Bayesian data analysis using R

Jouni Kerman and Andrew Gelman

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), and validation of the software used to fit the model.

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)). In this note, we describe our own entry in the "inference engine" sweepstakes but, perhaps more importantly, describe the ongoing development of some R packages that perform other aspects of Bayesian data analysis.

Umacs

Umacs (Universal Markov chain sampler) is an R package (to be released in Spring 2006) that facilitates the construction of the Gibbs sampler and Metropolis algorithm for Bayesian inference (Kerman, 2006b). 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 a procedure for generating starting points. Using these inputs, Umacs 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 random variable objects (see **rv**, below).

Umacs is 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).

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 logarithm 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.).

```
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.*

The program is customizable and modular so that users can define custom updating classes and more refined Metropolis implementations.

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 to a list of random variable objects (see *rv* below), which can be then attached to the search path.

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, see below). 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 2 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 2: *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 3 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).

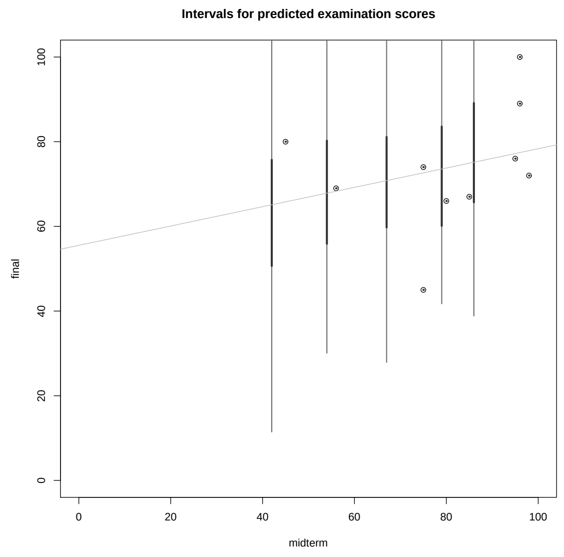

Figure 3: *A scatterplot of fifteen points* `(x,y)` *where five of the components of* `y` *are random, that is, represented by simulations and thus are drawn as intervals. Black vertical intervals represent the 50% uncertainty intervals and the gray ones the 95% intervals. (The light grey line is a regression line computed from the ten fixed points).*

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 parameters 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** (see below) 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).

R & B

The culmination of this research project is an R environment for Bayesian data analysis which would allow inference, model expansion and comparison, model checking, and software validation to be performed easily, using a high-level Bayesian graphical modeling language "*B*" adapted to R, with functions that operate on R objects that include *graphical models*, *parameters* (nodes), and *random variables*. *B* exists now only in conceptual level (Kerman, 2006a), and we plan for its first incarnation in R (called *R & B*) to be a simple version to demonstrate its possibilities. *B* is not designed to be tied to any particular inference engine but rather a general interface for doing Bayesian data analysis. Figure 4 illustrates a hypothetical interactive session using *R & B*.

```
## Model 1: A trivial model:
NewModel(1, "J", "theta", "mu", "sigma", "y")
Model(y)          <- Normal(J, theta, sigma)
Observation(y)    <- c(28,8,-3,7,-1,1,18,12)
Hypothesis(sigma) <- c(15,10,16,11,9,11,10,18)
Observation(J)    <- 8
Fit(1)
# Look at the inferences:
print(theta)
## Model 2: A hierarchical t model
NewModel(2, based.on.model=1, "V", "mu", "tau")
Model(theta) <- Normal(J, mu, V)
Model(V)     <- InvChisq(nu, tau)
Fit(2)
# Look at the new inferences:
plot(theta)
# Draw from posterior predictive distribution:
y.rep1 <- Replicate(y, model=1)
y.rep2 <- Replicate(y, model=2)
## Use the same models but
##   a new set of observations and hypotheses:
NewSituation()
Hypothesis(sigma) <- NULL # Sigma is now unknown.
Fit()
...
```

Figure 4: *A hypothetical interactive session using the high-level Bayesian language "*B*" in R (in development). Several models can be kept in memory. Independently of models, several "inferential situations" featuring new sets of observations and hypotheses (hypothesized values for parameters with assumed point-mass distributions) can be defined. Fitting a model launches an inference engine (usually, a sampler such as* **Umacs** *or BUGS) and stores the inferences as random variable objects. By default, parameters are given noninformative prior distributions.*

Conclusion

R is a powerful language for statistical modeling and graphics; however it is currently limited when it comes to Bayesian data analysis. Some packages are available for fitting models, but it remains awkward to work with the resulting inferences, alter or compare the models, check fit to data, or validate the software used for fitting. This article describes several of our research efforts, which we have made into R packages or plan to do so. We hope these packages will be useful in their own right and also will motivate future work by others integrating Bayesian modeling and graphical data analysis, so that Bayesian inference can be performed in the iterative data-analytic 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. Technical report, Department of Statistics, Columbia University, 2005.

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.