The Statistician (1994)
43, *No.* 1, *pp.* 169–177

A language and program for complex Bayesian modelling

By W. R. GILKS†, A. THOMAS and D. J. SPIEGELHALTER

Medical Research Council Biostatistics Unit, Cambridge, UK

[Received July 1992. Revised July 1993]

SUMMARY

Gibbs sampling has enormous potential for analysing complex data sets. However, routine use of Gibbs sampling has been hampered by the lack of general purpose software for its implementation. Until now all applications have involved writing one-off computer code in low or intermediate level languages such as C or Fortran. We describe some general purpose software that we are currently developing for implementing Gibbs sampling: BUGS (Bayesian inference using Gibbs sampling). The BUGS system comprises three components: first, a natural language for specifying complex models; second, an 'expert system' for deciding appropriate methods for obtaining samples required by the Gibbs sampler; third, a sampling module containing numerical routines to perform the sampling. S objects are used for data input and output. BUGS is written in Modula-2 and runs under both DOS and UNIX.

Keywords: Bayesian computation; Bayesian inference; Gibbs sampling; Graphical model; Statistical software

1. Introduction

Gibbs sampling (Geman and Geman, 1984; Hastings, 1970) is a technique for simulating samples from the joint posterior distribution of the unknown quantities in a statistical model. It has been shown to have enormous potential for the statistical analysis of complex data sets (see, for example, Gelfand and Smith (1990), Gelfand *et al.* (1990), Smith and Roberts (1993) and Gilks *et al.* (1993)). Gibbs sampling succeeds because it reduces the problem of dealing simultaneously with a large number of intricately related unknown parameters and missing data into a much simpler problem of dealing with one unknown quantity at a time, sampling each from its *full conditional* distribution (see Section 6). We pass over a description of the basic methodology of Gibbs sampling: a good introduction is given by Casella and George (1992).

Notwithstanding the growing popularity of Gibbs sampling, its routine use has been hampered by a lack of general purpose software for its implementation. Until now all applications have involved writing one-off computer code in low or intermediate level languages such as C or Fortran. In our experience, writing and debugging a Gibbs sampler for a moderately complex application can take anything from a few days to several weeks. Thereafter, modifying the program (e.g. to elaborate the model or to apply it to different data) might take several hours. This compares dismally with model fitting in GLIM or S, for which model specification may take just a few minutes, and modifications may take just a few seconds. The fact that increasing numbers of statisticians are expending considerable effort to use Gibbs sampling clearly indicates that currently available software is inadequate in many applications. Our aim is to make Gibbs sampling as easy to implement as generalized

† *Address for correspondence*: Medical Research Council Biostatistics Unit, Institute of Public Health, Robinson Way, Cambridge, CB2 2SR, UK.
E-mail: wally.gilks@mrc-bsu.cam.ac.uk

0039–0526/94/43169

linear models in GLIM or S. For this we are developing BUGS: Bayesian inference using Gibbs sampling. In this paper we describe our approach.

The main requirements of BUGS are that it should

(a) accommodate a very large class of models,
(b) enable models to be specified concisely and
(c) construct and sample from full conditional distributions automatically.

We discuss these requirements in the following sections. We begin with an example.

2. Example: job exposure matrices

Gilks and Richardson (1992) described an application of Gibbs sampling in occupational epidemiology. In this application the aim is to estimate disease risks associated with chronic exposure to industrial agents, such as chemicals, dust or fibres. A *disease study* has been conducted in which each individual's status for a particular disease has been recorded, but his exposure status on each of several agents is unknown. In another study called the *exposure study*, which is conducted on different individuals, exposure statuses have been recorded but disease statuses are unknown for each individual. Thus we have the usual ingredients for a logistic regression model: independent variables (exposure statuses) and a dependent variable (disease status); but unusually we know both of these for no one. The vital link between the two sets of individuals is *occupation*: we know the job description of every individual.

For the disease study individuals Gilks and Richardson (1992) proposed the following model:

$$D_i \sim \text{Bernoulli}(\theta_i) \tag{1}$$

where D_i is the disease status ($0 \equiv$ not diseased; $1 \equiv$ diseased) for the ith individual, θ_i is his probability of disease and

$$\text{logit}(\theta_i) = \beta_0 + \sum_k \beta_k E_{ik}, \tag{2}$$

where E_{ik} is the unobserved exposure status ($0 \equiv$ unexposed; $1 \equiv$ exposed) of individual i to agent k. Equations (1) and (2) together define a classical logistic regression model.

For the exposure study individuals, Gilks and Richardson (1992) proposed

$$m_{jk} \sim \text{binomial}(\pi_{jk}, n_j) \tag{3}$$

where n_j is the number of exposure study individuals in job type j, m_{jk} is the number of these who were exposed to agent k and π_{jk} is an exposure probability. The set $\{m_{jk}, n_j\}$ comprise a *job exposure matrix.*

The link between the two studies is then provided by

$$E_{ik} \sim \text{Bernoulli}(\pi_{j(i),k}) \tag{4}$$

where $j(i)$ denotes the job type of disease study individual i.

The Bayesian model specification is completed with priors

$$\beta_k \sim \text{normal}(\mu_k, \sigma_k^2) \tag{5}$$

and

$$\text{logit}(\pi_{jk}) = \phi_{jk} \tag{6}$$

with

$$\phi_{jk} \sim \text{normal}(\mu_k^*, \sigma_k^{*2}). \tag{7}$$

2.1. *Completing the model: directed acyclic graph*

The model set out in submodels (1)–(7) incompletely specifies the joint distribution of the model parameters and data. These submodels merely tie down a few conditional marginal distributions. However, in an intuitive sense, model (1)–(7) is already complete: if further structure had been required it could have been specified explicitly. Thus to complete the joint distribution of the parameters and data, one seeks some kind of assumption of 'independence' between the submodels. This is provided by the *directed Markov assumption* (Lauritzen *et al.*, 1990), which simply states that the joint distribution of all the model parameters and data is given by the product of all the submodels. Thus the joint distribution from model (1)–(7) is

$$\prod_i p(D_i|\theta_i) \prod_k p(\beta_k|\mu_k, \sigma_k) \prod_{ik} p(E_{ik}|\pi_{j(i),k}) \prod_{jk} p(m_{jk}|\pi_{jk}, n_j) \prod_{jk} p(\phi_{jk}|\mu^*_{jk}, \sigma^*_{jk}) \tag{8}$$

where deterministic equations (2) and (6) are assumed to hold.

That the directed Markov assumption is natural for model (1)–(7) is most easily seen from a *graph* of the model (Fig. 1), in which round nodes denote unobserved stochastic variables (which may be model parameters or missing exposures), square nodes with a single border denote observed stochastic variables (i.e. data), square nodes with a double border denote fixed quantities in prior distributions, triangles denote deterministic variables and *edges* (arrows) denote dependences specified in the submodels. Fig. 1 is a *directed acyclic graph* (Whittaker, 1990), since all the edges are directed and it is not possible, just by following the directions of the edges, to return to a node after leaving it. Thus there is a partial ordering of the nodes, and the directed Markov assumption is seen simply to be a natural Markov-type assumption on the partial ordering.

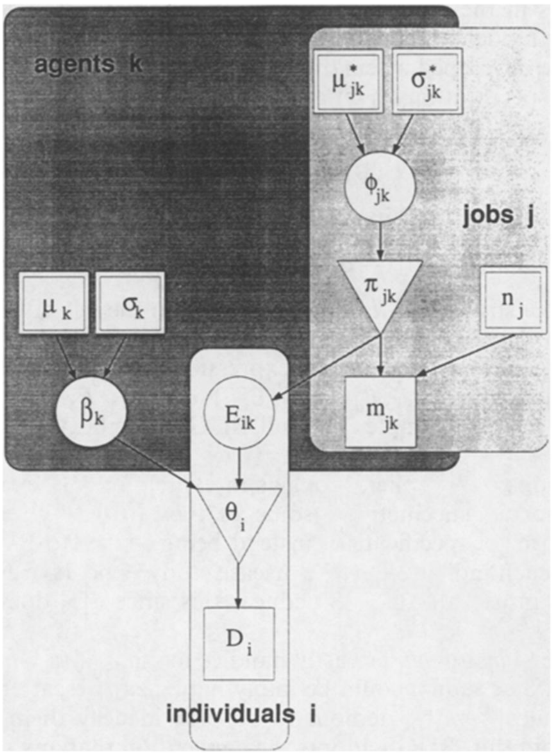

Fig. 1. Graphical representation of model (1)–(7)

We shall refer to the node on the left of a submodel, e.g. D_i in model (1), as a *child* of the nodes on the right of that submodel, and the nodes on the right of the submodel as the *parents* of the node on the left. A *stochastic parent* of a given node is a stochastic node which is an ancestor (in an obvious extension of terminology) of the given node, where only deterministic nodes intervene between the two. Likewise, a *stochastic child* of a given node is a stochastic node which is a descendant of the given node where only deterministic nodes intervene between the two. For example, from Fig. 1, θ_i is the only parent of D_i, and D_i is the child of θ_i; E_{ik} is one of the stochastic parents of D_i, and D_i is the stochastic child of E_{ik}.

We shall refer to a set of submodels such as model (1)–(7) corresponding to a directed acyclic graph, together with the directed Markov assumption, as a *directed acyclic graphical (DAG) model.* DAG models provide a very rich class of models which are particularly amenable to estimation by Gibbs sampling: see Sections 4 and 6. Within this class we would like the family of algebraic and stochastic forms for the individual submodels to be very general, accommodating non-linearity and all commonly used statistical distributions. Thus we have developed a user interface and corresponding facilitating algorithms, as we describe below.

3. Model specification

It is tempting to try to devise a graphical user interface to BUGS, allowing models to be specified through drawing graphs on the screen. However, a graph gives only structural information: the precise form of each submodel would still need to be specified. Moreover, a graph may not even give full structural information. For example in Fig. 1 the structure of the mapping of jobs onto individuals $j(i)$ is suppressed. To express the full model structure in the form of a graph on the screen would entail drawing one node for each of possibly several thousand variables in the model. Thus a graph, although of conceptual value, is of limited practical value for model specification.

Instead we have developed a semantic interface to BUGS which mimics as far as possible the natural model specification, as set out for example in equations (1)–(7). Thus the stochastic relationship (1) is specified by

```
D[i] ~ Bernoulli(theta[i]);                                        (9)
```

and the deterministic relationship (2) is specified by

```
Logit(theta[i]) <- beta[nA1] + InProd(beta[1: nA], E[i,]);.        (10)
```

Each of these model statements *declares* the node on the left of the expression. The '$\sim$' sign in expression (9) declares $D[i]$ to be a stochastic node, and the '$\leftarrow$' in expression (10) declares theta[i] to be a deterministic node. In expression (10), InProd denotes an inner product; beta[1: nA] denotes $\beta_1, \beta_2, \ldots, \beta_{nA}$ and $E[i,]$ denotes $E_1, E_2, \ldots, E_{nA}$, where nA is the number of agents in this example. The full BUGS specification of model (1)–(7) is given in Fig. 2.

Explicit 'for' loops and implicit loops as in expression (10) enable models to be specified succinctly. Also, to aid succinctness, some of the structural details of the model can be suppressed in the model specification, instead being read into BUGS as data. For example, the occupation of each individual in the disease study appears in Fig. 2 only in the form $J[i]$. $J[i]$ must be read in as data: BUGS deduces this since $J[i]$ does not appear on the left of any declaration.

In principle, the Gibbs sampler easily handles missing data by treating them as unknown model parameters. Since there could be many haphazardly scattered missing data in a given application, it would be rather tedious to have to identify them through individual model statements. To avoid this, BUGS adopts the convention that any stochastic variable declared on the left-hand side of a model statement, but which is not found in the data file, is to be

```
model Jobs;
  data in "bugs\dat\Jobs.dat";
  inits in "bugs\in\Jobs.in";
const
  nI = 1000, # number of individuals
  nA = 4, # number of agents
  nJ = 2, # number of jobs
  nA1 = nA + 1;
var
  D[nI],J[nI],theta[nI],E[nI,nA],pi[nJ,nA],
  phi[nJ,nA],mu_star[nJ,nA],tau_star[nJ,nA],
  m[nJ,nA],n[nA],beta[nA1],mu[nA1],tau[nA1];
{
  for (i in 1:nI)
  {
    # disease model
    D[i] ~ Bernoulli(theta[i]);
    Logit(theta[i]) <- beta[nA1] + InProd(beta[1:nA],E[i,]);
    for (k in 1:nA)
    {
      # exposure model
      E[i,k] ~ Bernoulli(pi[J[i],k]);
    }
  }
  for (j in 1:nJ)
  {
    for (k in 1:nA)
    {
      # measurement model
      m[j,k] ~ Binomial(pi[j,k],n[j]);
      Logit(pi[j,k]) <- phi[j,k];
      phi[j,k] ~ Normal(mu_star[j,k],tau_star[j,k]);
    }
  }
  for (k in 1:nA1)
  {
    beta[k] ~ Normal(mu[k],tau[k]);
  }
}
```

Fig. 2. Model specification in BUGS

treated as a free-model parameter in the Gibbs sampling and updated at each iteration. Any variables found in the data file are assumed known and will not be updated.

It is important to understand that model specification statements in BUGS are *declarative*, i.e. they are not instructions to perform calculations *per se*; rather, they are declarations, specifying the type of each node and its relationships with other nodes. Indeed, in the Gibbs sampling, $D[i]$ is never actually *sampled* from the Bernoulli distribution specified in expression (9), since $D[i]$ is fixed. Thus the order of model specification statements in BUGS is, to a considerable extent, arbitrary. For example, the 'for' loops in Fig. 2 could have been specified in a different order, perhaps starting with the measurement model loop. Equally, the order of statements within the inner loops in Fig. 2 is arbitrary.

4. Class of models

The class of models that we would like to be able to handle in BUGS is extensive. This class should include most if not all of the models that have been applied in the Gibbs sampling literature, including cluster analysis models (Gilks *et al.*, 1989), survival analysis models

(Clayton, 1991), image analysis models (Besag *et al.*, 1991), econometric models (Blattberg and George, 1991), generalized linear random effects models (Zeger and Karim, 1991; Dellaportas and Smith, 1993), complex genetic models (Sheehan and Thomas, 1993; Thompson and Guo, 1991; Thomas, 1992), disease mapping models (Bernardinelli and Montomoli, 1992; Clayton and Bernardinelli, 1992), hidden Markov models (Kirby, 1992), changepoint models (Carlin *et al.*, 1992); non-linear pharmacokinetic models (Wakefield *et al.*, 1994; Berzuini *et al.*, 1992) and constrained data models (Gelfand *et al.*, 1992).

5. Structure of BUGS

BUGS comprises three distinct modules: the parser, the code generator and the Gibbs sampler.

5.1. *Parser*

The parser is responsible for reading the user's code in the specification file and assimilating node declarations and dependences. For example, in Fig. 2, node $E[i, k]$ is set up as a stochastic node having a Bernoulli distribution, and a pointer to node pi$[J[i], k]$ is stored at node $E[i, k]$. Thus the parser builds up the partial ordering, the *node tree*, representing the full structure of the graphical model. For deterministic nodes, such as that defining theta$[i]$ in Fig. 2, the functional form of the deterministic relationship is stored in the form of a stack associated with the node, containing instructions in *reverse Polish* notation. For example, the expression $b * x + c$ would be stored as a stack: $b, x, *, c, +$, which will be interpreted as 'push b; push x; pop b and x; push b * x; push c; pop b * x and c; push b * x + c'.

The parser reads data from the data file specified by the user and stores these data at the relevant nodes. The parser also discerns which nodes are to be updated in Gibbs sampling and which are to remain fixed, as described in Section 3. Initial values for model parameters and any missing data are obtained from the user-defined inits file. Any free-model parameter that is not found in the inits file is assigned an initial value by forward sampling, provided that the user did not assign it an improper prior. Currently, all data and initial values read into BUGS must be stored in the input files in the form of S objects.

A number of consistency checks are carried out by the parser, e.g. to ensure that all nodes that do not appear on the left-hand side of a model statement are assigned values in the data file, and to check that each stochastic and deterministic function has the right number of arguments, of the right type.

5.2. *Code generator*

The code generator fills out the node tree constructed by the parser, to store at each node pointers to its stochastic children (see Section 2). For example, in the example in Fig. 2, pointers to nodes $E[i, k]$ for all individuals i in job j are stored at node phi$[j, k]$.

Having filled out the node tree, the code generator uses a small expert system to decide, for each node, how best to sample from its full conditional distribution in the Gibbs sampling module (see Section 6).

The code generator generates a low level language which is outputted to a system file. The contents of this file are intelligible to the user, although it should not be necessary for the user to concern himself with this. The low level language describes all the attributes of each node, including references to parents and stochastic children, as well as an indicator of the method to be used for sampling from its full conditional distribution. In this low level language, subscripts are not used: each node has equal status with all other nodes, regardless of whether it derived originally from a subscripted variable.

5.3. *Gibbs sampler*

The Gibbs sampler operates on the low level code produced by the code generator. In principle, the Gibbs sampler could be run on a machine other than that which produced the low level code, since the low level code can be read from a file. The Gibbs sampling module interprets the low level code, in particular the reverse Polish of deterministic nodes, constructs appropriate quantities relating to full conditional distributions (see Section 6), samples from full conditionals, controls the order in which nodes are resampled and prints results to a file in the form of S objects.

6. Full conditional distributions

6.1. *Construction of full conditionals*

The *full conditional distribution* of a stochastic node is its conditional distribution conditioning on the current values of all other stochastic nodes in the graph. For a DAG model, the full conditional distribution for a given stochastic node is proportional to the product of the submodel declaring the given node with the submodels declaring the stochastic children of the given node (see Section 2). For example, the full conditional for β_1 in model (1)–(7) is proportional to

$$p(\beta_1 | \mu_1, \sigma_1) \prod_i p(D_i | \theta_i) \tag{11}$$

where deterministic equation (2) holds. The proportionality constant, required to make the full conditional integrate to 1, will be a function of nodes other than β_1. However, methods of sampling from full conditionals used in BUGS (see below) do not require explicit evaluation of integration constants.

BUGS also has a mechanism for dealing with models which are not DAG models, e.g. image analysis models (Besag *et al.*, 1991) and disease mapping models (Bernardinelli and Montomoli, 1992; Clayton and Bernardinelli, 1992). There is no obvious rule here, analogous to the directed Markov assumption, for obtaining joint distributions from user-specified submodels. Moreover, it is easy to construct sets of submodels for which no consistent joint distribution exists. The rule used by BUGS in constructing the full conditional distribution for a given node in a non-DAG model is to form the product of the submodel declaring the given node with the submodels declaring the stochastic children of the given node, *excluding* any submodels declaring stochastic children which are also stochastic parents of the given node. This rule may not always make sense, but it allows BUGS to accommodate the above non-DAG models and would allow, for example, the full conditional for a given node to be supplied *explicitly* by the user.

Note that in expression (11) the full conditional for β_1 involves only a subset of the other nodes in the graph. This is generally true for DAG models and is important for computational efficiency. The ingredients required for constructing a full conditional for a given node are all identified explicitly through the list of pointers at that node in the low-level language produced by the code generator.

6.2. *Sampling from full conditionals*

The code generator contains a small expert system for deciding the best method of sampling from full conditionals. Currently the first choice is to identify conjugacy, where the full conditional reduces analytically to a well-known distribution, and to sample accordingly. For example, if $y \sim N(\mu, \sigma^2)$ and $\mu \sim N(\nu, \tau^2)$, then the full conditional for μ will be

$$N\left(\frac{y\tau^2 + \nu\sigma^2}{\tau^2 + \sigma^2}, \frac{1}{\sigma^{-2} + \tau^{-2}}\right).$$

The expert system recognizes a small number of canonical cases of conjugacy and also recognizes conjugacy in situations which reduce to a canonical case through linear transforms. For example, if μ is replaced with a linear form $a + b\mu$, then the full conditional for μ still reduces through conjugacy to

$$N\left\{\frac{b(y-a)\tau^2 + v\sigma^2}{b^2\tau^2 + \sigma^2}, \frac{1}{b^2\sigma^{-2} + \tau^{-2}}\right\}.$$

The method of second choice for sampling from full conditionals is adaptive rejection sampling. The original method of adaptive rejection sampling (Gilks and Wild, 1992) requires the density to be log-concave ($\mathrm{d}^2\{\log f(x)\}/\mathrm{d}x^2 \leqslant 0$) and the user to supply subroutines to calculate derivatives of the log-density. A second version (Gilks, 1992) does not use derivatives but still requires log-concavity. The most recent version (Gilks *et al.*, 1994) accommodates non-log-concavity through use of a generalized rejection function and a post-acceptance Hastings–Metropolis step (Hastings, 1970).

7. Discussion

The BUGS program is still under development, but a version is currently undergoing trials at several sites worldwide. We have several plans for further development. In the short term we plan to allow input of data which are not S objects, to allow specification of multivariate priors and to implement the method of Gilks *et al.* (1994) for sampling from non-log-concave full conditional distributions. In the longer term we hope to include facilities for diagnosing convergence of the Gibbs sampler, and for diagnosing model adequacy.

References

Bernardinelli, L. and Montomoli, C. (1992) Empirical Bayes vs fully Bayesian analysis of geographical variation in disease risk. *Statist. Med.*, **11**, 983–1007.

Berzuini, C., Bellazzi, R., Quaglini, S. and Spiegelhalter, D. J. (1992) Bayesian networks for patient monitoring. *Artif. Intell. Med.*, **4**, 243–260.

Besag, J., York, J. and Mollié, A. (1991) Bayesian image restoration, with two applications in spatial statistics (with discussion). *Ann. Inst. Statist. Math.*, **43**, 1–59.

Blattberg, R. C. and George, E. I. (1991) Shrinkage estimation of price and promotional elasticities: seemingly unrelated equations. *J. Am. Statist. Ass.*, **86**, 304–315.

Carlin, B. P., Gelfand, A. E. and Smith, A. F. M. (1992) Hierarchical Bayesian analysis of changepoint problems. *Appl. Statist.*, **41**, 389–405.

Casella, G. and George, E. I. (1992) Explaining the Gibbs sampler. *Am. Statistn*, **46**, 167–174.

Clayton, D. G. (1991) A Monte Carlo method for Bayesian inference in frailty models. *Biometrics*, **47**, 467–485.

Clayton, D. G. and Bernardinelli, L. (1992) Bayesian methods for mapping disease risk. In *Geographical and Environmental Epidemiology* (eds J. Cuzick and P. Elliott). Oxford: Oxford University Press.

Dellaportas, P. and Smith, A. F. M. (1993) Bayesian inference for generalized linear models and proportional hazards models via Gibbs sampling. *Appl. Statist.*, **42**, 443–459.

Gelfand, A. E., Hills, S. E., Racine-Poon, A. and Smith, A. F. M. (1990) Illustration of Bayesian inference in normal data models using Gibbs sampling. *J. Am. Statist. Ass.*, **85**, 972–985.

Gelfand, A. E. and Smith, A. F. M. (1990) Sampling based approaches to calculating marginal densities. *J. Am. Statist. Ass.*, **85**, 398–409.

Gelfand, A. E., Smith, A. F. M. and Lee, T. M. (1992) Bayesian analysis of constrained parameter and truncated data problems. *J. Am. Statist. Ass.*, **87**, 523–532.

Geman, S. and Geman, D. (1984) Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Trans. Pattn Anal. Mach. Intell.*, **6**, 721–741.

Gilks, W. R. (1992) Derivative-free adaptive rejection sampling for Gibbs sampling. In *Bayesian Statistics 4* (eds J. M. Bernardo, J. O. Berger, A. P. Dawid and A. F. M. Smith), pp. 641–649. Oxford: Clarendon.

Gilks, W. R., Best, N. G. and Tan, K. K. C. (1994) Adaptive rejection sampling from non log-concave densities. Submitted to *Appl. Statist.*

Gilks, W. R., Clayton, D. G., Spiegelhalter, D. J., Best, N. G., McNeil, A. J., Sharples, L. D. and Kirby, A. J. (1993) Modelling complexity: applications of Gibbs sampling in medicine. *J. R. Statist. Soc.* B, **55**, 39–52.

Gilks, W. R., Oldfield, L. and Rutherford, A. (1989) Statistical analysis. In *Leucocyte Typing IV: White Cell Differentiation Antigens* (eds W. Knapp, B. Dorken, W. R. Gilks, E. P. Rieber, R. E. Schmidt, H. Stein and A. E. G. Kr. von dem Borne), pp. 6–12. Oxford: Oxford University Press.

Gilks, W. R. and Richardson, S. (1992) Analysis of disease risks using ancillary risk factors, with application to job-exposure matrices. *Statist. Med.*, **11**, 1443–1463.

Gilks, W. R. and Wild, P. (1992) Adaptive rejection sampling for Gibbs sampling. *Appl. Statist.*, **41**, 337–348.

Hastings, W. K. (1970) Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, **57**, 97–109.

Kirby, A. J. (1992) Statistical modelling for the precursors of cervical cancer. *PhD Thesis.* University of Cambridge, Cambridge.

Lauritzen, S. L., Dawid, A. P., Larsen, B. N. and Leimer, H. G. (1990) Independence properties of directed Markov fields. *Networks*, **20**, 491–505.

Sheehan, N. and Thomas, A. (1993) On the irreducibility of a Markov chain defined on a space of genotype configurations by a sampling scheme. *Biometrics*, **49**, 163–175.

Smith, A. F. M. and Roberts, G. O. (1993) Bayesian computation via the Gibbs sampler and related Markov chain Monte Carlo methods. *J. R. Statist. Soc.* B, **55**, 3–23.

Thomas, D. C. (1992) Fitting genetic data using Gibbs sampling: an application to nevus counts in 38 Utah kindreds. *Cytgenet. Cell Genet.*, **59**, 228–230.

Thompson, E. A. and Guo, S. W. (1991) Evaluation of likelihood ratios for complex genetic models. *IMA J. Math. Appl. Med. Biol.*, **8**, 149–169.

Wakefield, J. C., Smith, A. F. M., Racine-Poon, A. and Gelfand, A. E. (1994) Bayesian analysis of linear and non-linear population models by using the Gibbs sampler. *Appl. Statist.*, **43**, 201–221.

Whittaker, J. (1990) *Graphical Models in Applied Multivariate Statistics.* New York: Wiley.

Zeger, S. L. and Karim, M. R. (1991) Generalized linear models with random effects; a Gibbs sampling approach. *J. Am. Statist. Ass.*, **86**, 79–86.