Available online at www.sciencedirect.com
ScienceDirect
Computational Statistics & Data Analysis 51 (2007) 5467–5470
COMPUTATIONAL STATISTICS & DATA ANALYSIS
www.elsevier.com/locate/csda

Short communication

AMCMC: An R interface for adaptive MCMC

Jeffrey S. Rosenthal*, [1]

Department of Statistics, University of Toronto, Toronto, Ont., Canada M5S 3G3

Received 15 February 2007; accepted 17 February 2007
Available online 3 March 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.

Keywords: Markov chain Monte Carlo; Adaptive algorithms; Statistical software; R/C interface

1. Introduction

Since at least the publication of Gelfand and Smith (1990), 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. Roberts and Rosenthal, 2001). While such choices can sometimes be made through human ingenuity, a long-standing goal in MCMC (see e.g. Green and Murdoch, 1998) 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. Rosenthal, 2004), but carefully designed adaption can be valid and effective (Haario et al., 2005; Andrieu and Moulines, 2003; Atchadé and Rosenthal, 2005; Roberts and Rosenthal, 2005, 2006; Andrieu and Atchadé, 2005; Giordani and Kohn, 2006), 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 (The BUGS Project), 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 (WinBUGS), and an alternative open-source version (OpenBUGS) may perhaps have stalled, so the future is unclear on this point.

Meanwhile, the statistical software R (The R Project for Statistical Computing) is now very well developed and widely used, and it is completely open source. Since R is an interpreted language, it runs somewhat slowly

* Fax: +1 416 978 5133.
E-mail address: jeff@math.toronto.edu
URL: http://probability.ca/jeff/.
[1] Supported in part by NSERC of Canada.

0167-9473/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.csda.2007.02.021

and is not ideal for running computationally intensive MCMC. However, the C programming language is freely compiled (usually with GCC, the GNU Compiler Collection), 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 (Geyer, 2005; Martin and Quinn, 2007), 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 (Roberts and Rosenthal, 2006, 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. Roberts and Rosenthal, 2001). 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 (Roberts and Rosenthal, 2006) 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 (Rosenthal, 2007) 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. Geyer, 2005). 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. Geyer, 2005).

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?