Sampling) project is a long-running project to provide a user-friendly language and environment for Bayesian inference. The first article, by Andrew Thomas and colleagues, describes the **BRugs** package which provides an R interface to the OpenBUGS engine. The second article by Andrew Thomas describes the BUGS language itself and the design philosophy behind it. Somewhat unusually for an article in R News, this article does not describe any R software, but it is included to highlight some of the differences in the way statistical models are represented in R and OpenBUGS.

The issue ends with an article by Jouni Kerman and Andrew Gelman, who give a personal perspective on what the next generation of Bayesian software may look like, and preview some of their own work in this area, notably the **rv** package for representing simulation-based random variables, and the forthcoming "Universal Markov Chain Sampler" package, **Umacs**.

Bibliography

A. E. Raftery, I. S. Painter, and C. T. Volinsky. BMA: An R package for bayesian model averaging. *R News*, 5(2):2–8, November 2005. URL http://CRAN.R-project.org/doc/Rnews/. 1

J. Yan. Fusing R and BUGS through Wine. *R News*, 4(2):19–21, September 2004. URL http://CRAN.R-project.org/doc/Rnews/. 1

Martyn Plummer
International Agency for Research on Cancer
Lyon, France
plummer@iarc.fr

Applied Bayesian Inference in R using MCMCpack

by Andrew D. Martin and Kevin M. Quinn

Introduction

Over the past 15 years or so, data analysts have become increasingly aware of the possibilities afforded by Markov chain Monte Carlo (MCMC) methods. This is particularly the case for researchers interested in performing Bayesian inference. Here, MCMC methods provide a fairly straightforward way for one to take a random sample approximately from a posterior distribution. Such samples can be used to summarize any aspect of the posterior distribution of a statistical model. While MCMC methods are extremely powerful and have a wide range of applicability, they are not as widely used as one might guess. At least part of the reason for this is the gap between the type of software that many applied users would like to have for fitting models via MCMC and the software that is currently available. **MCMCpack** (Martin and Quinn, 2005) is an R package designed to help bridge this gap.

Until the release of **MCMCpack**, the two main options for researchers who wished to fit a model via MCMC were to: a) write their own code in R, C, FORTRAN, etc., or b) write their own code (possibly relying heavily on the available example programs) using the BUGS language[1] in one of its various implementations (Spiegelhalter et al., 2004; Thomas, 2004; Plummer, 2005). While both of these options offer a great deal of flexibility, they also require non-trivial programming skills in the case of a) or the willingness to learn a new language and to develop some modest programming skills in the case of b). These costs are greater than many applied data analysts are willing to bear. **MCMCpack** is geared primarily towards these users.

The design philosophy of **MCMCpack** is quite different from that of the BUGS language. The most important design goal has been the implementation of MCMC algorithms that are model-specific. This comports with the manner in which people oftentimes think about finding software to fit a particular class of models rather than thinking about writing code from the ground up. The major advantage of such an approach is that the sampling algorithms, being hand-crafted to particular classes of models, can be made dramatically more efficient than black box approaches such as those found in the BUGS language, while remaining robust to poorly conditioned or unusual data. All the **MCMCpack** estimation routines are coded in C++ using the Scythe Statistical Library (Martin et al., 2005). We also think it is easier to call a single R function to fit a model than to code a model in the BUGS language. It should also be noted that **MCMCpack** is aimed primarily at so-

[1] The BUGS language is a general purpose language for simulation from posterior distributions of statistical models. BUGS exploits conditional independence relations implied by a particular graphical model in order to automatically determine an MCMC algorithm to do the required simulation. In order to fit a model, the user must specify a graphical model using either the BUGS language or (in the case of WinBUGS) a graphical user interface.

cial scientists. While some models (linear regression, logistic regression, Poisson regression) will be of interest to nearly all researchers, others (various item response models and factor analysis models) are especially useful for social scientists.

While we think **MCMCpack** has definite advantages over BUGS for many users, we emphasize that we view BUGS and **MCMCpack** as complimentary tools for the applied researcher. In particular, the greater flexibility of the BUGS language is perfect for users who need to build and fit custom probability models.

Currently **MCMCpack** contains code to fit the following models: linear regression (with Gaussian errors), a hierarchical longitudinal model with Gaussian errors, a probit model, a logistic regression model, a one-dimensional item response theory model, a K-dimensional item response theory model, a normal theory factor analysis model, a mixed response factor analysis model, an ordinal factor analysis model, a Poisson regression, a tobit regression, a multinomial logit model, a dynamic ecological inference model, a hierarchial ecological inference model, and an ordered probit model. The package also contains densities and random number generators for commonly used distributions that are not part of the standard R distribution, a general purpose Metropolis sampling algorithm, and some utilities for visualization and data manipulation. The package provides modular random number generators, including the L'Ecuyer generator which produces independent substreams, thus making (embarrassingly) parallel simulation using **MCMCpack** possible.

In the remainder of this article, we illustrate the user interface and functionality of **MCMCpack** with three examples.

User interface

The model fitting functions in **MCMCpack** have been written to be as similar as possible to the corresponding R functions for estimation of the models in question. This largely eliminates the need to learn a specialized model syntax for anyone who is even a novice user of R. For example, to fit a linear regression model with an improper uniform prior on the coefficient vector, an inverse gamma prior with shape and scale both equal to 0.0005 for the error variance, and the default settings for the parameters governing the MCMC algorithm, one would issue a function call nearly identical to the `lm()` command.

As an example, consider the `swiss` data that contains fertility and socioeconomic indicators from the 47 French-speaking provinces of Switzerland in 1888. To fit a Bayesian linear regression of fertility on a number of predictors in the dataset we use the Gibbs sampling algorithm to obtain a sample approximately from the appropriate posterior distribution. To do this with **MCMCpack** and to then summarize the results we issue the following command:

```
> data(swiss)
> posterior1 <- MCMCregress(Fertility ~
+                  Agriculture + Examination +
+                  Education + Catholic +
+                  Infant.Mortality,
+                  data=swiss)
> summary(posterior1)
```

The `MCMCregress()` function has a syntax similar to the `lm()` command. All model functions in **MCMCpack** return `mcmc` objects as defined by the **coda** package (Plummer et al., 2005). **MCMCpack** relies on coda to perform posterior summarization and convergence diagnostics on the simulated values. The summary method for `mcmc` objects prints various quantities of interest to the screen, including the posterior mean, standard deviation, and quantiles. The **coda** package provides a number of other facilities, including a plot method that produces marginal posterior kernel density plots and traceplots, and a suite of convergence diagnostics. See Figure 1 for the posterior summary for the Swiss fertility regression.

We note that diagnosing convergence is *critical* for any application employing MCMC methods. While we have ignored such diagnostics here for reasons of space, please note that simulation run lengths used in all of the examples have been chosen so that inferences are accurate.

Latent variable models in MCMCpack

One very active area of research in the field of political methodology involves modeling voting in committees using the spatial voting model. This explanation of voting asserts that actors have a preferred policy position (usually called an ideal point) in a K-dimensional issue space. For example, many European parliamentary systems are characterized by a two-dimensional model, with one dimension representing traditional left-right economic considerations, and the other dimension representing the issue of European integration. Voters cast votes on binary choices—one representing the status quo, the other an alternative policy. The goal of these models is to recover the ideal points of the actors, and a set of item-specific parameters that are functions of the alternative and status quo positions.

Under certain sets of assumptions, these empirical spatial voting models are the same as item response theory (IRT) models used in educational testing and psychometrics. The Bayesian approach to fitting these latent variable models provides many advantages over the frequentist approach. Model estimation is reasonably easy using data augmentation, identification of the model is straightforward,

```
> summary(posterior1)

Iterations = 1001:11000
Thinning interval = 1
Number of chains = 1
Sample size per chain = 10000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

                    Mean       SD  Naive SE Time-series SE
(Intercept)      67.0208 11.08133 0.1108133      0.1103841
Agriculture      -0.1724  0.07306 0.0007306      0.0007149
Examination      -0.2586  0.26057 0.0026057      0.0024095
Education        -0.8721  0.18921 0.0018921      0.0017349
Catholic          0.1040  0.03602 0.0003602      0.0002965
Infant.Mortality  1.0737  0.39580 0.0039580      0.0042042
sigma2           54.0498 12.68601 0.1268601      0.1566833

2. Quantiles for each variable:

                     2.5%      25%     50%      75%    97.5%
(Intercept)      45.53200 59.56526 67.0600 74.31604 88.87071
Agriculture      -0.31792 -0.22116 -0.1715 -0.12363 -0.02705
Examination      -0.76589 -0.43056 -0.2579 -0.08616  0.24923
Education        -1.24277 -0.99828 -0.8709 -0.74544 -0.49851
Catholic          0.03154  0.08008  0.1037  0.12763  0.17482
Infant.Mortality  0.28590  0.81671  1.0725  1.33767  1.85495
sigma2           34.57714 45.06332 52.3507 61.03743 83.85127
```

Figure 1: Posterior summary from coda for the Swiss fertility regression fit using `MCMCregress()`

auxiliary information can be included in the analysis through the use of priors, and one can discuss quantities of interest on the scale of probability (Clinton et al., 2004; Martin and Quinn, 2002). **MCMCpack** contains a number of latent variable models, including one-dimensional and K-dimensional IRT models and factor analysis models for continuous, ordinal, and mixed data.

To illustrate the one-dimensional IRT model in **MCMCpack**, we will use some data from the U.S. Supreme Court. **MCMCpack** contains a dataset (`SupremeCourt`) of the votes cast by the nine sitting justices on the 43 non-unanimous cases decided during the October 2000 term. The data are just a (43×9) matrix of zeros, ones, and missing values. To identify the polarity of the model we constrain the ideal points of two justices in our one-dimensional latent space. We constrain Justice Stevens (a well-known liberal) to have a negative ideal point, and Justice Scalia (perhaps the most conservative members of the Court) to have a positive ideal point. (The one-dimensional IRT model in **MCMCpack** is identified through constraints on the ideal points / subject abilities while the K-dimensional model is identified through constraints in the item parameters). We use the default priors on the item and subject parameters. To fit the model, we issue the following command:

```
> data(SupremeCourt)
> posterior2 <- MCMCirt1d(t(SupremeCourt),
+   theta.constraints=list(Stevens="-",
+   Scalia="+"), burnin=5000, mcmc=100000,
+   thin=10, verbose=500)
```

By default, `MCMCirt1d` only retains the ideal points in the posterior sample. One can optionally store the item parameters. We illustrate our results from this sample analysis in Figure 2. This figure shows the standard normal prior density on the ideal points, and the marginal posterior densities of the ideal points. Justice Stevens is the leftmost Justice, followed by Breyer, Ginsburg, and Souter, who are essentially indistinguishable. Justice O'Connor is the pivotal median justice, closely followed by Justice Kennedy. Chief Justice Rehnquist is next, following by Justices Thomas and Scalia. The posterior sample can be used to answer any number of important substantive questions (see, for example, Clinton et al., 2004).

Generic metropolis sampling

MCMCpack model functions allow users to choose prior distributions by picking the parameters of a particular parametric family that is specific to each

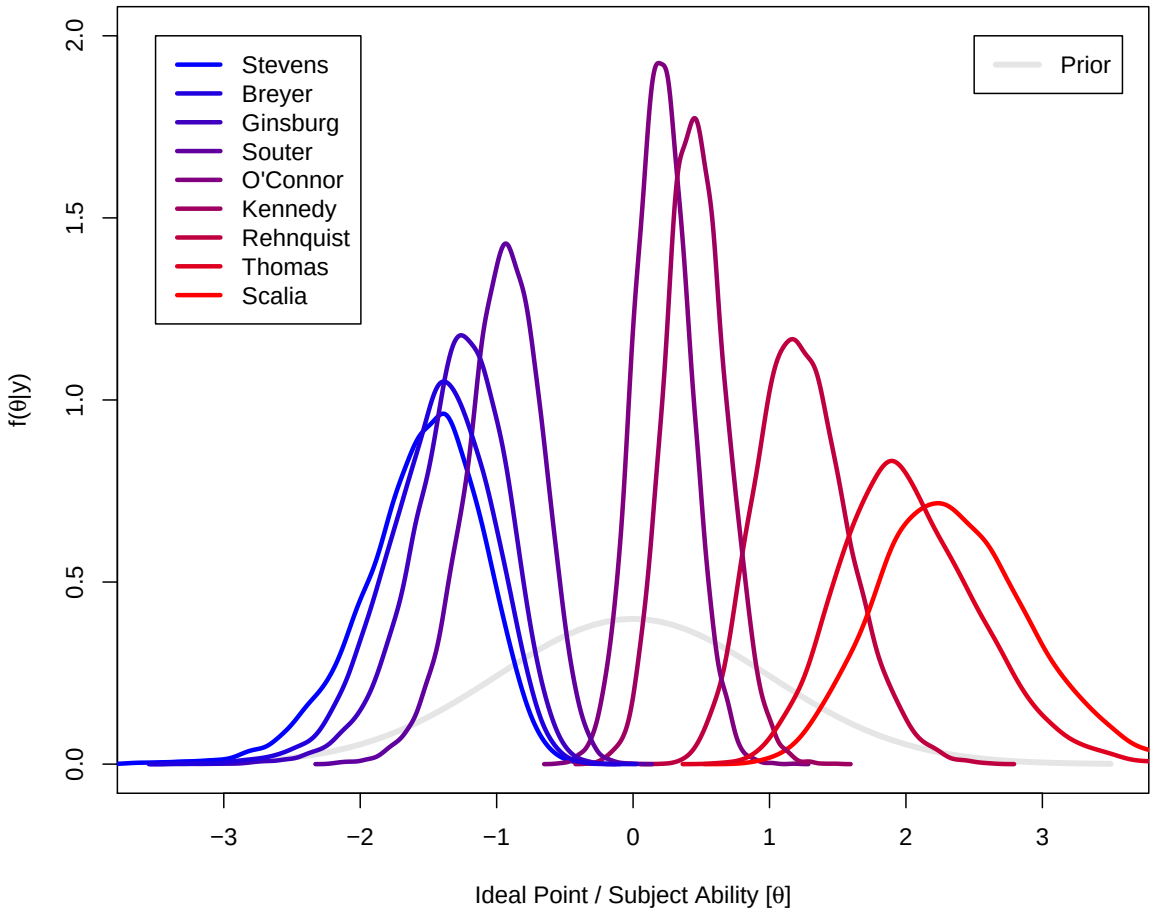

Figure 2: Posterior densities of ideal points for the U.S. Supreme Court justices, 2000 term, as estimated by `MCMCirt1d()`

model. This approach is reasonable for many applications, but at times users would like to use more flexible prior specifications. Similarly, there are numerous models that are not currently implemented *directly* in **MCMCpack**, but whose posterior densities are easy to write down up to a constant of proportionality. One area of **MCMCpack** that is currently under development is a set of functions to perform generic sampling from a user-supplied (log)-posterior distribution. While the user-supplied density is written as an R function, the simulation itself is performed in compiled C++ code, so it is much more efficient than doing the simulation in R itself.

While **MCMCpack** is not designed to be a general purpose sampling engine in the manner of BUGS, the ability to fit a models with relatively small numbers of parameters by specifying a (log)-posterior is very attractive to many social scientists who are accustomed to calculating maximum likelihood estimates using numerical optimization routines. For many of these researchers, writing an R function that evaluates a (log)-posterior is much more intuitive than specifying the equivalent graphical model in BUGS.

As an example, suppose one would like to fit a logistic regression model to the `birthwt` data from the **MASS** package (Venables and Ripley, 2002) with a complicated prior distribution. We assume our dichotomous dependent variable (the low birthweight indicator) $y_i \sim Bernouli(\pi_i)$ for observations $i = 1, \ldots, n$ with inverse-link function:

$$\pi_i = \frac{1}{1 + \exp(-x_i'\beta)}$$

The parameter vector β is of dimensionality $(p \times 1)$, and x_i is a column vector of observed covariates.

The data encodes risk factors associated with low birth weight. We prepare the data for analysis as follows:

```
> attach(birthwt)
> race <- factor(race, labels = c("white",
+  "black", "other"))
> ptd <- factor(ptl > 0)
> ftv <- factor(ftv)
> levels(ftv)[-(1:2)] <- "2+"
> bwt <- data.frame(low = factor(low), age,
+   lwt, race, smoke = (smoke > 0), ptd,
+   ht = (ht > 0), ui = (ui > 0), ftv)
> detach(birthwt)
```

We could obtain the maximum likelihood estimates with the command:

```
glm.out <- glm(low ~ ., binomial, bwt)
```

We will store these estimates and use them as starting values. We could also fit the model with `MCMClogit()` which assumes a multivariate normal prior on the β vector.

Suppose, however, that we would like to fit the model where our prior on β is:

$$p(\beta) \propto \mathbb{I}(\beta_6 > 0)\mathbb{I}(\beta_8 > \beta_7) \prod_{i=1}^{k} \frac{1}{2(1+(\beta_i/2)^2)}$$

This is an independent Cauchy prior with location parameter 0 and scale parameter 2 truncated to a sub-region of the parameter space. See Geweke (1986) for some situations where such constraints may be of interest.

To fit this model with **MCMCpack**, one has to code the log-posterior density in R. This function can be written as:

```
> logit.log.post <- function(beta){
+   ## constrain smoking coefficient to be
+   ## greater than zero
+   if (beta[6] <=0) return(-Inf)
+
+   ## constrain coefficient on ht to be
+   ## greater than coefficient on ptd
+   if (beta[8] <= beta[7]) return(-Inf)
+
+   ## form posterior
+   eta <- X %*% beta
+   p <- 1.0/(1.0+exp(-eta))
+   log.like <- sum(Y * log(p) +
+      (1-Y)*log(1-p))
+   log.prior <- sum(dcauchy(beta,
+      0, 2, log=TRUE))
+   return(log.like + log.prior)
+ }
```

Note that the argument to the function only contains the parameter vector. The data must be in the environment from which the function is called, which is done automatically by the model-fitting function. See the documentation for details. To prepare the data for analysis, we will build a vector that holds y_i and a matrix that holds x_i:

```
> Y.vec <- as.numeric(bwt$low) - 1
> X.mat <- model.matrix(glm.out)
```

The function we will use to simulate from the posterior is `MCMCmetrop1R()`. This function samples in a single block from a user-defined (log)-density using a random walk Metropolis algorithm with a multivariate normal proposal distribution. To simulate from the posterior distribution of this model, we issue the command:

```
> posterior3 <- MCMCmetrop1R(logit.log.post,
+   theta.init=coef(glm.out), burnin=1000,
+   mcmc=200000, thin=20, tune=.7,
+   Y=Y.vec, X=X.mat, verbose=500)
```

Here we use the results from the `glm()` function as our starting values. We choose a tuning parameter to produce an acceptance rate of about 25%. The data are passed with the `Y=Y.vec, X=X.mat` options. `MCMCmetrop1R()` puts these into the appropriate environment such that the data are available to the function when performing simulation.

As with all **MCMCpack** model functions, this code returns an `mcmc` object. Here, to summarize the results, we first label our variables, and then summarize the posterior:

```
> varnames(posterior3) <- colnames(X.mat)
> summary(posterior3)
```

We report just the posterior medians and central 95% credible intervals from the coda summary in Figure 3.

	2.5%	50%	97.5%
(Intercept)	-1.24909	0.70661	2.918533
age	-0.10475	-0.03132	0.040915
lwt	-0.02965	-0.01595	-0.003622
raceblack	0.10524	1.10791	2.176933
raceother	-0.09613	0.73250	1.606134
smokeTRUE	0.11367	0.78682	1.587147
ptdTRUE	0.35179	1.19072	2.076352
htTRUE	1.01740	2.01728	3.313555
uiTRUE	-0.21642	0.67817	1.573814
ftv1	-1.35557	-0.41235	0.459527
ftv2+	-0.75103	0.14322	1.009499

Figure 3: Posterior medians and 2.5th and 97.5th percentiles from the constrained logistic regression model fit using `MCMCmetrop1R()`

Future developments

MCMCpack is a work in progress and under current active development. Going forward, we intend to implement more standard models—especially those used commonly in the social sciences. To illustrate the use of **MCMCpack** for applied problems, we will provide detailed vignettes and more datasets. We also intend to continue improving the flexibility of **MCMCpack**. One approach to this is to give users the ability to provide non-standard priors to any of the standard model fitting functions. We also plan to expand the number of general-purpose sampling functions. The website for the **MCMCpack** project (`http://mcmcpack.wustl.edu`) contains a more detailed list of things to come. We welcome comments and suggestions from the R community about **MCMCpack** and how we can make it a better tool for applied Bayesian inference.

Acknowledgements

We gratefully acknowledge support from the United States National Science Foundation (Grants SES-0350646 and SES-0350613), the Department of Political Science and the Weidenbaum Center at Washington University, and the Department of Government and the Institute for Quantitative Social Science at Harvard University. Neither the Foundation, Washington University, nor Harvard University bear any responsibility for this software.

Bibliography

J. Clinton, S. Jackman, and D. Rivers. The statistical analysis of roll call data. *American Political Science Review*, 98:355–370, 2004. 4

J. Geweke. Exact inference in the inequality constrained normal linear regression model. *Journal of Applied Econometrics*, 1(2):127–141, 1986. 6

A. D. Martin and K. M. Quinn. Dynamic ideal point estimation via Markov chain Monte Carlo for the U.S. Supreme Court, 1953-1999. *Political Analysis*, 10:134–153, 2002. 4

A. D. Martin and K. M. Quinn. *MCMCpack: Markov chain Monte Carlo (MCMC) Package*, 2005. URL http://mcmcpack.wustl.edu. R package version 0.6-3. 2

A. D. Martin, K. M. Quinn, and D. B. Pemstein. *Scythe Statistical Library*, 2005. URL http://scythe.wustl.edu. Version 1.0. 2

M. Plummer. *JAGS: Just Another Gibbs Sampler*, 2005. URL http://www-fis.iarc.fr/~martyn/software/jags/. Version 0.8. 2

M. Plummer, N. Best, K. Cowles, and K. Vines. *coda: Output Analysis and Diagnostics for MCMC*, 2005. URL http://www-fis.iarc.fr/coda/. R package version 0.9-2. 3

D. Spiegelhalter, A. Thomas, N. Best, and D. Lunn. *WinBUGS*, 2004. URL http://www.mrc-bsu.cam.ac.uk/bugs/winbugs/. Version 1.4.1. 2

A. Thomas. *OpenBUGS*, 2004. URL http://mathstat.helsinki.fi/openbugs/. 2

W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer, New York, fourth edition, 2002. 5

Andrew D. Martin
Department of Political Science
Washington University in St. Louis, USA
admartin@wustl.edu

Kevin M. Quinn
Department of Government
Harvard University, USA
kevin_quinn@harvard.edu

CODA: Convergence Diagnosis and Output Analysis for MCMC

by Martyn Plummer, Nicky Best, Kate Cowles and Karen Vines

At first sight, Bayesian inference with Markov Chain Monte Carlo (MCMC) appears to be straightforward. The user defines a full probability model, perhaps using one of the programs discussed in this issue; an underlying sampling engine takes the model definition and returns a sequence of dependent samples from the posterior distribution of the model parameters, given the supplied data. The user can derive any summary of the posterior distribution from this sample. For example, to calculate a 95% credible interval for a parameter α, it suffices to take 1000 MCMC iterations of α and sort them so that $\alpha_1 < \alpha_2 < \ldots < \alpha_{1000}$. The credible interval estimate is then $(\alpha_{25}, \alpha_{975})$.

However, there is a price to be paid for this simplicity. Unlike most numerical methods used in statistical inference, MCMC does not give a clear indication of whether it has converged. The underlying Markov chain theory only guarantees that the distribution of the output will converge to the posterior in the limit as the number of iterations increases to infinity. The user is generally ignorant about how quickly convergence occurs, and therefore has to fall back on *post hoc* testing of the sampled output. By convention, the sample is divided into two parts: a "burn in" period during which all samples are discarded, and the remainder of the run in which the chain is considered to have converged sufficiently close to the limiting distribution to be used. Two questions then arise:

1. How long should the burn in period be?

2. How many samples are required to accurately