Bibliography

A. Gelfand and A. Smith. Sampling-based Approaches to Calculating Marginal Densities. *Journal of the American Statistical Association*, 85:398–409, 1990. 15

Insightful Corporation. *S-PLUS 6.2*. Insightful Corporation, Seattle, WA, USA, 2004. URL http://www.insightful.com. 14

M. Plummer. *JAGS: Version 0.90 manual*, 2005. URL http://www-ice.iarc.fr/~martyn/software/jags/. 13

M. Plummer, N. Best, K. Cowles, and K. Vines. *coda: Output Analysis and Diagnostics for MCMC*, 2005. R package version 0.10-3. 15

D. Spiegelhalter, N. Best, B. Carlin and A. van der Linde. Bayesian Measures of Complexity and Fit. *Journal of the Royal Statistical Society/B*, 64:583–639, 2002. 16

D. Spiegelhalter, A. Thomas, N. Best, and D. Lunn. *WinBUGS: User Manual, Version 2.10*. Medical Research Council Biostatistics Unit, Cambridge, 2005. 12, 15

S. Sturtz, U. Ligges, and A. Gelman. R2WinBUGS: A Package for Running WinBUGS from R. *Journal of Statistical Software*, 12(3):1–16, 2005. URL http://www.jstatsoft.org/. 13, 14

A. Thomas. *BRugs User Manual, Version 1.0*. Dept of Mathematics & Statistics, University of Helsinki, 2004. 12, 15

C. Weihs and U. Ligges. Parameter Optimization in Automatic Transcription of Music. In M. Spiliopoulou, R. Kruse, A. Nürnberger, C. Borgelt, and W. Gaul, editors, *From Data and Information Analysis to Knowledge Engineering*, pages 740–747, Berlin, 2006. Springer-Verlag. 15

Andrew Thomas, Bob O'Hara
Department of Mathematics & Statistics
University of Helsinki, Finland
ant@rni.helsinki.fi, bob.ohara@helsinki.fi

Uwe Ligges, Sibylle Sturtz
Fachbereich Statistik, SFB475
Universität Dortmund, Germany
<ligges,sturtz>@statistik.uni-dortmund.de

The BUGS Language

by Andrew Thomas

The BUGS language is a computer language not unlike the S language (Becker et al., 1988) in appearance, but it has a very different purpose.

Statistical models must be described before they can be used. A language to describe statistical models is needed by both the users of the model and the software that makes inference about the model. The language should be a formal language with well defined rules which can be processed automatically. It should not be concerned with the technology used to make inference about the model. We have developed a model description language called the BUGS language because of its use in the **B**ayesian inference **U**sing **G**ibbs **S**ampling (OpenBUGS) package. However, the BUGS language can be used outside the OpenBUGS software. For example, it is used in the JAGS package (Plummer, 2005) and has influenced other packages such as Bassist (Toivonen et al., 1999) and AUTOBAYES (Fisher and Schumann, 2003).

We choose to describe statistical models in terms of a joint probability distribution. Model description in terms of a joint probability distribution is both very general and very explicit. We consider these good points. We do not consider it a good idea to have a patchwork of specialized (maybe very elegant) notations for different types of model. We want to be able to combine small submodels to build larger models using a consistent notation. A small change to a model should not lead to a large change in the way that model is described. Examples of small changes to the model are: choice of sampling distribution, form of regression, covariate measurement error, missing data, interval censoring, *etc*. Explicitness is important in a model description language. There should be no doubt if two models are the same.

We hope the BUGS language will be useful to anyone who uses complex statistical models, and even to people who do not want to use the OpenBUGS package to make inference. Is the BUGS language really about statistics? OpenBUGS has many users who do not think of themselves primarily as statisticians, who are mainly interested in the deterministic skeleton of a model. We think that if a probabilistic model is used to explain observations, given this deterministic skeleton, that this is a form of statistics.

Influences

Formal languages have rules both for syntax and for semantics. For the BUGS language, syntax has been influenced by the S language (Becker et al., 1988) and

semantics by graphical models. A model described in the BUGS language looks like a piece of S code but the meaning is completely different. The BUGS language is declarative. It describes static relations beween quantities, not how to do calculations.

Many joint probability distributions can be written as a product of factors. This leads to a graphical notation for describing joint probability distributions. Each factor in the joint probability distribution is a function of several variables. It is possible to order these variables for each factor so that the factor is represented by a node in a directed acyclic graph (DAG) labeled by one of the variables of the factor with the remaining variables being parents of the node in the graph. Describing the DAG is equivalent to describing the joint probability distribution. A DAG can be described by specifying the parents of each node. In the simplest case, the factors are probability distribution functions whose parameters are given by the values of the node's parents. In the more general case the parameters of the distribution will be functions of the values of the node's parents.

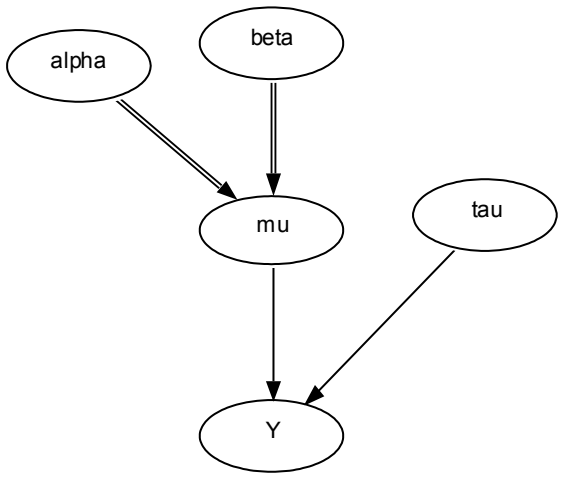

Figure 1: A simple directed acyclic graph

Consider the small graph in figure 1. This represents the joint probability distribution

```
P1(Y|mu, tau) P2(alpha) P3(beta) P4(tau)
```

where `P1`, `P2`, `P3` and `P4` are probability distributions associated with nodes in the graph. Nodes with solid arrows pointing into them represent stochastic relations and those with hollow arrows logical relations. Hence `mu` is some function of the values of `alpha` and `beta`.

Example 1: Growth curve in rats

An example of a simple model is the hierarchical linear growth curve model considered by Gelfand and Smith (1990). This model has a simple representation as a DAG (drawn with the DoodleBUGS editor), shown in figure 2. The plate, a rectangular box with four parallel lines along its bottom and right edges, is used as a metaphor for repetition.

To describe the DAG in the BUGS language, a textual language, we need two types of relation: stochastic relations and logical relations. The stochastic relations tell which probability distribution function is associated with which node in the model. The logical relations define how to calculate the values of the parameters of the probability distribution functions in terms of the values of the node's parents. For stochastic relations we use the tilde (`~`) as the relational operator and for logical relations the left pointing arrow (`<-`). A final element in the BUGS language is a notation for repetition. We use the notation

```
for (i in M:N) { ... }
```

where the statements between the braces are duplicated with the place holder `i` replaced by the integer values `M` through `N`. Comments in the BUGS language are any characters that follow the hash sign (`#`) up to the end of the line.

Written in the BUGS language, our example is

```
model
{
    for (i in 1:N) {
        for (j in 1:T) {
          Y[i,j] ~ dnorm(mu[i,j],tau.c)
          # linear growth curve
          mu[i,j] <- alpha[i]+beta[i]*(x[j]-xbar)
        }
        alpha[i] ~ dnorm(alpha.c,alpha.tau)
        beta[i] ~ dnorm(beta.c,beta.tau)
    }
    tau.c ~ dgamma(0.001,0.001)
    sigma <- 1/sqrt(tau.c)
    alpha.c ~ dnorm(0.0,1.0E-6)
    alpha.tau ~ dgamma(0.001,0.001)
    beta.c ~ dnorm(0.0,1.0E-6)
    beta.tau ~ dgamma(0.001,0.001)
    alpha0 <- alpha.c-xbar*beta.c
}
```

We make some comments about this model. The data `Y` consists of the weight of `N` rats measured at `T` time points. A linear model is fitted for each rat. The slope and intercept for each rat are drawn from normal distributions with unknown hyper parameters `alpha.c`, `alpha.tau`, `beta.c` and `beta.tau`. These hyper parameters are given vague priors.

The parameterization used by each distribution must be documented. For example, the `dnorm` distribution parameterizes the normal distribution in terms of its mean and precision (the reciprocal of the variance), not the standard deviation. Logical nodes

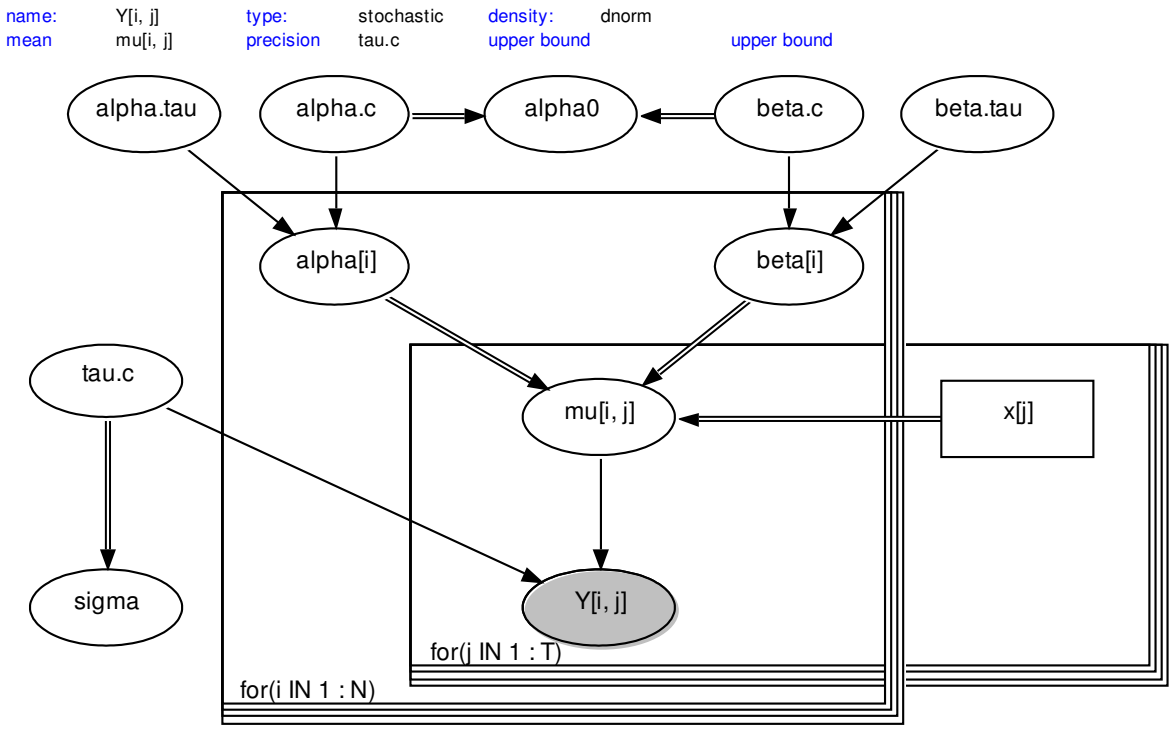

Figure 2: Directed Acyclic Graph for a hierarchical linear growth model

can be added to the model to calculate functions of stochastic nodes, for example `sigma` and `alpha0` in this model. We can easily change the model: the distribution of the data `Y` can be changed from the normal (`dnorm`) to, say, the t distribution (`dt`) to allow for outliers, or the linear growth curve relation for `mu` could be changed to a non-linear one, *etc*.

Example 2: Biopsy data

A slightly more complex model is the biopsy data considered by Spiegelhalter and Stovin (1983). In this model, the state of an internal organ (the heart) is probed by taking tissue samples with a hollow needle. The true (latent) state of the organ is at least as bad as the worst category of the tissue sample taken. Multiple tissue samples are taken from each organ giving rise to multinomial data. The probability vector of proportions in the multinomial is modeled as a mixture of Dirichlet distributions with the constraint that elements of the error matrix above the leading diagonal are zero (no false positives). This model is quite difficult to draw as a graph using the DoodleBUGS editor but easy to write in the BUGS language.

```
model
{
    for (i in 1:ns){
        nbiops[i] <- sum(biopsies[i,])
        true[i]  ~ dcat(p[ ])
        biopsies[i,1:4] ~
            dmulti(error[true[i],],nbiops[i])
    }
    error[2,1:2] ~ ddirch(prior[1:2])
    error[3,1:3] ~ ddirch(prior[1:3])
    error[4,1:4] ~ ddirch(prior[1:4])
    error[1,1] <- 1 error[1, 2] <- 0
    error[1,3] <- 0 error[1, 4] <- 0
    error[2,3] <- 0 error[2, 4] <- 0
    error[3,4] <- 0
    # prior for p
    p[1:4] ~ ddirch(prior[ ])
}
```

Note the use of variable indexing in the relation for biopsies: the variable `true[i]` takes a value in $\{1, 2, 3, 4\}$ and picks which row of the error matrix is used in the multinomial distribution. In the BUGS language, a variable index must always be a named quantity in the model. If the index is non variable, then an expression that evaluates to a constant can also be used. Nested indexing is allowed.

Data

Usually some of the quantities in the statistical model have fixed values: they are data. Within the BUGS language there is no distinction between quantities that are data and quantities about which inference is required. We put quantities with fixed values in a data set. The syntax we have chosen for data sets is the S list format containing scalars, vectors and multi dimensional arrays (in the form of structures).

If a quantity has both fixed components and components that need estimating, then the later will be represented as NAs in the data set. The OpenBUGS software processes both the model description language and the associated data sets to build the joint probability distribution.

Correctness

Using the BUGS language it is easy to write down a complex statistical model. But is the model correct? The model must be both syntactically and semantically correct. Checking syntactic correctness is quite easy: parsing the model will detect any errors and produce clear error messages. It is much harder to check that the description of a model in the BUGS language plus a data set (or data sets) defines a complete and consistent model. We take a constructive approach to this problem. OpenBUGS tries to compile the BUGS language into a detailed graph that represents the joint probability distribution. The completeness and consistency of this graph are then checked. This approach can detect many errors. However, users have found the error messages produced somewhat cryptic. Some typical cases of lack of consistancy are:

1. The data set defines the length of a vector quantity to be say L and the model uses a component of this vector quantity with a index greater than L.

2. Multiple definitions of a node in the model are given, for example the statement

```
for(i in 1:10){ x ~ dnorm(0, 1) }
```

Computation on the graph

A detailed representation of the graph of the model allows us to easily calculate the joint probability distribution. It also makes it easy to calculate the conditional distribution of a single node in the model, holding all other nodes fixed, in an efficient way. These single node conditional distributions are the basic building blocks of inference algorithms based on an extreme divide-and-conquer approach. Conditional distributions of blocks of nodes can be derived from the single node conditional distributions. These multi-node conditional distributions are useful for inference algorithms when the divide approach is not taken to extremes. The deviance of the model can be calculated from the distributions associated with data nodes (including censored observations) in the model. The OpenBUGS software tries to classify the functional form of the single node conditional distributions. The more detailed the classification of these single node conditional distributions, the wider the choice of algorithms that can be proved valid for statistical inference on the model. Markov Chain Monte Carlo (MCMC) simulation makes heavy use of the calculation of conditional distributions. This fact makes MCMC simulation a natural choice of inference technology to combine with the BUGS language. However other approaches to inference could be added to the OpenBUGS software.

Outlook

The BUGS language provides a uniform way of specifying complex statistical models. It allows a model to be worked on and shared by several people. The BUGS language can be used by other software that makes inference about complex models. The OpenBUGS software provides source code level access to the lexical and parsing tools used to process the BUGS language.

Different inference algorithms, for example the EM algorithm (Dempster et al., 1977), the variational algorithm (Jaakkola and Jordan, 2000) or particle filters (Doucet et al., 2001), could be built on top of the OpenBUGS software. We hope that the separation of model specification and parameter inference become more common in the future development of statistical software.

In many statistical packages, the idea of a model stays in the background; the emphasis is on fitting data. This makes interfacing the OpenBUGS software with, say, R conceptually difficult. R has data objects and functions for doing computation on data objects but not model description objects. **BRugs**, the R interface to OpenBUGS, has to read the model description from a file. This is less than ideal. In outline, the ideal way of interfacing R and OpenBUGS would be to have model description objects that could be translated into compiled model objects. Compiled model objects could then be passed to inference algorithms (MCMC etc) to give fitted model objects (for MCMC, something like an mcmc object from the **coda** package (Plummer et al., 2005)). Standard R functions could then be applied to the fitted model object to compute any derived quantity of interest. This is a long term program.

Bibliography

R. A. Becker, J. M. Chambers, and A. R. Wilks. *The new S language: A programming environment for data analysis and graphics*. Wadsworth & Brooks/Cole, Pacific Grove, Calif, 1988. 17

A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39:1, 1977. 20

A. Doucet, N. de Freitas, and N. Gordon, editors. *Sequential Monte Carlo in Practice*. Springer-Verlag, 2001. ISBN 0-387-95146-6. 20

B. Fisher and J. Schumann. Autobayes: A system for generating data analysis programs from statistical models. *Journal of Functional Programming*, 13(3): 483–508, May 2003. 17

A. Gelfand and A. Smith. Sampling-based approaches to calculating marginal densities. *Journal of the American Statistical Association*, 85:398–409, 1990. 18

T. Jaakkola and M. I. Jordan. Bayesian parameter estimation via variational methods. *Statistics and Computing*, 19:25–37, 2000. 20

M. Plummer. *JAGS Version 0.90 Manual*. International Agency for Research on Cancer, Lyon, France, September 2005. http://www-ice.iarc.fr/~martyn/software/jags. 17

M. Plummer, N. Best, K. Cowles, and K. Vines. *CODA: Output analysis and diagnostics for MCMC*, 2005. R package version 0.10-3. 20

D. J. Spiegelhalter and P. G. I. Stovin. An analysis of repeated biopsies following cardiac transplantation. *Statistics in Medicine*, 2:33–40, 1983. 19

H. Toivonen, H. Mannila, J. Seppänen, and K. Vasko. Bassist user's guide for version 0.8.3. Technical Report C-1999-36, Dept of Computer Science, University of Helsinki, 1999. http://www.cs.helsinki.fi/research/fdk/bassist/. 17

Andrew Thomas
Department of Mathematics & Statistics
University of Helsinki, Finland
ant@rni.helsinki.fi

Bayesian Data Analysis using R

by Jouni Kerman and Andrew Gelman

Introduction

Bayesian data analysis includes but is not limited to Bayesian inference (Gelman et al., 2003; Kerman, 2006a). Here, we take *Bayesian inference* to refer to posterior inference (typically, the simulation of random draws from the posterior distribution) given a fixed model and data. *Bayesian data analysis* takes Bayesian inference as a starting point but also includes fitting a model to different datasets, altering a model, performing inferential and predictive summaries (including prior or posterior predictive checks), and validation of the software used to fit the model.

The most general programs currently available for Bayesian inference are WinBUGS (BUGS Project, 2004) and OpenBUGS, which can be accessed from R using the packages **R2WinBUGS** (Sturtz et al., 2005) and **BRugs**. In addition, various R packages exist that directly fit particular Bayesian models (e.g. **MCMCPack**, Martin and Quinn (2005)). or emulate aspects of BUGS (JAGS). In this note, we describe our own entry in the "inference engine" sweepstakes but, perhaps more importantly, describe the ongoing development of some R packages that perform other aspects of Bayesian data analysis.

Umacs

Umacs (Universal Markov chain sampler) is an R package (to be released in Spring 2006) that facilitates the construction of the Gibbs sampler and Metropolis algorithm for Bayesian inference (Kerman, 2006b). The user supplies data, parameter names, updating functions (which can be some mix of Gibbs samplers and Metropolis jumps, with the latter determined by specifying a log-posterior density function), and a procedure for generating starting points. Using these inputs, Umacs writes a customized R sampler function that automatically updates, keeps track of Metropolis acceptances (and uses acceptance probabilities to tune the jumping kernels, following Gelman et al. (1995)), monitors convergence (following Gelman and Rubin (1992)), summarizes results graphically, and returns the inferences as random variable objects (see **rv**, below).

Umacs is customizable and modular, and can be expanded to include more efficient Gibbs/Metropolis steps. Current features include adaptive Metropolis jumps for vectors and matrices of random variables (which arise, for example, in hierarchical regression models, with a different vector of regression parameters for each group).

Figure 1 illustrates how a simple Bayesian hierarchical model (Gelman et al., 2003, page 451) can be fit using Umacs: $y_j \sim \mathrm{N}(\theta_j, \sigma_j^2)$, $j = 1, \ldots, J$ ($J = 8$), where σ_j are fixed and the means θ_j are given the prior $t_\nu(\mu, \tau)$. In our implementation of the Gibbs sampler, θ_j is drawn from a Gaussian distribution with a random variance component V_j. The conditional distributions of θ, μ, V, and τ can be calculated analytically, so we update them each by a direct (Gibbs) update. The updating functions are to be specified as R functions (here, `theta.update`,