STATISTICS IN MEDICINE
Statist. Med. (2009)
Published online in Wiley InterScience
(www.interscience.wiley.com) DOI: 10.1002/sim.3680

The BUGS project: Evolution, critique and future directions

David Lunn[1,*,†], David Spiegelhalter[1], Andrew Thomas[2] and Nicky Best[3]

[1]*Medical Research Council Biostatistics Unit, Institute of Public Health, University Forvie Site, Robinson Way, Cambridge CB2 0SR, U.K.*
[2]*School of Mathematics and Statistics, Mathematical Institute, North Haugh, St. Andrews, Fife KY16 9SS, U.K.*
[3]*Department of Epidemiology and Public Health, Imperial College London, St. Mary's Campus, Norfolk Place, London W2 1PG, U.K.*

SUMMARY

BUGS is a software package for Bayesian inference using Gibbs sampling. The software has been instrumental in raising awareness of Bayesian modelling among both academic and commercial communities internationally, and has enjoyed considerable success over its 20-year life span. Despite this, the software has a number of shortcomings and a principal aim of this paper is to provide a balanced critical appraisal, in particular highlighting how various ideas have led to unprecedented flexibility while at the same time producing negative side effects. We also present a historical overview of the BUGS project and some future perspectives. Copyright © 2009 John Wiley & Sons, Ltd.

KEY WORDS: BUGS; WinBUGS; OpenBUGS; Bayesian modelling; graphical models

1. INTRODUCTION

BUGS [1] is a software package for performing Bayesian inference using Gibbs sampling [2, 3]. The BUGS project began at the Medical Research Council Biostatistics Unit in Cambridge in 1989. Since that time the software has become one of the most popular statistical modelling packages, with, at the time of writing, over 30 000 registered users of *WinBUGS* (the Microsoft Windows incarnation of the software) worldwide, and an active on-line community comprising over 8000 members. Typing 'WinBUGS' into Google generates over 100 000 hits; in Google Scholar the figure is in excess of 5000; and simply searching Statistics in Medicine on-line gives nearly 100 hits.

BUGS has been just one part of the tremendous growth in the application of Bayesian ideas over the last 20 years. Prior to the widespread introduction of simulation-based methods, Bayesian ideas

*Correspondence to: David Lunn, Medical Research Council Biostatistics Unit, Institute of Public Health, University Forvie Site, Robinson Way, Cambridge CB2 0SR, U.K.
†E-mail: david.lunn@mrc-bsu.cam.ac.uk

Contract/grant sponsor: Medical Research Council; contract/grant number: U.1052.00.005

Received 31 March 2009
Accepted 19 June 2009

could only be implemented in circumstances in which solutions could be obtained in closed form in so-called conjugate analyses, or by ingenious but restricted application of numerical integration methods. BUGS has therefore been instrumental in raising awareness of Bayesian modelling among both academic and commercial communities internationally, and has greatly facilitated the routine analysis of many complex statistical problems. Numerous applications of the software can be found in the literature, in a wide array of application areas, including disease mapping [4], pharmacometrics [5], ecology [6], health-economics [7], genetics [8], archaeology [9], psychometrics [10], coastal engineering [11], educational performance [12], behavioural studies [13], econometrics [14], automated music transcription [15], sports modelling [16], fisheries stock assessment [17], and actuarial science [18]. The software is also used widely for the teaching of Bayesian modelling ideas to students and researchers the world over, and several texts use BUGS extensively for illustrating the Bayesian approach [19–26]. The importance of the software has been acknowledged in 'An International Review of U.K. Research in Mathematics' (`http://www.cms.ac.uk/irm/irm.pdf`).

Despite this apparent success, we are well aware that BUGS has some significant shortcomings and it is our intention here to provide a balanced critical appraisal of the software in its current form(s). We will also present an overview of the way in which the software has evolved, and a discussion of potential future directions for the BUGS project. We recognize that not all parts of the paper will be of interest to all readers, and so try to provide appropriate guidance.

The structure of this paper is as follows. In Section 2 we explain the intimate and vital connection between BUGS and graphical modelling, while in Section 3 we present an overview of the BUGS language: these sections could be skipped by those familiar to ideas behind the program. In Section 4 we describe how the software has evolved, which goes into more technical detail regarding the 'guts' of the program and how the different versions were developed and funded. The technical material is presented in a separate subsection (Section 4.2) and can be skipped by those who are less interested in how BUGS' capabilities have expanded over the years. Section 5 provides a critical appraisal and discusses features that should be irritatingly familiar to all those who have struggled with the program. A concluding discussion, including some future perspectives, is given in Section 6.

2. A BASIS IN GRAPHICAL MODELLING

The popularity of the software stems from its flexibility, which is due to the exploitation of graphical modelling theory. Consider a simple linear regression problem given by:

$$y_i \sim \mathrm{N}(\mu_i, \sigma^2), \quad \mu_i = \alpha + \beta x_i, \quad i = 1, \ldots, N$$

for responses y_i and observed values x_i of a single covariate ($i = 1, \ldots, N$). As we are working in a Bayesian setting we assign prior distributions to the unknown parameters, α, β and σ^2 (or σ, or $\log\sigma$, say). For example,

$$\alpha \sim \mathrm{N}(m_\alpha, v_\alpha), \quad \beta \sim \mathrm{N}(m_\beta, v_\beta), \quad \log\sigma \sim \mathrm{U}(a, b)$$

for fixed constants m_α, v_α, m_β, v_β, a and b. An alternative representation of this model is the *directed acyclic graph* (DAG; see [27], for example) shown in Figure 1, where each quantity in the model corresponds to a *node* and links between nodes show direct dependence. The graph is *directed* because each link is an arrow; it is *acyclic* because by following the arrows it is not possible to return to a node after leaving it.

DOI: 10.1002/sim

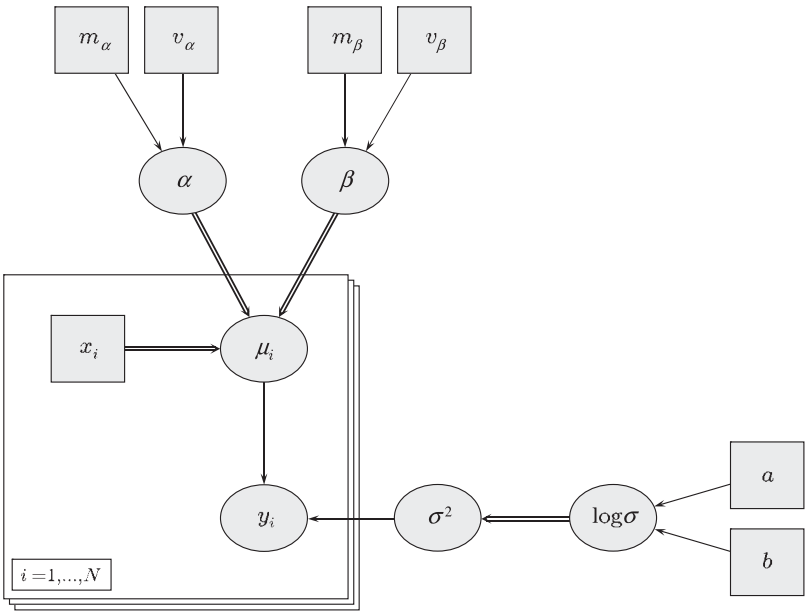

Figure 1. Directed acyclic graph (DAG) corresponding to linear regression example—see text for details.

The notation is defined as follows. Rectangular nodes denote known constants. Elliptical nodes represent either deterministic relationships (i.e. functions) or stochastic quantities, i.e. quantities that require a distributional assumption. Stochastic dependence and functional dependence are denoted by single-edged arrows and double-edged arrows, respectively. Repetitive structures, such as the 'loop' from $i = 1$ to N, are represented by 'plates', which may be nested if the model is hierarchical.

A node v is said to be a *parent* of node w if an arrow emanating from v points to w; furthermore, w is then said to be a *child* of v. We are primarily interested in stochastic nodes, i.e. the unknown parameters and the data. When identifying probabilistic relationships between these, deterministic links are collapsed and constants are ignored. Thus, the terms parent and child are usually reserved for the appropriate stochastic quantities. In the above example, the *stochastic parents* of each y_i are α, β and $\log\sigma$, whereas we can refer to μ_i and σ^2 as *direct parents*.

DAGs can be used to describe pictorially a very wide class of statistical models through describing the 'local' relationships between quantities. It is when these models become complicated that the benefits become obvious. DAGs communicate the essential structure of the model without recourse to a large set of equations. This is achieved by abstraction: the details of distributional assumptions and deterministic relationships are 'hidden'. This is conceptually similar to object-oriented programming (OOP) (see [28], for example), which realization has been instrumental in maximizing the flexibility of BUGS, as we will discuss later.

The joint distribution of all nodes V in *any* DAG can be factorized as follows [29]:

$$p(V) = \prod_{v \in V} p(v|\mathscr{P}_v) \tag{1}$$

DOI: 10.1002/sim

where $\mathscr{P}_v$ denotes the set of (stochastic) parents of node v. With regard to Bayesian inference, the data D and the unknowns θ, say, together form a partition of V, and so the joint posterior density $p(\theta|D) \propto p(V)$. For the purposes of Gibbs sampling we are interested in the full conditional distributions of each unknown stochastic node conditional on the values of all other nodes in the graph. For two arbitrary sets of nodes A and B, say, let $A \backslash B$ denote 'all elements of A except those in B'. Then the full conditional for a specific node w, say, is denoted $p(w|V \backslash w)$. As $\{w, V \backslash w\}$ is a also a partition of V, we have that the full conditional is also proportional to the right-hand side of (1). However, $p(w|V \backslash w)$ is a distribution in w only, and so we can ignore any factors not involving w, giving

$$p(w|V \backslash w) \propto p(w|\mathscr{P}_w) \times \prod_{v \in \mathscr{C}_w} p(v|\mathscr{P}_v)$$

where $\mathscr{C}_w$ denotes the children of w. The beauty of the factorization (1) is thus two-fold. First, we can write down the joint posterior for any DAG simply by knowing the relationship between each node and its parents. Second, the full conditional distribution for any node (or set of nodes) is a *local computation* on the graph, involving only the node-parent dependencies for the node of interest itself and its children. Thus, one only ever needs to consider a small part of the model at any given time, without needing to take account of the bigger picture. The BUGS software exploits these facts first by providing a language (the BUGS language) for specifying arbitrary child–parent relationships, and by 'inverting' these to determine the set of nodes relevant to each full conditional calculation: these comprise the children, parents and 'co-parents' collectively known as the 'Markov blanket'. The software also provides a mechanism whereby each node can communicate, to the 'inference engine', how it is related to its parents. With these facilities in place, Gibbs sampling on arbitrary graphs is conceptually straightforward.

3. THE BUGS LANGUAGE

As noted above, BUGS provides its own language for the textual specification of graphical models. The language is designed to mimic the mathematical specification of the model in terms of parent–child relationships. Stochastic relationships are denoted with '$\sim$' whereas logical/deterministic relationships are denoted with a '<-'. Repetitive structures are represented using 'for-loops', which may be nested if the model is hierarchical, say. The following code specifies the linear regression graph referred to in the previous section:

```
model {
   for (i in 1:N) {
     y[i]    ~ dnorm(mu[i], tau)
     mu[i] <- alpha + beta * x[i]
   }
   alpha     ~ dnorm(m.alpha, p.alpha)
   beta      ~ dnorm(m.beta, p.beta)
   log.sigma ~ dunif(a, b)
   sigma    <- exp(log.sigma)
   sigma.sq <- pow(sigma, 2)
   tau      <- 1 / sigma.sq
```


DOI: 10.1002/sim

```
  p.alpha  <- 1 / v.alpha
  p.beta   <- 1 / v.beta
}
```

The code should be reasonably self-explanatory but a few notes are required for clarification. First note that normal distributions (`dnorm(.,.)`) are parameterized in terms of precision (equal to 1/variance) as opposed to variance. This can be a source of confusion and reflects the fact that the software was originally designed to exploit mathematical conveniences (so-called conjugacy) where possible. Although this is no longer necessary and the need for an intuitive prior typically outweighs the benefit of any such conjugacy, the parameterization remains unchanged. In addition, note that the data, `y[1:N]`, `x[1:N]`, `N`, and the values of `m.alpha`, `v.alpha`, `m.beta`, `v.beta`, `a` and `b` in this case, are loaded into the system separately. Finally, note that the `pow(.,.)` function raises the first argument to the power of the second argument, and so the `sigma.sq` variable represents σ^2.

The BUGS language is a *declarative* language. This means that it does not matter in which order the various statements are made, which makes it easier for the system to interpret the model specification. The downside of this, however, is that there is no scope for 'logical' expressions such as IF–THEN–ELSE statements. There are ways around this, for example, by making use of the `step(.)` function (equal to 1/0 depending on the positive/negative status of the argument) to switch between modelling terms, but this still represents a significant limitation of the language. Despite this the language has proved enormously successful and capable of expressing a vast array of model-types.

To illustrate the flexibility afforded by the language, let us elaborate the model described above a little. First, suppose that the covariate `x[]` is subjected to measurement error. We could model this, for example, by adding the assumption `x[i]`$\sim$`dnorm(mu.x[i], tau.x)`, where the `mu.x[i]` terms are unknown and assigned an appropriate prior, e.g. `mu.x[i]`$\sim$`dnorm(m.x, p.x)` with the mean `m.x` and precision `p.x` known, and `tau.x`, for the sake of simplicity, also known. Now suppose, for instance, that the responses `y[]` represent growth data on a given individual, collected at times `x[]`. If we had such data on a number of individuals (indexed by $j=1,\ldots,K$) as opposed to just one, then we might wish to assume different individuals to have distinct but similar (exchangeable) parameters via $\alpha_j \sim \mathrm{N}(m_\alpha, v_\alpha)$, $\beta_j \sim \mathrm{N}(m_\beta, v_\beta)$, $j=1,\ldots,K$, where now m_α, v_α, m_β and v_β are unknown and assigned appropriate priors, e.g.

$$m_\alpha \sim \mathrm{N}(0, 100^2), \quad \sqrt{v_\alpha} \sim \mathrm{U}(0, 100), \quad m_\beta \sim \mathrm{N}(0, 100^2), \quad \sqrt{v_\beta} \sim \mathrm{U}(0, 100)$$

To implement this now hierarchical model we simply add lines to represent the priors for m_α, v_α, m_β and v_β, e.g. `m.aplha`$\sim$`dnorm(0, 0.0001)`, and nest any individual-specific statements within a loop over individuals (indexed by `j`):

```
for (j in 1:K) {
  for (i in 1:N) {
    y[i,j]   ~ dnorm(mu[i,j], tau)
    mu[i,j] <- alpha[j] + beta[j] * x[i]
  }
  alpha[j]   ~ dnorm(m.alpha, p.alpha)
  beta[j]    ~ dnorm(m.beta, p.beta)
}
```


DOI: 10.1002/sim

Suppose we wish to fit a non-linear growth curve, $\mu_{ij} = \alpha_j - \beta_j \gamma_j^{x_i}$, say, as opposed to a straight line. Then we would rewrite the definition of `mu[i,j]` as follows:

```
mu[i,j] <- alpha[j] - beta[j] * pow(gamma[j], x[i])
```

If the α_js and β_js are as before and we further suppose $\gamma_j \sim \mathrm{U}(0,1)$ for $j = 1, \ldots, K$, then we simply include the line `gamma[j]~dunif(0,1)` within the `j`-loop (but outside the `i`-loop). Finally, to robustify the model against any outlying observations we might wish to assume the `y[i,j]`s arise from a Student-*t* as opposed to a normal distribution. Then we would modify the assumed relationship between each `y[i,j]` and its parents to `y[i,j]~dt(mu[i,j],tau,d)`, where the degrees-of-freedom parameter `d` is either assumed to be known (e.g. `d <- 4`) or assigned an appropriate prior, Poisson, say, `d~dpois(...)`.

One of the reasons the BUGS language is so flexible is that it is open ended. It is quite straightforward to introduce new 'vocabulary', i.e. new distributions and functions, and indeed this can be done without any part of the existing software being modified or even recompiled. This is due to the object- and component-oriented [30] nature of the software, which is discussed throughout.

4. EVOLUTION OF BUGS

4.1. Inception and early years

In the early 1980s one focus of work in artificial intelligence concerned *expert systems*, which were programs designed to mimic expertise in complex situations. A basic principle was a separation of the *knowledge base* that encapsulated expertise, from the *inference engine* that controlled the response to new evidence and queries: there was agreement that a natural computer implementation of a knowledge base was in a *declarative* rather than a *procedural* form, so that the program would express local relationships between entities which could then be manipulated using the inference engine.

Where there was substantial disagreement and sometimes passionate argument was about how to deal with uncertainty. For example, extremely influential work from Stanford featured rule-based systems in which uncertainty was handled using a system of 'certainty factors' attached to rules and manipulated according to somewhat arbitrary principles—this was labeled as '*ad hoc* quantitative mumbo-jumbo' by a prominent statistician (Smith, discussion of [31]). The challenge was to come up with a more rigorous process based on probability theory but that still retained the attractive aspect of local computation. The potential role of graphical models, also to become known as Bayesian networks, was argued both in the U.S. [32] and in the U.K. [33], and exact means of propagating uncertainty in tree structures [34] and general networks [35] were developed.

Pearl [36] noted that simulation methods could also be used for approximate inference in directed graphical models, and Spiegelhalter and Lauritzen [37] showed that graphical models could also include parameters as nodes, hence facilitating general Bayesian inference. It became apparent that, by merging these ideas, the computations required for Bayesian inference were simply an iterative sequence of local calculations on the graph, and that each such calculation, viewed abstractly, was identical. OOP would therefore prove highly effective: not only does it provide a means of describing relationships between entities of different, but related, types, i.e. a *virtual graphical*

DOI: 10.1002/sim

model (VGM),[‡] but it also allows abstract computation, vastly simplifying implementation (and maintenance) of the algorithm (see [38] for details). Putting these ideas together led to the concept of a general program that could handle arbitrary models and would use simple Gibbs sampling to simulate from nodes conditional on their neighbours in the graph. A start was made on the BUGS program in 1989 with the appointment of Andrew Thomas to the MRC Biostatistics Unit: it is rather extraordinary that at the same time the classic MCMC work of Gelfand and Smith [3] was being carried out 80 miles away in Nottingham but entirely independently and from a rather different starting point.

The language chosen for implementation was Modula-2 [39]. Actually this is not an object-oriented language but it can be used to a similar effect. In addition it is modular (as one might expect from its name), which facilitates the hiding of detail, an important concept in designing complex architectures, for much the same reasons as it is effective in graphical modelling. An obvious alternative might have been C++; however, as Andrew Thomas points out '*I am unable to program in* C++. *If I was clever enough to program in* C++ *I would have a very good job in say banking.*'

A prototype on a PC was demonstrated at the 4th Bayesian Valencia meeting in April 1991 [40], and a Unix version at the Practical Bayesian Statistics meeting in Nottingham in 1992 [1]. Version 0.1 for Unix was released in February 1993 and further refinements followed until a stable Version 0.5 in 1995. Early versions were distributed on disks and a voluntary fee of £25 or $40 solicited. A large impetus was the INSERM workshop on MCMC methods in 1993 which had a follow-up workshop in Cambridge using BUGS and led to the *MCMC in Practice* book [41] (which, rather remarkably, is still selling well). Throughout this period interest in MCMC was steadily growing, and the portfolio of examples grew, including the standard lip-cancer spatial mapping example which was made to work on the day before the Basel–Amsterdam Bayesian Riverboat conference in 1993.

The name BUGS shows that only simple Gibbs sampling was used at first, and even then only using simple inversion, conjugate updating, or adaptive rejection sampling for log-concave full conditionals. The parameterizations for each distribution were chosen for computational convenience, which is why the normal ended up being parameterized in terms of precision (inverse-variance). After some time a grid-based Metropolis–Hastings algorithm [42, 43] was implemented as a somewhat crude means of performing more general sampling. However, it was not until the project moved (in 1996) to Imperial College, London, that the sampling capabilities of the software were expanded. We describe such technological developments in the following subsection, which can be skipped by uninterested readers. In short, we describe how and why a Windows version was developed, and how various modelling challenges in fields such as pharmacokinetics, disease mapping and genetic epidemiology led to improved algorithms and greater flexibility.

4.2. Technological evolution

4.2.1. A stand-alone (*Windows*) *version.* In the mid-1990s a decision was taken to create a stand-alone version of the software, one which could provide interactive diagnostic and inference tools. This led to a new choice of programming language, for two main reasons. First the Microsoft Windows operating system had been rapidly gaining massive popularity and it seemed an ideal

[‡]Simplistically speaking, a VGM uses *objects* and *pointer variables*, respectively, to represent nodes and directed links in the graph.

DOI: 10.1002/sim

environment for the kind of fully interactive software that was planned; it was also thought that we would reach a much wider audience through this medium. Second, the parallels between OOP and graphical modelling were becoming more and more apparent and it was thought that these could be better exploited. In addition, the use of a *component-oriented* framework [30] would facilitate the growth of the BUGS-language-vocabulary, by allowing new distributions and functions to be implemented with relative ease.

The programming language chosen was Component Pascal [44], implemented within a *Rapid Application Development* environment known as *BlackBox Component Builder* (`http://www.oberon.ch/blackbox.html`). Component Pascal follows the tradition of Pascal but is actually incompatible with it. The language encompasses both procedural and object-oriented paradigms and is modular, again making for well-structured architectures and greater ease of maintenance. An in-built *Garbage Collector* [45] ensures reliable memory management and a *run-time linking* facility allows components to be loaded *on demand*, which feature is key in ensuring open-endedness of the BUGS language.

The BlackBox framework supports rapid application development by providing infrastructure for creating the features inherent with interactive software, such as menu commands, dialogue boxes and graphics. It is very powerful but much of the functionality is provided at a very low level. For example, the graphics subsystem is so low-level that an entirely new subsystem has had to be built on top of this for rendering statistical plots in a reasonably intuitive manner. It has been difficult to justify investing too much time in such developments due to the academic nature of BUGS' funding over the years. Hence, the software's graphics have always been a little basic, although, hopefully, they are adequate.

4.2.2. Expansion of capabilities. Shortly after the first stand-alone version (WinBUGS) was functional, work began on expanding its capabilities. The first task was to implement a general purpose Metropolis–Hastings sampler for updating continuous scalar quantities whose full conditional distribution is neither available in closed form nor log-concave. The proposal distribution is a normal distribution centred at the current point with a self-tuning variance adjusted every 100 iterations with a view to (ultimately) achieving an acceptance rate of between 20 and 40 per cent, an intuitive balance between the regularity of movement and the distance travelled in each move. These limits are fairly arbitrary but are based loosely on formal analysis [46].

The self-tuning variance is encapsulated within a Metropolis–Hastings *updater object*, an instance of which is coupled to each node in the graph requiring Metropolis updates. Hence, a distinct and independently adapted proposal variance is used for each such node, so that the sampling of each full conditional is 'optimized' as opposed to using a globally adapted variance which may perform poorly for some nodes. The initial size of the proposal variance is small, e.g. 10^{-4}. This ensures regular movements (high acceptance) towards the target mode(s), which allows reliable location of the vicinity of the target density in a reasonably short time. As the target density is homed-in upon, the proposal variance is gradually increased until it is of a similar, but somewhat larger, size to that of the target distribution.

The self-tuning Metropolis updater was initially developed as a part of the PKBugs project, which aimed to provide an interface (to WinBUGS) for pharmacokinetic modelling. Not only did PKBugs provide a test-bed for serious non-linear modelling, whereby the reliability of the Metropolis updater's adaption algorithm could be honed, but it also served as a proof-of-concept study for fully exploiting the software's various extensibility features. In particular, PKBugs showed that the BUGS language could easily be expanded to include (arbitrarily) complex functions, and

DOI: 10.1002/sim

that specialized interfaces for describing the statistical model and for presenting the output in different ways could be implemented in a straightforward manner.

To further the range of non-linear modelling possibilities, other forms of Metropolis proposal distribution for sampling from distributions with restricted support (interval or semi-infinite) were experimented with, until Radford Neal's slice sampler was invented [47]. Like the Metropolis updater this needs adapting over time, by estimating the typical size of a 'slice'. The procedure can be easily automated, however, and, again like the Metropolis updater, the tuning parameter can be encapsulated within an updater object so that each full conditional is adapted to independently.

At this point, WinBUGS seemed capable of fitting pretty much anything we could throw at it, and work was completed on the graphics. This provided a good opportunity to try pushing the system further by developing GeoBUGS, a specialized interface for spatial modelling, which would provide new distributions for spatially correlated quantities and graphical devices for mapping them as well as for facilitating model specification. The majority of such distributions actually correspond to undirected sub-graphs. However, they are easily accommodated within the directed graphical framework of BUGS by simply thinking of them as multivariate quantities represented by a single node in the graph with complex internal structure—if all internal nodes can be updated together, then the internal structure is unimportant for the purposes of performing the encompassing Gibbs scheme.

In another development, an extension of the PKBugs project led to consideration of how we might implement models described in terms of differential equations, which are important in infectious disease epidemiology and mathematical biology [48, 49], for example, as well as in the motivating pharmacokinetic/toxicokinetic problems that we considered [50]. This work uncovered a significant limitation of the way in which BUGS implemented virtual graphical models (VGMs), an inability to accommodate vector-valued functions. For example, at this point the matrix-inverse function could only provide one element of the inverted matrix for each inversion, requiring $p(p+1)/2$ inversions to obtain the whole matrix. Clearly, such inefficiencies had to be circumvented to make numerical differential equation solving across a grid of time-points feasible. The problem here was that there was no mechanism in place for remembering when calculations had already been performed. One of the most significant advances in the design of the BUGS VGM was to implement such a memory mechanism, whereby logical nodes would know whether any of their stochastic parents had been updated since their last evaluation and thus require reevaluating. This was achieved by all stochastic nodes broadcasting an appropriate message to their descendants whenever their value was updated. This paved the way for implementing more and more complex functions efficiently and also inspired an improvement to the software's Metropolis updaters—a similar mechanism was used to ensure that complex functions saved their current values, in case of rejection, before any relevant Metropolis proposals.

At this point, we had had a great deal of experience in implementing new distributions and functions and realized that much of the work involved, although complicated, was somewhat routine. It became apparent that we could build a module hierarchy for implementing new modelling components whereby only the most basic function-specific or distribution-specific details need be specified directly by the user, and all of the less specific detail would be hidden beneath. This led to the *WinBUGS Development Interface* (WBDev [51]). In many cases, WBDev could offer the user significant (sometimes massive) run-speed improvements, due to 'hard-wiring' of the new distribution/function into the system using compiled code as opposed to specifying the new component via the BUGS language, which is interpreted at run-time. Other advantages included tidier (and less error-prone) BUGS models due to 'packaging' of complex code, and the facility to

DOI: 10.1002/sim

exploit a comprehensive computer language for specifying the new component as opposed to the somewhat limited BUGS language, e.g. piecewise functions could be specified using IF–THEN–ELSE constructs rather than via BUGS' `step(.)` function. The WBDev project also served as a proof-of-concept for making the software open source.

4.2.3. Divergence of established and experimental versions. In 2004 Andrew Thomas began work on an open-source version of the software at the University of Helsinki. The OpenBUGS project initially had three main aims. First, decoupling of core functionality and the user interface, to facilitate use of the software from within other environments. The most significant output from this research arm was BRugs, an interface that allows BUGS to be used interactively from within R. Several other packages exist for interfacing with *WinBUGS* through other software, e.g. SAS, Stata, Excel, but the beauty of BRugs is that it is fully interactive and BUGS output can be accessed mid-analysis for exploiting the full power of R in generating diagnostic plots, for instance. A second aim of the OpenBUGS project was to make the software more platform-independent, and to this end a Linux-based version, namely LinBUGS, has been developed; currently, this can be run on any Intel®-chipped machine. Third, and perhaps most importantly, the software was to be taken forwards with an experimental version in which new ideas could be tried without detriment to users requiring an established/stable version (WinBUGS). This has led to a version that is much more amenable to extension in terms of its range of updating algorithms. Indeed this development highlights one of the fundamental differences between WinBUGS and OpenBUGS. Both versions classify the full conditional distribution of each node to be updated, but whereas WinBUGS uses a one-to-one mapping of classification to updating method, OpenBUGS works through a list of available updating methods, typically in order of increasing generality, and allocates methods to all nodes for which the method is appropriate. This latter approach can cope far better with an ever-expanding range of methods, which fits well with an open-source philosophy. Another significant outcome of this research is that infrastructure for parallelizing the software now exists.

Meanwhile, the development of WinBUGS at Imperial College was centred more and more around specific applications. For example, reversible jump methods were implemented for a generic class of problems [52], but the main focus of this work became applications in genetic epidemiology [53] and pharmacokinetics [54]. In addition, the differential equation solving capabilities were strengthened to accommodate more complex models in pharmacokinetics and make them more applicable to infectious disease modelling. Hence, the two versions of the software diverged somewhat, each with their own advanced features unavailable in the other. At the time of writing, we are moving to rectify this and facilitate the migration of WinBUGS' advanced features over to the OpenBUGS framework, which represents the future of the project, as discussed in Section 6, although WinBUGS will always remain available as a stable version for routine use.

4.3. Funding history

In almost two decades, the BUGS project has received a total of $\sim$1.3 million U.K. pounds in direct funding, as well as indirect long-term support for senior staff, including 5 years funding from The Academy of Finland for AT to work on OpenBUGS. As outlined in Table I, the U.K. grant funding has come from three of the U.K.'s research councils (Medical Research Council, Economic and Social Research Council, Engineering and Physical Sciences Research Council) as well as the Wellcome Trust and a pharmaceutical company (AstraZeneca). While most grants have been targeted at a specific development goal, each has contributed generally to the overall

DOI: 10.1002/sim

Table I. U.K. funding history of the BUGS project.

Time-scale	Funding body	Main purpose
1989–1993	MRC	Salary for programmer (AT)
1993–1996	ESRC	Analysis of complex longitudinal data
1996–1999	EPSRC	Development of PKBugs (pharmacokinetics)
1997–1999	ESRC	Development of GeoBUGS (spatial modelling)
1999–2004	MRC	Modelling complexity in biomedical research
2003–2004	AstraZeneca	Development of differential equation interface
2003–2006	MRC	Development of tools for handling genetic data
2006–2007	Wellcome Trust	Consolidation of previous work + dissemination

evolution of the software. The project now attracts some core funding from MRC via DS's and DL's work at the Biostatistics Unit in Cambridge, which is currently focussed on building an appropriate infrastructure for the further development of OpenBUGS (see Section 6).

5. CRITICAL APPRAISAL

In this section we attempt to identify the reasons for BUGS' success. We also show how the same positive aspects of its design also lead to some of its main shortcomings. The aim is to provide some insight into the origins of BUGS' various idiosyncrasies.

5.1. Flexibility

It seems that the most fundamental reason behind BUGS' success is its flexibility and generality. The software's Gibbs sampling scheme is designed so that it can be applied to any DAG and, in principle, any directed graph can be specified using the BUGS language, since the language is entirely open-ended and new distributions and functions can thus be added as required. We delve deeper into how these things are possible below but we first highlight the issues that they cause:

- Any model can be specified, regardless of whether it makes any sense. For example, Bernoulli trials cannot be over-dispersed as the variance is determined by the mean, but there is nothing to stop people fitting a model such as

```
y[i]           ~ dbern(mu[i])
logit(mu[i]) <- dnorm(m.mu, p.mu)
```

  and getting meaningless results. We have seen publications that do this. Similarly, we have frequently seen 'pointless' prior distributions being given, for example

```
beta   ~ dnorm(m.beta, p.beta)
m.beta ~ dnorm(0, 0.0001)
p.beta ~ dgamma(0.001, 0.001)
```

  instead of directly specifying values for `m.beta` and `p.beta`: this construction does induce some sort of prior on `beta` but adds a completely superfluous layer to the model.

  In particular, there is no mechanism in place to ensure that a given model is identifiable. Such mechanisms may seem desirable but any attempt to police use of the language would invariably

DOI: 10.1002/sim

restrict freedom, which goes against the philosophy of BUGS somewhat. (The ingenuity of users is such that there will always be models that we could not have anticipated when defining the rules.) It can be quite reasonable, for example, to deliberately over-parameterize hierarchical models, making certain parameters non-identifiable but greatly improving the efficiency of estimation for identifiable contrasts (see [19, Chapter 19], for instance). For example, in a repeated measures model with data $y_{ij} \sim \mathrm{N}(\mu + u_i, \sigma^2)$ and $u_i \sim \mathrm{N}(0, \phi^2)$, we can rescale the second-stage random effects by an unknown (and non-identifiable) factor, $u_i = \alpha b_i$, say, which can prevent the Gibbs sampler getting stuck.

- Similarly, the BUGS language does not discriminate between models that the software is good at fitting and those for which it may perform poorly. For example, the language easily accommodates autocorrelated time-series models, even those with longitudinal structure on the variances, but the algorithms selected to update the parameters of such models can often struggle. More generally, there is usually very little contextual information to be gleaned from the model specification, and BUGS' MCMC algorithms are required to be reasonably general to cope with the vast array of possible models. These facts together mean that it is very difficult to fully optimize the updating process for a given model: BUGS will almost always run slower than bespoke code.
- The same model can typically be specified in different ways, sometimes leading to (dramatically) different behaviours. A good example of this, which also highlights the previous point, is the autocorrelated model with data $y_i \sim \mathrm{N}(\mu_i, \sigma^2)$, $i = 1, \ldots, N$, and $\mu_i \sim \mathrm{N}(f(\mu_{i-1}), \phi^2)$ for some appropriate function $f(.)$. We can specify this either as written, which gives a *hierarchically centred* model, or, equivalently, via $\mu_i = f(\mu_{i-1}) + u_i$ with $u_i \sim \mathrm{N}(0, \phi^2)$. In cases where $f(.)$ contains more than one instance of μ_{i-1}, e.g. $f(\mu_{i-1}) = \beta\mu_{i-1}/(1-\mu_{i-1})$, then the latter approach, with anything more than a few data ($N > 30$, say), will cause BUGS to 'hang' when building the internal representation of the model. We still do not fully understand why this happens, although BUGS is presumably attempting to construct an excessively complex graph, where the excess complexity grows exponentially with N. (Note that it is usually building complex/large graphs that causes BUGS to hang during compilation of the model.) We have established, however, that both models appear to give the same results for small N, although the hierarchically centred model runs much faster and has far better convergence properties.
- The range of possibilities afforded by the BUGS language means that there are certain modelling scenarios that occur only very rarely, and so bugs in the source code may go unnoticed for long periods of time. For example, one error in updating a Pareto distribution (with uniform likelihood) took over 15 years to discover. Bizarrely, someone else noticed the same error within the same week!

It is also worth noting here that BUGS provides much flexibility with respect to the way in which the simulation is carried out, for example, imposing no limits on the number of Gibbs iterations required and very few on the range of samples that can be used for inference. Hence, the software can be used in a non-rigorous manner for performing exploratory analyses, say. The downside of this, of course, is that the user has to take responsibility for convergence-related decisions, and so, to some extent, they need to have a reasonable understanding of the underlying methodology. BUGS is also flexible in allowing the user to choose which quantities should be monitored during the simulation. This is actually out of necessity since all monitored samples are stored in the computer's RAM for quick access, and to store all values sampled for every variable in a complex

DOI: 10.1002/sim

model would soon lead to the computer running out of memory. Hence, the user must again be careful, to monitor only quantities of interest. (Note that more memory-efficient monitors are available for nodes where approximate inference is sufficient.)

5.2. *Popularity*

The popularity brought about largely by BUGS' generality/flexibility is itself partly responsible for the viral spreading of certain poor practices throughout Bayesian statistics. In particular, such widespread use has led to a degree of ubiquity of potentially inappropriate priors, through, ironically, a lack of exploitation of the software's flexibility. Many people enter the world of Bayesian ideas through WinBUGS, and novice users naturally tend to copy more experienced users' code; hence, 'lazy' priors are easily perpetuated. Examples include:

- The Gamma(ε, ε) prior may be reasonable as a prior for precision parameters for observations (it is approximately the Jeffreys prior which might be better approximated by a uniform prior on the log-variance), but is generally inappropriate for the precision of random effects, particularly for continuous data. It arose in a time when the mathematical convenience that it offered was valuable but, with modern MCMC methods, this is no longer the case. However, it still survives due to early use in our own BUGS examples.
- It is easy to fit huge models to sparse data, and have all the conclusions driven unwittingly by casually made prior assumptions.
- The `I(,)` mechanism for dealing with censored data can be used for modelling truncated prior distributions *provided they have no unknown parameters*—if there are unknown parameters the inferences will be wrong.

All these errors can be made, and results obtained, without any warning from the program. Sometimes lack of convergence can give a hint that something is wrong, but generally it is the user's (generally untrained) responsibility to identify the problem, which is why the manual is headed by a large public-health warning that '*Gibbs sampling can be dangerous!*'

Another factor contributing to BUGS' popularity is the facility to automatically compute the *Deviance Information Criterion* (DIC), which offers a straightforward means of comparing different models. This can be considered an adaptation of the Akaike Information Criterion to Bayesian models incorporating prior information, whether through fully-specified proper prior distributions or hierarchical models. It features two elements: the posterior mean of the deviance as a measure of fit, and an assessment p_D of the 'effective number of parameters' as a measure of complexity. Adding these components gives the DIC, and the model with the smallest DIC might be considered preferable in the sense of being likely to make better short-term predictions on data of a similar structure to that already observed.

DIC has proved to be very popular, and the original paper [55] is currently the third highest-cited paper in the whole of the mathematical sciences in the last 10 years. But it is not without problems. The method of assessing p_D is not invariant to transformations of the parameters and it can give inappropriate results if there are highly non-normal posterior distributions of the parameters on which prior distributions have been placed. Alternative forms of p_D have been suggested, and the theoretical basis for DIC has been questioned, in particular a suggestion that there is insufficient penalization for complexity [56]. Furthermore, it is still unclear what an appropriate procedure for categorical parameters, as in mixture models [57], might be. We consider DIC as a useful tool, but its implementation could be improved and perhaps guidance for its appropriate use built into the program.

DOI: 10.1002/sim

5.3. Use of object-orientation

The software's ability to update any DAG is largely due to its exploitation of OOP. Put very simply, OOP facilitates abstract computation and so the general factorization properties of DAGs can be fully exploited. Even when writing code for the Gibbs scheme we only need to think in abstract terms, of visiting each stochastic node in the graph and navigating to its children, asking each node in turn for its contribution to the relevant full conditional (or its parameters). We need not care here about the nature of each node's distributional or functional form—responsibility for providing the correct information rests with the node objects themselves, which from the point-of-view of the Gibbs scheme are identical, but have much specialized detail hidden within.

Use of OOP and related ideas also makes it possible for the BUGS language to be open ended. Because the software works at an abstract level, it can be designed to make use of modelling components (distributions and functions) without having to know anything about them. Hence, new functions and distributions can be added at any time, and because the software can load new components *on demand*, it does not even need recompiling when new components are added. The same is true with new updating algorithms, which has greatly facilitated the expansion of BUGS' capabilities over the years.

One downside of object-orientation in systems as complex as BUGS is that it can make debugging somewhat difficult. In more traditional programming settings, there would typically be some kind of master program specifying a list of commands to be executed in sequence. A debugger can then be implemented to step through each command in turn and provide current information on parameters of interest. In contrast, objects are generally designed only to make/respond to various requests of/from other objects—they have little or no concept of the order in which external events occur. Consequently, an object-oriented program typically behaves as a complex chain-reaction and the path back to the source of any error may be extremely convoluted. This can make for indecipherable error messages, error messages that have nothing to do with the actual error, and impenetrable 'trap' windows, all of which even the developers of the software sometimes struggle to understand. For example, an 'undefined real result' trap lists the sequence of 'function calls' that led to some impossible calculation, such as the square root of a negative number. The calculation in question may have been attempted by an updater object trying to evaluate some function of interest. To figure out what the updater was trying to do, which node it was dealing with at the time, what the current values of that node and those in its Markov blanket are, and why these might have led to some inconsistency (even if it were that simple), just from a list of function calls and the current values of some 'internal' parameters, is a challenge indeed. As developers of the software we have access to special tools for probing these 'traps' for more information but it can still be difficult sometimes. It has been particularly difficult to fix any of these traps, or replace them with more meaningful error messages, since they represent unanticipated errors—things that were not *meant* to go wrong—it is virtually impossible to get hold of enough contextual information to provide a message that might be of some use to the user.

It should be noted at this point that the object-oriented framework selected for developing BUGS was perhaps, in retrospect, not the best choice. A natural extension of object-orientation is *component*-orientation, whereby the development of distinct software components that can be deployed independently is facilitated; for example, a graphics/equation editor or a spell-checker that can be used from within many other packages, such as a word processor or presentation software. The Component Pascal framework underlying BUGS is component-oriented (as the name

DOI: 10.1002/sim

suggests) but was dominated by Java, its nearest competitor, which offered the (vastly) more popular C-style syntax as well as platform independence. However, at the time, Java could not support serious number-crunching and it was thought that with the explosion in personal computing, Unix might become substantially less popular, somewhat lessening the need for platform independence. Component-oriented systems are best exploitable when there is a large user-base and one can make use of other developers' components (reusing code as opposed to having to continually reinvent the wheel). The increased uptake of Java made it even more popular and Component Pascal is now somewhat obsolete (although much loved by its users). This has led to substantial problems in terms of capacity building, i.e. encouraging contributions to BUGS from third-parties as well as recruiting new developers.

5.4. Context-independence of modelling components

Both object- and component-orientation facilitate the development of components that can be used in arbitrary settings. Such context-independence has contributed enormously to the generality/flexibility of BUGS. For example, the majority of distributions in BUGS can be used as priors, likelihood, predictive distributions, and, with the use of the `cut` function, 'distributional constants' [58]; they may also be used at any level of a (hierarchical, say) model. However, the freedom that context-independence offers also creates scope for (unintentional) misuse when components have been designed imperfectly. Examples include:

- Certain distributions can only be used in particular contexts; for example, neither the Wishart nor the Dirichlet distribution, in WinBUGS, should be used as a likelihood, i.e. it is not possible to learn about their parameters (although a trick does exist in the case of the Dirichlet). The user is free to attempt to use such distributions incorrectly, and the software *should* generate a warning, but with the vast array of possibilities there is no guarantee that all such misuses will be caught; this, of course, could lead to unreliable results (without anyone being aware). In a similar vein, the Poisson distribution was once 'upgraded' (for reasons that have been lost in the mists of time) so that it could be used as a prior for continuous quantities, e.g. non-integer-valued data. However, several users have pointed out that in some settings this can lead to non-integer values being sampled for discrete quantities for which a conventional (i.e. discrete) Poisson prior was intended.
- Context independence allows the various algorithms used in BUGS to be switched around, e.g. Metropolis could be used instead of rejection sampling for certain models. This, however, was not fully anticipated in the design of WinBUGS. For the sake of efficiency, many of the samplers used in WinBUGS make assumptions about the context in which they are being used, which do not necessarily hold if applied in alternative settings. For example, a Metropolis updater will assume that any real number is a suitable candidate value for the variable being updated, because the updater was originally designed only for distributions with support on the whole real line. If it is then applied to a node whose full conditional has restricted support, the node may calculate its full conditional density incorrectly (since it assumes, for efficiency, that it will not receive inappropriate input), potentially leading to acceptance of a prohibited value. OpenBUGS, on the other hand, has been designed from a fully component-oriented perspective so that MCMC samplers can be used in arbitrary settings.

DOI: 10.1002/sim

6. DISCUSSION AND FUTURE PERSPECTIVES

The current thrust of the BUGS project is towards focussing *all* development efforts on OpenBUGS. At the time of writing there exist only two people, worldwide, with an intimate understanding of the BUGS architecture (AT+DL). This architecture is complex and to simply download the source code and learn how to develop it would be a formidable task. There are numerous researchers and organizations willing to contribute, but to invest the time and resources required just to get to the point of being able to start development work is perhaps not economically viable. To circumvent this problem we are in the process of setting up a charitable foundation to act as a conduit for funding of development work, whereby all funds can be centralized and invested in a small number of developers who can be trained in the internal workings of BUGS and subsequently be responsible for multiple developments. The foundation will have a board of directors responsible for prioritizing proposed developments, which will all be for the common good of users (as opposed to specialized ventures, though we would actively encourage such developments 'externally').

For most of its life BUGS has enjoyed a relatively competition-free environment. MLwiN [59] has provided MCMC facilities for some years now, but only for a restricted class of models. Indeed, for more complex models, MLwiN will actually write a BUGS script. More recently, however, more general-purpose contenders have emerged. In particular, Just Another Gibbs Sampler (JAGS [60]) is a 'BUGS clone', which is implemented in C++ and will therefore run on many platforms as well as being more amenable to user-extension. JAGS has also highlighted and corrected some potential flaws in the BUGS language; for example, JAGS achieves better separation of modelling assumptions from data, with respect to censored observations, for instance. More competition arises from Microsoft's 'Infer.NET' package (`http://research.microsoft.com/en-us/um/cambridge/projects/infernet/`), which is a very general graphical modelling package, supporting many types of graph (directed or undirected) and providing numerous algorithms for inference. However, we anticipate that the focus here will be more on efficient processing of large problems using adequate approximations within a machine-learning context, as opposed to precise statistical inference requiring accurate assessment of second-order (error) properties. A Bayesian MCMC package has also recently been implemented within the SAS software [61].

Meanwhile, there is an ever expanding range of interfaces for using BUGS from other software, including Excel, GenStat, Matlab, R, SAS and Stata. There are also numerous packages to aid in converting BUGS input/output from/to ArcGIS, ArcView, Emacs, Excel, Mathematica, Matlab, R/S and SAS. In addition, there exist packages for automatically generating and running scripts using Perl and Python. And several specialized interfaces for running complex models in WinBUGS are available from `www.winbugs-development.org.uk`. These include facilities for implementing pharmacokinetic models, reversible jump methods, differential equation models, and for writing your own specialized function/distribution (to be incorporated into the BUGS language).

Despite several shortcomings, such as the lack of IF–THEN–ELSE logic and a somewhat cumbersome mechanism for specifying censoring, the BUGS language has proved enormously successful. It can describe effectively a vast array of different model types, even when the software is not well suited to analysing them. For example, the language can accommodate many time-series (autoregressive) models but Gibbs sampling is notoriously problematic in these settings, with excessive autocorrelation typically apparent in the simulated chains. As BUGS continues to evolve, it is mainly the algorithms implemented in the software that are changing, to offer a more

DOI: 10.1002/sim

tailored solution to each individual problem, as opposed to the language itself, which we hope will, with some tweaks, continue long into the future, as a general means of describing graphical models.

REFERENCES

1. Gilks WR, Thomas A, Spiegelhalter DJ. A language and program for complex Bayesian modelling. *The Statistician* 1994; **43**:169–178.
2. Geman S, Geman D. Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis* 1984; **6**:721–741.
3. Gelfand AE, Smith AFM. Sampling-based approaches to calculating marginal densities. *Journal of the American Statistical Association* 1990; **85**:398–409.
4. Lawson AB, Browne WJ, Vidal Rodeiro CL. *Disease Mapping with WinBUGS and MLwiN. Statistics in Practice*. Wiley: Chichester, 2004.
5. Mu S, Ludden TM. Estimation of population pharmacokinetic parameters in the presence of non-compliance. *Journal of Pharmacokinetics and Pharmacodynamics* 2003; **30**:53–81.
6. McCarthy MA, Parris KM. Clarifying the effect of toe clipping on frogs with Bayesian statistics. *Journal of Applied Ecology* 2004; **41**:780–786.
7. O'Hagan A, Stevens JW, Montmartin J. Bayesian cost-effectiveness analysis from clinical trial data. *Statistics in Medicine* 2001; **20**:733–753.
8. Scurrah KL, Palmer LJ, Burton PR. Variance components analysis for pedigree-based censored survival data using generalized linear mixed models (GLMMs) and Gibbs sampling in BUGS. *Genetic Epidemiology* 2000; **19**:127–148.
9. Millard A, Gowland R. A Bayesian approach to the estimation of age of humans from toothwear. *Archeologia e Calcolatori* 2002; **13**:197–210.
10. Eaves L, Erkanli A, Silberg J, Angold A, Maes HH, Foley D. Application of Bayesian inference using Gibbs sampling to item-response theory modeling of multi-symptom genetic data. *Behavior Genetics* 2005; **35**:765–780.
11. Milheiro-Oliveira P. Bayesian statistical methods for modeling and prediction of major landslides in coastal cliffs. *Coastal Engineering Journal* 2007; **49**:45–61.
12. Ayers E, Junker B. IRT modeling of tutor performance to predict end-of-year exam scores. *Educational and Psychological Measurement* 2008; **68**:972–987.
13. Chevrolat J-P, Golmard J-L, Ammar S, Jouvent R, Boisvieux J-F. Modelling behavioral syndromes using Bayesian networks. *Artificial Intelligence in Medicine* 1998; **14**:259–277.
14. Meyer R, Yu J. BUGS for a Bayesian analysis of stochastic volatility models. *Econometrics Journal* 2000; **3**:198–215.
15. Weihs C, Ligges U. Parameter optimization in automatic transcription of music. In *From Data and Information Analysis to Knowledge Engineering*: *Proceedings of the 29th Annual Conference of the Gesellschaft für Klassifikation e.V. University of Magdeburg*, 9–11 March 2005, Spiliopoulou M, Kruse R, Borgelt C, Nürnberger A, Gaul W (eds), Studies in Classification, Data Analysis, and Knowledge Organization. Springer: Berlin, 2006; 740–747.
16. Swartz TB, Gill PS, Beaudoin D, deSilva BM. Optimal batting orders in one-day cricket. *Computers and Operations Research* 2006; **33**:1939–1950.
17. Wyatt RJ. Mapping the abundance of riverine fish populations: integrating hierarchical Bayesian models with a geographic information system (GIS). *Canadian Journal of Fisheries and Aquatic Sciences* 2003; **60**:997–1006.
18. Katsis A, Ntzoufras I. Bayesian hypothesis testing for the distribution of insurance claim counts using the Gibbs sampler. *Journal of Computational Methods in Science and Engineering* 2005; **5**:201–214.
19. Gelman A, Hill J. *Data Analysis using Regression and Multilevel/Hierarchical Models*. Cambridge University Press: Cambridge, 2007.
20. Carlin BP, Louis TA. *Bayesian Methods for Data Analysis* (3rd edn). CRC Press: Boca Raton, 2008.
21. Congdon P. *Bayesian Statistical Modelling*. Wiley: Chichester, 2001.
22. Lawson AB. *Bayesian Disease Mapping*: *Hierarchical Modeling in Spatial Epidemiology*. CRC Press: Boca Raton, 2009.
23. Woodworth GG. *Biostatistics*: *A Bayesian Introduction*. Wiley: New York, 2004.
24. Broemeling LD. *Bayesian Biostatistics and Diagnostic Medicine*. CRC Press: Boca Raton, 2007.

DOI: 10.1002/sim

25. Ntzoufras I. *Bayesian Modeling using WinBUGS*. Wiley: New York, 2009.
26. Gill J. *Bayesian Methods*: *A Social and Behavioral Sciences Approach*. Chapman & Hall, CRC: London, Boca Raton, 2002.
27. Spiegelhalter DJ. Bayesian graphical modelling: a case-study in monitoring health outcomes. *Applied Statistics* 1998; **47**:115–133.
28. Stefik M, Bobrow DG. Object-oriented programming: themes and variations. *The AI Magazine* 1985; **6**(4):40–62.
29. Lauritzen SL, Dawid AP, Larsen BN, Leimer HG. Independence properties of directed Markov fields. *Networks* 1990; **20**:491–505.
30. Szyperski C. Component-oriented programming: a refined variation of object-oriented programming. *The Oberon Tribune* 1995; **1**:1–5.
31. Spiegelhalter DJ, Knill-Jones RP. Statistical and knowledge-based approaches to clinical decision-support systems, with an application in gastroenterology (with Discussion). *Journal of the Royal Statistical Society*, *Series A* 1984; **147**:35–77.
32. Pearl J. Reverend Bayes on inference engines: a distributed hierarchical approach. *Proceedings of the American Association for Artificial Intelligence National Conference on AI*, Pittsburgh, 1982; 133–136.
33. Spiegelhalter DJ. Probabilistic reasoning in predictive expert systems. In *Uncertainty in Artificial Intelligence*, Kanal LN, Lemmer J (eds). North-Holland: Amsterdam, 1986; 47–68.
34. Kim JH, Pearl J. A computational model for combined causal and diagnostic reasoning in inference systems. *Proceedings of the IJCAI-83*, Karlsruhe, Germany, 1983; 190–193.
35. Lauritzen SL, Spiegelhalter DJ. Local computations with probabilities on graphical structures and their application to expert systems (with Discussion). *Journal of the Royal Statistical Society*, *Series B* 1988; **50**:157–224.
36. Pearl J. Evidential reasoning using stochastic simulation. *Artificial Intelligence* 1987; **32**:245–257.
37. Spiegelhalter DJ, Lauritzen SL. Sequential updating of conditional probabilities on directed graphical structures. 1990; 20:579–605.
38. Lunn DJ, Thomas A, Best N, Spiegelhalter D. WinBUGS—a Bayesian modelling framework: concepts, structure, and extensibility. *Statistics and Computing* 2000; **10**:325–337.
39. Wirth N. *Programming in Modula-2*. Springer: Santa Clara, 1983.
40. Thomas A, Spiegelhalter DJ, Gilks WR. BUGS: a program to perform Bayesian inference using Gibbs sampling. In *Bayesian Statistics*, Bernardo JM, Berger JO, Dawid AP, Smith AFM (eds), vol. 4. Oxford University Press: Oxford, U.K., 1992; 837–842.
41. Gilks WR, Richardson S, Spiegelhalter DJ. *Markov Chain Monte Carlo in Practice*. Chapman & Hall: London, 1996.
42. Metropolis N, Rosenbluth AW, Rosenbluth MN, Teller AH, Teller E. Equations of state calculations by fast computing machines. *Journal of Chemical Physics* 1953; **21**:1087–1091.
43. Hastings WK. Monte Carlo sampling-based methods using Markov chains and their applications. *Biometrika* 1970; **57**:97–109.
44. Oberon Microsystems Inc. Component Pascal Language Report. *Technical Report*, Oberon Microsystems Inc., Zurich, 2001.
45. Jones R, Lins R. *Garbage Collection*: *Algorithms for Automatic Dynamic Memory Management*. Wiley: New York, 1996.
46. Gelman A, Gilks WR, Roberts GO. Weak convergence and optimal scaling of random walk Metropolis algorithms. *Annals of Applied Probability* 1997; **7**:110–120.
47. Neal RM. Markov chain Monte Carlo methods based on 'slicing' the density function. *Technical Report 9722*, Department of Statistics, University of Toronto, 1997.
48. Hethcote HW. The mathematics of infectious diseases. *SIAM Review* 2000; **42**:599–653.
49. Jones DS, Sleeman BD. *Differential Equations and Mathematical Biology*. Chapman & Hall: Boca Raton, 2003.
50. Lunn DJ. Bayesian analysis of population pharmacokinetic/pharmacodynamic models. In *Probabilistic Modeling in Bioinformatics and Medical Informatics*, Husmeier D, Dybowski R, Roberts S (eds). Springer: London, 2005; 351–370.
51. Lunn DJ. WinBUGS Development Interface (WBDev). *ISBA Bulletin* 2003; **10**(3):10–11.
52. Lunn DJ, Best N, Whittaker JC. Generic reversible jump MCMC using graphical models. *Statistics and Computing* 2008; DOI: 10.1007/s11222-008-9100-0.
53. Lunn DJ, Whittaker JC, Best N. A Bayesian toolkit for genetic association studies. *Genetic Epidemiology* 2006; **30**:231–247.
54. Lunn DJ. Automated covariate selection and Bayesian model averaging in population PK/PD models. *Journal of Pharmacokinetics and Pharmacodynamics* 2008; **35**:85–100.

DOI: 10.1002/sim

55. Spiegelhalter DJ, Best NG, Carlin BP, van der Linde A. Bayesian measures of model complexity and fit (with Discussion). *Journal of the Royal Statistical Society*, *Series B* 2002; **64**:583–639.
56. Plummer M. Penalized loss functions for Bayesian model comparison. *Biostatistics* 2008; **9**:523–539.
57. Celeux G. Mixture models for classification. In *Advances in Data Analysis*, Decker R, Lenz HJ (eds). Springer: Berlin, 2007; 3–14.
58. Lunn D, Best N, Spiegelhalter D, Graham G, Neuenschwander B. Combining MCMC with 'sequential' PKPD modelling. *Journal of Pharmacokinetics and Pharmacodynamics* 2009; DOI: 10.1007/s10928-008-9109-1.
59. Rasbash J, Browne W, Goldstein H, Yang M, Plewis I, Healy M, Woodhouse G, Draper D, Langford I, Lewis T. *A User's Guide to MLwiN* (2nd edn). Institute of Education: London, 2000.
60. Plummer M. *JAGS Version 1.0.3 Manual*. IARC: Lyon, France, 2008.
61. SAS Institute Inc. *SAS/STAT 9.2 User's Guide*. SAS Institute Inc.: Cary, NC, 2008.

DOI: 10.1002/sim