Tools for Computing

TOOLS FOR BAYESIAN DATA ANALYSIS IN R

Jouni Kerman, Novartis Pharma AG, Switzerland
Andrew Gelman, Columbia University, USA
jouni@kerman.com

Introduction

Bayesian data analysis includes but is not limited to Bayesian inference (Gelman et al., 2003; Kerman, 2006a). Here, we take *Bayesian inference* to refer to posterior inference (typically, the simulation of random draws from the posterior distribution) given a fixed model and data. *Bayesian data analysis* takes Bayesian inference as a starting point but also includes fitting a model to different datasets, altering a model, performing inferential and predictive summaries (including prior or posterior predictive checks).

These tasks require a flexible computing environment that allows us to fit a Bayesian probability model (generating simulations from the joint posterior distribution), but also to manipulate and summarize simulations graphically and numerically.

The most general programs currently available for Bayesian inference are WinBUGS (BUGS Project, 2004) and OpenBUGS, which can be accessed from R using the packages **R2WinBUGS** (Sturtz et al., 2005) and **BRugs**. In addition, various R packages exist that directly fit particular Bayesian models (e.g. **MCMCPack**, Martin and Quinn (2005)), or emulate aspects of BUGS (JAGS). In this article, we describe the ongoing development of two R packages that perform important aspects of Bayesian data analysis.

Umacs

Umacs (Universal Markov chain sampler) is an R package (to be released) that facilitates the construction of the Gibbs sampler and Metropolis algorithm for Bayesian inference (Kerman, 2006b). Writing one's own Gibbs/Metropolis sampler is sometimes necessary for large problems that cannot be fit using programs like BUGS.

Two programs implementing Gibbs samplers differ essentially just by their updating functions. Different Metropolis samplers sample from different posterior functions, but have a similar program structure. Umacs provides the necessary program structure around user-supplied Gibbs updating functions or Metropolis samplers, writing a complete, customized sampler function in R, ready to be run.

The user supplies data, parameter names, updating functions (which can be some mix of Gibbs samplers and Metropolis jumps, with the latter determined by specifying a log-posterior density function), and procedures for generating starting points. Using these inputs, Umacs generates (writes) a customized R sampler function that automatically updates, keeps track of Metropolis acceptances (and uses acceptance probabilities to tune the jumping kernels, following Gelman et al. (1995)), monitors convergence (following Gelman and Rubin (1992)), summarizes results graphically, and returns the inferences as arrays of simulations, or as simulation-based random variable objects (see **rv**, below).

Umacs is customizable and modular, and can be expanded to include more efficient Gibbs/Metropolis steps. Current features include adaptive Metropolis jumps for vectors and matrices of random variables (which arise, for example, in hierarchical regression models, with a different vector of regression parameters for each group). Real-time trace plots can be defined for any scalar parameters or for the convergence statistics, if desired (Figure 5).

Figure 1 illustrates how a simple Bayesian hierarchical model (Gelman et al., 2003, page 451) can be fit using Umacs: $y_j \sim \mathrm{N}(\theta_j, \sigma_j^2)$, $j = 1, \ldots, J$ ($J = 8$), where σ_j are fixed and the means θ_j are given the prior $t_\nu(\mu, \tau)$. In our implementation of the Gibbs sampler, θ_j is drawn from a Gaussian distribution with a random variance component V_j. The conditional distributions of θ, μ, V, and τ can be calculated analytically, so we update them each by a direct (Gibbs) update. The updating functions are to be specified as R functions (here, `theta.update`, `V.update`, `mu.update`, etc.). The degrees-of-freedom parameter ν is also unknown, and updated using a Metropolis algorithm. To implement this, we only need to supply a function calculating the log-

```
s <- Sampler(
  J       = 8,
  sigma.y = c(15, 10, 16, 11,  9, 11, 10, 18),
  y       = c(28,  8, -3,  7, -1,  1, 18, 12),
  theta   = Gibbs(theta.update,theta.init),
  V       = Gibbs(V.update, V.init),
  mu      = Gibbs(mu.update,mu.init),
  tau     = Gibbs(tau.update, tau.init),
  nu      = SMetropolis(log.post.nu, nu.init),
  Trace("theta[1]")
)
```

Figure 1: *Invoking the Umacs* `Sampler` *function to generate an R Markov chain sampler function* `s(...)`. *Updating algorithms are associated with the unknown parameters* $(\theta, V, \mu, \tau, \nu)$. *Optionally, the non-modeled constants and data (here* J, σ, y*) can be localized to the sampler function by defining them as parameters; the function* `s` *then encapsulates a complete sampling environment that can be even moved over and run on another computer without worrying about the availability of the data variables. The "virtual updating function"* `Trace` *displays a real-time trace plot for the specified scalar variable (thus updating the the graphical window which acts as a parameter).*

arithm of the posterior function; Umacs supplies the code. We have several Metropolis classes for efficiency; `SMetropolis` implements the Metropolis update for a scalar parameter. These "updater-generating functions" (`Gibbs` and `SMetropolis`) also require an argument specifying a function returning an initial starting point for the unknown parameter (here, `theta.init`, `mu.init`, `tau.init`, etc.).

The function produced by `Sampler` runs a given number of iterations and a given number of chains; if we are not satisfied with the convergence, we may resume iteration without having to restart the chains. It is also possible to add chains. The length of the burn-in period that is discarded is user-definable and we may also specify the desired number of simulations to collect, automatically performing thinning as the sampler runs.

Once the pre-specified number of iterations are done, the sampler function returns the simulations wrapped in an object which can be coerced into a plain matrix of simulations or into a list of random variable objects (see **rv**, below), which can be then attached to the search path.

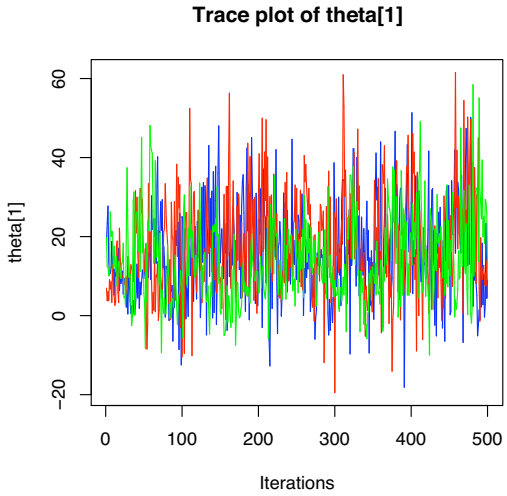

Figure 2: *Real-time trace plot of the scalar component* θ_1 *in Umacs; different colors refer to different chains. It is possible to define any number of trace plots for any scalars in the model. A trace plot behaves conceptually just like a parameter that is updated during each iteration of the Gibbs sampler. In practice, we update the graph every 10 or 50 iterations not to slow down the sampler.*

rv

rv is an R package that defines a new simulation-based *random variable* class in R along with various mathematical and statistical manipulations (Kerman and Gelman, 2005). The program creates an object class whose instances can be manipulated like numeric vectors and arrays. However, each element in a vector contains a hidden dimension of simulations: the `rv` objects can thus be thought of being approximations of random variables. That is, a random

scalar is stored internally as a vector, a random vector as a matrix, a random matrix as a three-dimensional array, and so forth. The random variable objects are useful when manipulating and summarizing simulations from a Markov chain simulation (for example those generated by Umacs). They can also be used in simulation studies (Kerman, 2005). The number of simulations stored in a random variable object is user-definable.

The `rv` objects are a natural extension of numeric objects in R, which are conceptually just "random variables with zero variance"—that is, constants. Arithmetic operations such as `+` and `^` and elementary functions such as `exp` and `log` work with `rv` objects, producing new `rv` objects.

These random variable objects work seamlessly with regular numeric vectors: for example, we can impute random variable `z` into a regular numeric vector `y` with a statement like `y[is.na(y)] <- z`. This converts `y` automatically into a random vector (`rv` object) which can be manipulated much like any numeric object; for example we can write `mean(y)` to find the distribution of the arithmetic mean function of the (random) vector `y` or `sd(y)` to find the distribution of the sample standard deviation statistic.

The default `print` method of a random variable object outputs a summary of the distribution represented by the simulations for each component of the argument vector or array. Figure 3 shows an example of a summary of a random vector `z` with five random components.

```
> z
     name mean   sd      Min  2.5%  25%  50%  75% 97.5% Max
[1] Alice 59.0 27.3 ( -28.66  1.66 42.9 59.1 75.6   114 163 )
[2]   Bob 57.0 29.2 ( -74.14 -1.98 38.3 58.2 75.9   110 202 )
[3] Cecil 62.6 24.1 ( -27.10 13.25 48.0 63.4 76.3   112 190 )
[4]  Dave 71.7 18.7 (   2.88 34.32 60.6 71.1 82.9   108 182 )
[5] Ellen 75.0 17.5 (   4.12 38.42 64.1 75.3 86.2   108 162 )
```

Figure 3: *The* `print` *method of an* `rv` *(random variable) object returns a summary of the mean, standard deviation, and quantiles of the simulations embedded in the vector.*

Standard functions to plot graphical summaries of random variable objects are being developed. Figure 4 shows the result of a statement `plot(x,y)` where `x` are constants and `y` is a random vector with 10 constant components (shown as dots) and five random components (shown as intervals).

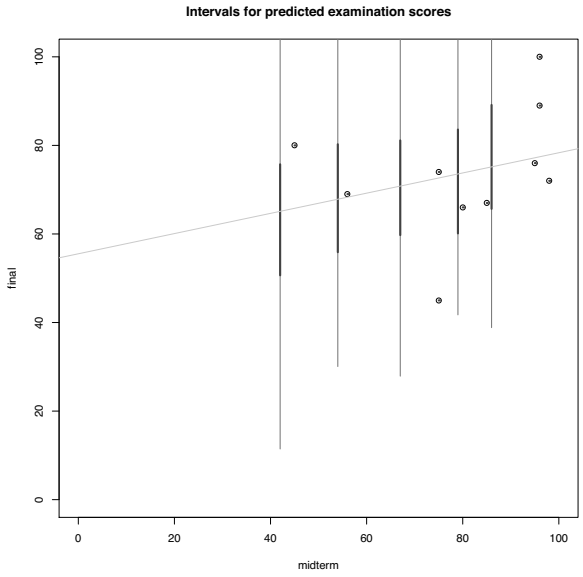

Figure 4: *A scatterplot of fifteen points* `(x,y)` *where five of the components of* `y` *are random, that is, represented by simulations and thus are drawn as intervals. Black vertical intervals represent the 50% posterior intervals and the gray ones the 95% intervals. This plot was simply obtained by a command* `plot(x,y)` *(with appropriate supplementary arguments). The light grey line is a regression line computed from the ten fixed points, included for reference.*

Many methods on `rv` objects have been written, for example `E(y)` returns the individual means (expectations) of the components of a random vector `y`. A statement `Pr(z[1]>z[2])` would give an estimate of the probability of the event $\{z_1 > z_2\}$.

Random-variable generating functions generate new `rv` objects by sampling from standard distributions, for example `rvnorm(n=10, mean=0, sd=1)` would return a random vector representing 10 draws from the standard normal distribution. What makes these functions interesting is that we can give them pa-

rameters that are also random, that is, represented by simulations. If y is modeled as $\mathrm{N}(\mu, \sigma^2)$ and the random variable objects `mu` and `sigma` represent draws from the joint posterior distribution of (μ, σ)—we can obtain these if we fit the model with **Umacs** or BUGS for example—then a simple statement like `rvnorm(mean=mu, sd=sigma)` would generate a random variable representing draws from the posterior predictive distribution of y. A single line of code thus will in fact perform Monte Carlo integration of the joint density of $(y^{\mathrm{rep}}, \mu, \sigma)$, and draw from the resulting distribution $p(y^{\mathrm{rep}}|y) = \int\int \mathrm{N}(y^{\mathrm{rep}}|\mu, \sigma)p(\mu, \sigma|y)\,\mathrm{d}\mu\,\mathrm{d}\sigma$. (We distinguish the observations y and the unobserved random variable y^{rep}, which has the same conditional distribution as y).

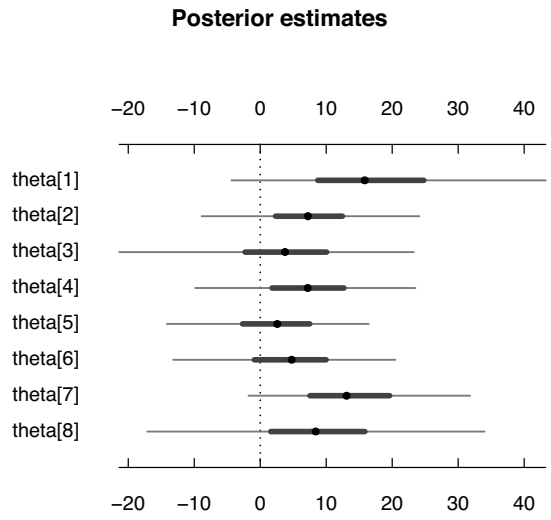

Figure 5: *A posterior interval plot of the vector $\theta = (\theta_1, \ldots, \theta_8)$ fitted using Umacs in the previous section; the resulting object was coerced into an* `rv` *object* `theta`*, and then displayed in graphical form using a command that plots the components of the argument vector vertically. Since the arguments of* `theta` *are random variables, they are plotted as intervals; constants would be plotted as single points, indicating no posterior uncertainty. The thick lines in the middle are 50% posterior intervals and the thinner lines are 95% posterior intervals of the corresponding scalar components of θ. The dots are posterior means. This kind of a graph is especially useful for displaying estimates from a hierarchical model.*

Summary

Most of the work of writing a standard Gibbs/Metropolis sampler can be produced automatically; Umacs makes this possible by writing a customized sampler given only the updating functions or log-posterior functions relevant to the model. The user-defined parameters are embedded into standard looping structures and Metropolis updating routines, saving the trouble of writing the program from scratch. This saves time and makes debugging the sampler program easier.

Once posterior simulations are generated, it is awkward to work with the resulting inferences, display them graphically, generate posterior probability statements or generate predictions, since the inferences are in the form of numerical arrays of simulations and not accessible directly as random variables. The package 'rv' provides a new simulation-based random variable object class, which makes the job of manipulating and summarizing posterior inferences easier and provides the foundation of a "Bayesian programming environment." Using random variable objects instead of arrays of simulations saves time and effort in writing—and understanding—program code.

We hope these packages will be useful and also will motivate future work by others, so that Bayesian inference can be performed in the interactive spirit of R.

Acknowledgements

We thank Tian Zheng, Shouhao Zhao, Yuejing Ding, and Donald Rubin for help with the various programs and the National Science Foundation for financial support.

Bibliography

BUGS Project. BUGS: Bayesian Inference Using Gibbs Sampling. `http://www.mrc-bsu.cam.ac.uk/bugs/`, 2004.

A. Gelman and D. Rubin. Inference from iterative simulation using multiple sequences (with discussion). *Statistical Science*, 7:457–511, 1992.

A. Gelman, G. Roberts, and W. Gilks. Efficient Metropolis jumping rules. In J. M. Bernardo, J. O. Berger, A. P. Dawid, and A. F. M. Smith, editors, *Bayesian Statistics 5*. Oxford University Press, 1995.

A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin. *Bayesian Data Analysis*. Chapman & Hall/CRC, London, 2nd edition, 2003.

J. Kerman. Using random variable objects to compute probability simulations. Technical report, Department of Statistics, Columbia University, 2005.

J. Kerman. An integrated framework for Bayesian graphic modeling, inference, and prediction. Technical report, Department of Statistics, Columbia University, 2006a.

J. Kerman. Umacs: A Universal Markov Chain Sampler. Technical report, Department of Statistics, Columbia University, 2006b.

J. Kerman and A. Gelman. Manipulating and summarizing posterior simulations using random variable objects. To appear in Statistics and Computing.

A. D. Martin and K. M. Quinn. `MCMCpack` 0.6-6. `http://mcmcpack.wustl.edu/`, 2005.

S. Sturtz, U. Ligges, and A. Gelman. R2WinBUGS: A package for running WinBUGS from R. *Journal of Statistical Software*, 12(3):1–16, 2005. ISSN 1548-7660.

Jouni Kerman
Methodology Biostatistics, Novartis Pharma AG, Switzerland

Andrew Gelman
Department of Statistics
Columbia University, NY, USA

WANTED: NEWSLETTER CO-EDITOR, STATS GRAPHICS

The Statistical Computing and Graphics Newsletter (SCGN) needs a new co-Editor on the Stat Graphics side. This is a great opportunity to serve the Statistical Graphics Section and the ASA in general. Co-editing it is a volunteer job with many rewards.

The Newsletter is a joint product of the Statistical Computing and Statistical Sections of the ASA, hence having two editors, one for Stats Computing and another for Graphics. There are two issues per year: one in the Fall and one in the Spring. The spring issue contains a lot of information about the upcoming ASA meetings, other meetings sponsored by the two sections, announcements of the competition awards and feature articles that anticipate future trends in Stats Computing and Graphics. The Fall issue talks about what happened in those past meetings, announces the competitions and also contains feature articles of high interest. Both the Fall and the Spring issues contain other interesting news and the Chair's columns plus some special columns, depending on availability of contributions for them.

The Editors of SCGN select contributions from different authors after extensive review and decide the final contents of the newsletter and what format the newsletter will have. They follow up on authors to guarantee a timely delivery once their article is accepted, collect news, gather columns from contributors and make sure that everything is done in a timely fashion and appropriately. All this material is then edited and entered into a newsletter semi-template (currently in Pages, a product of Apple's iWorks, but not necessarily so for ever). After the Executive Committees of both sections have approved, and the authors have proofed their pieces, the Newsletter is then posted online and Section members are notified that it is ready. Lately we have also been sending a postcard through regular mail, and will continue to do so.

This is a volunteer job with lots of room for creativity and for making the ASA sections you are part of visible to a wider group of statisticians.

If you are interested in becoming a Co-editor, please contact the Statistics Graphics Chair, Paul Murrell by email. His email address is

p.murrell@auckland.ac.nz