LaplacesDemon Tutorial

Byron Hall
STATISTICAT,LLC

January 8, 2011

Abstract

LaplacesDemon, usually referred to as Laplace's Demon, is a contributed R package for Bayesian inference, and is freely available on the Comprehensive R Archive Network (CRAN). Laplace's Demon allows the choice of four MCMC algorithms to update a Bayesian model according to a user-specified model function. The user-specified model function enables Bayesian inference for any model form, provided the user specifies the likelihood. Laplace's Demon also attempts to assist the user by creating and offering R code, based on a previous model update, that can be copy/pasted and executed. Posterior predictive checks and many other features are included as well. Laplace's Demon seeks to be generalizable and user-friendly to Bayesians...especially Laplacians.

Keywords. Adaptive, AM, Bayesian Inference, Delayed Rejection, DR, DRAM, DRM, LaplacesDemon, Laplace's Demon, Markov chain Monte Carlo, MCMC, Metropolis, R, Random Walk, Random-Walk, STATISTICAT.

Disclaimer. Demonic references are used only to add flavor to the software and its use, and in no way endorse beliefs in demons.

1 Introduction

Bayesian inference is named after Reverend Thomas Bayes (1702-1761) for developing Bayes' theorem, which was published posthumously after his death in 1763. This was the first instance of what would be called inverse probability.

Unaware of Bayes, Pierre-Simon Laplace (1749-1827) independently developed Bayes' theorem and first published his version in 1774, eleven years after Bayes, in one of Laplace's first major works (Laplace, 1774, p.366-367). In 1812, Laplace (1749-1827) introduced a host of new ideas and mathematical techniques in his book, Theorie Analytique des Probabilites. Before Laplace, probability theory was solely concerned with developing a mathematical analysis of games of chance. Laplace applied probabilistic ideas to many scientific and practical problems. Although Laplace is not the father of probability, Laplace may be considered the father of the field of probability.

In 1814, Laplace published his "Essai philosophique sur les probabilites", which introduced a mathematical system of inductive reasoning based on probability. In it, the Bayesian interpretation of probability was developed independently by Laplace, much more thoroughly

than Bayes, so some "Bayesians" refer to Bayesian inference as Laplacian inference. This is a translation of a quote in the introduction to this work:

> "We may regard the present state of the universe as the effect of its past and the cause of its future. An intellect which at a certain moment would know all forces that set nature in motion, and all positions of all items of which nature is composed, if this intellect were also vast enough to submit these data to analysis, it would embrace in a single formula the movements of the greatest bodies of the universe and those of the tiniest atom; for such an intellect nothing would be uncertain and the future just like the past would be present before its eyes" (Laplace, 1814).

The 'intellect' has been referred to by future biographers as Laplace's Demon. In this quote, Laplace expresses his philosophical belief in hard determinism and his wish for a computational machine that is capable of estimating the universe.

This article is an introduction to an `R` package called `LaplacesDemon`, which was designed without consideration for hard determinism, but instead with a lofty goal toward facilitating high-dimensional Bayesian (or Laplacian) inference, posing as its own intellect that is capable of impressive analysis. The `LaplacesDemon` `R` package is often referred to as Laplace's Demon. This article guides the user through installation, data, specifying a model, initial values, updating Laplace's Demon, summarizing and plotting output, posterior predictive checks, general suggestions, discusses independence and observability, covers details of the algorithm, software comparisons, discusses large data sets and speed, explains future goals, and presents references.

Herein, it is assumed that the reader has basic familiarity with Bayesian inference, numerical approximation, and `R`. If any part of this assumption is violated, then suggested sources include Gelman et al. (2004) and Crawley (2007).

To wit, it is suspected that if Laplace's Demon actually existed, then from time to time it would chant:

> *One package to rule them all,*
> *One package to find them,*
> *One package to bring them all and in the darkness bind them*

2 Installation

To obtain Laplace's Demon, simply open `R` and install the `LaplacesDemon` package from a CRAN mirror:

```
> install.packages("LaplacesDemon")
```

A goal in developing Laplace's Demon was to minimize reliance on other packages or software. Therefore, the usual `dep=TRUE` argument does not need to be used, because `LaplacesDemon` does not depend on anything other than base `R`. Once installed, simply use the `library` or

`require` function in `R` to activate the `LaplacesDemon` package and load its functions into memory:

```
> library(LaplacesDemon)
```

```
By using LaplacesDemon, invoked on its own or through another
package, you accept the license agreement in the LaplacesDemon
LICENSE file and at http://www.statisticat.com/laplacesdemon.html
```

Although Laplace's Demon is freely available, a software license restricts it from commercial use, except when a license is purchased from STATISTICAT, LLC[1].

3 Data

Laplace's Demon requires data that is specified in a list. As an example, there is a data set called `demonsnacks` that is provided with the `LaplacesDemon` package. For no good reason, other than to provide an example, the log of `Calories` will be fit as an additive, linear function of the remaining variables. Since an intercept will be included, a vector of 1's is inserted into design matrix **X**.

```
> data(demonsnacks)
> N <- NROW(demonsnacks)
> J <- NCOL(demonsnacks)
> y <- log(demonsnacks$Calories)
> X <- cbind(1, as.matrix(demonsnacks[, c(1, 3:10)]))
> for (j in 2:J) {
+     X[, j] <- (X[, j] - mean(X[, j]))/(2 * sd(X[, j]))
+ }
> parm.names <- rep(NA, J + 1)
> for (j in 1:J) {
+     parm.names[j] <- paste("beta[", j, "]", sep = "")
+ }
> parm.names[J + 1] <- "log.tau"
> MyData <- list(J = J, X = X, parm.names = parm.names, y = y)
```

There are J=10 independent variables (including the intercept), one for each column in design matrix **X**. However, there are 11 parameters, since the residual precision, tau, must be included as well. The reason why it is called `log.tau` will be explained later. Each parameter must have a name specified in the vector `parm.names`, and parameter names must be included with the data. Also, note that each predictor has been centered and scaled, as per Gelman (2008). This is a crude method of centering and scaling, where more elegant methods exist, but it is used here because it is easy to see how it is done.

[1]To obtain a license, send an email to `statisticat@gmail.com` or visit `http://www.statisticat.com/`.

4 Specifying a Model

To use Laplace's Demon, the user must specify a model. Let's consider a simple linear regression, which is often denoted as:

$$y_i \sim N(u_i, \sigma^2)$$
$$u_i = \beta X_i$$

For a Bayesian model, the notation for the residual variance, σ^2, is often replaced with the residual precision, τ^{-1}. Prior probabilities are specified for β and τ:

$$\beta_i \sim N(0, 0.001)$$
$$\tau \sim \Gamma(0.001, 0.001)$$

To specify a model, the user must create a function called `Model`. Here is an example for a simple, linear regression model:

```
> Model <- function(parm, Data) {
+     beta.mu <- rep(0, J)
+     beta.tau <- rep(0.001, J)
+     tau.alpha <- 0.001
+     tau.beta <- 0.001
+     beta <- rep(0, J)
+     for (j in 1:J) {
+         beta[j] <- parm[j]
+     }
+     tau <- exp(parm[J + 1])
+     beta.prior <- rep(0, J)
+     for (j in 1:J) {
+         beta.prior[j] <- dnorm(beta[j], beta.mu[j], 1/sqrt(beta.tau[j]),
+             log = TRUE)
+     }
+     tau.prior <- dgamma(tau, tau.alpha, tau.beta, log = TRUE)
+     mu <- beta %*% t(X)
+     LL <- sum(dnorm(y, mu, 1/sqrt(tau), log = TRUE))
+     LP <- LL + sum(beta.prior) + tau.prior
+     Modelout <- list(LP = LP, Dev = -2 * LL, Monitor = c(tau,
+         mu[1]), yhat = mu)
+     return(Modelout)
+ }
```

Laplace's Demon iteratively maximizes the log of the joint posterior density as specified in this `Model` function. In Bayesian inference, the log of the joint posterior density is proportional to the sum of the log-likelihood and log of the prior densities:

$$\log[p(\theta|y)] \propto \log[p(y|\theta)] + \log[p(\theta)]$$

where θ is a set of parameters, y is the data, $p(\theta|y)$ is the joint posterior density, $p(y|\theta)$ is the likelihood, and $p(\theta)$ is the set of prior densities.

During each iteration in which Laplace's Demon is maximizing the log of the joint posterior density, Laplace's Demon passes two arguments to `Model`: `parm` and `Data`, where `parm` is short for the set of parameters, and `Data` is the data. These arguments are specified in the beginning of the function:

```
Model <- function(parm, Data)
```

Then, the Model function is evaluated and the log of the joint posterior density is calculated as `LP`, and returned to Laplace's Demon in a list called `Modelout`, along with the deviance (`Dev`), a vector (`Monitor`) of any variables desired to be monitored in addition to the parameters, and y[rep] (`yhat`) or replicates of y. All arguments must be returned. Even if there is no desire to observe the deviance and any monitored variable, a scalar must be placed in the second position of the `Modelout` list, and at least one element of a vector for a monitored variable. This can be seen in the end of the function:

```
LP <- LL + sum(beta.prior) + tau.prior

Modelout <- list(LP=LP, Dev=-2*LL, Monitor=c(tau,mu[1]), yhat=mu)

return(Modelout)
```

The rest of the function specifies the prior parameters, parameters, log of the prior densities, and calculates the log-likelihood.

The prior parameters specify the parameters for the prior distributions. Since design matrix $\mathbf{X}$ has J=10 independent variables (including the intercept), there are 10 `beta` parameters and a `tau` parameter for residual precision, the inverse of the variance. Each of the `J beta` parameters will be distributed normally according to mean `beta.mu` and precision `beta.tau`, and the additional `tau` parameter will be gamma-distributed according to `tau.alpha` and `tau.beta`. Here are the specifications for the prior parameters:

```
beta.mu <- rep(0,J)

beta.tau <- rep(1.0E-3,J)

tau.alpha <- 1.0E-3

tau.beta <- 1.0E-3
```

Since Laplace's Demon passes a vector of parameters called `parm` to `Model`, the function needs to know which parameter is associated with which element of `parm`. For this, the vector `beta` is declared, and then each element of `beta` is populated with the value associated in the corresponding element of `parm`. The reason why `tau` is exponentiated will, again, be explained later.

```
beta <- rep(0,J)

for (j in 1:J) beta[j] <- parm[j]

tau <- exp(parm[J+1])
```

To work with the log of the prior densities and according to the assigned names of the parameters and prior parameters, they are specified as follows:

```
beta.prior <- rep(0,J)

for (j in 1:J)

 beta.prior[j] <- dnorm(beta[j], beta.mu[j], 1/sqrt(beta.tau[j]), log=TRUE)

tau.prior <- dgamma(tau, tau.alpha, tau.beta, log=TRUE)
```

It is important to reparameterize all parameters to be real-valued. For example, a positive-only parameter such as variance should be transformed with a `log` function, and a proportion p can be transformed to the real line by a `logit` function, such as `logit(p) = log(p/(1-p))`. Laplace's Demon will attempt to increase or decrease the value of each parameter to maximize `LP`, without consideration for the distributional form of the parameter. In the above example, the residual precision `tau` receives a gamma-distributed prior of the form:

$$\tau \sim \Gamma(0.001, 0.001)$$

In this specification, `tau` cannot be negative. By reparameterizing `tau` as

`tau <- exp(parm[J+1])`,

Laplace's Demon will increase or decrease `parm[J+1]`, which is effectively `log(tau)`). Now it is possible for Laplace's Demon to decrease `log(tau)`) below zero without causing an error or violating its gamma-distributed specification.

Finally, everything is put together to calculate `LP`, the log of the joint posterior density. The expectation vector `mu` is the inner product (`%*%`) of the vector `beta` and the transposed design matrix, `t(X)`. Expectation vector `mu`, vector `y`, and scalar `tau` are used to estimate the sum of the log-likelihoods, where:

$$y_i \sim N(u_i, \tau^{-1})$$

and as noted before, the log of the joint posterior density is:

$$\log[p(\theta|y)] \propto \log[p(y|\theta)] + \log[p(\theta)]$$

```
mu <- beta %*% t(X)

LL <- sum(dnorm(y, mu, 1/sqrt(tau), log=TRUE)

LP <- LL + sum(beta.prior) + tau.prior
```

Specifying the model in the `Model` function is the most involved aspect for the user of Laplace's Demon. But it has been designed so it is also incredibly flexible, allowing a wide variety of Bayesian models to be specified.

Missing values can be estimated in Laplace's Demon, but each missing value must be specified as a parameter in `Model` so that an initial value is assigned.

5 Initial Values

Laplace's Demon requires a vector of initial values for the parameters. Each initial value is a user-specified starting point for a parameter. In this example, there are 11 parameters, and with no prior knowledge, it is a good idea to set them equal to zero or use a random function. The first 10 parameters, the `beta` parameters, have been set equal to zero, and the remaining parameter, `log.tau`, has been set equal to `log(1)`, which is equal to zero; this visually reminds me that I am working with the log of `tau`, rather than `tau`. The order of the elements of the vector of initial values must match the order of the parameters associated with each element of `parm` passed to `Model`.

```
> Initial.Values <- c(rep(0, J), log(1))
```

6 Laplace's Demon

Compared to specifying the model in `Model`, the actual use of Laplace's Demon is very easy. Since Laplace's Demon is stochastic, or involves pseudo-random numbers, it's a good idea to set a 'seed' for pseudo-random number generation, so results can be reproduced. Pick any number you like, but there's only one number appropriate for a demon:

```
> set.seed(666)
```

As with any `R` package, the user can learn about a function by entering a question mark, followed by the name of the function. To learn the details of the `LaplacesDemon` function, enter:

```
> ?LaplacesDemon
```

Here is one of many possible ways to begin:

```
> Fit <- LaplacesDemon(Model, Data = MyData, Adaptive = 900,
+     Covar = NULL, DR = 1, Initial.Values, Iterations = 10000,
+     Periodicity = 10, Status = 1000, Thinning = 10)
```

In this example, an output object called `Fit` will be created as a result of using the `LaplacesDemon` function. `Fit` is an object of class `demonoid`, which means that since it has been assigned a customized class, other functions have been custom-designed to work with it. Laplace's Demon offers four MCMC algorithms (which are explained in the *Details* section). The above example declares the Delayed Rejection Adaptive Metropolis (DRAM) algorithm.

The above example tells the `LaplacesDemon` function to maximize the user-specified `Model` function, given a data set called `MyData`, and according to several settings.

- The `Adaptive=900` argument indicates that a non-adaptive MCMC algorithm will begin, and that it will become adaptive at the 900th iteration. Beginning with the 900th iteration, the MCMC algorithm will estimate the proposal variance or covariance based on the history of the chains.

- The `Covar=NULL` argument indicates that a user-specified variance vector or covariance matrix has not been supplied, so the algorithm will begin with its own estimate.
- The `DR=1` argument indicates that delayed rejection will occur, such that when a proposal is rejected, an additional proposal will be attempted, thus potentially delaying rejection of proposals.
- The `Initial.Values` argument requires a vector of initial values for the parameters.
- The `Iterations=10000` argument indicates that `LaplacesDemon` will update 10,000 times before completion.
- The `Periodicity=10` argument indicates that once adaptation begins, the algorithm will adapt every 10 iterations.
- The `Status=1000` argument indicates that a status message will be printed to the `R` console every 1,000 iterations.
- Finally, the `Thinning=10` argument indicates that only every nth iteration will be retained in the output, and in this case, every 10th iteration will be retained.

By running the `LaplacesDemon` function, the following output was obtained:

```
> Fit <- LaplacesDemon(Model, Data = MyData, Adaptive = 900,
+     Covar = NULL, DR = 1, Initial.Values, Iterations = 10000,
+     Periodicity = 10, Status = 1000, Thinning = 10)

Laplace's Demon was called on Sat Jan  8 22:47:56 2011

Performing initial checks...
Algorithm: Delayed Rejection Adaptive Metropolis

Laplace's Demon is beginning to update...
Iteration: 1000,   Proposal: Multivariate
Iteration: 2000,   Proposal: Multivariate
Iteration: 3000,   Proposal: Multivariate
Iteration: 4000,   Proposal: Multivariate
Iteration: 5000,   Proposal: Multivariate
Iteration: 6000,   Proposal: Multivariate
Iteration: 7000,   Proposal: Multivariate
Iteration: 8000,   Proposal: Multivariate
Iteration: 9000,   Proposal: Multivariate

Assessing Stationarity
Assessing Thinning and Effective Size
Creating Summaries
Creating Output

Laplace's Demon has finished.
```

Laplace's Demon finished quickly, though it had a small data set (N=39), few parameters (K=11), and the model was very simple. At each status of 1000 iterations, the proposal was multivariate, so it did not have to resort to independent proposals. The output object, `Fit`, was created as a list. As with any `R` object, use `str()` to examine its contents:

```
> str(Fit)
```

To access any of these values in the output object `Fit`, simply append a dollar sign and the name of the item. For example, here is how to access the observed acceptance rate:

```
> Fit$Acceptance.Rate
```

```
[1] 0.191
```

7 Summarizing Output

The output object, `Fit`, has many components. The (copious) contents of `Fit` can be printed to the screen with the usual `R` functions:

```
> Fit
> print(Fit)
```

Both return the same output, which is:

```
> Fit
```

```
Call:
LaplacesDemon(Model = Model, Data = MyData, Adaptive = 900, Covar = NULL,
    DR = 1, Initial.Values = Initial.Values, Iterations = 10000,
    Periodicity = 10, Status = 1000, Thinning = 10)

Acceptance Rate: 0.191
Adaptive: 900
Algorithm: Delayed Rejection Adaptive Metropolis
Covar: (NOT SHOWN HERE)
DIC of all samples (Dbar): 86.907
DIC of all samples (pD): 719.76
DIC of all samples (DIC): 806.67
DIC of stationary samples (Dbar): 85.158
DIC of stationary samples (pD): 53.232
DIC of stationary samples (DIC): 138.39
DR: 1
Iterations: 10000
Minutes of run-time: 0.42
Model: (NOT SHOWN HERE)
```

```
Monitor: (NOT SHOWN HERE)
Parameters (Number of): 11
Periodicity: 10
Posterior1: (NOT SHOWN HERE)
Posterior2: (NOT SHOWN HERE)
Recommended Burn-In of Thinned Samples: 301
Recommended Burn-In of Un-thinned Samples: 3010
Recommended Thinning: 270
Status is displayed every 1000 iterations
Summary1: (SHOWN BELOW)
Summary2: (SHOWN BELOW)
Thinned Samples: 1000
Thinning: 10


Summary of All Samples
              Mean       SD     MCSE Eff.Size         LB   Median
beta[1]   5.022744  0.29788 0.017538   288.49  4.7825736  5.04047
beta[2]  -0.449700  0.40129 0.025770   242.48 -1.2192368 -0.45520
beta[3]  -0.354680  0.96424 0.078525   150.78 -2.2047573 -0.31601
beta[4]  -0.092932  0.69866 0.051057   187.25 -1.4984657 -0.10772
beta[5]  -0.403650  0.55309 0.039374   197.32 -1.4227603 -0.45082
beta[6]  -0.472748  0.31433 0.019925   248.87 -1.0715801 -0.45303
beta[7]   2.221839  0.63308 0.053801   138.47  0.8041232  2.24437
beta[8]   0.603436  0.46815 0.030160   240.94 -0.4162341  0.61453
beta[9]  -0.180499  0.56446 0.035346   255.03 -1.3330820 -0.16966
beta[10]  1.555348  0.82088 0.063613   166.52 -0.1322433  1.58307
log.tau   0.620962  0.30480 0.023012   175.43  0.0068253  0.65190
Deviance 86.907163 37.94114 2.026465   350.54 74.3851161 83.12989
Monitor   1.955169  0.59055 0.038829   231.31  0.9898689  1.92412
Monitor   4.157063  0.30662 0.015166   408.75  3.7150806  4.17152
                UB
beta[1]    5.25693
beta[2]    0.36491
beta[3]    1.47275
beta[4]    1.27766
beta[5]    0.74765
beta[6]    0.14696
beta[7]    3.38278
beta[8]    1.48273
beta[9]    0.84930
beta[10]   3.06140
log.tau    1.12403
Deviance 113.01651
Monitor    3.23246
Monitor    4.63016


Summary of Stationary Samples
              Mean       SD      MCSE Eff.Size        LB    Median
```

```
beta[1]    5.039152  0.12158 0.0055652   477.26  4.784144  5.040819
beta[2]   -0.494324  0.39615 0.0274562   208.18 -1.269493 -0.496514
beta[3]   -0.454290  0.97750 0.0652184   224.64 -2.524612 -0.407956
beta[4]   -0.082261  0.69068 0.0497506   192.73 -1.503985 -0.051364
beta[5]   -0.421718  0.53882 0.0343869   245.53 -1.445971 -0.441650
beta[6]   -0.490788  0.31037 0.0220482   198.16 -1.112198 -0.463762
beta[7]    2.294935  0.55973 0.0332093   284.08  1.135636  2.263510
beta[8]    0.635673  0.47593 0.0330957   206.79 -0.336534  0.633569
beta[9]   -0.168345  0.55218 0.0340447   263.07 -1.307631 -0.169823
beta[10]   1.631995  0.78472 0.0526515   222.13  0.130607  1.631624
log.tau    0.635934  0.27423 0.0208941   172.26  0.084972  0.652988
Deviance 85.158210 10.31812 0.6065567   289.37 74.385690 83.157250
Monitor    1.979997  0.60232 0.0424282   201.53  1.005590  1.922430
Monitor    4.154517  0.21805 0.0113182   371.16  3.738744  4.164022
                UB
beta[1]    5.273698
beta[2]    0.311042
beta[3]    1.319463
beta[4]    1.257107
beta[5]    0.733734
beta[6]    0.094813
beta[7]    3.468211
beta[8]    1.578397
beta[9]    0.913065
beta[10]   3.199257
log.tau    1.167378
Deviance 110.720403
Monitor    3.331427
Monitor    4.584599
```

Several items are labeled as `NOT SHOWN HERE`, due to their size, such as the covariance matrix `Covar` or the stationary posterior samples `Posterior2`. As usual, these can be printed to the screen by appending a dollar sign, followed by the desired item, such as:

```
> Fit$Posterior2
```

Although a lot can be learned from the above output, notice that it completed 10000 iterations of 11 variables in 0.42 minutes. Of course this was fast, since there were only 39 records, and the form of the specified model was simple. As discussed later, Laplace's Demon does better than most other MCMC software with large numbers of records, such as 100,000.

In `R`, there is usually a `summary` function associated with each class of output object. The `summary` function usually summarizes the output. For example, with frequentist models, the `summary` function usually creates a table of parameter estimates, complete with p-values.

Since this is not a frequentist package, p-values are not part of any table, and the marginal posterior distributions of the parameters and other variables have already been summarized in `Fit`, there is no point to have an associated `summary` function. Going one more step

toward useability, `LaplacesDemon` has a `Consort` function, where the user consorts with Laplace's Demon about the output object. For example:

```
> Consort(Fit)
```

This produces two kinds of output. The first section is identical to `print(Fit)`, but by consorting with Laplace's Demon, it also produces a second section called `Demonic Suggestion`.

```
> Consort(Fit)

#############################################################
# Consort with Laplace's Demon                              #
#############################################################
Call:
LaplacesDemon(Model = Model, Data = MyData, Adaptive = 900, Covar = NULL,
    DR = 1, Initial.Values = Initial.Values, Iterations = 10000,
    Periodicity = 10, Status = 1000, Thinning = 10)

Acceptance Rate: 0.191
Adaptive: 900
Algorithm: Delayed Rejection Adaptive Metropolis
Covar: (NOT SHOWN HERE)
DIC of all samples (Dbar): 86.907
DIC of all samples (pD): 719.76
DIC of all samples (DIC): 806.67
DIC of stationary samples (Dbar): 85.158
DIC of stationary samples (pD): 53.232
DIC of stationary samples (DIC): 138.39
DR: 1
Iterations: 10000
Minutes of run-time: 0.42
Model: (NOT SHOWN HERE)
Monitor: (NOT SHOWN HERE)
Parameters (Number of): 11
Periodicity: 10
Posterior1: (NOT SHOWN HERE)
Posterior2: (NOT SHOWN HERE)
Recommended Burn-In of Thinned Samples: 301
Recommended Burn-In of Un-thinned Samples: 3010
Recommended Thinning: 270
Status is displayed every 1000 iterations
Summary1: (SHOWN BELOW)
Summary2: (SHOWN BELOW)
Thinned Samples: 1000
Thinning: 10
```

```
Summary of All Samples
              Mean       SD     MCSE Eff.Size         LB   Median
beta[1]   5.022744  0.29788 0.017538   288.49  4.7825736  5.04047
beta[2]  -0.449700  0.40129 0.025770   242.48 -1.2192368 -0.45520
beta[3]  -0.354680  0.96424 0.078525   150.78 -2.2047573 -0.31601
beta[4]  -0.092932  0.69866 0.051057   187.25 -1.4984657 -0.10772
beta[5]  -0.403650  0.55309 0.039374   197.32 -1.4227603 -0.45082
beta[6]  -0.472748  0.31433 0.019925   248.87 -1.0715801 -0.45303
beta[7]   2.221839  0.63308 0.053801   138.47  0.8041232  2.24437
beta[8]   0.603436  0.46815 0.030160   240.94 -0.4162341  0.61453
beta[9]  -0.180499  0.56446 0.035346   255.03 -1.3330820 -0.16966
beta[10]  1.555348  0.82088 0.063613   166.52 -0.1322433  1.58307
log.tau   0.620962  0.30480 0.023012   175.43  0.0068253  0.65190
Deviance 86.907163 37.94114 2.026465   350.54 74.3851161 83.12989
Monitor   1.955169  0.59055 0.038829   231.31  0.9898689  1.92412
Monitor   4.157063  0.30662 0.015166   408.75  3.7150806  4.17152
                UB
beta[1]    5.25693
beta[2]    0.36491
beta[3]    1.47275
beta[4]    1.27766
beta[5]    0.74765
beta[6]    0.14696
beta[7]    3.38278
beta[8]    1.48273
beta[9]    0.84930
beta[10]   3.06140
log.tau    1.12403
Deviance 113.01651
Monitor    3.23246
Monitor    4.63016


Summary of Stationary Samples
              Mean       SD      MCSE Eff.Size        LB    Median
beta[1]   5.039152  0.12158 0.0055652   477.26  4.784144  5.040819
beta[2]  -0.494324  0.39615 0.0274562   208.18 -1.269493 -0.496514
beta[3]  -0.454290  0.97750 0.0652184   224.64 -2.524612 -0.407956
beta[4]  -0.082261  0.69068 0.0497506   192.73 -1.503985 -0.051364
beta[5]  -0.421718  0.53882 0.0343869   245.53 -1.445971 -0.441650
beta[6]  -0.490788  0.31037 0.0220482   198.16 -1.112198 -0.463762
beta[7]   2.294935  0.55973 0.0332093   284.08  1.135636  2.263510
beta[8]   0.635673  0.47593 0.0330957   206.79 -0.336534  0.633569
beta[9]  -0.168345  0.55218 0.0340447   263.07 -1.307631 -0.169823
beta[10]  1.631995  0.78472 0.0526515   222.13  0.130607  1.631624
log.tau   0.635934  0.27423 0.0208941   172.26  0.084972  0.652988
Deviance 85.158210 10.31812 0.6065567   289.37 74.385690 83.157250
Monitor   1.979997  0.60232 0.0424282   201.53  1.005590  1.922430
Monitor   4.154517  0.21805 0.0113182   371.16  3.738744  4.164022
                UB
```

```
beta[1]     5.273698
beta[2]     0.311042
beta[3]     1.319463
beta[4]     1.257107
beta[5]     0.733734
beta[6]     0.094813
beta[7]     3.468211
beta[8]     1.578397
beta[9]     0.913065
beta[10]    3.199257
log.tau     1.167378
Deviance  110.720403
Monitor     3.331427
Monitor     4.584599

Demonic Suggestion

Due to the combination of the following conditions,

1. Delayed Rejection Adaptive Metropolis
2. The acceptance rate (0.191) is within the interval [0.15,0.5].
3. At least one target MCSE is >= 6.27% of its marginal posterior
   standard deviation.
4. Each target distribution has an effective sample size of
   at least 100.
5. Each target distribution became stationary by
   301 iterations.

Laplace's Demon has not been appeased, and suggests
copy/pasting the following R code into the R console,
and running it.

Initial.Values <- Fit$Posterior1[Fit$Thinned.Samples,]
Fit <- LaplacesDemon(Model, Data=MyData, Adaptive=0,
     Covar=Fit$Covar, DR=0, Initial.Values, Iterations=270000,
     Periodicity=0, Status=23810, Thinning=270)

Laplace's Demon is finished consorting.
```

The `Demonic Suggestion` is a very helpful section of output. When Laplace's Demon was developed initially in late 2010, there were not to my knowledge any tools of Bayesian inference that make suggestions to the user.

Before making its `Demonic Suggestion`, Laplace's Demon considers and presents five conditions: the algorithm, acceptance rate, MCSE, effective sample size, and stationarity. There are 48 combinations of these five conditions, though many combinations lead to the same conclusion. In addition to these conditions, there are other suggested values, such as a recommended number of iterations or values for the `Periodicity` and `Status` arguments. The suggested value for `Status` is seeking to print a status message every minute when the expected time is longer than a minute, and is based on the time in minutes it took, the

number of iterations, and the recommended number of iterations. This estimate is fairly accurate for non-adaptive algorithms, and is hard to estimate for adaptive algorithms. But, back to the really helpful part...

If these five conditions are not satisfactory, then Laplace's Demon is not appeased, and suggests it should continue updating, and that the user should copy/paste and execute its suggested `R` code. Here are the criteria it measures against. The final algorithm must be non-adaptive, so that the Markov Property holds (this is covered in the *Details* section). The acceptance rate is considered satisfactory if it is within the interval [15%,50%]. MCSE is considered satisfactory for each target distribution if it is less than 6.27% of the standard deviation of the target distribution. This allows the true mean to be within 5% of the area under a normal distribution around the estimated mean. The effective sample size is considered satisfactory for each target distribution if it is at least 100, which is usually enough to describe 95% probability intervals. And finally, each variable must be estimated as stationary.

Notice that since stationarity has been estimated beginning with the 301st iteration, the suggested `R` code changes from `Adaptive=900` to `Adaptive=0`. The suggestion is to abandon the adaptive MCMC algorithm in favor of a non-adaptive algorithm, specifically a Random-Walk Metropolis (RWM). It is also replacing the initial values with the latest values of the parameter chains, and is suggesting to begin with the latest covariance matrix. Some of the arguments in the suggested `R` code seem excessive, such as `Iterations=270000` and `Thinning=270`. For the sake of the example, the suggested `R` code will be run:

```
> Initial.Values <- Fit$Posterior1[Fit$Thinned.Samples, ]
> Fit <- LaplacesDemon(Model, Data = MyData, Adaptive = 0,
+     Covar = Fit$Covar, DR = 0, Initial.Values, Iterations = 270000,
+     Periodicity = 0, Status = 23970, Thinning = 270)

Laplace's Demon was called on Sat Jan  8 22:48:22 2011

Performing initial checks...
Adaptation will not occur due to the Adaptive argument.
Adaptation will not occur due to the Periodicity argument.
Algorithm: Random-Walk Metropolis

Laplace's Demon is beginning to update...
Iteration: 23970,   Proposal: Multivariate
Iteration: 47940,   Proposal: Multivariate
Iteration: 71910,   Proposal: Multivariate
Iteration: 95880,   Proposal: Multivariate
Iteration: 119850,   Proposal: Multivariate
Iteration: 143820,   Proposal: Multivariate
Iteration: 167790,   Proposal: Multivariate
Iteration: 191760,   Proposal: Multivariate
Iteration: 215730,   Proposal: Multivariate
Iteration: 239700,   Proposal: Multivariate
Iteration: 263670,   Proposal: Multivariate

Assessing Stationarity
```

```
Assessing Thinning and Effective Size
Creating Summaries
Creating Output

Laplace's Demon has finished.
```

Next, the user consorts with Laplace's Demon:

```
> Consort(Fit)

#############################################################
# Consort with Laplace's Demon                              #
#############################################################
Call:
LaplacesDemon(Model = Model, Data = MyData, Adaptive = 0, Covar = Fit$Covar,
    DR = 0, Initial.Values = Initial.Values, Iterations = 270000,
    Periodicity = 0, Status = 23970, Thinning = 270)

Acceptance Rate: 0.117
Adaptive: 270001
Algorithm: Random-Walk Metropolis
Covar: (NOT SHOWN HERE)
DIC of all samples (Dbar): 82.839
DIC of all samples (pD): 17.859
DIC of all samples (DIC): 100.70
DIC of stationary samples (Dbar): 82.839
DIC of stationary samples (pD): 17.859
DIC of stationary samples (DIC): 100.70
DR: 0
Iterations: 270000
Minutes of run-time: 3.32
Model: (NOT SHOWN HERE)
Monitor: (NOT SHOWN HERE)
Parameters (Number of): 11
Periodicity: 270001
Posterior1: (NOT SHOWN HERE)
Posterior2: (NOT SHOWN HERE)
Recommended Burn-In of Thinned Samples: 1
Recommended Burn-In of Un-thinned Samples: 270
Recommended Thinning: 270
Status is displayed every 23970 iterations
Summary1: (SHOWN BELOW)
Summary2: (SHOWN BELOW)
Thinned Samples: 1000
Thinning: 270


Summary of All Samples
              Mean      SD      MCSE Eff.Size         LB   Median
```

```
beta[1]   5.049414 0.11703 0.0037007   1000.0  4.8213813  5.04903
beta[2]  -0.456951 0.38159 0.0120668   1000.0 -1.1866647 -0.44209
beta[3]  -0.387225 0.91224 0.0288476   1000.0 -2.2137711 -0.37603
beta[4]  -0.088007 0.69464 0.0219666   1000.0 -1.4570598 -0.08442
beta[5]  -0.364783 0.51354 0.0162397   1000.0 -1.3624402 -0.35843
beta[6]  -0.493701 0.29823 0.0094307   1000.0 -1.0941874 -0.47841
beta[7]   2.263348 0.56054 0.0177259   1000.0  1.1800345  2.25029
beta[8]   0.627717 0.42579 0.0134647   1000.0 -0.2091316  0.62871
beta[9]  -0.204996 0.61125 0.0193293   1000.0 -1.4862898 -0.19935
beta[10]  1.553713 0.77324 0.0244520   1000.0 -0.0052482  1.56880
log.tau   0.663916 0.27930 0.0088322   1000.0  0.0901297  0.67766
Deviance 82.838804 5.97653 0.1911866    977.2 73.9313406 82.07288
Monitor   2.012494 0.55143 0.0174378   1000.0  1.0827196  1.95672
Monitor   4.184054 0.20695 0.0065444   1000.0  3.7786554  4.18750
               UB
beta[1]   5.286891
beta[2]   0.285416
beta[3]   1.346461
beta[4]   1.237535
beta[5]   0.667524
beta[6]   0.085168
beta[7]   3.421534
beta[8]   1.462383
beta[9]   0.972693
beta[10]  3.024874
log.tau   1.177234
Deviance 96.543775
Monitor   3.220192
Monitor   4.582045


Summary of Stationary Samples
             Mean      SD      MCSE Eff.Size          LB   Median
beta[1]   5.049414 0.11703 0.0037007   1000.0  4.8213813  5.04903
beta[2]  -0.456951 0.38159 0.0120668   1000.0 -1.1866647 -0.44209
beta[3]  -0.387225 0.91224 0.0288476   1000.0 -2.2137711 -0.37603
beta[4]  -0.088007 0.69464 0.0219666   1000.0 -1.4570598 -0.08442
beta[5]  -0.364783 0.51354 0.0162397   1000.0 -1.3624402 -0.35843
beta[6]  -0.493701 0.29823 0.0094307   1000.0 -1.0941874 -0.47841
beta[7]   2.263348 0.56054 0.0177259   1000.0  1.1800345  2.25029
beta[8]   0.627717 0.42579 0.0134647   1000.0 -0.2091316  0.62871
beta[9]  -0.204996 0.61125 0.0193293   1000.0 -1.4862898 -0.19935
beta[10]  1.553713 0.77324 0.0244520   1000.0 -0.0052482  1.56880
log.tau   0.663916 0.27930 0.0088322   1000.0  0.0901297  0.67766
Deviance 82.838804 5.97653 0.1911866    977.2 73.9313406 82.07288
Monitor   2.012494 0.55143 0.0174378   1000.0  1.0827196  1.95672
Monitor   4.184054 0.20695 0.0065444   1000.0  3.7786554  4.18750
               UB
beta[1]   5.286891
beta[2]   0.285416
```

```
beta[3]   1.346461
beta[4]   1.237535
beta[5]   0.667524
beta[6]   0.085168
beta[7]   3.421534
beta[8]   1.462383
beta[9]   0.972693
beta[10]  3.024874
log.tau   1.177234
Deviance 96.543775
Monitor   3.220192
Monitor   4.582045

Demonic Suggestion

Due to the combination of the following conditions,

1. Random-Walk Metropolis
2. The acceptance rate (0.117) is below 0.15.
3. Each target MCSE is < 6.27% of its marginal posterior
   standard deviation.
4. Each target distribution has an effective sample size of
   at least 100.
5. Each target distribution became stationary by
   1 iterations.

Laplace's Demon has not been appeased, and suggests
copy/pasting the following R code into the R console,
and running it.

Initial.Values <- Fit$Posterior1[Fit$Thinned.Samples,]
Fit <- LaplacesDemon(Model, Data=MyData, Adaptive=0,
     Covar=Fit$Covar, DR=1, Initial.Values, Iterations=270000,
     Periodicity=270, Status=81325, Thinning=270)

Laplace's Demon is finished consorting.
```

In 3.32 minutes, Laplace's Demon updated 270000 iterations, retaining every 270th iteration due to thinning, and reported an acceptance rate of 0.117, which is low enough to trigger a suggestion to continue updating. However, notice that all other criteria have been met: MCSE's are sufficiently small, effective size is sufficiently large, and stationarity was estimated beginning with the first iteration.

Notice `DR=1` in the suggested `R` code. Together with `Adaptive=0`, this indicates that a Delayed Rejection Metropolis (DRM) algorithm is suggested. DRM is discussed in the *Details* section, but here it is enough to state that in the case of a rejected proposal, the DRM algorithm will make a second proposal before simply moving on to the next iteration. This serves to increase the acceptance rate, and is obviously why Laplace's Demon suggests DRM.

The low acceptance rate suggests that the proposal distribution may not be optimal, and that the chains may not have mixed well. Even though Laplace's Demon has not been appeased due to the acceptance rate, everything else looks good. Since the algorithm was RWM, the Markov property holds, so let's look at some plots.

8 Plotting Output

Laplace's Demon has a `plot.demonoid` function to enable its own customized plots with `demonoid` objects. The variable `BurnIn` (below) may be left as it is so it will show only the stationary samples (samples that are no longer trending), or set equal to one so that all samples can be plotted. In this case, it will already be one, so I will leave it alone. The function also enables the user to specify whether or not the plots should be saved as a .pdf file, and allows the user to limit the number of parameters plotted, in case the number is very large and only a quick glance is desired.

```
> BurnIn <- Fit$Rec.BurnIn.Thinned
```

```
> plot(Fit, BurnIn, MyData, PDF = FALSE, Parms = Fit$Parameters)
```

There are 3 plots for each parameter, the deviance, and each monitored variable (which in this example are `tau` and `mu[1]`). The leftmost plot is a trace-plot, showing the history of the value of the parameter according to the iteration. The middlemost plot is a kernel density plot. The rightmost plot is an ACF or autocorrelation function plot, showing the autocorrelation at different lags. The chains look stationary (do not exhibit a trend), the kernel densities look Gaussian, and the ACF's show low autocorrelation.

If all is well, then the Markov chains should be studied with MCMC diagnostics, and finally, further assessments of model fit should be estimated with posterior predictive checks, showing how well (or poorly) the model fits the data. When the user is satisfied, marginal posterior samples may be used for inference.

When predicting the logarithm of `y` (Calories) with the `demonsnacks` data, the best fitting variables are `beta[6]` (Sodium), `beta[7]` (Total.Carbohydrate), and `beta[10]` (Protein). Overall, Laplace's Demon seems to have done well, eating `demonsnacks` for breakfast.

9 Posterior Predictive Checks

A posterior predictive check is a method to assess discrepancies between the model and the data (Gelman, Meng, & Stern, 1996). To perform posterior predictive checks with Laplace's Demon, simply use the `predict` function:

```
> Pred <- predict(Fit, Model, MyData)
```

This creates `Pred`, which is an object of class `demonoid.ppc` (where ppc is short for posterior predictive check) that is a list which contains `y` and `yhat`. If the data set that was used

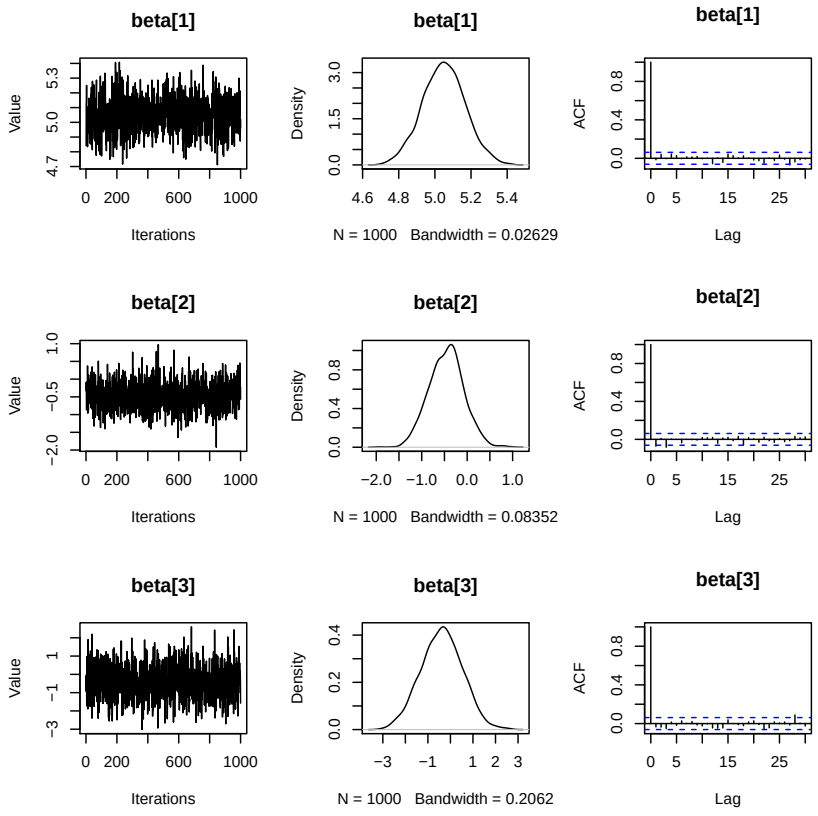

Figure 1: Plots of Marginal Posterior Samples

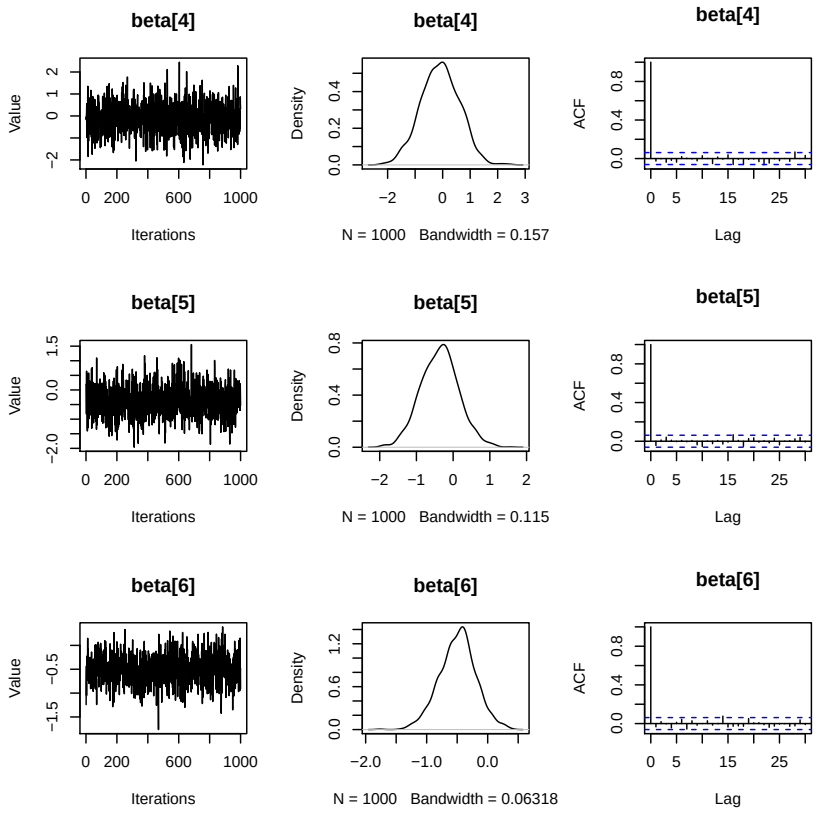

Figure 2: Plots of Marginal Posterior Samples

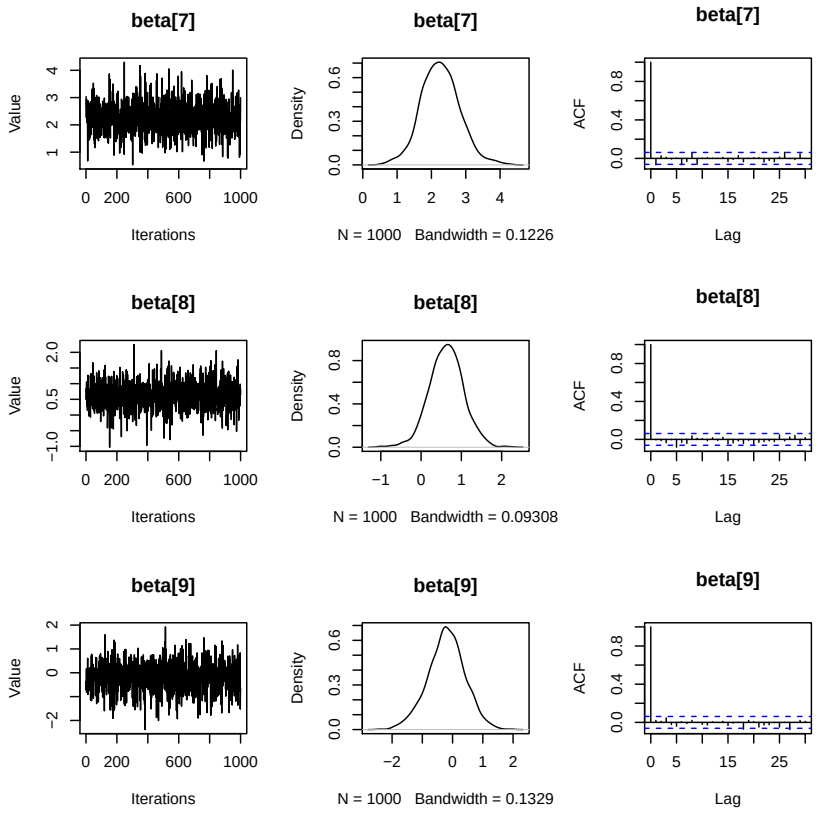

Figure 3: Plots of Marginal Posterior Samples

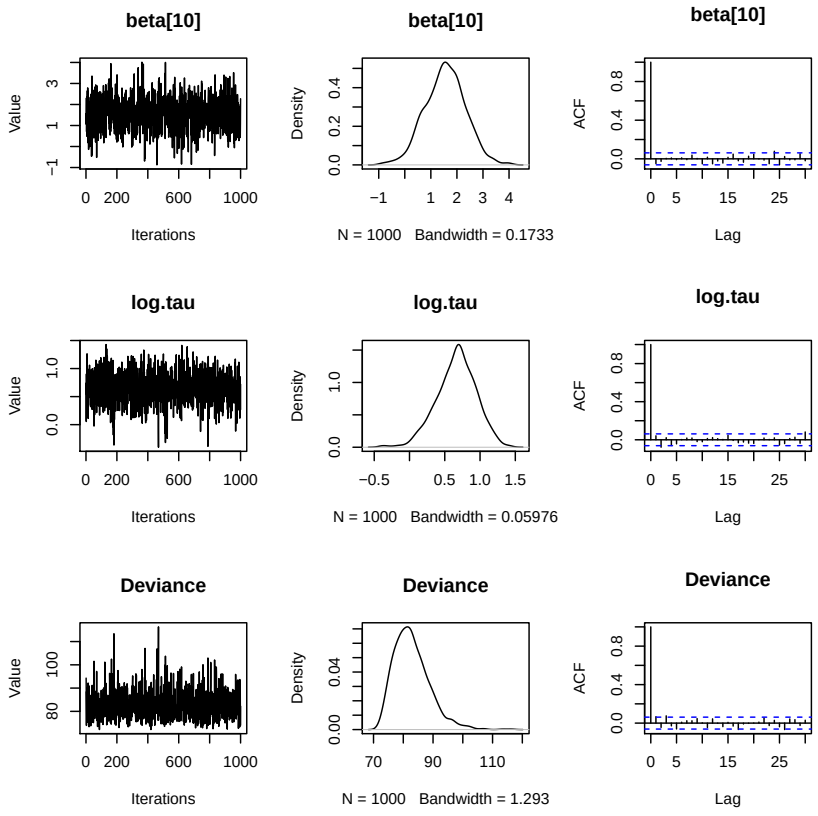

Figure 4: Plots of Marginal Posterior Samples

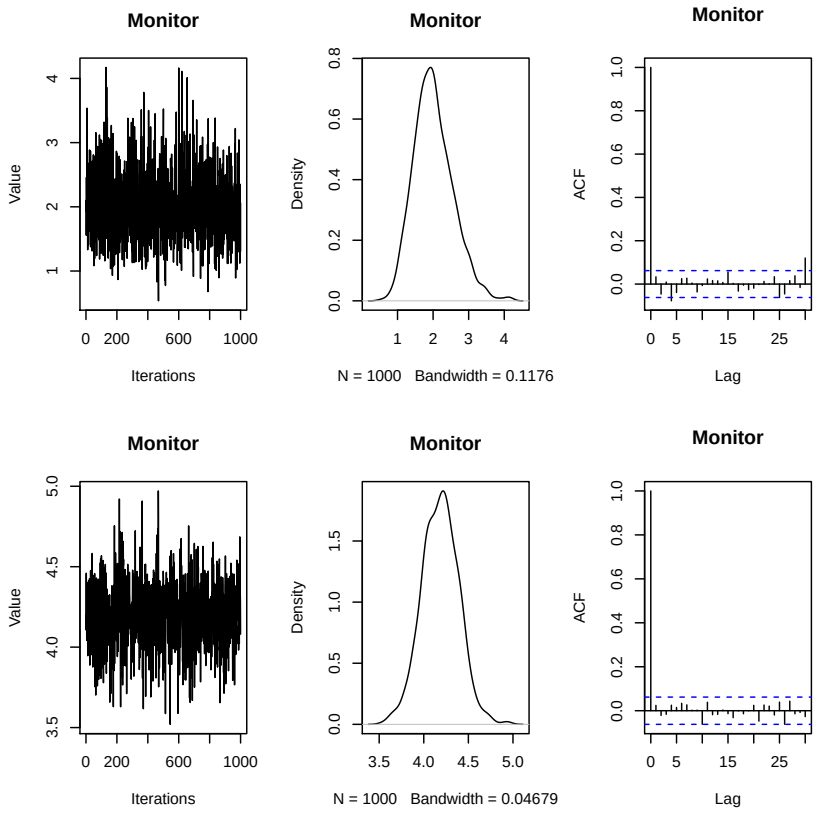

Figure 5: Plots of Marginal Posterior Samples

to estimate the model is supplied in `predict`, then replicates of `y` (also called y[rep]) are estimated. If a new data set is supplied in `predict`, then new, unobserved instances of `y` (called y[new]) are estimated. Note that with new data, a `y` vector must still be supplied, and if unknown, can be set to something sensible such as the mean of the `y` vector in the model.

The posterior predictive distribution may be summarized:

```
> summary(Pred)
```

The `summary` consists of: `Chi.Square` is a chi-square model fit statistic considering `y` and `yhat`. `Concordance` is a scalar value that indicates the percentage of records of `y` that are within the probability interval [2.5%,97.5%] of y[rep]. `Summary` is a n x 7 matrix with a row for each record of `y`, and 7 columns: y, Mean, SD, LB, Median, UB, and p.value. The `p.value` is the posterior predictive p-value, as described in Gelman et al. (1996). Lastly, `yhat` is a n x s matrix with a row for each record of `y` and a column for each sample of the marginal posterior samples.

The `predict` function calls the `Model` function once for each set of stationary samples in `Fit$Posterior2`. Each set of samples is used to calculate `mu`, which is the expectation of `y`, and `mu` is reported here as `yhat`. When there are few discrepancies between `y` and y[rep], the model is considered to fit the data well.

Since `Pred$yhat` is a large (39 x 1000) matrix, let's look at the summary of it:

```
> summary(Pred)

Chi-square:  10.514
Concordance:  0.79487
Records:
        y   Mean      SD     LB Median     UB p.value
1  4.1744 4.1858 0.20745 3.7666 4.1920 4.5825   0.528
2  5.3613 5.2842 0.41313 4.4357 5.2836 6.1401   0.422
3  6.0890 5.2860 0.54018 4.2064 5.2759 6.3511   0.073
4  5.2983 5.1487 0.33365 4.5135 5.1476 5.8109   0.323
5  4.4067 4.0780 0.25029 3.5634 4.0875 4.5620   0.095
6  2.1972 3.8151 0.20894 3.4179 3.8199 4.1925   1.000
7  5.0106 4.5501 0.19063 4.1550 4.5538 4.9156   0.007
8  1.6094 3.8739 0.20686 3.4748 3.8756 4.2477   1.000
9  4.3438 4.2327 0.24522 3.7474 4.2328 4.7040   0.310
10 4.8122 4.7294 0.23588 4.2739 4.7328 5.1903   0.365
11 4.1897 4.4224 0.20614 4.0342 4.4246 4.8335   0.871
12 4.9200 4.5453 0.18446 4.1925 4.5503 4.9075   0.021
13 4.7536 4.3881 0.18856 4.0157 4.3904 4.7433   0.021
14 4.1271 4.1687 0.18095 3.7941 4.1713 4.5310   0.598
15 3.7136 4.1051 0.20306 3.6997 4.1075 4.4819   0.969
16 4.6728 4.4048 0.23932 3.9246 4.4051 4.8544   0.120
17 6.9305 7.1952 0.55131 6.0893 7.1920 8.2542   0.691
18 5.0689 4.8039 0.25718 4.2872 4.8084 5.3145   0.150
19 6.7754 6.3296 0.49831 5.3259 6.3341 7.2686   0.184
```

```
20 6.5539 7.2003 0.49003 6.2070 7.2029 8.1268   0.905
21 4.8903 5.3860 0.34824 4.7289 5.3837 6.0568   0.926
22 4.4427 4.2681 0.28390 3.6997 4.2768 4.8475   0.268
23 2.8332 3.0960 0.49883 2.0924 3.1091 4.1072   0.719
24 4.7875 4.9456 0.25055 4.4570 4.9456 5.4365   0.735
25 6.9334 7.2636 0.63901 6.0545 7.2398 8.5084   0.689
26 6.1800 6.0313 0.62848 4.7884 6.0231 7.2665   0.402
27 5.6525 5.3098 0.30975 4.6709 5.3136 5.9039   0.122
28 5.4293 4.4772 0.21382 4.0358 4.4801 4.8793   0.000
29 5.6348 5.5100 0.72067 4.1228 5.5060 6.9243   0.419
30 4.2627 4.0762 0.21579 3.6299 4.0770 4.4659   0.202
31 3.8918 4.0767 0.26180 3.5731 4.0855 4.5707   0.763
32 6.6134 6.6024 0.40611 5.7858 6.6144 7.3666   0.502
33 4.9200 4.4179 0.19352 4.0450 4.4151 4.7964   0.005
34 6.5410 6.4162 0.48677 5.4232 6.4300 7.3295   0.406
35 6.3456 6.4251 0.49918 5.4736 6.4227 7.3687   0.565
36 3.7377 4.0564 0.26623 3.5152 4.0569 4.5845   0.890
37 7.3563 7.9073 0.64082 6.6483 7.9128 9.1841   0.810
38 5.7398 4.7739 0.17252 4.4214 4.7783 5.1074   0.000
39 5.5175 5.1398 0.27099 4.6408 5.1426 5.6803   0.083
```

`Chi.Square` can be interpreted as a traditional chi-square for model fit. As the value becomes larger, the model exhibits worse fit.

`Concordance` is a summary of the posterior predictive p-values for all elements of `y`, indicating a percentage of how many records had `y` within the 95% probability interval of `yhat`. In this case, roughly 1% of the time, `y` is within the 95% probability interval of yhat. All models are wrong, but some fit better than others. Every statistician should strive for the best fitting model, but a good and arbitrary goal is to attempt to achieve at least 95% concordance. This suggests that the model should be attempted again under different conditions, such as using different predictors, or specifying a different form to the model.

The last part of the summarized output reports `y`, information about the distribution of `yhat`, and the posterior predictive p-value. The mean prediction of `y[1]`, or yrep[1], given the model and data, is 4.186. Most importantly, `p.value[1]` is 0.528, indicating that 52.8% of the time, `yhat[1,]` was greater than `y[1]`, or that `y[1]` is close to the mean `yhat[1,]`. Contrast this with the 6th record, where `y[6]`=2.197 and `p.value[6]`=1. Therefore, `yhat[6,]` was not a good replication of `y[6]`, because the distribution of `yhat[6,]` is always greater than `y[6]`. While `y[6]` is 2.197, `yhat[6,]` has a mean of 3.815 and is within the 95% probability interval [3.418, 4.192]. Clearly, the 95% probability interval of `yrep[6,]` is above `y[6]` 100% of the time, indicating a strong discrepancy between the model and data, in this case.

The last part of this summary may be viewed graphically as well. Rather than observing plots for each of 39 records or rows, only the first 9 will be shown here:

```
> plot(Pred, Rows = c(1:9))
```

These posterior predictive checks indicate that there is plenty of room to improve this model.

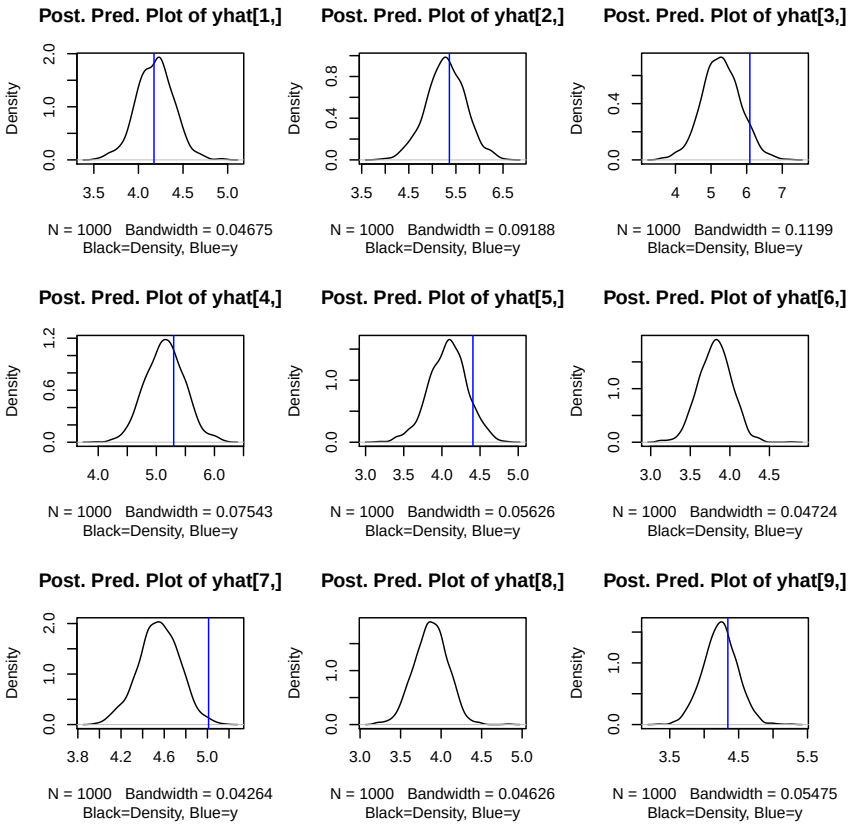

Figure 6: Plots of Marginal Posterior Samples

10 General Suggestions

Following are general suggestions on how best to use Laplace's Demon:

- As suggested by Gelman (2008), continuous predictors should be centered and scaled. Here is an explicit example in `R` of how to center and scale a single predictor called `x01`: `x01.cs <- (x01 - mean(x01)) / (2*sd(x01))`.
- Do not forget to reparameterize any bounded parameters in `Model` to be real-valued.
- MCMC is a stochastic method of numerical approximation, and as such, results may differ with each run due to the use of pseudo-random number generation. It is good practice to set a seed so that each update of the model may be reproduced. Here is an example in `R`: `set.seed(666)`.
- Once a model has been specified in `Model`, it may be tempting to specify a large number of iterations and thinning in `LaplacesDemon`, and simply let the model update a long time, hoping for convergence. Instead, it is wise to begin with few iterations such as `Iterations=20`, set `Adaptive=0` (preventing adaptation), and set `Thinning=1`. User-error in specifying the `Model` function will be frustrating otherwise.
- After studying updates with few iterations, the first "actual" update should be long enough that adaptation begins to occur, and that enough iterations occur after the first adaptation to allow the user to study the adaptation. In the supplied example, adaptation was allowed to begin at the 900th iteration (`Adaptive=900`), but also occurred with `Periodicity=10`, so every 10th iteration, adaptation occurred. It is also wise to use delayed rejection to assist with the acceptance rate when the algorithm may begin far from its solution, so set `DR=1`.
- If adaptation does not seem to improve estimation or the initial movement in the chains is worse than expected, then consider changing the initial values. Initial values are most effective when the starting points are close to the target distributions. When initial values are far enough away from the target distributions to be in low-probability regions, the algorithm may take longer than usual, and will struggle more as the proposal covariance matrix approaches near-singularity. If there is no information available to make a better selection, then randomize the initial values. Centered and scaled predictors also help by essentially standardizing the possible range of the target distributions.
- If Laplace's Demon exhibits an unacceptably low acceptance rate (say, arbitrarily, lower than 15%) and is having a hard time exploring after significant iterations, then investigate the latest proposal covariance matrix by entering `Fit$Covar`. Chances are that the diagonal, the variances, are large. In this case, it may be best to set `Covar=NULL` for the next time it continues to update, which will begin by default with a scaled identity matrix that should get more movement in the chains. As is usual practice, the latest sampled values should also replace the initial values, so it begins from the last point, but with larger proposal variances. The chains will mix better the closer they get to their target distributions. The user can confirm that Laplace's Demon is making progress and moving overall in the right direction by observing the trace-plot of the deviance. If it is decreasing run after run, then the model is continuously fitting better and better, and one sign of convergence will be when the deviance seems to become stationary or no longer shows a trend.

- `Demonic Suggestion` is intended as an aid, not an infallible replacement for critical thinking. As with anything else, its suggestions are based on assumptions, and it is the responsibility of the user to check those assumptions. For example, the `Geweke.Diagnostic` may indicate stationarity (lack of a trend) when it does not exist, and this most likely occurs when too few thinned samples remain. Or, the `Demonic Suggestion` may indicate that the next update may need to run for a million iterations in a complex model, requiring weeks to complete. Is this really best for the user?
- Use a two-phase approach with Laplace's Demon, where the first phase consists of using the AM or DRAM algorithm to achieve stationary samples that seem to have converged to the target distributions (convergence can never be determined with MCMC, but some instances of non-convergence can be observed). Once it is believed that convergence has occurred, continue Laplace's Demon with `Adaptive=0` so that adaptation will not occur. The final samples should again be checked for signs of non-convergence and, if satisfactory, used for inference.
- The desirable number of final, thinned samples for inference depends on the required precision of the inferential goal. A good general goal is to end up with 1,000 thinned samples, where the effective sample size is at least 100.
- Debates exist in MCMC literature as to whether to update one chain or multiple chains with different, randomized initial values. Laplace's Demon is not designed to simultaneously update multiple chains, because its algorithms are likely to produce multiple long chains, as opposed to Gibbs sampling, which should converge faster when its conditions are met. Therefore, one long chain is suggested here. Nonetheless, if multiple chains are desired, then Laplace's Demon can be updated a series of times, each beginning with different initial values, until multiple output objects of class `demonoid` exist with stationary samples, if time allows.

11 Independence and Observability

For the user, one set of advantages of Laplace's Demon compared to many other available methods is that it was designed with independence and observability in mind. By independence, it is meant that a goal was to minimize the dependence on other software. Laplace's Demon is performed completely within base `R` (though of course the `LaplacesDemon` package is required). From personal experience, I've used multiple packages to achieve goals before, and have been trapped when one of those packages failed to keep pace with other changes.

All functions in Laplace's Demon are written entirely in `R`, so the user can easily observe or manipulate the algorithm or functions. For example, to print the code for `LaplacesDemon` to the `R` console, simply enter:

```
> LaplacesDemon
```

12 Details

Laplace's Demon accomplishes numerical approximation with Markov chain Monte Carlo (MCMC) algorithms. There are a large number of MCMC algorithms, too many to review

here. Popular families (which are often non-distinct) include Gibbs sampling, Metropolis-Hastings, Random-Walk Metropolis (RWM), slice sampling, and many others, including hybrid algorithms. All MCMC algorithms are known as special cases of the Metropolis-Hastings algorithm (Metropolis, et al., 1953). Regardless of the algorithm, the goal in Bayesian inference is to maximize the joint posterior distribution and collect samples of the target distributions, which are marginal posterior distributions, later to be used for inference.

While designing Laplace's Demon, the primary goal in numerical approximation was generalization. Random-Walk Metropolis (RWM) is widely regarded as the most generalizable MCMC algorithm, though it requires that the proposal variance is tuned manually. Gibbs sampling requires conditional sampling of conjugate distributions, so it is precluded from non-conjugate sampling in its purest form. Slice sampling samples a distribution by sampling uniformly from the region under the plot of its density function. Therefore, it is more appropriate with bounded distributions that cannot approach infinity. There are valid ways to tune the RWM algorithm as it updates. This is known by many names, including adaptive Metropolis and adaptive MCMC, among others. Laplace's Demon uses the most generalizable family of MCMC algorithms: RWM and other modified forms of it.

12.1 Block Updating

Usually, there is more than one target distribution, in which case it must be determined whether it is best to sample from target distributions individually, in groups, or all at once. Block updating refers to splitting a multivariate vector into groups called blocks, so each block may be treated differently. A block may contain one or more variables. Advantages of block updating are that a different MCMC algorithm may be used for each block (or variable, for that matter), creating a more specialized approach, and the acceptance of a newly proposed state is likely to be higher than sampling from all target distributions at once in high dimensions. Disadvantages of block updating are that correlations probably exist between variables between blocks, and each block is updated while holding the other blocks constant, ignoring these correlations of variables between blocks. Without simultaneously taking everything into account, the algorithm may converge slowly or never arrive at the proper solution. Also, as the number of blocks increases, more computation is required, which slows the algorithm. In general, block updating allows a more specialized approach at the expense of accuracy, generalization, and speed. Laplace's Demon avoids block updating.

12.2 Random-Walk Metropolis

The basic idea of RWM is that each iterative estimate of a parameter is part of a changing state, and is influenced only by the previous state. The succession of states or iterations constitutes a Markov chain, where the current state is influenced only by the previous state. A proposed future estimate, called a proposal, is used to calculate the joint posterior density, and a ratio of the proposed to the current joint posterior density, called α, is compared to a random number drawn uniformly from the interval [0,1]. In practice, the log of the joint posterior density is used, so $\log(\alpha)$ is the proposal density minus the current density. The proposed state is accepted, replacing the current state with probability 1 when the proposed state is an improvement over the current state, and may still be accepted if the logarithm of a random draw from a uniform distribution is less than $\log(\alpha)$. Otherwise, the proposed state is rejected, and the current state is repeated so that another proposal may be estimated

at the next iteration. By comparing $\log(\alpha)$ to the log of a random number when $\log(\alpha)$ is not an improvement, random-walk behavior is included in the algorithm.

Random-walk behavior is desirable because it allows the algorithm to explore, and hopefully avoid getting trapped in undesirable regions. On the other hand, random-walk behavior is undesirable because it takes longer to converge to the target distribution while the algorithm explores. The algorithm generally progresses in the right direction, but may periodically wander away. Such exploration may uncover multi-modal target distributions, which other algorithms may fail to recognize, and then converge incorrectly. With enough iterations, RWM is guaranteed theoretically to converge to the correct target distribution, regardless of the starting point of each parameter, provided the proposal variance for each proposal is sensible.

Multiple parameters usually exist, and therefore correlations may occur between the parameters. All MCMC algorithms in Laplace's Demon are modified to attempt to estimate multivariate proposals, thereby taking correlations into account through a covariance matrix. If a failure is experienced in attempting to estimate multivariate proposals, then Laplace's Demon temporarily resorts to independent proposals by estimating univariate variances, and will continue to attempt to return to multivariate proposals at each iteration.

Throughout the RWM algorithm, the proposal covariance or variance remains fixed. The user may enter a vector of proposal variances or a proposal covariance matrix, and if neither is supplied, then Laplace's Demon estimates both before it begins, based on the number of variables.

The acceptance or rejection of each proposal should be observed at the completion of the RWM algorithm as the acceptance rate, which is the number of acceptances divided by the total number of iterations. If the acceptance rate is too high, then the proposal variance or covariance is too small. In this case, the algorithm will take longer than necessary to find the target distribution and the samples will be highly autocorrelated. If the acceptance rate is too low, then the proposal variance or covariance is too large, and the algorithm is ineffective at exploration. In the worst case scenario, no proposals are accepted and the algorithm fails to move. Under theoretical conditions, the optimal acceptance rate for a sole, independent and identically distributed (IID), Gaussian, marginal posterior distribution is 0.44 or 44%. The optimal acceptance rate for an infinite number of distributions that are IID and Gaussian is 0.234 or 23.4%.

12.3 Delayed Rejection Metropolis

The Delayed Rejection Metropolis (DRM or DR) algorithm is a RWM with one, small twist. Whenever a proposal is rejected, the DRM algorithm will try one or more alternate proposals, and correct for the probability of this conditional acceptance. By delaying rejection, autocorrelation in the chains may be decreased, and the algorithm is encouraged to move. Currently, Laplace's Demon will attempt one alternate proposal when using the DRAM (see below) or DRM algorithm. The additional calculations may slow each iteration of the algorithm in which the first set of proposals is rejected, but it may also converge faster. For more information on DRM, see Mira (2001).

DRM may be considered to be an adaptive MCMC algorithm, because it adapts the proposal based on a rejection. Here, I have to admit that I don't know if DRM violates the Markov

property (see below). For the purposes of Laplace's Demon, DRM is not considered to be an adaptive MCMC algorithm, because it is not adapting to the target distribution, other than making more attempts with the same type of algorithm. However, if the DRM is considered to violate the Markov property (again, see below), then its use will be changed here, accordingly. DRM is rarely suggested by Laplace's Demon, though the combination of DRM and AM, called DRAM, is suggested frequently.

12.4 Adaptive Metropolis

In traditional, non-adaptive RWM, the Markov property is satisfied, creating valid Markov chains, but it is difficult to manually optimize the proposal variance or covariance, and it is crucial that it is optimized for good mixing of the Markov chains. Adaptive MCMC may be used to automatically optimize the proposal variance or covariance based on the history of the chains, though this violates the Markov property, which declares the proposed state is influenced only by the current state. To retain the Markov property, and therefore valid Markov chains, a two-phase approach may be used, in which adaptive MCMC is used in the first phase to arrive at the target distributions while violating the Markov property, and non-adaptive DRM or RWM is used in the second phase to sample from the target distributions for inference, while possessing the Markov property.

There are too many adaptive MCMC algorithms to review here. All of them adapt the proposal variance to improve mixing. Some adapt the proposal variance to also optimize the acceptance rate (which becomes difficult as dimensionality increases), minimize autocorrelation, or optimize a scale factor. Laplace's Demon uses a variation of the Adaptive Metropolis (AM) algorithm of Haario, Saksman, and Tamminen (2001).

Given the number of dimensions (d) or parameters, the optimal scale of the proposal variance, also called the jumping kernel, has been reported as $\texttt{2.4}/\sqrt{d}$ based on the asymptotic limit of infinite-dimensional Gaussian target distributions that are independent and identically-distributed (Gelman, Roberts, and Gilks, 1996). In applied settings, each problem is different, so the amount of correlation varies between variables, target distributions may be non-Gaussian, the target distributions may be non-IID, and the scale should be optimized. Laplace's Demon uses a scale that is accurate to more decimals: $2.381204/\sqrt{d}$, even though Haario et al. use a different form: $2.4^2/d$. There are algorithms in statistical literature that attempt to optimize this scale, and it is hoped that these algorithms will be included in Laplace's Demon in the future.

Haario et al. (2001) tested their algorithm with up to 200 dimensions or parameters, so it is capable of large-scale Bayesian inference. The version in Laplace's Demon should be capable of more dimensions than the AM algorithm as it was presented, because when Laplace's Demon experiences an error in multivariate AM, it defaults to independent adaptive proposals. Although independent adaptive proposals should take longer to converge, the algorithm is limited in dimension only by the RAM of the computer.

For multivariate adaptive tuning, the formula across `K` parameters and `t` iterations is:

$$\Sigma^* = (\phi_K cov(\theta_{[1:t,1:K]})) + (\phi_K 1.0E-5\mathbf{I}_K)$$

where ϕ_K is the scale according to `K` parameters, 1.0E-5 is a small constant to ensure the proposal covariance matrix is positive definite (does not have zero or negative variance on

the diagonal), and $\mathbf{I}_K$ is a K x K identity matrix. The initial proposal covariance matrix, when none is provided, defaults to the scaling component multiplied by its identity matrix: $phi_K\mathbf{I}_K$.

For independent adaptive tuning, the formula across K parameters and t iterations is:

$$\sigma^{*2}_k = \phi_k var(\theta_{[1:t,k]}) + \phi_k 1.0E-5$$

Each element in the initial vector of proposal variances is set equal to the asymptotic scale according to its dimensions: ϕ_k.

In both the multivariate and independent cases, the AM algorithm begins with a fixed proposal variance or covariance that is either estimated internally or supplied by the user. Next, the algorithm begins, and it does not adapt until the iteration is reached that is specified by the user in the `Adaptive` argument of `LaplacesDemon`. Then, the algorithm will adapt with every `n` iterations according to the `Periodicity` argument. Therefore, the user has control over when the AM algorithm begins to adapt, and how often it adapts. The value of the `Adaptive` argument in Laplace's Demon is chosen subjectively by the user according to their confidence in the accuracy of the initial proposal covariance or variance. The value of the `Periodicity` argument is chosen by the user according to their patience: when the value is 1, the algorithm will adapt continuously, which will be slower to calculate. The AM algorithm adapts the proposal covariance or variance according to the observed covariance or variance in the entire history of all parameter chains, as well as the scale factor.

As recommended by Haario, et al. (2001), there are two tricks that may be used to assist the AM algorithm in the beginning. Although Laplace's Demon does not use the suggested "greedy start" method, it uses the second suggested trick of shrinking the proposal as long as the acceptance rate is less than 5%. Haario, et al. (2001) suggest loosely that if "it has not moved enough during some number of iterations, the proposal could be shrunk by a constant factor". For each iteration that the acceptance rate is less than 5% and that the AM algorithm is used but the current iteration is prior to adaptation, Laplace's Demon multiplies the proposal covariance or variance by 99%. Over pre-adaptive time, this encourages a smaller proposal covariance or variance to increase the acceptance rate so that when adaptation begins, the observed covariance or variance of the chains will not be constant, and then shrinkage will cease and adaptation will take it from there.

12.5 Delayed Rejection Adaptive Metropolis

The Delayed Rejection Adaptive Metropolis (DRAM) algorithm is merely the combination of both DRM (or DR) and AM (Haario, Laine, Mira, and Saksman, 2006). DRAM has been demonstrated as robust in extreme situations where DRM or AM fail separately. Haario, et al. (2006) present an example involving ordinary differential equations in which least squares could not find a stable solution, and DRAM did well.

12.6 Afterward

Once the model is updated, the `Geweke.Diagnostic` function of Geweke (1992) is iteratively applied to successively smaller tail-sections of the thinned samples to assess stationarity (or lack of trend). When all parameters are estimated as stationary beyond a given iteration, the previous iterations are suggested to be considered as burn-in and discarded. The number of thinned samples is divided into cumulative 10% groups, and the `Geweke.Diagnostic` is applied by beginning with each cumulative group.

There is debate in statistical literature over the importance of Monte Carlo Standard Error (MCSE). Here, it is considered important enough to be one of five main criteria to appease Laplace's Demon. Most literature recommends using one of several competing batch methods to estimate MCSE, arguing that the simple method (`MCSE =` $\sigma/\sqrt{m}$) is biased and reports less error (where `m` is the effective sample size). I have calculated both the simple method and non-overlapping batch MCSE's on a wide range of applied models, and noted just as many cases of the simple method producing higher MCSE's as lower MCSE's. As far as Laplace's Demon is concerned, the simple method is used to estimate MCSE, but it is open to debate.

13 Software Comparisons

There is now a wide variety of software to perform MCMC for Bayesian inference. Perhaps the most common is BUGS, which is an acronym for Bayesian Using Gibbs Sampling (Lunn, Spiegelhalter, Thomas, and Best, 2009). BUGS has several versions. A popular variant is JAGS, which is an acronym for Just Another Gibbs Sampler (Plummer, 2003). The only other comparisons made here are with some `R` packages (`AMCMC`, `mcmc`, `MCMCpack`, and `UMACS`) and SAS. Many other `R` packages use MCMC, but are not intended as general-purpose MCMC software. Hopefully I have not overlooked any general-purpose MCMC packages in `R`.

WinBUGS has been the most common version of BUGS, though it is no longer developed. BUGS is an intelligent MCMC engine that is capable of numerous MCMC algorithms, but prefers Gibbs sampling. According to its user manual, WinBUGS 1.4 uses Gibbs sampling with full conditionals that are continuous, conjugate, and standard. For full conditionals that are log-concave and non-standard, derivative-free Adaptive Rejection Sampling (ARS) is used. Slice sampling is selected for non-log-concave densities on a restricted range, and tunes itself adaptively for 500 iterations. Seemingly as a last resort, an adaptive MCMC algorithm is used for non-conjugate, continuous, full conditionals with an unrestricted range. The standard deviation of the Gaussian proposal distribution is tuned over the first 4,000 iterations to obtain an acceptance rate between 20% and 40%. Samples from the tuning phases of both Slice sampling and adaptive MCMC are ignored in the calculation of all summary statistics, although they appear in trace-plots.

The current version of BUGS, OpenBUGS, allows the user to specify an MCMC algorithm from a long list for each parameter (Lunn et al., 2009). This is a step forward, overcoming what is perceived here as an over-reliance on Gibbs sampling. However, if the user does not customize the selection of the MCMC sampler, then Gibbs sampling will be selected for full conditionals that are continuous, conjugate, and standard, just as with WinBUGS.

Based on years of almost daily experience with WinBUGS and JAGS, which are excellent software packages for Bayesian inference, Gibbs sampling is selected too often in these automatic, MCMC engines. A suggestion for BUGS and JAGS would be to attempt Gibbs sampling and abandon it if correlations are too high. An advantage of Gibbs sampling is that the proposals are accepted with probability 1, so convergence may be faster. Unfortunately, Gibbs sampling is not as generalizable, because it can function only when certain conjugate distributional forms are known *a priori*. Moreover, Gibbs sampling was avoided for Laplace's Demon because it doesn't perform well with correlated variables or parameters, which usually exist, and I have been bitten by that *bug* many times.

The BUGS and JAGS families of MCMC software are excellent. BUGS is capable of several things that Laplace's Demon is not. For example, BUGS automatically handles missing values in the dependent variable, where Laplace's Demon requires specifications for each one in the `Model` function. BUGS also allows the user to specify the model graphically as a directed acyclic graph (DAG) in Doodle BUGS. Lastly, many textbooks in several fields have been written that are full of WinBUGS examples.

The four MCMC algorithms in Laplace's Demon are generalizable, and generally robust to correlation between variables or parameters. The disadvantage is slower convergence when correlations are low. The advantages, however, are faster convergence when correlations are high, and more confidence in the results.

At the time this article was written, the `AMCMC` package in `R` is unavailable on CRAN, but may be downloaded from the author's website. This download is best suited for a Linux, Mac, or UNIX operating system, because it requires the `gcc` C compiler, which is unavailable in Windows. It performs adaptive Metropolis-within-Gibbs (Roberts and Rosenthal, 2007), and uses C language for significantly faster sampling. Metropolis-within-Gibbs is not as generalizable as adaptive MCMC. Otherwise, if the user wishes to see the code of the `AMCMC` sampler, then the user must also be familiar with C language.

Also in `R`, the `mcmc` package offers RWM with multivariate Gaussian proposals and allows batching, as well as a simulated tempering algorithm, but it does not have any adaptive algorithms.

The `MCMCpack` package in `R` takes a canned-function approach to RWM, which is convenient if the user needs the specific form provided, but is otherwise not generalizable. General-purpose RWM is included, but adaptive algorithms are not. It also offers the option of Laplace Approximation to optimize initial values, though the algorithm is evaluated in `optim`, which has not performed well in my testing of Laplace Approximations.

At the time this article was written, Gelman's `UMACS` package has been removed from CRAN. It became outdated due to lack of interest, and did not include an adaptive MCMC algorithm, as far as I know.

In SAS 9.2, an experimental procedure called PROC MCMC has been introduced. It is undeniably a rip-off of BUGS (including its syntax), though OpenBUGS is much more powerful, tested, and generalizable. Since SAS is proprietary, the user cannot see or manipulate the source code, and should expect much more from it than OpenBUGS or any open-source software, given the absurd price.

14 Large Data Sets and Speed

An advantage of Laplace's Demon compared to other MCMC software is that the model is specified in a way that takes advantage of `R`'s vectorization. BUGS and JAGS, for example, require models to be specified so that each record of data is processed one by one inside a 'for loop', which significantly slows updating with larger data sets. In contrast, Laplace's Demon avoids 'for loops' wherever possible. For example, a data set of 100,000 rows and 16 columns (the dependent variable, a column vector of 1's for the intercept, and 14 predictors) was updated 1,000 times with `Adaptive=2`, `DR=0`, and `Periodicity=10` in 1.55 minutes by Laplace's Demon, according to a simple, linear regression[2]. It was nowhere near convergence, but try to run 100,000 rows of comparable data for 1,000 iterations in BUGS or JAGS, and tell me how long it took!

However, with small data sets, other MCMC software (`AMCMC` is a good example) can be faster than Laplace's Demon, if it is programmed in a faster language such as Component Pascal, C, or C++. I have not studied all MCMC algorithms in `R`, but most are probably programmed in C and called from `R`. And Laplace's Demon could be much faster if programmed in C as well.

When the non-adaptive algorithm updates in Laplace's Demon, the expected speed of an iteration should not differ depending on how many iterations it has previously updated. However, the adaptive algorithm will slow as iterations are updated, because each time it adapts, it is adapting to the covariance of the entire history of the chains. As the history increases, the calculations take longer to complete, and the expected speed of an adaptive iteration decreases, compared to earlier adaptive iterations. If time is of the essence and the algorithm needs to be adaptive, then it may be best to make multiple, shorter updates in place of one, longer update.

15 Future Goals

Laplace's Demon is useful software for Bayesian inference. Nonetheless, there are several future goals to improve it. There are a bewildering number of methods for numerical approximation. For example, not only are there a large number of MCMC algorithms, some of which are newer and take additional things into account, there are other methods of approximation, including Laplace Approximation (also called Laplace's Method), Expectation Maximization (EM) and Variational Bayes, Approximate Bayesian Computation (ABC), and other methods such as iterative quadrature.

Currently, Laplace's Demon offers custom variations of four MCMC algorithms. It is likely that future versions of Laplace's Demon will include other MCMC algorithms for the user to select. Additionally, a deterministic algorithm such as Laplace Approximation may become optional to initially approximate the solution prior to using MCMC, thus speeding convergence.

To this end, I have not yet found a generalizable Laplace Approximation algorithm. Laplace Approximation is available as the `laplace` function in the `LearnBayes` package, and also as an option in the `MCMCpack` package for some MCMC models. Both functions are solved

[2]These updates were performed on a 2010 System76 Pangolin Performance laptop with 64-bit Debian Linux and 8GB RAM.

with the `optim` function of base `R`, and both exhibit error that increases unacceptably with the number of parameters, usually beyond 6. Using the `BB` package for optimization, I have been able to accurately estimate approximately 60 parameters in a simple model.

It is possible that the algorithms may also be included in C language, so the user can select whichever language is preferred, probably C for speed. Last but not least, I'm sure my `R` code could be more efficient.

Please send all constructive criticism and reports of software bugs to `statisticat@gmail.com`.

16 References

Bayes, T. and Price, R. (1763). An Essay towards solving a Problem in the Doctrine of Chance. By the late Rev. Mr. Bayes, communicated by Mr. Price, in a letter to John Canton, M. A. and F. R. S. Philosophical Transactions of the Royal Society of London, 53, p. 370–418.

Crawley, M.J. (2007). The R Book. John Wiley & Sons Ltd: West Sussex, England.

Gelman, A. (2008). Scaling Regression Inputs by Dividing by Two Standard Deviations. Statistics in Medicine, 27, p. 2865–2873.

Gelman, A., Carlin, J.B., Stern, H.S., and Rubin, D. (2004). Bayesian Data Analysis, Second Edition. Chapman & Hall: Boca Raton, FL.

Gelman, A., Meng, X.L., and Stern, H. (1996). Posterior Predictive Assessment of Model Fitness via Realized Discrepancies. Statistica Sinica, 6, p. 733–807.

Gelman, A., Roberts, G.O., and Gilks, W.R. (1996). Efficient Metropolis Jumping Rules. Bayesian Statistics, 5, p. 599–608.

Geweke, J. (1992). Evaluating the Accuracy of Sampling-Based Approaches to the Calculation of Posterior Moments. Bayesian Statistics, 4, p. 1–31.

Haario, H., Laine, M., Mira, A., and Saksman, E. (2006). DRAM: Efficient Adaptive MCMC. Statistical Computing, 16, p. 339–354.

Haario, H., Saksman, E., and Tamminen, J. (2001). An Adaptive Metropolis Algorithm. Bernouli, 7(2), p. 223–242.

Laplace, P.S. (1774). Memoire sur la probabilite des causes par les evenements. Mem. Acad. Sci. Paris, 6, 621–656. English translation in 1986 as "Memoir on the probability of the causes of events", Statist. Sci., 1, p. 359–378.

Laplace, P.S. (1812). Theorie Analytique des Probabilites. Paris: Courcier. Reprinted as Oeuvres Completes de Laplace, 7, 1878–1912. Paris: Gauthier-Villars.

Laplace, P.S. (1814). Essai philosophique sur les probabilites. English translation in 1902 as "A Philosophical Essay on Probabilities".

Lunn, D., Spiegelhalter, D., Thomas, A., and Best, N. (2009). The BUGS project: Evolution, critique, and future directions. Statistics in Medicine, 28, p. 3049–3067.

Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., and Teller, E. (1953). Equation of

State Calculations by Fast Computing Machines. Journal of Chemical Physics, 21, p. 1087–1092.

Mira, A. (2001). On Metropolis-Hastings Algorithms with Delayed Rejection. Metron, Vol. LIX, n. 3–4, p. 231–241.

Plummer, M. (2003). JAGS: A Program for Analysis of Bayesian Graphical Models Using Gibbs Sampling. Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003), March 20–22, Vienna, Austria. ISBN 1609–395X.

R Development Core Team (2010). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3–900051–07–0, URL `http://www.R-project.org/`.

Roberts, G.O., and Rosenthal, J.S. (2007). Examples of Adaptive MCMC. Comp. Stat. & Data Anal., 51, p. 5467–5470.