Bayesian Poisson Regression using the Gibbs Sampler: Sensitivity Analysis through Dynamic Graphics

Hani Doss*
Department of Statistics
Ohio State University
Columbus, Ohio 43210

B. Narasimhan†
Division of Science
Penn State Erie, The Behrend College
Erie, Pennsylvania 16563

August, 1994

Abstract

In a Bayesian analysis one fixes a prior on the unknown parameter, observes the data, and obtains the posterior distribution of the parameter given the data. For a number of problems the posterior cannot be obtained in closed form and one uses instead the Markov chain simulation method, which in effect produces a sequence of random variables distributed approximately from the posterior distribution. These random variables can be used to estimate the posterior or features of it like the posterior expectation and variance. Unfortunately, the Markov chain simulation method requires non-negligible computer time and this precludes consideration of a large number of priors and an interactive analysis. We present a computing environment within which one can interactively change the prior and immediately see the corresponding changes in the posterior. The environment is based on the object-oriented programming language `LISP-STAT` and an importance sampling procedure which enables one to use the output of one or a small number of Markov chains to obtain estimates of the posterior for a large class of priors. The environment is developed for the particular case of Bayesian Poisson regression but the programs have been deliberately structured in such a way that the environment can be easily modified to handle a wide range of Bayesian problems requiring use of Markov chain simulation.

Key words and phrases: Dynamic graphics, Gibbs sampling, Importance sampling.

*Research supported by Air Force Office of Scientific Research Grant F49620-94-1-0028.

†Research partially supported by Graduate School Grant-in-Aid-of-Research, University of Minnesota, Minneapolis, while the author was at the University of Minnesota, Morris.

1 Introduction and Summary

The Bayesian paradigm is summarized as follows. There is an unknown parameter θ. We put a prior distribution ν on θ. We observe the data $\boldsymbol{Y}$, and compute $\nu_{\boldsymbol{Y}}$, the posterior distribution of θ given the data. Unfortunately, the prior distribution ν that we put down is almost always only an approximation to our true opinion about θ, and therefore, the posterior $\nu_{\boldsymbol{Y}}$ is not an exact representation of the opinion that we have on θ after having seen the data. If the posterior does not change much when one changes the prior then one gets a feeling of reassurance—a different investigator with a slightly different prior may not even bother to recompute the posterior for his prior. On the other hand, if the posterior changes significantly when one changes the prior, then it is important to record that fact, so that for example more time is spent on prior elicitation. Therefore, in almost any problem in which one carries out a serious data analysis, one wants to calculate the posterior distribution for a large number of prior distributions, especially in the exploratory stages of the analysis.

Markov chain simulation is a computational method for obtaining posterior distributions that has received considerable recent attention, because it broadens the range of problems that can be analyzed in the Bayesian framework. See Tierney (1994) and Geyer (1992) for recent reviews. In a typical application the method requires non-negligible computer time, and this is unfortunate, since this precludes consideration of a large number of priors.

The objective of this paper is to introduce a computing environment within which one can interactively change the prior ν and immediately see the corresponding changes in the posterior $\nu_{\boldsymbol{Y}}$. Although this environment has been developed for the particular case of Bayesian Poisson regression (described in detail below), the programs have been written explicitly so that they may be easily modified to handle a wide range of Bayesian problems requiring use of Markov chain simulation. This is discussed further in Section 5.

We now give a simplified description of our approach, which is based on an application of importance sampling. Suppose that we are interested in the class of priors ν_h, where h varies over the hyperparameter space $\mathcal{H}$. To simplify our explanation, suppose that we are interested not in the entire posterior $\nu_{h,\boldsymbol{Y}}$, but in the expectation $\int t(\theta)\,\nu_{h,\boldsymbol{Y}}(d\theta)$, for some function t. For a particular value h_0, the Markov chain simulation method produces a sequence of random variables $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(G)}$ with distribution approximately $\nu_{h_0,\boldsymbol{Y}}$. The expectation $\int t(\theta)\,\nu_{h_0,\boldsymbol{Y}}(d\theta)$ can be estimated by the average $(1/G)\sum_{g=1}^{G} t(\theta^{(g)})$.

Our program begins by fixing a value h_0 of the hyperparameter and running a Markov chain corresponding to the prior ν_{h_0}, producing $\theta^{(1)}, \ldots, \theta^{(G)}$. Given h, a new hyperparameter value of interest, we compute weights $w_h^{(g)}$, $g = 1, \ldots, G$ and estimate $\int t(\theta)\,\nu_{h,\boldsymbol{Y}}(d\theta)$ by the weighted average $\sum_{g=1}^{G} w_h^{(g)} t(\theta^{(g)})$. The computation of the weights $w_h^{(g)}$, which requires only knowledge of the sequence $\theta^{(g)}$, $g = 1, \ldots, G$ and of the two priors ν_{h_0} and ν_h, can be done rapidly enough fast to permit interactive exploration of the effect of changing the prior.

(In actual practice, the importance sampling method based on a single hyperparameter value h_0 works well only for h close enough to h_0 so that not just a few of the weights dominate the average. For this reason it is better to base the inference on the output

of k Markov chains, corresponding to appropriately located hyperparameters $h_1, \ldots, h_k$. We do this using a method described in a recent paper by Geyer (1994). A detailed description of how this is done is given in Section 3.)

The Poisson regression model we consider is

$$Y_i \sim \text{Poisson}(\lambda_i), \quad i = 1, 2, \ldots, n \tag{1.1}$$

where

$$\lambda_i = \exp\Big(\sum_{j=1}^{p} x_{ij}\beta_j\Big). \tag{1.2}$$

In (1.2), the x_{ij}'s are covariates and $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_p)$ is the vector of unknown parameters. Equation (1.2) specifies the "canonical link" function, in the language of generalized linear models. We take our prior on $\boldsymbol{\beta}$ to be the product of independent normals.

With this choice of prior and of the link function, it turns out that the Gibbs sampler is a convenient Markov chain to run. We emphasize that the dynamic graphics requires only the output of one (or a few) Markov chains, and is completely unaffected by how these chains are generated. A user can choose different priors and/or link functions as long as he is able to produce Markov chain output to start the program.

This paper is organized as follows. In Section 2 we show how Gibbs sampling is easy to implement in our model. In Section 3 we discuss importance sampling, and show how to pool the output of k chains to obtain a distribution with respect to which importance sampling can be carried out. The main contribution of this work is described in Section 4. In Section 4.1 we outline the main features of our programs and explain how to use them; we also give two examples. Section 4.2 is more technical and explains the object-oriented structure of our programs. It is necessary to understand this structure in order to use the programs in a more flexible way than in our illustrations, and to modify them to handle other problems involving importance sampling in Markov chain Monte Carlo. We conclude with some final remarks in Section 5.

2 The Gibbs Sampling Algorithm

We begin by reviewing the Gibbs sampling algorithm and then show how it can be used in our model. The Gibbs sampler is an algorithm for estimating the joint distribution $\pi = \pi_{X_1,\ldots,X_p}$ of the random variables $(X_1, \ldots, X_p)$ which works iteratively by updating the coordinates one at a time, as follows. Assume that the conditional distributions $\pi_{X_i|\{X_j, j\neq i\}}$, $i = 1, \ldots, p$ are known or at least that it is possible to generate random observations from these conditional distributions. Fix an arbitrary starting point $X_1^{(0)}, \ldots, X_p^{(0)}$ in the support of π, generate $X_1^{(1)}$ from $\pi_{X_1|\{X_j, j\neq 1\}}(\cdot, X_2^{(0)}, \ldots, X_p^{(0)})$, generate $X_2^{(1)}$ from $\pi_{X_2|\{X_j, j\neq 2\}}(X_1^{(1)}, \cdot, X_3^{(0)}, \ldots, X_p^{(0)})$, and so on until $X_p^{(1)}$ is generated from $\pi_{X_p|\{X_j, j\neq p\}}(X_1^{(1)}, \ldots, X_{p-1}^{(1)}, \cdot)$. This completes one iteration of the scheme. After m iterations we obtain a random variable $(X_1^{(m)}, \ldots, X_p^{(m)})$. The sequence $(X_1^{(j)}, \ldots, X_p^{(j)})$, $j = 1, 2, \ldots$ is a Markov chain, and it is not difficult to see that $\pi_{X_1,\ldots,X_p}$ is a stationary distribution of the chain. If one can establish that this chain converges to its stationary distribution, then for large m, $(X_1^{(m)}, \ldots, X_p^{(m)})$ has a distribution which is approximately

equal to $\pi_{X_1,\ldots,X_p}$. For our purposes we can estimate $\pi_{X_1,\ldots,X_p}$ for example by running one very long chain and after an initial "burn-in" period, retaining every l^{th} value, with l large enough to ensure that there is little correlation between subsequent values. For a discussion of how to use the output of a Markov chain, see Geyer (1992).

We now show how the Gibbs sampler can be used in our situation. The likelihood function for the model given by (1.1) and (1.2) is

$$l_{\boldsymbol{Y}}(\boldsymbol{\beta}) = \Big(\prod_{i=1}^{n} Y_i!\Big)^{-1} \exp\bigg(-\sum_{i=1}^{n} \exp\Big(\sum_{j=1}^{p} x_{ij}\beta_j\Big) + \sum_{j=1}^{p} \beta_j \sum_{i=1}^{n} x_{ij} Y_i \bigg). \tag{2.1}$$

We take the prior ν on $\boldsymbol{\beta}$ to be the product of independent normals with means a_j and variance b_j, i.e. the joint density of the β_j's is given by

$$\nu'(\beta_1, \beta_2, \ldots, \beta_p) = \prod_{j=1}^{p} \frac{1}{(2\pi b_j)^{\frac{1}{2}}} \exp\Big(-\frac{1}{2b_j}(\beta_j - a_j)^2 \Big).$$

Therefore, the posterior distribution of the β_j's given the data has joint density

$$\nu'_{\boldsymbol{Y}}(\beta_1, \beta_2, \ldots, \beta_p) \propto \exp\bigg(-\sum_{j=1}^{p} \frac{1}{2b_j}\beta_j^2 + \sum_{j=1}^{p} d_j \beta_j - \sum_{i=1}^{n} \exp\Big(\sum_{j=1}^{p} x_{ij}\beta_j\Big)\bigg), \tag{2.2}$$

where $d_j = a_j/b_j + \sum_{i=1}^{n} x_{ij} Y_i$, $j = 1, 2, \ldots, p$. Thus, denoting the conditional density of β_s, given the rest of the β_j's and the data by $\nu'_{\boldsymbol{Y},s}$ we see from (2.2) that

$$\nu'_{\boldsymbol{Y},s}(\beta_1, \beta_2, \ldots, \beta_p) \propto \exp\bigg(\frac{1}{2b_s}\beta_s^2 + d_s\beta_s - \sum_{i=1}^{n} \exp(c_i + x_{is}\beta_s)\bigg), \tag{2.3}$$

where $c_i = \sum_{j \neq s} x_{ij}\beta_j$. The features of this density that we need are that it is known up to a normalizing constant, and it is log-concave since

$$\frac{d^2}{d\beta_s^2} \ln \nu'_{\boldsymbol{Y},s}(\beta_1, \beta_2, \ldots, \beta_p) = -\frac{1}{b_s} - \sum_{i=1}^{n} x_{is}^2 \exp(c_i + x_{is}\beta_s) < 0.$$

The Adaptive Rejection Sampling technique of Gilks and Wild (1992) can be used to generate observations from any density having these two properties. This technique essentially constructs upper and lower hulls made up of piecewise exponential densities between which the target density is squeezed. The exponentials can be easily generated and a rejection technique is used to return samples according to the target density.

Convergence of this Gibbs sampler is established in Appendix A.1.

3 Importance Sampling

Suppose that for each $\ell = 1, \ldots, k$, we run a Markov chain corresponding to the prior ν_{h_ℓ}, and obtain as output the values $\boldsymbol{\beta}^{(\ell,1)}, \ldots, \boldsymbol{\beta}^{(\ell,G)}$. For each $\ell = 1, \ldots, k$ and $g = 1, \ldots, G$,

the distribution of $\boldsymbol{\beta}^{(\ell,g)}$ is (approximately) $\nu_{h_\ell,\boldsymbol{Y}}$. Now let h be a new hyperparameter value and suppose that we wish to estimate

$$I_h(t) = \int t(\boldsymbol{\beta})\, \nu_{h,\boldsymbol{Y}}(d\boldsymbol{\beta}). \tag{3.1}$$

for some $\nu_{h,\boldsymbol{Y}}$-integrable function t. We would like to do importance sampling with respect to the probability measure

$$\mu_{\text{I.S.}}(d\boldsymbol{\beta}) = \frac{1}{k}\sum_{\ell=1}^{k} \nu_{h_\ell,\boldsymbol{Y}}(d\boldsymbol{\beta}) = \frac{1}{k}\sum_{\ell=1}^{k} c(h_\ell) l_{\boldsymbol{Y}}(\boldsymbol{\beta})\, \nu_{h_\ell}(d\boldsymbol{\beta}). \tag{3.2}$$

(In (3.2), $c(h_\ell)$ is the normalizing constant corresponding to $l_{\boldsymbol{Y}}(\boldsymbol{\beta})\, \nu_{h_\ell}(d\boldsymbol{\beta})$.) Unfortunately, the constants $c(h_\ell)$ are not known. Suppose temporarily that the vector $(c(h_1), \ldots, c(h_k))$ was known *up to an overall multiplicative constant*. That is, let $d(h_l) = c(h_l)/c(h_1)$ for $\ell = 1, \ldots, k$ and suppose we knew $d(h_2), \ldots, d(h_k)$ in the identity

$$(c(h_1), \ldots, c(h_k)) = c(h_1) \cdot (d(h_1), \ldots, d(h_k)). \tag{3.3}$$

(Note that this is trivially true for the case $k = 1$.) We would then be able to write

$$\begin{aligned}
\int t(\boldsymbol{\beta})\, \nu_{h,\boldsymbol{Y}}(d\boldsymbol{\beta}) &= \int t(\boldsymbol{\beta}) \left[\frac{d\nu_{h,\boldsymbol{Y}}}{d\mu_{\text{I.S.}}}\right](\boldsymbol{\beta})\, \mu_{\text{I.S.}}(d\boldsymbol{\beta}) \\
&= \int t(\boldsymbol{\beta}) \frac{c(h) l_{\boldsymbol{Y}}(\boldsymbol{\beta}) \nu_h'(\boldsymbol{\beta})}{\frac{1}{k}\sum_{\ell=1}^{k} c(h_1)\cdot d(h_\ell) l_{\boldsymbol{Y}}(\boldsymbol{\beta}) \nu_{h_\ell}'(\boldsymbol{\beta})}\, \mu_{\text{I.S.}}(d\boldsymbol{\beta}) \\
&= \frac{\displaystyle\int t(\boldsymbol{\beta}) \frac{c(h)}{c(h_1)} \frac{\nu_h'(\boldsymbol{\beta})}{\frac{1}{k}\sum_{\ell=1}^{k} d(h_\ell)\nu_{h_\ell}'(\boldsymbol{\beta})}\, \mu_{\text{I.S.}}(d\boldsymbol{\beta})}{\displaystyle\int \frac{c(h)}{c(h_1)} \frac{\nu_h'(\boldsymbol{\beta})}{\frac{1}{k}\sum_{\ell=1}^{k} d(h_\ell)\nu_{h_\ell}'(\boldsymbol{\beta})}\, \mu_{\text{I.S.}}(d\boldsymbol{\beta})} \\
&= \frac{\displaystyle\int t(\boldsymbol{\beta}) \frac{\nu_h'(\boldsymbol{\beta})}{\sum_{\ell=1}^{k} d(h_\ell)\nu_{h_\ell}'(\boldsymbol{\beta})}\, \mu_{\text{I.S.}}(d\boldsymbol{\beta})}{\displaystyle\int \frac{\nu_h'(\boldsymbol{\beta})}{\sum_{\ell=1}^{k} d(h_\ell)\nu_{h_\ell}'(\boldsymbol{\beta})}\, \mu_{\text{I.S.}}(d\boldsymbol{\beta})}
\end{aligned} \tag{3.4}$$

where in the third equality in (3.4) we have used the fact that the integral in the denominator is just 1. This is the ratio of two integrals with respect to $\mu_{\text{I.S.}}$ each of which can be estimated from the sequence $\{\boldsymbol{\beta}^{(\ell,g)}\}_{\ell,g}$. Letting

$$\tilde{v}_h(\boldsymbol{u}) = \frac{\nu_h'(\boldsymbol{u})}{\sum_{\ell=1}^{k} d(h_\ell)\nu_{h_\ell}'(\boldsymbol{u})}, \tag{3.5}$$

we may estimate the numerator and the denominator by

$$\frac{1}{kG}\sum_{\ell=1}^{k}\sum_{g=1}^{G} t(\boldsymbol{\beta}^{(\ell,g)})\tilde{v}_h(\boldsymbol{\beta}^{(\ell,g)}) \quad \text{and} \quad \frac{1}{kG}\sum_{\ell=1}^{k}\sum_{g=1}^{G} \tilde{v}_h(\boldsymbol{\beta}^{(\ell,g)})$$

respectively. Thus, if we define

$$\tilde{w}_h^{(\ell,g)} = \frac{\tilde{v}_h(\boldsymbol{\beta}^{(\ell,g)})}{\sum_{q=1}^{k}\sum_{r=1}^{G}\tilde{v}_h(\boldsymbol{\beta}^{(q,r)})}, \tag{3.6}$$

we see that $I_h(t)$ may be estimated by

$$\tilde{I}_h(t) = \sum_{\ell=1}^{k}\sum_{g=1}^{G}\tilde{w}_h^{(\ell,g)}t(\boldsymbol{\beta}^{(\ell,g)}). \tag{3.7}$$

Note from (3.5) that the calculation of $\tilde{v}_h(\boldsymbol{\beta}^{(\ell,g)})$ requires only the values of the prior densities ν_h' and $\nu_{h_1}', \ldots, \nu_{h_k}'$ at $\boldsymbol{\beta}^{(\ell,g)}$, and that the sequence $\{\sum_{\ell=1}^{k} d(h_\ell)\nu_{h_\ell}'(\boldsymbol{\beta}^{(q,r)})\}_{q,r}$ can be precomputed and stored. It is this feature of the calculation that enables rapid recomputation of the weights as we change the value of h.

Unfortunately, in general it is not possible to know the values $d(h_2), \ldots, d(h_k)$. However, Geyer (1994) proposes an interesting method for estimating these. A brief description of his method is as follows. Consider the *set* $\mathcal{M} = \{\boldsymbol{\beta}^{(\ell,g)},\ \ell = 1, \ldots, k;\ g = 1, \ldots, G\}$. If we select at random an element of that set and note that its value is $\boldsymbol{e}$ then the probability that this element came from the ℓ^{th} Markov chain is

$$\frac{c(h_\ell)l(\boldsymbol{e})\nu_{h_\ell}'(\boldsymbol{e})}{\sum_{q=1}^{k}c(h_q)l(\boldsymbol{e})\nu_{h_q}'(\boldsymbol{e})} = \frac{c(h_\ell)\nu_{h_\ell}'(\boldsymbol{e})}{\sum_{q=1}^{k}c(h_q)\nu_{h_q}'(\boldsymbol{e})} = \frac{d(h_\ell)\nu_{h_\ell}'(\boldsymbol{e})}{\sum_{q=1}^{k}d(h_q)\nu_{h_q}'(\boldsymbol{e})}.$$

Treating now the vector $(d(h_2), \ldots, d(h_k))$ as an unknown parameter, we can form the "likelihood"

$$p(d(h_2), \ldots, d(h_k)) = \prod_{\ell=1}^{k}\prod_{g=1}^{G}\frac{d(h_\ell)\nu_{h_\ell}'(\boldsymbol{\beta}^{(\ell,g)})}{\sum_{q=1}^{k}d(h_q)\nu_{h_q}'(\boldsymbol{\beta}^{(\ell,g)})}. \tag{3.8}$$

Of course, the expression in (3.8) is not a likelihood in the usual sense because when we select kG elements without replacement from the set $\mathcal{M}$ we no longer have independence. Nevertheless, Geyer (1994) shows that (under certain regularity conditions) the vector $(\hat{d}(h_2), \ldots, \hat{d}(h_k))$ that maximizes (3.8), behaves much like ordinary maximum likelihood estimates. In particular, $G^{1/2}\Big((\hat{d}(h_2), \ldots, \hat{d}(h_k)) - (d(h_2), \ldots, d(h_k))\Big)$ has an asymptotic normal distribution.

Since $d(h_2), \ldots, d(h_k)$ are unknown, in actual fact, we do not form (3.5) and (3.6), but rather form

$$\hat{v}_h(\boldsymbol{u}) = \frac{\nu_h'(\boldsymbol{u})}{\sum_{\ell=1}^{k}\hat{d}(h_\ell)\nu_{h_\ell}'(\boldsymbol{u})} \quad \text{and} \quad \widehat{w}_h^{(\ell,g)} = \frac{\hat{v}_h(\boldsymbol{\beta}^{(\ell,g)})}{\sum_{q=1}^{k}\sum_{r=1}^{G}\hat{v}_h(\boldsymbol{\beta}^{(q,r)})},$$

and estimate the integral in (3.1) by

$$\hat{I}_h(t) = \sum_{\ell=1}^{k}\sum_{g=1}^{G}\widehat{w}_h^{(\ell,g)}t(\boldsymbol{\beta}^{(\ell,g)}). \tag{3.9}$$

The complete data produced by the ℓ^{th} Markov chain is a sequence of dependent random vectors. In our situation we will need to quickly manipulate G points from each

of the k chains, and this puts a limit on how large G can be. For this reason, we would like to have the G points from each chain to be nearly independent, and this can be achieved either by running one long chain and retaining only every l^{th} vector, for a large value of l, or running G independent chains and retaining the last vector in each. In the rest of this paper, we assume that for each ℓ, the sequence $\boldsymbol{\beta}^{(\ell,1)}, \ldots, \boldsymbol{\beta}^{(\ell,G)}$ is iid from $\nu_{h_\ell, \boldsymbol{Y}}$. Moreover, we assume that the k sequences $\{\boldsymbol{\beta}^{(1,1)}, \ldots, \boldsymbol{\beta}^{(1,G)}\}, \ldots, \{\boldsymbol{\beta}^{(k,1)}, \ldots, \boldsymbol{\beta}^{(k,G)}\}$ are independent.

From a practical point of view, one would like to know the range of h's for which (3.9) gives stable estimates. In Appendix A.2 we consider this question and also give a formula for estimating the standard error of (3.9).

In actual practice we are often interested in estimating $\nu'_{h,\boldsymbol{Y},j}$, the density of the marginal posterior distribution of β_j. This density cannot be written in the form (3.9) for any function t. We note that the estimate (3.9) can be written as $\int t(\boldsymbol{\beta})\, F_{W_h}(d\boldsymbol{\beta})$ where F_{W_h} is the discrete distribution that gives mass $\widehat{w}_h^{(\ell,g)}$ to $\boldsymbol{\beta}^{(\ell,g)}$. By analogy with kernel density estimation, we may estimate $\nu'_{h,\boldsymbol{Y},j}$ by the convolution $K \star F^{(j)}_{W_h}$ where $F^{(j)}_{W_h}$ is the distribution on R^1 that gives mass $\widehat{w}_h^{(\ell,g)}$ to $\beta_j^{(\ell,g)}$ and K is a suitable kernel.

We note that it would be more efficient to estimate $\nu'_{h,\boldsymbol{Y},j}(\cdot)$ by the "mixture estimate" $\sum_{\ell=1}^{k}\sum_{g=1}^{G} \widehat{w}_h^{(\ell,g)} \nu'_{\boldsymbol{Y},s}(\cdot, \beta_2^{(\ell,g)}, \ldots, \beta_p^{(\ell,g)})$ (see (2.3)). Unfortunately, we cannot use this estimate because $\nu'_{\boldsymbol{Y},s}(\cdot, \beta_2^{(\ell,g)}, \ldots, \beta_p^{(\ell,g)})$ is not available in closed form.

4 Dynamic Graphics

Our programs are written in the language `LISP-STAT` (Tierney 1990). This is a powerful object-oriented environment for statistical computing. It is very useful for managing several graphical windows in an integrated manner as is required for our purpose.

Our code consists of several `Lisp` programs and auxiliary programs written in C that handle the reweighting. The installation of `LISP-STAT` must have "dynamic loading" enabled. This is a facility that allows one to call C programs from `LISP-STAT`. Since the reweighting requires substantial computing effort it is necessary that this be done in a compiled language, such as C, in order to have the posteriors updated "immediately". A `Lisp` program that calculates the estimates of the constants $d(h_1), \ldots, d(h_k)$ in equation (3.3) is included. Instructions on how to obtain and install the software are given in Section 5.

This section is organized as follows. Section 4.1 gives an introduction to use of the software. It includes two examples, one involving a standard Poisson regression model, and the other involving a hierarchical Poisson model. There is no difficulty in using the programs on this last model, even though it cannot be put into the framework of Poisson regression (and that is part of our reason for choosing this model). Section 4.2 discusses the design of our programs and gives the information required to extend the methods described here to models other than those discussed in this paper.

4.1 Usage of the Programs

Both a line-oriented interface and a user-friendly graphical interface with dialog boxes are available. The graphical interface is actually a front end to the line interface. Use of the graphical interface does not require any knowledge of `LISP-STAT`. Users with some knowledge of `LISP-STAT` will eventually gravitate towards the line-oriented interface because there are many inputs required in a data analysis, and specifying all these via dialog boxes is slower than specifying them through the command line. We mention that the graphical interface actually creates a file containing commands which can be used in subsequent sessions to avoid encountering the dialog boxes once again—one simply invokes `LISP-STAT` with the file name as an argument. The line-oriented interface is illustrated in the examples in Sections 4.1.1 and 4.1.2. Here we discuss the graphical interface. The software is so organized that if one types in what we show in the following discussion, one duplicates the results shown in our figures. To invoke the graphical interface, one types:

```
% xlispstat graphical-interface
```

where `%` denotes the Unix system prompt. The dialog box shown on the left in Figure 1 comes up.

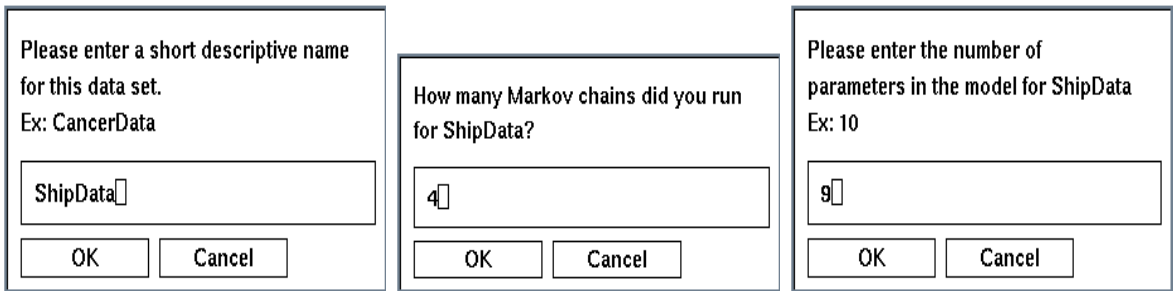


Figure 1: Some dialog boxes of the graphical interface.

Since one could conceivably be dealing with many data sets in a single `LISP-STAT` run, the software prompts the user for a short descriptive name which is used to identify the data set—every window that comes up will have this name in the title. In addition, this name (with the extension `.lsp`) is used as the name of the file containing the `LISP-STAT` commands that can speed up subsequent runs. We have typed in the string "ShipData" in the left dialog box in Figure 1. (Therefore, typing `xlispstat ShipData.lsp` at the Unix prompt in subsequent runs will bypass the need for dialog boxes.) Upon dismissing the first dialog box by clicking on the `OK` button, the dialog box shown in the middle of Figure 1 pops up and the user is asked to enter the number of Markov chains used; we have entered 4. Then, the dialog box shown in the right of the figure asks the user for the number of parameters in the model; we have entered 9. After this, the large dialog box shown in Figure 2 asks the user to enter some additional information needed by the program. The figure is mostly self-explanatory (the inputs in the second column are the estimates of the constants d in (3.3)). At this point, if there are many parameters

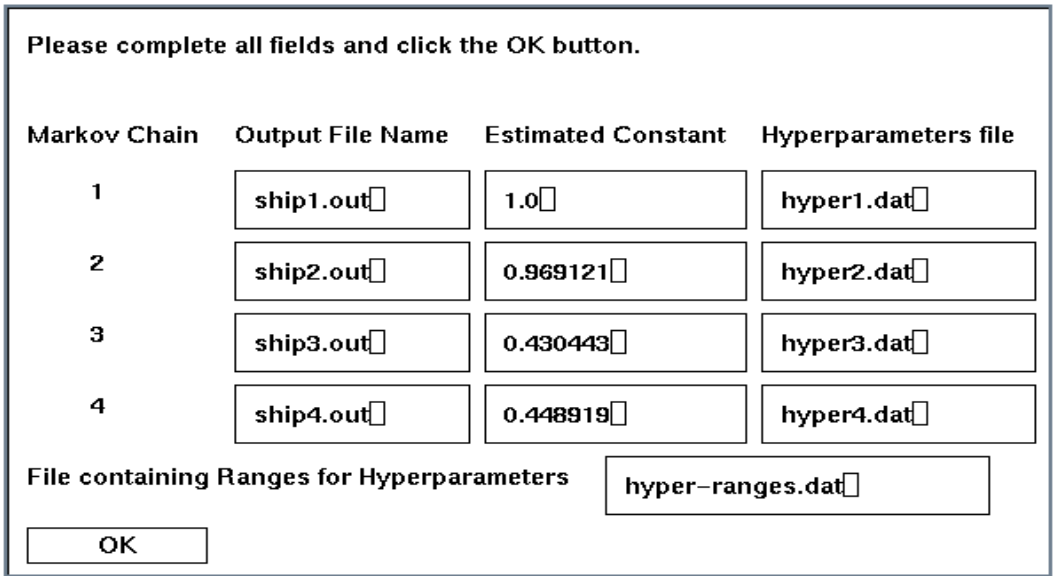


Figure 2: The dialog box for specifying file names and constants.

and hyperparameters, dialog boxes come up which give the user the option of selectively concentrating on certain parameters and hyperparameters; these are not shown here. Finally, a set of windows such as the ones shown in Figure 3 come up. This figure pertains to an example which is discussed in detail in Section 4.1.1. Here we need to know only that the model in this example has 12 parameters, $\beta_0, \beta_1, \ldots, \beta_{11}$, three of which are aliased to zero (so effectively 9 parameters). For our purposes we have chosen to concentrate on a subset of three parameters: β_0, β_2, and β_9.

Figure 3 shows a window titled `ShipData:Master-Object`; it will be referred to as the master window. It controls all others. Figure 3 also shows two small dialog boxes at the bottom. The leftmost box, which pops up when the button titled `More...` in the master window is clicked, can be used to reselect the set of windows to display. Double clicking on the `Window List` displays the dialog box in the bottom middle. The figure shows checkboxes which are checked off for the parameters β_0, β_2 and β_9, indicating that only those windows will be shown. The user can concentrate on any other group of parameters by checking other boxes and clicking the `OK` button. Double clicking on the `Statistics` button creates a table like Table 2. The `Weight Watcher` button in the master window can be used to warn the user when estimates from reweighting can be inaccurate (see Figure 4). At present, the weight watcher shows only a boxplot of the weights; however, it can easily be modified to display instead the value of the maximum weight, or the standard deviation of the weights. Also, if one is interested in estimating the expectation of a particular function $t(\boldsymbol{\beta})$ using $\hat{I}_h(t)$ in (3.9), it is possible to display estimates of the *Monte Carlo* standard deviation of $\hat{I}_h(t)$ (refer to Appendix A.2) in addition to the boxplot. See Section 4.2. Finally, the `Reset` button is provided for convenience—it can be used to reset the values of the hyperparameters to the initial values.

The sliders in the master window enable the user to change the hyperparameter values. As the hyperparameter values change, the windows are refreshed and new estimates of the kernel density estimates are drawn. Notice that the windows displaying marginal posterior density estimates show some statistics in the upper right corner. By default, these are the estimated posterior mean and standard deviation. One can add any other statistic to this list—details of how this may be done appear in Section 4.2. As the hyperparameters are changed, the values of these statistics also change. When a single Markov chain is used, the statistics are displayed in two columns. One column contains the statistics for the unweighted output of the chain, and the other for the reweighted output.

The options button in the upper left corner of each window lets one change certain features of the kernel density estimates via a menu; Figure 4 shows the results of clicking on the `Options` button of the window for β_0. One can change the kernel type, the kernel window width, the number of points at which the kernel density estimate is computed (the default is 30), and the printing format of the statistics, by double clicking on the appropriate menu item. The kernel window width menu item pops up a slider for changing the window width. There are five kernel choices: Biweight, Gaussian, Triangular, Uniform, and Epanechnikov. The default is Gaussian. The same `Options` button also allows a user to decide whether to superimpose the changes as the hyperparameters are changed.

To quit the program, the user clicks on the close button in the master window.

4.1.1 Ship Data

McCullagh and Nelder (1989) give data on a study of wave damage to cargo ships. In order to set standards for hull construction, it was necessary to study the risk of damage associated with the three classifying factors: Ship Type (A–E), Year of Construction (1960–64, 1965–69, 1970–74, 1975–79) and Period of Operation (1960–74, 1975–79). The data are shown in Table 1. For each of 34 ships, the table gives the number of damage incidents (the response variable), the level of the three classifying factors, and also the covariate "aggregate months of service", which is an "offset" (i.e. the parameter corresponding to it is assumed known). McCullagh and Nelder fit the 9-parameter Poisson regression model with categorical predictors

$$\begin{aligned} \log(\lambda) \;=\; & \beta_0 \;+\; \log(\text{aggregate months of service}) \\ & +\; (\text{effect due to ship type}) \\ & +\; (\text{effect due to year of construction}) \\ & +\; (\text{effect due to period of operation}). \end{aligned}$$

Table 2 shows the maximum likelihood estimates obtained by McCullagh and Nelder.

We considered the priors π_1, π_2, π_3, and π_4 shown in Table 2. The Gibbs sampler was used to produce a sample of 500 independent points from the posterior corresponding to each prior. We estimated the values $d(h_2), d(h_3)$, and $d(h_4)$ in equation (3.3) to be $0.969, 0.430$, and 0.449 to three decimal places ($d(h_1) = 1$).

The code used to invoke our programs is shown below. We have numbered the lines of code for easy reference; the numbers do not form part of the program. The commands below are the ones that the graphical interface produces and stores in a file for future use.

Ship Type	Year of construction	Period of operation	Aggregate months service	Number of damage incidents
A	1960–64	1960–74	127	0
A	1960–64	1975–79	63	0
A	1965–69	1960–74	1095	3
A	1965–69	1975–79	1095	4
A	1970–74	1960–74	1512	6
A	1970–74	1975–79	3353	18
A	1975–79	1975–79	2244	11
B	1960–64	1960–74	44882	39
B	1960–64	1975–79	17176	29
B	1965–69	1960–74	28609	58
B	1965–69	1975–79	20370	53
B	1970–74	1960–74	7064	12
B	1970–74	1975–79	13099	44
B	1975–79	1975–79	7117	18
C	1960–64	1960–74	1179	1
C	1960–64	1975–79	552	1
C	1965–69	1960–74	781	0
C	1965–69	1975–79	676	1
C	1970–74	1960–74	783	6
C	1970–74	1975–79	1948	2
C	1975–79	1975–79	274	1
D	1960–64	1960–74	251	0
D	1960–64	1975–79	105	0
D	1965–69	1960–74	288	0
D	1965–69	1975–79	192	0
D	1970–74	1960–74	349	2
D	1970–74	1975–79	1208	11
D	1975–79	1975–79	2051	4
E	1960–64	1960–74	45	0
E	1965–69	1960–74	789	7
E	1965–69	1975–79	437	7
E	1970–74	1960–74	1157	5
E	1970–74	1975–79	2161	12
E	1975–79	1975–79	542	1

Table 1: Data on number of reported damage accidents and aggregate months of service by ship type, year of construction and period of operation. Source: McCullagh and Nelder (1989).

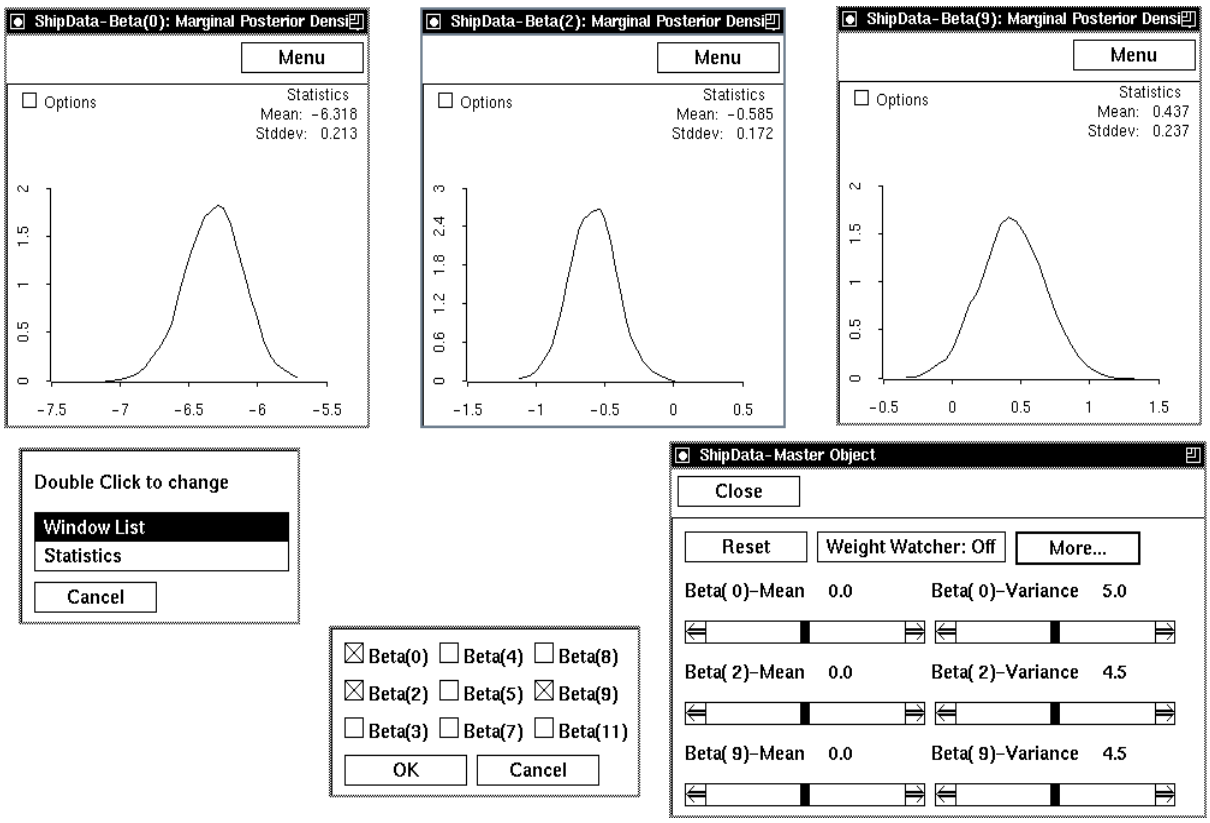


Figure 3: The posterior marginal densities for the Ship data using Normal priors.

Minimal acquaintance with these commands is sufficient to enable the user to edit the file to obtain useful modifications.

```
1. (require "master")
2. (defvar *param-labels* '("Beta(0)" "Beta(2)" "Beta(3)" "Beta(4)"
          "Beta(5)" "Beta(7)" "Beta(8)" "Beta(9)" "Beta(11)"))
3. (def d-files '("ship1.out" "ship2.out" "ship3.out" "ship4.out"))
4. (def hlist
    '(((-10 10) (-3 9) (-3 9) (-3 8) (-3 8) (3 9) (3 9) (3 9) (3 8))
      ((-10 10) (-3 9) (3 9) (3 8) (3 8) (3 9) (3 9) (3 9) (3 8))
      ((5.0 10) (3 9) (3 9) (3 8) (3 8) (-3 9) (-3 9) (-3 9) (-3 8))
      ((5.0 10) (3 9) (-3 9) (-3 8) (-3 8) (-3 9)(-3 9) (-3 9) (-3 8))))
5. (def rlist '((-10 10) (0 10) (-3 3) (0 9) (-3 3) (0 9) (-3 3) (0 8)
      (-3 3) (0 8) (-3 3) (0 9) (-3 3) (0 9) (-3 3) (0 9) (-3 3) (0 8)))
6. (def hnames
    '("Beta( 0)-Mean" "Beta( 0)-Variance" "Beta( 2)-Mean"
      "Beta( 2)-Variance" "Beta( 3)-Mean" "Beta( 3)-Variance"
      "Beta( 4)-Mean" "Beta( 4)-Variance" "Beta( 5)-Mean"
      "Beta( 5)-Variance" "Beta( 7)-Mean" "Beta( 6)-Variance"
```

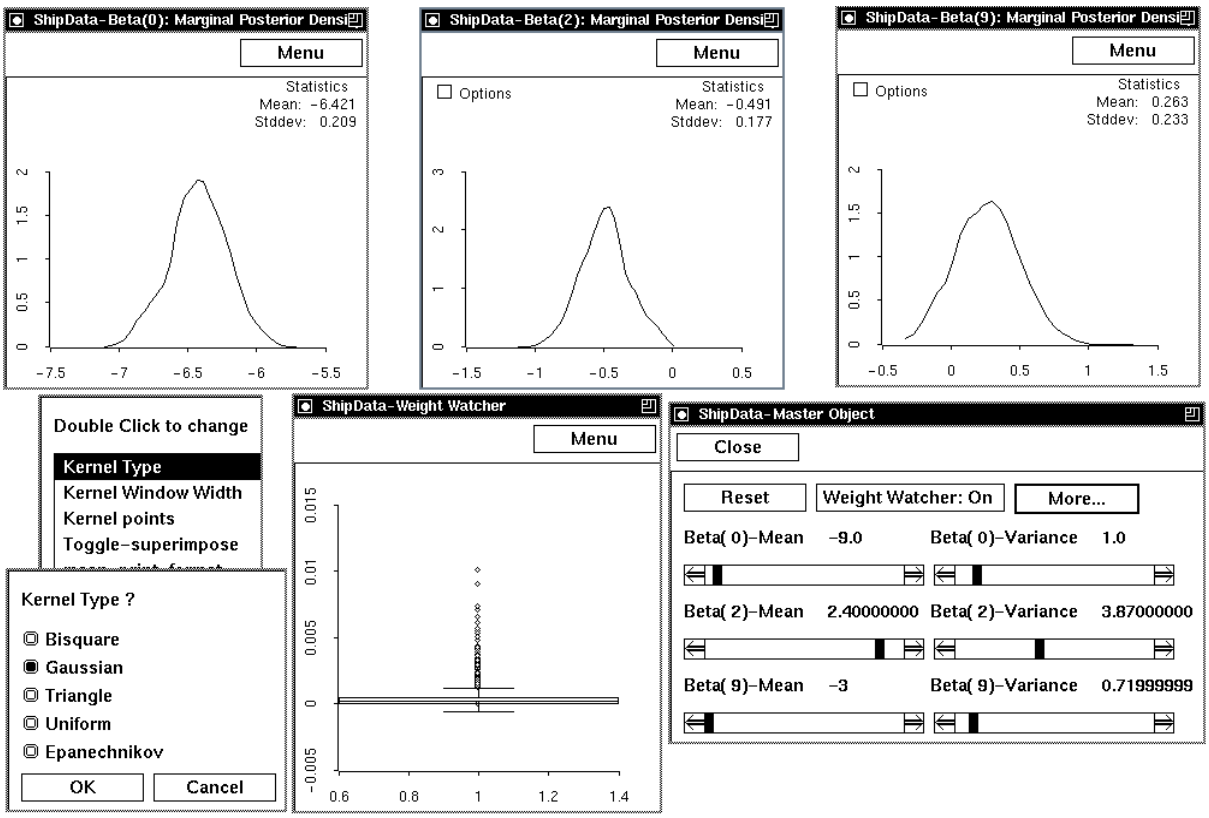


Figure 4: The effect of changing the hyperparameters for the Ship Data.

```
      "Beta( 8)-Mean" "Beta( 8)-Variance" "Beta( 9)-Mean"
      "Beta( 9)-Variance" "Beta(11)-Mean" "Beta(11)-Variance"))
 7. (def wlist '(t t nil nil nil nil nil t nil))
 8. (def slist '((t t) (t t) nil nil nil nil nil nil
                 nil nil nil nil (t t) nil nil))
 9. (def c '(1.0 0.969 0.430 0.449))
10. (setf posterior (send master-proto :new :data d-files :constants c
                          :p 9 :hyper-param-names hnames
                          :identifier "ShipData" :hyper-vals hlist
                          :hyper-range-list rlist :hyper-show-list slist
                          :create-window-list wlist))
```

Line 1 loads our programs if they are not already loaded. Line 2 defines some meaningful names for our parameters such as `Beta(0)`, `Beta(2)`, etc. Line 3 defines a list of names of files containing the outputs of the Gibbs sampler. For example, the file `ship1.out` should contain the output of the Gibbs sampler run using prior π_1, `ship2.out` should contain the output of the Gibbs sampler run at prior π_2, and so on. Line 4 defines a variable `hlist` which is a list of hyperparameter values at which the Gibbs samplers were run. This gives the program information about the priors used for each data file. Since

Table 2: Estimates obtained by maximum likelihood and reweighting mixtures. The priors π_1, π_2, π_3, π_4 and π are the product of the densities in the corresponding rows; "Prior π (direct)" refers to estimates obtained by running only one Markov chain corresponding to π.

	Parameters								
Method	β_0	β_2	β_3	β_4	β_5	β_7	β_8	β_9	β_{11}
MLE									
Est.	−6.41	−0.54	−0.69	−0.08	0.33	0.70	0.82	0.45	0.38
Std. Err.	0.22	0.23	0.43	0.38	0.31	0.19	0.22	0.30	0.15
Prior π_1									
Density	$N(-10,10)$	$N(-3,9)$	$N(-3,9)$	$N(-3,8)$	$N(-3,8)$	$N(3,9)$	$N(3,9)$	$N(3,9)$	$N(3,8)$
Post. mean	−6.38	−0.57	−0.76	−0.16	0.27	0.69	0.80	0.50	0.37
Post. Std. Dev.	.22	.17	.34	.30	.24	.15	.17	.24	.12
Prior π_2									
Density	$N(-10,10)$	$N(-3,9)$	$N(3,9)$	$N(3,8)$	$N(-3,8)$	$N(3,9)$	$N(3,9)$	$N(3,9)$	$N(3,8)$
Post. mean	−6.43	−0.51	−0.65	−0.06	0.34	0.70	0.80	0.51	0.37
Post. Std. Dev.	.22	.18	.33	.30	.24	.15	.17	.24	.12
Prior π_3									
Density	$N(5,10)$	$N(3,9)$	$N(3,9)$	$N(3,8)$	$N(3,8)$	$N(-3,9)$	$N(-3,9)$	$N(-3,9)$	$N(-3,8)$
Post. mean	−6.34	−0.56	−0.69	−0.08	0.32	0.64	0.73	0.42	0.37
Post. Std. Dev.	.22	.18	.33	.30	.24	.15	.17	.24	.12
Prior π_4									
Density	$N(5,10)$	$N(3,9)$	$N(-3,9)$	$N(-3,8)$	$N(-3,8)$	$N(-3,9)$	$N(-3,9)$	$N(-3,9)$	$N(-3,8)$
Post. mean	−6.31	−0.59	−0.77	−0.16	0.29	0.63	0.73	0.42	0.37
Post. Std. Dev.	.21	.17	.34	.30	.24	.15	.17	.24	.12
Prior π									
Density	$N(-9,1)$	$N(2.4,3.87)$	$N(0,4.5)$	$N(0,4)$	$N(0,4)$	$N(0,4.5)$	$N(0,4.5)$	$N(-3,.72)$	$N(0,4)$
Post. mean	−6.42	−0.49	−0.66	−0.01	0.36	0.63	0.71	0.26	0.43
Post. Std. Dev.	.21	.18	.33	.30	.24	.14	.17	.23	.12
Prior π (direct)									
Post. mean	−6.45	−0.48	−0.66	−0.01	0.33	0.66	0.76	0.30	0.43
Post. Std. Dev.	.21	.17	.33	.30	.24	.15	.17	.23	.12

we used independent normal priors on each of the β's, we have a total of $2 \times 9 = 18$ hyperparameters. For each of these hyperparameters, a range for the sliders to vary in is specified as a list in the variable `rlist` on line 5. Thus the mean of the normal prior on β_0 can be varied anywhere from -10 to 10 and the variance can be varied from 0 to 10, the mean of the normal prior on β_2 can be varied from -3 to 3 and the variance from 0 to 9, etc. We provide a list of names for our hyperparameters in the variable `hnames` in line 6.

With 9 parameters, it would take a non-trivial amount of computation to bring up windows for each parameter. On line 7, the variable `wlist` determines which windows come up. In Lisp, `t` stands for true and `nil` stands for false. Since we choose to concentrate only on three β's, we have three `t`'s in the list.

Since we also have a large number of hyperparameters, we decided to concentrate on three pairs, each pair associated with a normal prior on a β—our choices in the code correspond to β_0, β_1 and β_9. Line 8 defines which hyperparameters are to be manipulated using sliders. The sublists in the list, indicated by items grouped in parentheses, tell the program how to group the sliders for the hyperparameters. Thus hyperparameters 1 and 2 will be together in one row, as will others corresponding to the `(t t)`'s in the list. Line 9 lists the estimated constants discussed earlier and finally in line 10, we create a master object which will enable us to do the dynamic graphics.

Table 2 and Figure 4 are a "snapshot" of the program output, for the particular prior π specified in the last row in Table 2. The Bayes estimates (means of the posteriors) are shown in Table 2. The estimates in all rows except the first and last are obtained by reweighting the output of all four Markov chains. The estimates in the last row were obtained by running one Markov chain corresponding to the prior π. This was done as a consistency check. We see that the comparison is excellent.

4.1.2 Pump Data

George, Makov, and Smith (1993) provide a complete hierarchical Bayesian analysis of the pump failure data analyzed by Gaver and O'Muircheartaigh (1987). The data, shown in Table 3, give the number of failures Y_i and the length of operation time t_i (in thousands of hours) for 10 pumps in a nuclear power plant.

Observation	1	2	3	4	5	6	7	8	9	10
t_i	94.3	15.7	62.9	126	5.24	31.4	1.05	1.05	2.1	10.5
y_i	5	1	5	14	3	19	1	1	4	22

Table 3: Pump failure data of Gaver and O'Muircheartaigh (1987).

George et. al. use a Gamma-Poisson hierarchical model as follows. Given θ_i, the number of failures Y_i has the Poisson$(\theta_i t_i)$ distribution, and is independent of Y_j, $j \neq i$. Given α and β, the failure rates θ_i are iid from the Gamma(α, β) distribution with density proportional to $\theta_i^{\alpha-1} \exp(-\beta\theta_i)$. They used four priors, $\pi_1, \pi_2, \pi_3, \pi_4$, on the pair (α, β), with densities

1. $\pi_1'(\alpha, \beta) \propto \exp(-\alpha) \times \beta^{-.9} \exp(-\beta)$,
2. $\pi_2'(\alpha, \beta) \propto \exp(-\alpha/100) \times \beta^{-.9} \exp(-\beta)$,
3. $\pi_3'(\alpha, \beta) \propto \exp(-\alpha) \times \beta^{-.9} \exp(-\beta/100)$, and
4. $\pi_4'(\alpha, \beta) \propto \exp(-\alpha/100) \times \beta^{-.9} \exp(-\beta/100)$.

Although this is not a regression model, we can still apply our methodology. We considered three priors with the following densities:

1. $\nu_1'(\alpha, \beta) \propto \exp(-\alpha/50) \times \beta^{-.5} \exp(-.5\beta)$,
2. $\nu_2'(\alpha, \beta) \propto \exp(-\alpha/90) \times \beta^{-.75} \exp(-1.5\beta)$, and
3. $\nu_3'(\alpha, \beta) \propto \exp(-\alpha/50) \times \beta^{-.2} \exp(-0.08\beta)$.

Note that these priors are very different from those used by George et. al. The Gibbs sampler was used to produce a sample of 500 independent points from the posterior corresponding to each prior. The points were stored in files called `pump1.out`, `pump2.out`, and `pump3.out`. We estimated the values $d(h_2)$ and $d(h_3)$ in equation (3.3) to be 0.312 and 10.812 ($d(h_1) = 1$ as usual). The following piece of code was used to invoke our routines—it is very similar to the code used earlier.

```
 1. (defvar *reweighting-code* '("pump_weights.o" "pump_hmix.o"))
 2. (require "master")
 3. (defvar *param-labels*
     '("Theta(1)" "Theta(2)" "Theta(3)" "Theta(4)" "Theta(5)" "Theta(6)"
       "Theta(7)" "Theta(8)" "Theta(9)" "Theta(10)" "Alpha" "Beta"))
 4. (def d-files '("pump1.out" "pump2.out" "pump3.out"))
 5. (def hlist '(((50.0) (0.5 0.5)) ((90.0) (0.25 1.5))
                 ((0.5) (0.8 0.08))))
 6. (def rlist '((1 100) (0 1) (0 1)))
 7. (def hnames '("Alpha-Mean" "Beta-Shape" "Beta-Scale"))
 8. (def wlist '(t nil nil nil nil nil nil nil nil nil t t))
 9. (def init '(1.0 0.1 1.0))
10. (def c '(1.0 0.312 10.812))
11. (setf posterior (send master-proto :new :data d-files :constants c
                          :p 12 :hyper-param-names hnames
                          :identifier "Pump Data" :hyper-vals hlist
                          :initial-hyper-vals init :hyper-range-list rlist
                          :create-window-list wlist))
```

In implementing our scheme for sensitivity analysis, the only real difference between this hierarchical model and the regression model given by (1.1) and (1.2) with Normal priors lies in the formulas for calculating the weights. Line 1 reflects this fact, and informs the program to use the files `pump_weights.o` and `pump_hmix.o` (produced by C programs) which calculate the importance weights. If one omits this line, the formulas for the regression model would be used by default, which would give inappropriate results.

Again, we chose to concentrate on a subset of the parameters, namely θ_1, α, and β (lines 3 and 8). The code allows `Alpha-mean` to be varied between 1 and 100, and `Beta-scale` and `Beta-Shape` between 0 and 1 (line 6). We also arranged for the initial values of the hyperparameters to reflect the prior π_1 of George et. al. (line 9).

The initial output is shown in Figure 5(a). Figure 5(b) shows the windows when the hyperparameters are changed to match the prior π_4. Table 4 compares the estimates of the parameters obtained by reweighting to those obtained by George et. al. Even with only 500 samples for each of 3 chains, the estimates are quite close to the values obtained by a direct run of one Gibbs sampler.

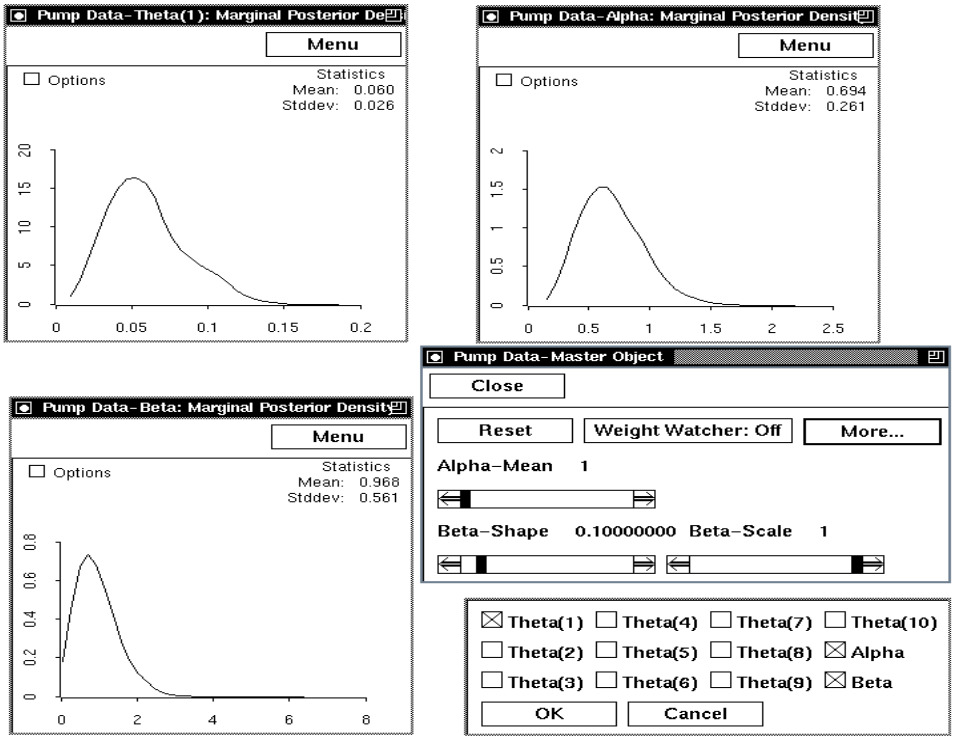
Prior	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7	θ_8	θ_9	θ_{10}	α	β
π_1	.060	.102	.089	.116	.602	.609	.891	.894	1.588	1.994	.7	.9
	.060	.101	.089	.116	.588	.609	.840	.878	1.576	1.989	.7	1.0
π_2	.061	.106	.090	.117	.603	.609	.884	.886	1.560	1.981	.8	1.0
	.061	.106	.090	.116	.589	.607	.839	.868	1.557	1.977	.8	1.1
π_3	.061	.107	.091	.117	.584	.605	.806	.808	1.453	1.936	.8	1.4
	.060	.107	.090	.116	.573	.598	.754	.778	1.427	1.915	.8	1.5
π_4	.062	.113	.093	.118	.585	.604	.791	.789	1.398	1.905	1.0	1.6
	.061	.112	.091	.116	.574	.595	.747	.763	1.389	1.891	.9	1.7

Table 4: Comparison of estimates obtained by George et. al. (1993) to those obtained by reweighting. The estimates obtained by reweighting are shown in every second row.

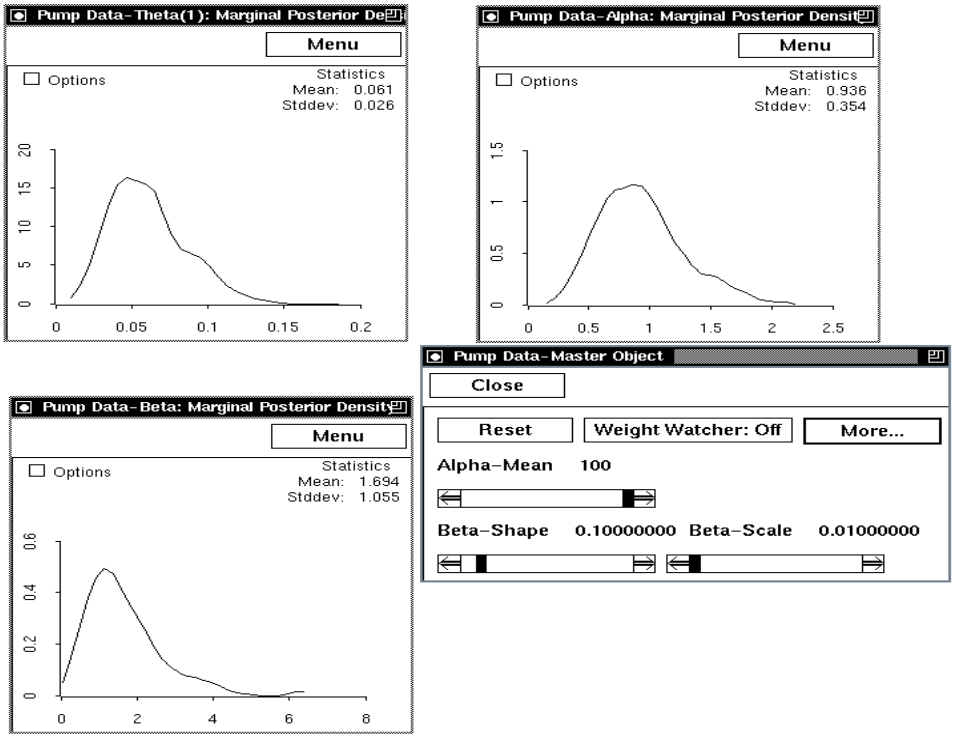
4.2 The Implementation

We now turn to details regarding the implementation of the programs. Our programs use a *master-slave* design, where there is one slave for every parameter in the model and a single master controls all the slaves. To implement the functionality necessary in the master and the slave, we created several object prototypes. It is helpful to think of an object prototype as an entity that embodies both the data structures *and* the associated algorithms. The `LISP-STAT` system provides some frequently needed object prototypes; `scatterplot-proto` and `dialog-proto` are two such. The object prototypes we created were built on top of these. There are two main object prototypes in our programs: `master-proto`, corresponding to the master, and `slave-proto`, corresponding to the slave. Figure 6 shows the pedigree of the prototypes. The arrows in the figure point to descendants. Thus, `master-proto` is a descendant of `dialog-proto`, and `slave-proto` is a descendant of `kde-proto`, which itself is a descendant of `scatterplot-proto`. At times, we use the phrase "object *A inherits from* object *B*" to mean that *A* is a *descendant of B*. It must be noted that when *A* inherits from *B*, it inherits both *B*'s data strutures and algorithms.

The `kde-proto` prototype, which we created, provides the data structures and algorithms for doing basic weighted kernel density estimation. Since `slave-proto` inherits from `kde-proto`, every slave "knows" how to calculate kernel density estimates. Also,



(a) Prior π_1.



(b) Prior π_4.

Figure 5: The set of windows for two priors, (a) Prior π_1, and (b) Prior π_4.

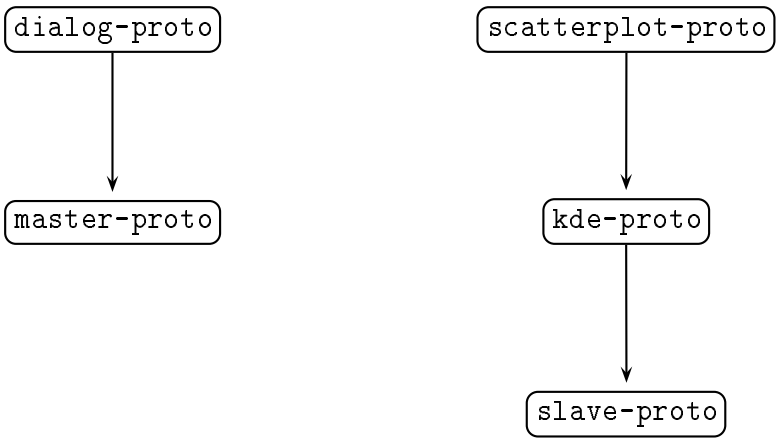


Figure 6: The pedigree of objects. Arrows point to descendants.

since `kde-proto` inherits from `scatterplot-proto` which "knows" how to draw graphs, etc., so does `slave-proto`. Since every parameter in our model can be identified with an instance of `slave-proto`, we have, in a programming sense, "intelligent" model parameters. For example, one could ask any parameter what its estimated mean and standard deviation are, and the parameter would "answer" with the values. The `master-proto` prototype is responsible for coordinating the behavior of the slaves by providing the required synchronization.

When one defines a new prototype in `LISP-STAT`, one may provide "slots" in the prototype for storing various quantities that are associated with any instance of the prototype. In addition, one can also embody the prototype with *methods*, i.e., algorithms for doing various tasks. It is customary to also provide *accessor* methods that access the values stored in the slots. We describe some important methods of `master-proto` that are often needed in other situations.

`:data` Returns the data from all the Markov chains as one long list.

`:slaves` Returns a list of instances of the slave objects, one for each β_j, $j = 1, 2, \ldots, p$.

`:hyper-vals` Returns a list of the current values of the hyperparameters. When the user changes any hyperparameter with a slider, this list will reflect the change.

`:org-hyper-vals` Returns a list of lists of hyperparameter values at which the Markov chains were run.

`:weights` Returns the current importance weights of the data points used in calculating posterior quantities. As the hyperparameters are changed these weights are automatically recalculated.

`:synchronize` Synchronizes the slaves so that all the windows are updated.

In addition, the method `:toggle-weight-watcher` toggles the weight watcher on and off. The default weight watcher is a boxplot of the importance weights. However, by following

the comments in our code, one could easily compute a statistic such as $\sum_{i=1}^{kG}(w_i-(kG)^{-1})^2$ and display it dynamically to watch the weights.

Every slave object contains information particular to a specific parameter. The user almost never manipulates the `slave-proto`; the master automatically creates and kills instances of the slave prototype as the situation demands. However, some methods of the slave prototype also come in handy when adapting our routines for other situations. A brief discussion of the important method follows.

`:master` Returns the master of this object. This allows for a slave to communicate with its master.

`:weights` Returns a list of current importance weights. This is nothing but an accessor method for the master's slot.

`:data` Returns the data specific to this slave from all Markov chains as one long list.

`:stats-messages` Returns a list of method names, each of which can be sent to the slave for getting the respective reweighted Monte Carlo estimate of the statistic.

There is a provision in the code that allows for displaying the value of any calculated statistic in every window. For example, suppose one wishes to display, in addition to the mean and standard deviation, Monte Carlo estimates of the posterior expectation of β_i^2. This is easily done in three steps by editing the file `slave-stats.lsp`, as follows.

1. We write a method, say `:mean-beta-sq`, for the object `slave-proto` that returns this quantity.

```
(defmeth slave-proto :mean-beta-sq ()
   (let ((data (send self :data))
         (wts (send self :weights)))
      (sum (* data data wts))))
```

2. We change the list `*stats-to-show*` in the file `slave-opts.lsp` to include the string `mean-beta-sq`. For example, if `*stats-to-show*` was previously `'("mean" "stddev")`, we change it to `'("mean" "stddev" "mean-beta-sq")`.

3. We change the `*stats-print-formats*` variable to reflect how the quantity should be printed. For example, if `*stats-print-formats*` was previously `'((6 3) (6 3))`, we change it to `'((6 3) (6 3) (6 3))` to indicate that our new statistic should be printed in a field of width 6 with 3 decimal places.

The new statistic will automatically appear on every parameter window. It is also possible to do some fine tuning for printing the statistic using the `Options` overlay on the windows.

5 Concluding Remarks

Although we concentrate on the Poisson regression model (1.1) and (1.2) with Normal priors, nothing in our programs assumes a particular model or a particular family of priors. When dealing with a new problem, it will be necessary only to derive the formulas for the importance weights and to code them as C programs. In most cases, the effort in doing this is minimal. It also follows that our programs can be used to perform sensitivity analyses where instead of changing the priors, one changes the likelihood, for example by perturbing the data points. Of course it must also be noted that before running our programs, the output of the Markov chains, and estimates of the constants $d(h_1), \ldots, d(h_k)$ in equation (3.3) must be available. The estimation of the constants can be done by any method the user chooses. In particular, one can use our approach (contained in the files `ship_constants.lsp` and `pump_constants.lsp`) as a template.

We conclude with some indications as to the speed of the dynamic graphics. Our experiments were conducted on a workstation doing about 8 million floating point operations per second and having 48MB of RAM. For the ship data, which involved 9 parameters and samples of 500 from each of 4 Gibbs samplers, the machine took 2 seconds to update 3 windows when any hyperparameter was changed. With samples of size 200, the updating was done in less than a second.

`LISP-STAT` is freely available by from `umnstat.stat.umn.edu`. Our software is available from `statlib`. Users of the popular `Mosaic` program can use the Universal Resource Locator (URL) `http://lib.stat.cmu.edu/xlispstat/bpois.shar`. To obtain our software by `ftp` type

```
% ftp lib.stat.cmu.edu
```

Use `statlib` for the username and type in your e-mail address for the password. Then, in `ftp`, type:

```
ftp> cd xlispstat
ftp> get bpois.shar
ftp> quit
```

Put the file `bpois.shar` in a new directory and in that directory type

```
% sh bpois.shar
```

Now follow the detailed instructions in the file named `README`.

We have tested the programs only on Unix workstations, but we expect them to require only minimal changes to work on other platforms where dynamic loading is enabled.

Appendix

A.1 Convergence of the Algorithm

Here we show that our Gibbs sampler converges to the posterior distribution (2.2) from any starting point $\boldsymbol{\beta}^{(0)}$. To be more specific, let $P_{\boldsymbol{b}}$ denote the distribution of the chain $\boldsymbol{\beta}^{(1)}, \boldsymbol{\beta}^{(2)}, \ldots$ given that $\boldsymbol{\beta}^{(0)} = \boldsymbol{b}$. We will show that as $G \to \infty$

$$\sup_A |P_{\boldsymbol{b}}(\boldsymbol{\beta}^{(G)} \in A) - \nu_{\boldsymbol{Y}}(A)| \to 0 \quad \text{for all } \boldsymbol{b} \tag{A.1}$$

and for every $\nu_{\boldsymbol{Y}}$-integrable function t

$$P_{\boldsymbol{b}}\left\{ \frac{1}{G}\sum_{g=1}^{G} t(\boldsymbol{\beta}^{(g)}) \to \int t(\boldsymbol{\beta})\, \nu_{\boldsymbol{Y}}(d\boldsymbol{\beta}) \right\} = 1 \quad \text{for all } \boldsymbol{b}. \tag{A.2}$$

Conditions (A.1) and (A.2) will follow from Theorem 6 of Athreya, Doss, and Sethuraman (1992) if we can check the hypotheses of that theorem, which in our setup amount to showing that there exists sets A_j, $j = 1, \ldots, p$ and a $\delta > 0$ such that for each $s = 1, \ldots, p$

$$\nu'_{\boldsymbol{Y},s}(\beta_1, \beta_2, \ldots, \beta_p) > 0 \quad \text{if } \beta_1 \in A_1, \ldots, \beta_s \in A_s,\ \beta_{s+1}, \ldots, \beta_p \text{ are arbitrary}$$

and

$$\nu'_{\boldsymbol{Y},s}(\beta_1, \beta_2, \ldots, \beta_p) > \delta \quad \text{for all } \beta_1 \in A_1, \ldots, \beta_p \in A_p.$$

Since by (2.3), $\nu'_{\boldsymbol{Y},s}$ is continuous in $\beta_1, \beta_2, \ldots, \beta_p$ and always positive, it is trivial to produce such sets and such a δ (the sets $A_j = [-1, 1]$ would do). (Actually, Theorem 6 of Athreya, Doss, and Sethuraman (1992) implies only that there exists a set D_0 of $\nu_{\boldsymbol{Y}}$-probability one, such that (A.1) and (A.2) hold for $\boldsymbol{b} \in D_0$. However, the exceptional null set is easily handled since for every $\boldsymbol{b}$, $P_{\boldsymbol{b}}(\boldsymbol{\beta}^{(1)} \in D_0) = 1$, so (A.1) and (A.2) hold as stated.)

It seems likely that the chain is geometrically ergodic, but we have not been able to show that. The problem is that to establish a "drift condition" (see Proposition 1 of Tierney (1994) or Theorem 2.1 of Chan (1993)) we need to calculate certain expectations with respect to the conditional densities (2.3), but this is difficult because the normalizing constant in (2.3) (a complicated function of $\{\beta_j;\ j \neq s\}$) is not known.

A.2 Asymptotic Normality of the Estimate (3.9)

We now discuss the problem of identifying the range of hyperparameter values h for which the estimate $\hat{I}_h(t)$ in (3.9) is asymptotically normal. Consider first the case $k = 1$. Thus, assume we have $\boldsymbol{\beta}^{(1)}, \ldots, \boldsymbol{\beta}^{(G)} \overset{\text{iid}}{\sim} \nu_{h_1,\boldsymbol{Y}}$.

Let $v_h(\boldsymbol{u}) = \nu'_h(\boldsymbol{u})/\nu'_{h_1}(\boldsymbol{u})$. Consider the sequence

$$G^{1/2} \begin{pmatrix} \dfrac{1}{G}\displaystyle\sum_{g=1}^{G} t(\boldsymbol{\beta}^{(g)}) v_h(\boldsymbol{\beta}^{(g)}) - \int t(\boldsymbol{\beta}) v_h(\boldsymbol{\beta})\, \nu_{h_1,\boldsymbol{Y}}(d\boldsymbol{\beta}) \\ \dfrac{1}{G}\displaystyle\sum_{g=1}^{G} v_h(\boldsymbol{\beta}^{(g)}) - \int v_h(\boldsymbol{\beta})\, \nu_{h_1,\boldsymbol{Y}}(d\boldsymbol{\beta}). \end{pmatrix} \tag{A.3}$$

Asymptotic joint normality of this sequence implies asymptotic normality of $G^{1/2}(\hat{I}_h(t) - I_h(t))$ by the delta method. By the Cramér-Wold device, we see that the sequence (A.3) has an asymptotic bivariate normal distribution if

$$\int \Big(t(\boldsymbol{\beta})v_h(\boldsymbol{\beta})\Big)^2 \, \nu_{h_1,\boldsymbol{Y}}(d\boldsymbol{\beta}) < \infty \tag{A.4}$$

and

$$\int (v_h(\boldsymbol{\beta}))^2 \, \nu_{h_1,\boldsymbol{Y}}(d\boldsymbol{\beta}) < \infty. \tag{A.5}$$

Let $h_1 = (a_1^{(1)}, b_1^{(1)}, \ldots, a_p^{(1)}, b_p^{(1)})$ and $h = (a_1, b_1, \ldots, a_p, b_p)$. Using (2.1) we see that the integral in (A.5) is proportional to

$$\begin{aligned} &\int \frac{\prod_{j=1}^p \exp\Big(-\frac{(\beta_j - a_j)^2}{b_j}\Big)}{\prod_{j=1}^p \exp\Big(-\frac{(\beta_j - a_j^{(1)})^2}{2b_j^{(1)}}\Big)} \exp\Big(-\sum_{i=1}^n \exp\Big(\sum_{j=1}^p x_{ij}\beta_j\Big) + \sum_{j=1}^p \beta_j A_j\Big)\, d\boldsymbol{\beta} \\ &\qquad \propto \int \prod_{j=1}^p \exp\Big(-\frac{\beta_j^2}{b_j} + \frac{\beta_j^2}{2b_j^{(1)}}\Big) \exp\Big(-\sum_{i=1}^n \exp\Big(\sum_{j=1}^p x_{ij}\beta_j\Big) + \sum_{j=1}^p \beta_j B_j\Big) d\boldsymbol{\beta} \end{aligned} \tag{A.6}$$

where $A_j = \sum_{i=1}^n x_{ij} Y_i$ and $B_j = A_j + 2a_j - a_j^{(1)}$. Now from the form of the last integral in (A.6) it is easy to see that the integral in (A.5) is finite if

$$b_j < 2b_j^{(1)} \quad \text{for } j = 1, \ldots, p. \tag{A.7}$$

(When $b_j > 2b_j^{(1)}$ for at least one j the integral diverges. For h on the "boundary", we need to have $B_j < 0$ for all j such that $b_j = 2b_j^{(1)}$.)

If (A.7) holds and there exists a constant K and a δ with $\delta < \min_j\{1/b_j - 1/2b_j^{(1)}\}$ such that

$$|t(\boldsymbol{\beta})| < K \exp(\delta \|\boldsymbol{\beta}\|^2) \quad \text{for all } \boldsymbol{\beta}, \tag{A.8}$$

then (A.4) holds as well. Condition (A.8) is a very mild restriction on the growth rate of $t(\cdot)$.

Consider now the case $k > 1$, and let $h_\ell = (a_1^{(\ell)}, b_1^{(\ell)}, \ldots, a_p^{(\ell)}, b_p^{(\ell)})$ for $\ell = 1, \ldots, k$. For each $j = 1, \ldots, p$, let jmax be the index ℓ of the largest value of $b_j^{(1)}, \ldots, b_j^{(k)}$, (i.e. $b_j^{(\text{jmax})} = \max_\ell b_j^{(\ell)}$). We deal first with the estimate $\tilde{I}_h(t)$ defined in (3.7). Recall that $\tilde{v}_h$ is defined in (3.5). Note that for some constant C,

$$(\tilde{v}_h(\boldsymbol{\beta}))^2 \le C \frac{\prod_{j=1}^p \exp\Big(-\frac{(\beta_j - a_j)^2}{b_j}\Big)}{\prod_{j=1}^p \exp\Big(-\frac{(\beta_j - a_j^{(\text{jmax})})^2}{b_j^{(\text{jmax})}}\Big)}.$$

For each ℓ, the expectation $\int (\tilde{v}_h(\boldsymbol{\beta}))^2 \, \nu_{h_\ell,\boldsymbol{Y}} \, d\boldsymbol{\beta}$ is handled by using the fact that

$$\exp\Big(-\frac{\beta_j^2}{b_j} + \frac{\beta_j^2}{b_j^{(\text{jmax})}} - \frac{\beta_j^2}{2b_j^{(\ell)}}\Big) \le \exp\Big(\beta_j^2\Big\{\frac{-1}{b_j} + \frac{1}{2b_j^{(\text{jmax})}}\Big\}\Big).$$

Thus, we see that $\tilde{I}_h(t)$ is asymptotically normal if

$$b_j < \max_\ell 2b_j^{(\ell)} \quad \text{for } j = 1, \ldots, p$$

and t satisfies

$$|t(\boldsymbol{\beta})| < K \exp(\delta \|\boldsymbol{\beta}\|^2) \quad \text{for some } \delta < \min_j \{1/b_j - 1/(2 \max_\ell b_j^{(\ell)})\}.$$

Finally, asymptotic normality of $\tilde{I}_h(t)$ implies that of $\hat{I}_h(t)$ by Theorem 4 of Geyer (1994) (and the asymptotic variances of $\hat{I}_h(t)$ and $\tilde{I}_h(t)$ are equal).

References

Athreya, K. B., Doss, H. and Sethuraman, J. (1992). On the convergence of the Markov chain simulation method. Preprint.

Chan, K. S. (1993). Asymptotic behavior of the Gibbs sampler. *J. Amer. Statist. Assoc.* **88** 320–326.

Doss, H. (1994). Bayesian nonparametric estimation for incomplete data via successive substitution sampling. *Ann. Statist.* (to appear).

Gaver, D. and O'Muircheartaigh, I. (1987). Robust empirical Bayes analyses of event rates. *Technometrics* **29** 1–15.

George, E. I., Makov, I. E. and Smith, A. F. M. (1993). Conjugate likelihood distributions. *Scandinavian J. Stat.* **20** 147–156.

Geyer, C. J. (1992). Practical Markov Chain Monte Carlo (with discussion). *Statist. Sci.* **7** 473–511.

Geyer, C. J. (1994). Reweighting Monte Carlo mixtures. *J. Amer. Statist. Assoc.* (to appear).

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.

McCullagh, P. and Nelder, J. A. (1989). *Generalized Linear Models.* Second Edition, Chapman and Hall, London.

Tierney, L. (1990). *Lisp-Stat.* Wiley, New York.

Tierney, L. (1994). Markov chains for exploring posterior distributions (with discussion). *Ann. Statist.* (to appear).