a lost simpler and is not a moving target. Secondly, classic BUGS comes with a large collection of example scripts that can be run on Linux (my platform of choice) and which can therefore be used as a test suite. A third reason for choosing classic BUGS concerns the BUGS language itself. This will be explained in section 3.

JAGS is written in C++ allowing an object-oriented style, which is extremely useful in this context. At the same time, it allows existing software written in C and Fortran to be

used. Indeed, the project would not be possible without the library Rmath, which provides random number generators and the d-p-q-r functions for calculating the density, quantile, distribution function and random numbers for many univariate distributions.

2.3 Architecture

2.3.1 Graphs and Nodes

The fundamental object in a graphical model is a node, which represents a variable (observed or unobserved) in the model. Nodes have children and parents, and a dimension attribute. Several subclasses of node are defined that should be familiar to all users of BUGS:

Stochastic nodes which have a distribution whose parameters are determined by the node's parents.

Logical nodes that are deterministic functions of their parents.

Constant nodes that have a fixed value with no associated distribution.

In addition, there is are two subclasses which are less obvious to the user, but still necessary:

Array Nodes are containers that may be tiled with other nodes.

Subset Nodes are derived from other nodes by subscripting. The dimension of a subset node is fixed, but the subscripts may be stochastic.

A Graph is a container class for nodes. The Graph class has member functions to determine the properties of the graph: such as whether it is closed (with no links to nodes outside the graph) or whether is it acyclic.

2.3.2 Models, Samplers and Monitors

A model associates a graph with a set of samplers and a set of monitors.

A sampler defines a method of updating a graph. At the moment, the only defined samplers are Gibbs samplers, which act on a single node. The work horse function for Gibbs sampling is the Adaptive Rejection Metropolis sampler, which uses software written by Wally Gilks.

A monitor records the sampled values and summarizes them. Currently, the only monitor available is a "trace monitor" that records the sampled value every nth iteration. The trace monitor can dump its contents to an output stream.

2.3.3 The scanner, parser and compiler

The scanner, parser and compiler work together to read the data files and the file that describes the model in the BUGS language. Development of the scanner and compiler is greatly facilitated by the Unix tools lex and yacc (which I have used in their GNU forms flex and bison). The parser creates several tree structures which are interrogated by the compiler to create the graph.

2.3.4 The console

The console aims to reproduce the basic scripting language of classic BUGS so that the sampler can be run in batch mode. Again, it is implemented using flex and yacc.

2.4 Current status and future plans

I am currently working toward the first milestone of running the examples in vol 1 of the BUGS manual [10]. The second milestone is to run the second volume [11]. Some additional work is required to get JAGS to print comprehensible error messages when things go wrong. When these tasks are complete, the program will be ready for release 1.0.

Future plans for the development of JAGS involve integration with R. It would be useful to have a Bayesian graphical model package for R that could do the following in a transparent way:

- Define the model in the R language
- Use R data structures

- Control and interrogate the sampler from R
- Return the sampled values to R for summary and analysis.

These goals could be best achieved by having a loadable module containing the sampler and using the R foreign language interface. Aside from the technical issue of integration between JAGS and R, there is an important issue of how the model is described. Before modifying JAGS as an R package I would like to explore some changes to the BUGS language. The remainder of this paper is concerned with some ideas for modification.

3 A vectorized language for Bayesian graphical models

The BUGS language offers a flexible way of describing graphical models. However, if the ultimate goal is to be able to describe a Bayesian graphical model within R, then the BUGS language has a shortcoming. It relies heavily on subscripting, and this does not fit well with the principle of computing on "whole objects" which is fundamental to the S language.

To be more specific, there are two distinct uses of subscripting in the BUGS language. The first, and most common, is to represent structures that are identically calculated or distributed, via a "for" loop. The second use is to define the value of one node in terms of another via *nested indexing*. The first usage can be thought of as iteration and the second as recursion. I believe that the iterative use of indexing can be avoided with some fairly minor changes to the BUGS language.

I admit that these ideas are rather speculative. They need to be implemented before they can be tested properly.

3.1 The classic BUGS language

The first step in the construction of a Bayesian graphical model is to describe the conditional independence relationships between the various quantities (observed and unobserved) using a directed acyclic graph (DAG). The specification of the model is completed by providing a probability distribution that is Markov on this graph. The complex joint distribution is thus broken down into a set of simpler parent-child relationships. The BUGS language is a declarative language for describing these parent child relationships that is loosely based on S.

A full description of the BUGS language can be found in the classic BUGS manual [12], but some of the salient features can be illustrated by the simple linear regression model that is supplied with the BUGS program [10].

```
model line;
const  N = 5;  # number of observations
var    x[N], Y[N], mu[N], alpha, beta, tau, x.bar;
data   x, Y in "line.dat";
inits in "line.in";
{
   for (i in 1:N) {
      mu[i] <- alpha + beta*(x[i] - x.bar);
      Y[i]   ~ dnorm(mu[i], tau);
   }
   x.bar <- mean(x[]);
   alpha  ~ dnorm(0.0, 1.0E-4);
   beta   ~ dnorm(0.0, 1.0E-4);
   tau    ~ dgamma(1.0E-3, 1.0E-3);
}
```

The model statement gives the name of the model. This is followed by declaration of constants. The variables in the model are then declared. These may be scalars (e.g. alpha) or arrays (e.g. x[N]). The next two lines read in the data and initial values. The last part of the model specifications lists the relations between the nodes which may be stochastic ($\sim$) or deterministic ($\leftarrow$). A peculiarity of the BUGS language is that there is no distinction between

data and parameters in the specification of the model. Whether or not a node is regarded as data depends entirely on whether values are supplied for it in the data file.

It is worth noting that the language has been modified in WinBUGS in several ways. Constants have been abolished: these are now defined in the data file (a variable that appears in the data file but not on the left hand side of a relation must be constant node). There is no need to declare the variables since the variable names and their dimensions can be inferred from the relations.

3.2 For loops

The existence of "for" loops in the BUGS language is somewhat incongruous, since it is a declarative language, not a procedural one. For loops can be thought of as a kind of macro that succinctly describes repetitive structures. For example, the loop:

```
for (i in 1:2) {
  mu[i] <- alpha + beta*(x[i] - x.bar);
  Y[i]   ~ dnorm(mu[i], tau);
}
```

can be thought of as expanding to

```
mu[1] <- alpha + beta*(x[1] - x.bar);
Y[1]   ~ dnorm(mu[1], tau);
mu[2] <- alpha + beta*(x[2] - x.bar);
Y[2]   ~ dnorm(mu[2], tau);
```

3.3 Named dimensions

"For" loops can be eliminated by using the notion of a named dimension. The idea is rather simple and requires only a minor change to the BUGS language. The idea is that declaration of constants is replaced with declaration of the dimension names and their corresponding sizes. When array nodes are declared, the extent of the array must be given in terms of a previously defined dimension. This can be combined with a number of syntactic rules to allow unambiguous relations to be defined between arrays of nodes.

3.4 Logical relations

If two arrays in an expression are conforming (i.e. their dimensions match), then operations on the arrays are performed element-wise. When dimensions do not match, the expression is replicated to fill out the missing dimensions (essentially taking the outer product). For example

```
dim FOO = 3, BAR = 4;
var x[FOO], y[BAR], z[FOO, BAR];
{
  z <- x + y;
}
```

is equivalent to

```
{
  for (i in 1:3) {
    for (j in 1:4) {
      z[i,j] <- x[i] + y[j];
      }
    }
  }
}
```

Two arrays are conforming if they have the same dimensions in a different order. This means that

```
dim FOO=10, BAR=20;
var x[FOO, BAR], y[BAR, FOO];
{
  y <- x + 1;
}
```

is allowed, and is equivalent to

```
{
  for (i in 1:10) {
    for (j in 1:20) {
      x[i, j] <- y[j, i] + 1;
      }
    }
}
```

A corollary of this flexibility is the restriction that an array may not have two dimensions that are the same, unless the it is symmetric in those two dimensions. This allows the use of

```
var V[FOO, FOO];
```

to declare a variance matrix, for example. But square matrices that are not symmetric must have distinct dimensions. For example if P is a misclassification matrix (a non-negative matrix whose row sum to 1), then it must be declared as

```
dim TRUE=5, OBSERVED=5;
var P[TRUE, OBSERVED];
```

3.5 Stochastic relations

Stochastic relations are replicated in the same way as logical relations, but the replicates are (*a priori*) independent. For example

```
dim FOO = 10;
var y[FOO];
{
  y ~ dnorm(0.0, 1.0E-3);
}
```

is equivalent to

```
{
  for (i in 1:10) {
    y[i] ~ dnorm(0.0, 1.0E-3);
  }
}
```

To give a complete example, the "line" example above could be expressed as:

```
dim  N = 5;
var x[N], Y[N], mu[N], alpha, beta, tau, x.bar;
{
  mu <- alpha + beta*(x - x.bar);
  Y   ~ dnorm(mu, tau);
  x.bar <- mean(x);
  alpha ~ dnorm(0.0, 1.0E-4);
  beta ~ dnorm(0.0, 1.0E-4);
  tau ~ dgamma(1.0E-3, 1.0E-3);
}
```

3.6 Discussion

The notion of using named dimensions comes from two sources. The first source is the convention used in the S language that shorter vectors are replicated, when necessary, to the length of a longer vector. The second source is the "method of dimensions" used in physics to check the integrity of equations.

Vectorization of the BUGS language comes at the price of having to declare dimension names and to associate each array in the model with a set of dimensions. Hence this modification uses two features of the language that have been eliminated in WinBUGS. This explains why I am not attempting to replicate the WinBUGS language.

For an eventual R interface, the declarations could be dropped from the model definition if the dimensions, and their sizes, were attributes of R data structures that could be interrogated by the sampler. Arrays in R already have a dimension attribute. It suffices to make the dimnames attribute a named vector. This appears to be perfectly legal

```
R> a <- matrix(1:4, 2, 2)
R> dimnames(a) <- list(foo=NULL, bar=NULL)
R> a
      bar
foo    [,1] [,2]
  [1,]    1    3
  [2,]    2    4
```

although the names are not necessarily preserved by operations on arrays (e.g. t() will drop them). Whether it is worth the price of adding extra attributes to R objects in order to simplify the model remains to be seen.

This work is very much in progress, and some of the ideas expressed here may have to be revised. The aim of this project is not to offer a comprehensive library for MCMC, like the HYDRA library [13], but to offer a simple drop-in replacement for classic BUGS, and to be able to explore ideas in Bayesian modelling.

One topic that I have not touched on is speed. I have not made any attempt to optimize JAGS, so it is too early to say if it will be competitive with BUGS in this respect. Ultimately this will determine whether it becomes widely used or remains a personal research tool.

References

[1] Spiegelhalter D, Thomas A, Best N, and Lunn D. *WinBUGS 1.4 manual*, 2002.

[2] Spiegelhalter D, Best N, Carlin B, and van der Linde A. Bayesian measures of model complexity and fit (with discussion). *Journal of the Royal Statistical Society, Series B*, 64:583–640, 2002.

[3] Best NG, Cowles K, and Vines SK. *CODA: convergence diagnosis and output analysis software for Gibbs sampling output, vesion 0.3.* MRC Biostatistics Unit, Cambridge, 1995.

[4] Jackson CH, Sharples LD, Thompson SG, Duffy SW, and Couto E. Multi-state models for disease progression with classification error. *JRSS (D)*, (in press).

[5] Sharples L. Use of the Gibbs sampler to estimate transition reates between grades of coronary disease following cardiac transplantation. *Statistics in Medicine*, 12:1155–1169, 1993.

[6] Guihenneuc-Jouyaux C, Richardson S, and Longini Jr IM. Modelling markers of disease progression by a hidden Markov process: application to characterizing CD4 cell decline. *Biometrics*, 56:733–741, 2000.

[7] Bladt M, Gonzalez A, and Lauritzen SL. The estimation of phase-type related functionals using Markov Chain Monte Carlo methods. *Scandinavian Actuarial Journal*, (in press).

[8] Plummer M. Local sensitivity in Bayesian graphical models. http://www-fis.iarc.fr/∼martyn/papers.

[9] Plummer M. Some criteria for Bayesian model choice. http://www-fis.iarc.fr/∼martyn/papers.

[10] Spiegelhalter D, Thomas A, Best N, and Gilks W. *BUGS 0.5 Examples, Volume 1.* MRC Biostatistics Unit, 1996.

[11] Spiegelhalter D, Thomas A, Best N, and Gilks W. *BUGS 0.5 Examples, Volume 2.* MRC Biostatistics Unit, 1996.

[12] Spiegelhalter D, Thomas A, Best N, and Gilks W. *BUGS: Bayesian inference using Gibbs sampling, version 0.5.* MRC Biostatistics Unit, Cambridge.

[13] Warnes GR. HYDRA: A Java library for Markov Chain Monte Carlo. In Hornik K and Leisch F, editors, *DSC 2001. Proceedings of the 2nd International Workshop on Distributed Statistical Computing*, March 2001.