AMCMC: An R interface for adaptive MCMC

by

Jeffrey S. Rosenthal*

(February 2007)

Abstract. We describe AMCMC, a software package for running adaptive MCMC algorithms on user-supplied density functions. AMCMC provides the user with an R interface, which in turn calls C programs for faster computations. The user can supply the density and functionals either as R objects, or as auxiliary C files. We describe experiments which illustrate that for fast performance in high dimensions, it is best that the latter option be used.

1. Introduction.

Since at least the publication of [6], Markov chain Monte Carlo (MCMC) algorithms have been extremely widely used in statistics. Since many related MCMC algorithms are available, a major issue is the appropriate choice of algorithm and tuning parameters (e.g. [16]). While such choices can sometimes be made through human ingenuity, a long-standing goal in MCMC (see e.g. [10]) is to make such choices, and the application of MCMC, more routine.

One recent development in this direction has been work on *adaptive* MCMC algorithms, which get the computer to update tuning parameters and other choices as the algorithm runs. Naive adaption can destroy ergodicity (e.g. [20]), but carefully designed adaption can be valid and effective ([11], [2], [3], [17], [18], [1], [9]), and may hold promise for wider application of MCMC methods in the future.

One limitation of widespread use of adaptive MCMC methods is software. While some general-purpose MCMC software is available, notably the widely-used BUGS [4], such software generally does not take advantage of adaptive techniques. And while it may conceivably incorporate adaption at some later stage, the main version of BUGS runs only on a proprietary operating system [22], and an alternative open-source version [14] may perhaps have stalled, so the future is unclear on this point.

*Department of Statistics, University of Toronto, Toronto, Ontario, Canada M5S 3G3. Email: `jeff@math.toronto.edu`. Web: `http://probability.ca/jeff/` Supported in part by NSERC of Canada.

Meanwhile, the statistical software R [15] is now very well developed and widely used, and it is completely open source. Since R is an interpreted language, it runs somewhat slowly and is not ideal for running computationally intensive MCMC. However, the C programming language is freely compiled (usually with [7]), runs very quickly, and can be called from R using the built-in .C( ) and .Call( ) functions. So, it seems reasonable to develop new software to run MCMC from R, with cross-calls to C.

There have been some previous MCMC packages written for R ([8], [12]), but they do not use adaptive techniques, and also they require evaluating the target density directly within R (see Section 4). In this paper, we describe (Section 3) a new software package, AMCMC, for this purpose. We also consider (Section 4) issues of how best to harness the speed of C from within R.

2. The Adaptive Metropolis-within-Gibbs Algorithm.

MCMC concerns itself primarily with estimating the expected value of a given functional, with respect to a given (usually high-dimensional) density function. One way of performing this estimation is with an adaptive Metropolis-within-Gibbs algorithm ([18], Section 3).

Specifically, for each variable i in turn, the addition of a $N(0, \sigma_i^2)$ random quantity to that variable is proposed. The proposal is then accepted with the usual Metropolis probability, $\min[1, \ \pi(\text{new}) \,/\, \pi(\text{old})]$, otherwise it is rejected and variable i remains as it was.

This Metropolis step is performed for each variable in turn, and repeated some fixed number of times, in each "batch". At the end of each batch, each of the σ_i^2 proposal variances is adapted, i.e. modified by a small amount to better balance the fraction of acceptances (cf. [16]). This leads to new Metropolis-within-Gibbs algorithms with different, hopefully better scalings σ_i^2.

In runs of dimension as high as 500, this algorithm performed very well [18] when programmed directly in C. The question is whether such algorithmic success can be combined with the ease and interactivity of R.

3. The AMCMC Package.

The AMCMC package [21] is written in R and C. It allows a user to specify a target density function π, and a desired real-valued functional h. The package then estimates the expected value of that functional with respect to that density function, using the adaptive Metropolis-within-Gibbs algorithm described above.

The package allows for numerous other quantities: the batch length, the number of batches to be performed, the Markov chain's initial value, the fraction of initial output to be discarded as "burn-in", etc. Each of these quantities can be specified by the user if desired, or left to its built-in default value if the user prefers. The result is an R function that is easy to use, but which gives the user significant control if desired.

The package provides an R function, which in turn calls a C program using .Call( ). Normally, the C program then it turn makes calls to R to evaluate new density and functional values whenever needed, similar to other R packages (e.g. [8]). This raises the question of whether some of C's speed is being sacrificed by using R for function evaluation.

To deal with this problem, AMCMC also allows the user to optionally specify their own density and functional directly in C, by modifying a simple auxiliary C file. This does require that the user specify certain lines of C code, but the amount needed is quite minimal. If the user chooses to do this, then the R function can be told to do the function evaluation directly in C, resulting in much faster running speed, as we now discuss.

4. Timing: R versus C.

Since R is an interpreted language, it runs much slower than C in general. So, a common practice is for R functions to in turn call C programs for computations. One obstacle with MCMC algorithms is that, even with efficient programming, one new target density value must be evaluated every time a new proposal is considered, and one new functional value must be evaluated every time a new state is accepted (after the burn-in period). If those functions are defined as R objects, then the C program must in turn call R to do the evaluation (e.g. [8]).

As discussed above, AMCMC provides an optional interface to allow the user to specify their functions directly in C. The question is, how much speed-up does this provide?

We tested this on the target density for the 20-dimensional statistical model analysed in [19], based on the models of [5] and [13] for baseball hitting percentage data. The run involved 1000 different batches, each consisting of 10 updates of each of the 20 different variables, for a total of 200,000 individual Metropolis steps. Each step required computing both the functional value (simply the first coordinate squared), and the log target density (a rather complicated formula involving sums of logs of various normal and gamma distributions). For consistency, all runs were done with verboselevel = 3.

We ran the AMCMC package on a modern personal Linux computer. In all case the user interface was in R, and the main algorithm and loops were (of course) in C. However, we tested each of four different configurations: where the density and functional evaluations are both done in C, where one is done in C and the other in R, and where both are done in R. For comparison, we also re-programmed the overall algorithm to run purely in R, i.e. without using the AMCMC package and without any calls to C at all.

We then ran each of these five configurations (only) once, and recorded the resulting estimates and running times. The results were as follows:

algorithm	density	functional	estimate	time (secs)
C	C	C	0.3931286	2.57
C	C	R	0.3907571	2.90
C	R	C	0.3926776	259.81
C	R	R	0.3926751	260.16
R	R	R	0.3928319	306

Looking at the table, we see that all configurations computed roughly the same estimate, close to 0.392. So, that is a nice confirmation that the algorithm is indeed running correctly and giving accurate answers in all configurations.

However, the running times are vastly different. Interestingly, there is only a modest speed-up (less than a minute) from running the overall algorithm in C instead of R. And, there is very small speed-up (about one-third of a second) from computing the functional in C instead of R. However, there is *tremendous* speed-up from computing the density function in C instead of R: over four minutes, or a factor of about 100.

We learn from this that the amount of "overhead" in making function calls to R from C (as with the functional evaluation) is very small indeed. What affects the running time is the actual intensive computations: somewhat from the loops, accept/reject, etc. of the overall algorithm, and especially from the complicated evaluation of the target density function. Doing these intensive computations in C rather than R results in very large speed-up.

5. Conclusion.

AMCMC appears to be a promising package for applying recent advances in adaptive MCMC to general target densities, incorporating the speed of C while providing the interactivity and convenience of R. However, to achieve excellent speed-up, it is necessary to use AMCMC's option of specifying the target density function directly in an auxiliary C file. It is to be hoped that others will add to AMCMC (which is freely available [21]) and similar packages, to make adaptive MCMC algorithms more convenient and wide-spread in the future.

More generally, it appears that any packages that wish to combine C's speed with R's convenience would be wise to allow for all intensive computations to (optionally) be done directly within C.

References

[1] C. Andrieu and Y.F. Atchadé (2005), On the efficiency of adaptive MCMC algorithms. Preprint.

[2] C. Andrieu and E. Moulines (2003), On the ergodicity properties of some adaptive Markov Chain Monte Carlo algorithms. Preprint.

[3] Y.F. Atchadé and J.S. Rosenthal (2005), On Adaptive Markov Chain Monte Carlo Algorithms. Bernoulli **11**, 815–828.

[4] The BUGS Project. At: http://www.mrc-bsu.cam.ac.uk/bugs/

[5] B. Efron and C. Morris (1975), Data analysis using Stein's estimator and its generalizations. J. Amer. Stat. Assoc., Vol. **70**, No. **350**, 311-319.

[6] A.E. Gelfand and A.F.M. Smith (1990), Sampling based approaches to calculating marginal densities. J. Amer. Stat. Assoc. **85**, 398–409.

[7] GCC, the GNU Compiler Collection. At: http://gcc.gnu.org/

[8] C.J. Geyer (2005), MCMC R Package. At: http://www.stat.umn.edu/geyer/mcmc/

[9] P. Giordani and R. Kohn (2006), Adaptive independent Metropolis-Hastings by fast estimation of mixtures of normals. Preprint.

[10] P.J. Green and D.J. Murdoch (1998), Exact sampling for Bayesian inference: towards general purpose algorithms. In Bayesian Statistics **6**, J. M Bernardo et al. (eds.), Oxford University Press. 301–321

[11] H. Haario, E. Saksman, and J. Tamminen (2005), Componentwise adaptation for high dimensional MCMC. Comput. Stat. **20**, 265–274.

[12] A.D. Martin and K.M. Quinn (2007), MCMCpack. At: http://mcmcpack.wustl.edu/wiki/index.php/Main_Page

[13] C. Morris (1983), Parametric empirical Bayes confidence intervals. Scientific Inference, Data Analysis, and Robustness, 25-50.

[14] OpenBUGS. At: http://mathstat.helsinki.fi/openbugs/

[15] The R Project for Statistical Computing. At: http://www.r-project.org/

[16] G.O. Roberts and J.S. Rosenthal (2001), Optimal scaling for various Metropolis-Hastings algorithms. Stat. Sci. **16**, 351–367.

[17] G.O. Roberts and J.S. Rosenthal (2005), Coupling and Ergodicity of Adaptive MCMC. Preprint.

[18] G.O. Roberts and J.S. Rosenthal (2006), Examples of Adaptive MCMC. Preprint.

[19] J.S. Rosenthal (1996), Convergence of Gibbs sampler for a model related to James-Stein estimators. Stat. and Comput. **6**, 269–275.

[20] J.S. Rosenthal (2004), Adaptive MCMC Java Applet. At: http://probability.ca/jeff/java/adapt.html

[21] J.S. Rosenthal (2007), The AMCMC package. At: http://probability.ca/amcmc/

[22] WinBUGS. At: http://www.mrc-bsu.cam.ac.uk/bugs/winbugs/contents.shtml