LaplacesDemon: An R Package for Bayesian Inference

Byron Hall
STATISTICAT, LLC

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 Laplace Approximation and 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, Delayed Rejection, DR, DRAM, DRM, Gradient Ascent, Laplace Approximation, LaplacesDemon, Laplace's Demon, Markov chain Monte Carlo, MCMC, Metropolis, Optimization, R, Random Walk, Random-Walk, STATISTICAT.

Bayesian inference is named after Reverend Thomas Bayes (1702-1761) for developing Bayes' theorem, which was published posthumously after his death (Bayes and Price 1763). This was the first instance of what would be called inverse probability[1].

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 introduced a host of new ideas and mathematical techniques in his book, *Theorie Analytique des Probabilites*, (Laplace 1812). 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 (Laplace 1814). In it, the

[1]'Inverse probability' refers to assigning a probability distribution to an unobserved variable, and is in essence, probability in the opposite direction of the usual sense. Bayes' theorem has been referred to as "the principle of inverse probability". Terminology has changed, and the term 'Bayesian probability' has displaced 'inverse probability'. The adjective "Bayesian" was introduced by R. A. Fisher as a derogatory term.

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 (R Development Core Team 2010) package called **LaplacesDemon** (Hall 2011), which was designed without consideration for hard determinism, but instead with a lofty goal toward facilitating high-dimensional Bayesian (or Laplacian) inference[2], 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, and explains future goals.

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 the vignette entitled "Bayesian Inference" that comes with the **LaplacesDemon** package, Gelman, Carlin, Stern, and Rubin (2004), and Crawley (2007).

1. 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)
```

[2]Even though the **LaplacesDemon** package is dedicated to Bayesian inference, frequentist inference may be used instead with the same functions by omitting the prior distributions and maximizing the likelihood.

```
LaplacesDemon: Software for Bayesian Inference

``Probability theory is nothing but common sense reduced to
calculation'' (Pierre-Simon Laplace, 1814).

Laplace's Demon is ready for you.
```

2. 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] <- CenterScale(X[, j])
+ }
> mon.names <- c("LP", "tau")
> 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, mon.names = mon.names,
+     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). Laplace's Demon provides a `CenterScale` function to center and scale predictors[3].

Laplace's Demon will consider using Laplace Approximation, and part of this consideration includes determining the sample size. The user must specify the number of observations in the data as either a scalar `n` or `N`. If these are not found by the `LaplaceApproximation` or `LaplacesDemon` functions, then it will attempt to determine sample size as the number of rows in `y` or `Y`.

[3]Centering and scaling a predictor is `x.cs <- (x - mean(x)) / (2*sd(x))`.

3. Specifying a Model

Laplace's Demon is capable with any Bayesian model for which the likelihood is specified[4]. To use Laplace's Demon, the user must specify a model. Let's consider a linear regression model, which is often denoted as:

$$y \sim N(\mu, \sigma^2)$$
$$\mu = \mathbf{X}\beta$$

The dependent variable, y, is normally distributed according to expectation vector μ and scalar variance σ^2, and expectation vector μ is equal to the inner product of design matrix $\mathbf{X}$ and parameter vector β.

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:

$$\beta_j \sim N(0, 1000), \quad j = 1, \ldots, J$$
$$\tau \sim \Gamma(0.001, 0.001)$$

Each of the J β parameters is assigned an uninformative prior probability distribution that is normally-distributed according to $\mu = 0$ and $\sigma^2 = 1000$, where the precision is $\tau = 0.001$. The large variance or small precision indicates a lot of uncertainty about each β, and is hence a uninformative distribution. The residual precision τ is gamma-distributed according to two parameters of its distribution: $\alpha = 0.001$ and $\beta = 0.001$.

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

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

[4]Examples of numerous Bayesian models may be found in the "Examples" vignette that comes with the **LaplacesDemon** package.

Laplace's Demon iteratively maximizes the logarithm of the unnormalized joint posterior density as specified in this `Model` function. In Bayesian inference, the logarithm of the unnormalized joint posterior density is proportional to the sum of the log-likelihood and logarithm 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, $\propto$ means 'proportional to'[5], $p(\Theta|y)$ 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 logarithm of the unnormalized 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 a list of data. These arguments are specified in the beginning of the function:

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

Then, the `Model` function is evaluated and the logarithm of the unnormalized 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, y^{rep} (`yhat`) or replicates of y, and the parameter vector `parm`. 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(LP,tau),
    yhat=mu, parm=parm)
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 column vectors (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,Data$J)
beta.tau <- rep(1.0E-3,Data$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.

[5]For those unfamiliar with $\propto$, this symbol simply means that two quantities are proportional if they vary in such a way that one is a constant multiplier of the other. This is due to an unspecified constant of proportionality in the equation. Here, this can be treated as 'equal to'.

```
beta <- parm[1:Data$J]
tau <- exp(parm[Data$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 <- dnorm(beta, beta.mu, 1/sqrt(beta.tau), 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 allowed to range from $-\infty$ to ∞, and be transformed in the `Model` function with the `exp` function, which will force it to positive values. A parameter θ that needs to be bounded in the model, such as in the interval [1,5], can be transformed to that range with a logistic function, such as $1+4[\exp(\theta)/(\exp(\theta)+1)]$. Alternatively, each parameter may be constrained in the `Model` function. 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[Data$J+1])
```

Laplace's Demon will increase or decrease `parm[Data$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 logarithm of the unnormalized joint posterior density. The expectation vector `mu` is the inner product (`%*%`) of the vector `beta` and the transposed design matrix, `t(Data$X)`. Expectation vector `mu`, vector `Data$y`, and scalar `tau` are used to estimate the sum of the log-likelihoods, where:

$$y \sim N(\mu, \tau^{-1})$$

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

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

```
mu <- beta %*% t(Data$X)
LL <- sum(dnorm(Data$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 the `Model` function so that an initial value is assigned.

4. Initial Values

Laplace's Demon requires a vector of initial values for the parameters. Each initial value is a starting point for the estimation of a parameter. When all initial values are set to zero, Laplace's Demon will optimize initial values using a step-adaptive gradient ascent algorithm in the `LaplaceApproximation` function. Laplace Approximation is asymptotic with respect to sample size, so it is inappropriate in this example with a sample size of 39 and 11 parameters. Laplace's Demon will not use Laplace Approximation when the sample size is not at least five times the number of parameters. Otherwise, the user may prefer to optimize initial values in the `LaplaceApproximation` function before using the `LaplacesDemon` function. When Laplace's Demon receives initial values that are not all set to zero, it will begin to update each parameter.

In this example, there are 11 parameters. With no prior knowledge, it is a good idea either to randomize each initial value within an interval, say -3 to 3, or set all of them equal to zero and let the `LaplaceApproximation` function optimize the initial values, provided there is sufficient sample size. Here, the `LaplaceApproximation` function will be introduced in the `LaplacesDemon` function, so 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`, and is merely a personal preference. 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 the `Model` function.

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

5. Laplace's Demon

Compared to specifying the model in the `Model` function, 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[6]:

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

As with any R package, the user can learn about a function by using the `help` function and including the name of the desired function. To learn the details of the **LaplacesDemon** function, enter:

```
> help(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)
```

[6]Demonic references are used only to add flavor to the software and its use, and in no way endorse beliefs in demons. This specific pseudo-random seed is often referred to, jokingly, as the 'demon seed'.

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 Laplace Approximation and four MCMC algorithms (which are explained in section 11). The above example did not use Laplace Approximation due to small sample size, and instead used the Delayed Rejection Adaptive Metropolis (DRAM) algorithm for updating.

This example tells the **LaplacesDemon** function to maximize the first component in the list output from the user-specified `Model` function, given a data set called `Data`, 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 the `LaplacesDemon` function 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 `n`th 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 Sun Mar  6 22:12:55 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 ESS
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 single-component proposals. The output object, `Fit`, was created as a list. As with any R object, use `str()` to examine its structure:

```
> str(Fit)
```

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

```
> Fit$Acceptance.Rate

[1] 0.3158
```

6. 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: 0.3158
Adaptive: 900
Algorithm: Delayed Rejection Adaptive Metropolis
Covar: (NOT SHOWN HERE; diagonal shown instead)
 [1] 0.084759 0.081693 0.695060 0.355258 0.155647 0.052688
 [7] 0.148212 0.131299 0.170847 0.368145 0.103156

CovarDHis: (NOT SHOWN HERE)
DIC of all samples (Dbar): 91.273
DIC of all samples (pD): 968.64
DIC of all samples (DIC): 1059.9
DIC of stationary samples (Dbar): 86.046
DIC of stationary samples (pD): 106.96
DIC of stationary samples (DIC): 193.00
DR: 1
Initial Values:
 [1] 0 0 0 0 0 0 0 0 0 0 0

Iterations: 10000
Log-Marginal Likelihood: -89.763
Minutes of run-time: 0.37
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: 290
Status is displayed every 1000 iterations
Summary1: (SHOWN BELOW)
Summary2: (SHOWN BELOW)
Thinned Samples: 1000
Thinning: 10


Summary of All Samples
              Mean       SD     MCSE     ESS         LB
beta[1]    4.98575  0.41962 0.060709  47.776    4.55879
beta[2]   -0.52199  0.39939 0.032998 146.492   -1.31310
beta[3]   -0.76316  1.16481 0.211942  30.205   -3.74508
beta[4]    0.05211  0.82495 0.168197  24.056   -1.27877
beta[5]   -0.54739  0.55229 0.052224 111.836   -1.66639
```

```
beta[6]   -0.51761  0.31742 0.020297 244.575   -1.15368
beta[7]    2.33422  0.53727 0.039787 182.354    1.24725
beta[8]    0.69312  0.50935 0.063835  63.666   -0.24318
beta[9]   -0.19174  0.56505 0.041635 184.184   -1.21528
beta[10]   1.87953  0.84446 0.093782  81.081    0.41744
log.tau    0.57753  0.44770 0.072955  37.658   -0.25642
Deviance  91.27320 44.01450 3.487605 159.271   74.34135
LP       -96.88605 21.90373 1.683197 169.343 -133.61290
tau        1.95984  0.76054 0.070424 116.626    0.72221
             Median         UB
beta[1]    5.033012   5.285545
beta[2]   -0.505730   0.233485
beta[3]   -0.669652   1.143706
beta[4]   -0.021369   2.346119
beta[5]   -0.527179   0.484156
beta[6]   -0.519699   0.075785
beta[7]    2.349993   3.496501
beta[8]    0.658268   1.831501
beta[9]   -0.176907   0.851550
beta[10]   1.814351   3.787706
log.tau    0.623115   1.135299
Deviance  83.617162 167.446819
LP       -93.056475 -88.744643
tau        1.894297   3.505554


Summary of Stationary Samples
              Mean       SD     MCSE    ESS          LB
beta[1]    5.04557  0.11868 0.004881 591.23    4.814571
beta[2]   -0.52362  0.39048 0.035129 123.56   -1.311910
beta[3]   -0.47931  0.89817 0.054157 275.05   -2.167081
beta[4]   -0.11374  0.62000 0.032336 367.62   -1.282226
beta[5]   -0.47736  0.54689 0.045461 144.71   -1.527810
beta[6]   -0.49279  0.27033 0.017765 231.55   -1.023104
beta[7]    2.30835  0.52407 0.042463 152.32    1.222653
beta[8]    0.59179  0.45094 0.033123 185.34   -0.295607
beta[9]   -0.15060  0.55303 0.043271 163.35   -1.165710
beta[10]   1.74735  0.76890 0.053745 204.68    0.371212
log.tau    0.65451  0.27090 0.018185 221.92    0.095373
Deviance  86.04567 14.62590 0.851933 294.74   74.070956
LP       -94.33643  7.31275 0.411682 315.53 -110.073588
tau        2.01495  0.63118 0.039498 255.36    0.944870
             Median         UB
beta[1]    5.04186   5.299255
beta[2]   -0.50511   0.189954
beta[3]   -0.49805   1.296241
beta[4]   -0.13297   1.078877
```

```
beta[5]    -0.48085   0.556542
beta[6]    -0.49886   0.053826
beta[7]     2.32787   3.421885
beta[8]     0.57468   1.501888
beta[9]    -0.13048   0.888065
beta[10]    1.71997   3.302169
log.tau     0.66436   1.139950
Deviance   82.89749 117.858803
LP       -92.79119 -88.656286
tau        1.98189   3.489243
```

Several components 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 component, such as:

```
> Fit$Posterior2
```

Although a lot can be learned from the above output, notice that it completed 10000 iterations of 11 variables in 0.37 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 (see section 13).

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 with the `LaplacesDemon` function, 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.

Consorting with Laplace's Demon 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: 0.3158
Adaptive: 900
```

```
Algorithm: Delayed Rejection Adaptive Metropolis
Covar: (NOT SHOWN HERE; diagonal shown instead)
 [1] 0.084759 0.081693 0.695060 0.355258 0.155647 0.052688
 [7] 0.148212 0.131299 0.170847 0.368145 0.103156

CovarDHis: (NOT SHOWN HERE)
DIC of all samples (Dbar): 91.273
DIC of all samples (pD): 968.64
DIC of all samples (DIC): 1059.9
DIC of stationary samples (Dbar): 86.046
DIC of stationary samples (pD): 106.96
DIC of stationary samples (DIC): 193.00
DR: 1
Initial Values:
 [1] 0 0 0 0 0 0 0 0 0 0 0

Iterations: 10000
Log-Marginal Likelihood: -89.763
Minutes of run-time: 0.37
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: 290
Status is displayed every 1000 iterations
Summary1: (SHOWN BELOW)
Summary2: (SHOWN BELOW)
Thinned Samples: 1000
Thinning: 10


Summary of All Samples
              Mean       SD     MCSE     ESS         LB
beta[1]    4.98575  0.41962 0.060709  47.776    4.55879
beta[2]   -0.52199  0.39939 0.032998 146.492   -1.31310
beta[3]   -0.76316  1.16481 0.211942  30.205   -3.74508
beta[4]    0.05211  0.82495 0.168197  24.056   -1.27877
beta[5]   -0.54739  0.55229 0.052224 111.836   -1.66639
beta[6]   -0.51761  0.31742 0.020297 244.575   -1.15368
beta[7]    2.33422  0.53727 0.039787 182.354    1.24725
beta[8]    0.69312  0.50935 0.063835  63.666   -0.24318
beta[9]   -0.19174  0.56505 0.041635 184.184   -1.21528
beta[10]   1.87953  0.84446 0.093782  81.081    0.41744
```

```
log.tau    0.57753  0.44770 0.072955  37.658   -0.25642
Deviance  91.27320 44.01450 3.487605 159.271   74.34135
LP       -96.88605 21.90373 1.683197 169.343 -133.61290
tau        1.95984  0.76054 0.070424 116.626    0.72221
              Median         UB
beta[1]     5.033012   5.285545
beta[2]    -0.505730   0.233485
beta[3]    -0.669652   1.143706
beta[4]    -0.021369   2.346119
beta[5]    -0.527179   0.484156
beta[6]    -0.519699   0.075785
beta[7]     2.349993   3.496501
beta[8]     0.658268   1.831501
beta[9]    -0.176907   0.851550
beta[10]    1.814351   3.787706
log.tau     0.623115   1.135299
Deviance   83.617162 167.446819
LP        -93.056475 -88.744643
tau         1.894297   3.505554


Summary of Stationary Samples
               Mean       SD     MCSE    ESS          LB
beta[1]     5.04557  0.11868 0.004881 591.23    4.814571
beta[2]    -0.52362  0.39048 0.035129 123.56   -1.311910
beta[3]    -0.47931  0.89817 0.054157 275.05   -2.167081
beta[4]    -0.11374  0.62000 0.032336 367.62   -1.282226
beta[5]    -0.47736  0.54689 0.045461 144.71   -1.527810
beta[6]    -0.49279  0.27033 0.017765 231.55   -1.023104
beta[7]     2.30835  0.52407 0.042463 152.32    1.222653
beta[8]     0.59179  0.45094 0.033123 185.34   -0.295607
beta[9]    -0.15060  0.55303 0.043271 163.35   -1.165710
beta[10]    1.74735  0.76890 0.053745 204.68    0.371212
log.tau     0.65451  0.27090 0.018185 221.92    0.095373
Deviance   86.04567 14.62590 0.851933 294.74   74.070956
LP        -94.33643  7.31275 0.411682 315.53 -110.073588
tau         2.01495  0.63118 0.039498 255.36    0.944870
             Median         UB
beta[1]     5.04186   5.299255
beta[2]    -0.50511   0.189954
beta[3]    -0.49805   1.296241
beta[4]    -0.13297   1.078877
beta[5]    -0.48085   0.556542
beta[6]    -0.49886   0.053826
beta[7]     2.32787   3.421885
beta[8]     0.57468   1.501888
beta[9]    -0.13048   0.888065
```

```
beta[10]   1.71997   3.302169
log.tau    0.66436   1.139950
Deviance  82.89749 117.858803
LP       -92.79119 -88.656286
tau        1.98189   3.489243

Demonic Suggestion

Due to the combination of the following conditions,

1. Delayed Rejection Adaptive Metropolis
2. The acceptance rate (0.3158) 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 (ESS)
   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=35,
     Covar=Fit$Covar, DR=0, Initial.Values, Iterations=290000,
     Periodicity=918, Status=27027, Thinning=290)

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, Monte Carlo standard error (MCSE), effective sample size (ESS), and stationarity. There are 48 combinations of these five conditions, though many combinations lead to the same conclusions. 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 unsatisfactory, 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 section 11). The acceptance rate is considered satisfactory if it is within the interval [15%,50%][7]. 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 Gaussian distribution around the estimated mean. ESS 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=290000` and `Thinning=290`. For the sake of the example and saving the reader from a few pages of output, the suggested R code will not be run and the following will be run instead:

```
> Initial.Values <- Fit$Posterior1[Fit$Thinned.Samples,
+     ]
> Fit <- LaplacesDemon(Model, Data = MyData, Adaptive = 0,
+     Covar = Fit$Covar, DR = 0, Initial.Values, Iterations = 180000,
+     Periodicity = 0, Status = 10000, Thinning = 180)

Laplace's Demon was called on Sun Mar  6 22:13:17 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: 10000,   Proposal: Single-Component
Iteration: 20000,   Proposal: Multivariate
Iteration: 30000,   Proposal: Multivariate
Iteration: 40000,   Proposal: Multivariate
Iteration: 50000,   Proposal: Multivariate
Iteration: 60000,   Proposal: Multivariate
Iteration: 70000,   Proposal: Multivariate
Iteration: 80000,   Proposal: Multivariate
Iteration: 90000,   Proposal: Multivariate
Iteration: 100000,   Proposal: Multivariate
Iteration: 110000,   Proposal: Multivariate
Iteration: 120000,   Proposal: Multivariate
Iteration: 130000,   Proposal: Multivariate
```

[7]While Spiegelhalter, Thomas, Best, and Lunn (2003) recommend updating until the acceptance rate is within the interval [20%,40%], and Roberts and Rosenthal (2001) suggest [10%,40%], the interval recommended here is [15%,50%].

```
Iteration: 140000,   Proposal: Multivariate
Iteration: 150000,   Proposal: Multivariate
Iteration: 160000,   Proposal: Multivariate
Iteration: 170000,   Proposal: Multivariate

Assessing Stationarity
Assessing Thinning and ESS
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 = 180000,
    Periodicity = 0, Status = 10000, Thinning = 180)

Acceptance Rate: 0.17264
Adaptive: 180001
Algorithm: Random-Walk Metropolis
Covar: (NOT SHOWN HERE; diagonal shown instead)
 [1] 0.084759 0.081693 0.695060 0.355258 0.155647 0.052688
 [7] 0.148212 0.131299 0.170847 0.368145 0.103156

CovarDHis: (NOT SHOWN HERE)
DIC of all samples (Dbar): 82.767
DIC of all samples (pD): 17.566
DIC of all samples (DIC): 100.33
DIC of stationary samples (Dbar): 82.855
DIC of stationary samples (pD): 18.021
DIC of stationary samples (DIC): 100.88
DR: 0
Initial Values:
 [1]  5.223185 -0.258280 -0.127803  0.051125 -0.727624 -1.010162
 [7]  2.047204  0.069733 -0.329800  2.074757  0.305603

Iterations: 180000
Log-Marginal Likelihood: NaN
Minutes of run-time: 1.65
Model: (NOT SHOWN HERE)
```

```
Monitor: (NOT SHOWN HERE)
Parameters (Number of): 11
Periodicity: 180001
Posterior1: (NOT SHOWN HERE)
Posterior2: (NOT SHOWN HERE)
Recommended Burn-In of Thinned Samples: 101
Recommended Burn-In of Un-thinned Samples: 18180
Recommended Thinning: 180
Status is displayed every 10000 iterations
Summary1: (SHOWN BELOW)
Summary2: (SHOWN BELOW)
Thinned Samples: 1000
Thinning: 180


Summary of All Samples
               Mean      SD      MCSE     ESS         LB
beta[1]    5.038749 0.11646 0.0036829 1000.00   4.804800
beta[2]   -0.459686 0.36099 0.0127183  805.61  -1.206870
beta[3]   -0.450027 0.94921 0.0322922  864.03  -2.266950
beta[4]   -0.087439 0.71256 0.0225333 1000.00  -1.498926
beta[5]   -0.432082 0.51110 0.0181564  792.41  -1.461060
beta[6]   -0.477028 0.29573 0.0100647  863.38  -1.044564
beta[7]    2.256190 0.54976 0.0191678  822.64   1.138339
beta[8]    0.613356 0.43754 0.0145697  901.85  -0.240514
beta[9]   -0.158728 0.58660 0.0195076  904.22  -1.298208
beta[10]   1.661952 0.77912 0.0271239  825.09   0.122949
log.tau    0.667271 0.28286 0.0089448 1000.00   0.060415
Deviance  82.766960 5.92724 0.2130495  774.01  73.743243
LP       -92.705809 2.79787 0.1004620  775.62 -99.230567
tau        2.011161 0.55770 0.0176360 1000.00   1.051018
             Median        UB
beta[1]    5.041494   5.25577
beta[2]   -0.456871   0.22997
beta[3]   -0.479041   1.40472
beta[4]   -0.052237   1.33462
beta[5]   -0.428363   0.57156
beta[6]   -0.476297   0.07648
beta[7]    2.274999   3.29131
beta[8]    0.627241   1.46009
beta[9]   -0.168592   0.95889
beta[10]   1.654995   3.17903
log.tau    0.676537   1.16755
Deviance  81.935652  96.66287
LP       -92.297541 -88.45990
tau        1.944181   3.22751
```

```
Summary of Stationary Samples
               Mean      SD      MCSE    ESS         LB
beta[1]    5.037064 0.11662 0.0038874 900.00   4.801999
beta[2]   -0.457591 0.36406 0.0121354 900.00  -1.225101
beta[3]   -0.447784 0.95503 0.0318345 900.00  -2.275635
beta[4]   -0.071333 0.71772 0.0239241 900.00  -1.483336
beta[5]   -0.426331 0.51569 0.0171897 900.00  -1.495565
beta[6]   -0.483570 0.30108 0.0100360 900.00  -1.048696
beta[7]    2.261568 0.55191 0.0188148 860.48   1.126839
beta[8]    0.616376 0.43182 0.0143938 900.00  -0.205416
beta[9]   -0.179314 0.58955 0.0196518 900.00  -1.298250
beta[10]   1.656385 0.78816 0.0283596 772.37   0.124178
log.tau    0.666857 0.28514 0.0095047 900.00   0.055071
Deviance  82.854714 6.00355 0.2257224 707.40  73.733602
LP       -92.748294 2.83437 0.1067622 704.82 -99.275690
tau        2.009609 0.56238 0.0187462 900.00   1.046095
             Median         UB
beta[1]    5.040274   5.256832
beta[2]   -0.450436   0.228595
beta[3]   -0.465163   1.387763
beta[4]   -0.026182   1.366174
beta[5]   -0.416428   0.576283
beta[6]   -0.489486   0.084237
beta[7]    2.289064   3.284359
beta[8]    0.625241   1.496912
beta[9]   -0.208225   0.971124
beta[10]   1.649771   3.198003
log.tau    0.678813   1.175322
Deviance  81.985021  96.805919
LP       -92.318844 -88.449905
tau        1.944181   3.264851

Demonic Suggestion

Due to the combination of the following conditions,

1. Random-Walk Metropolis
2. The acceptance rate (0.17264) is within the interval [0.15,0.5].
3. Each target MCSE is < 6.27% of its marginal posterior
   standard deviation.
4. Each target distribution has an effective sample size (ESS)
   of at least 100.
5. Each target distribution became stationary by
   101 iterations.

Laplace's Demon has been appeased, and suggests
```

```
the marginal posterior samples should be plotted
and subjected to any other MCMC diagnostic deemed
fit before using these samples for inference.

Laplace's Demon is finished consorting.
```

In 1.65 minutes, Laplace's Demon updated 180000 iterations, retaining every 180th iteration due to thinning, and reported an acceptance rate of 0.173. Notice that all criteria have been met: MCSE's are sufficiently small, ESS's are sufficiently large, and stationarity was estimated beginning with the first iteration. Since the algorithm was RWM, the Markov property holds, so let's look at some plots.

7. Plotting Output

Laplace's Demon has a `plot.demonoid` function to enable its owncustomized 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 three 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.

Another useful plot is called the caterpillar plot, which plots a horizontal representation of three quantiles (2.5%, 50%, and 97.5%) of each selected parameter from the posterior samples summary. The caterpillar plot will attempt to plot the stationary samples first (`Fit$Summary2`), but if stationary samples do not exist, then it will plot all samples (`Fit$Summary1`). Here, only the first ten parameters are selected for a caterpillar plot:

```
> caterpillar.plot(Fit, Parms = 1:10)
```

When predicting the logarithm of `y` (Calories) with the `demonsnacks` data, the caterpillar plot shows that 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.

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

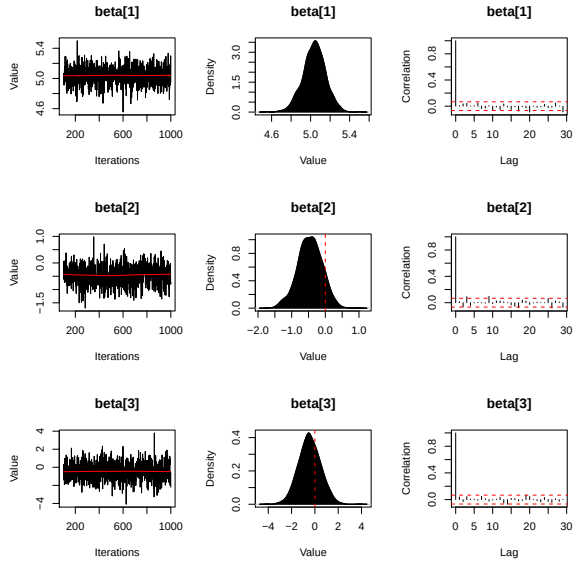

Figure 1: Plots of Marginal Posterior Samples

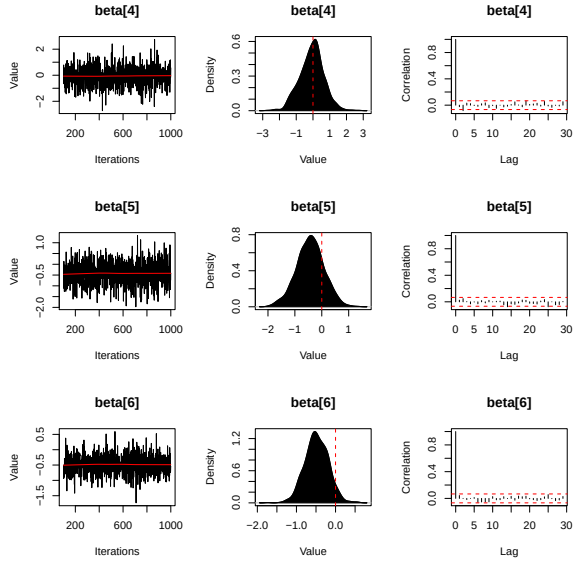

Figure 2: Plots of Marginal Posterior Samples

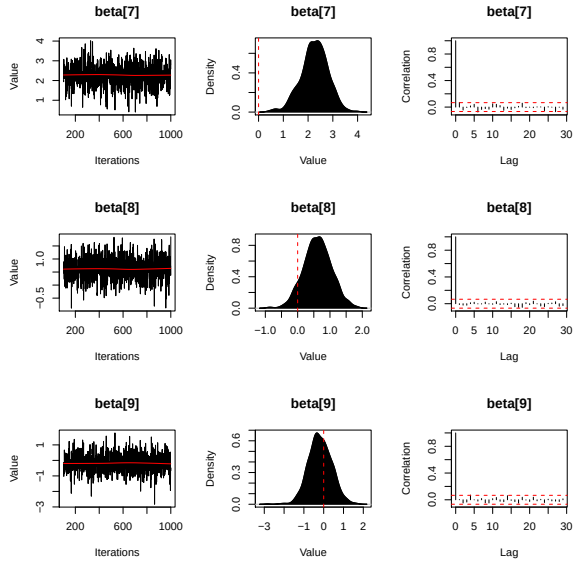

Figure 3: Plots of Marginal Posterior Samples

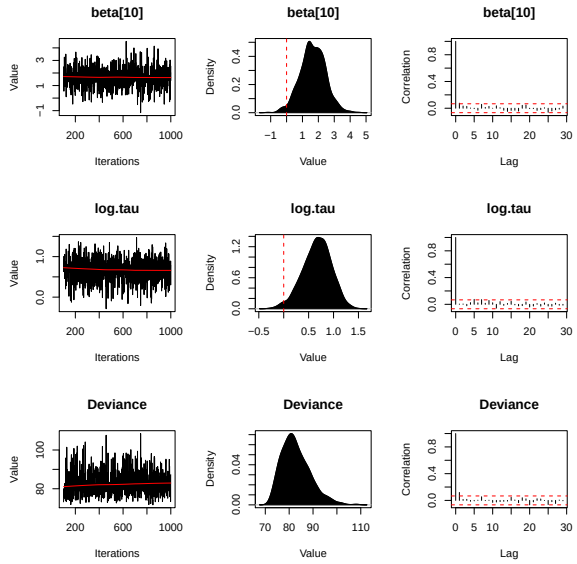

Figure 4: Plots of Marginal Posterior Samples

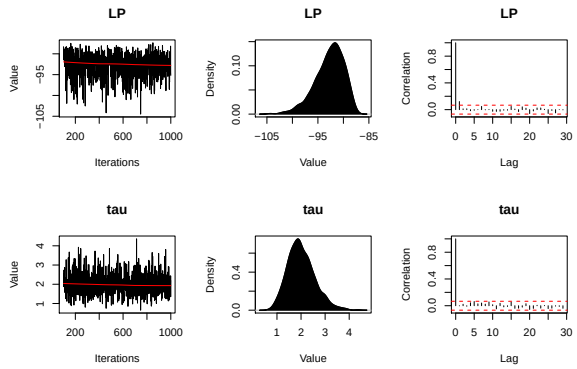

Figure 5: Plots of Marginal Posterior Samples

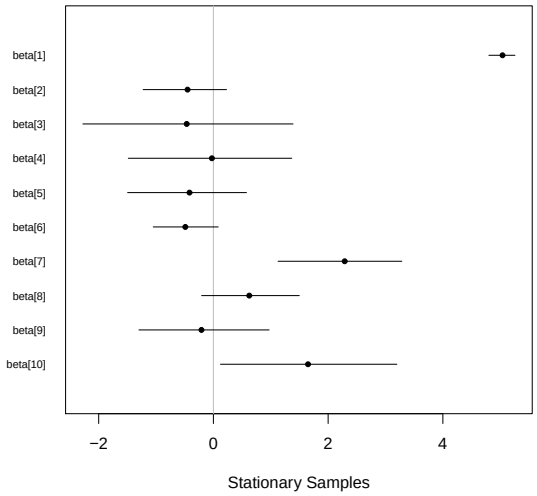

Figure 6: Caterpillar Plot

how well (or poorly) the model fits the data. When the user is satisfied, the `BayesFactor` function may be useful in selecting the best model, and the marginal posterior samples may be used for inference.

8. Posterior Predictive Checks

A posterior predictive check is a method to assess discrepancies between the model and the data (Gelman, Meng, and Stern 1996a). 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 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 `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 well to the data.

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

```
> summary(Pred)

Concordance:  0.84615
Discrepancy Statistic:  0
Records:
        y   Mean      SD     LB Median     UB        PQ Discrep
1  4.1744 4.1554 0.19958 3.7569 4.1573 4.5457 0.4633333      NA
2  5.3613 5.2779 0.40995 4.5077 5.2846 6.0940 0.4333333      NA
3  6.0890 5.2488 0.56046 4.1031 5.2828 6.2971 0.0588889      NA
4  5.2983 5.1287 0.32527 4.5225 5.1432 5.7738 0.3077778      NA
5  4.4067 4.0714 0.24256 3.6150 4.0778 4.5495 0.0744444      NA
6  2.1972 3.7847 0.21294 3.3722 3.7846 4.2027 1.0000000      NA
7  5.0106 4.5314 0.18901 4.1594 4.5242 4.9193 0.0100000      NA
8  1.6094 3.8441 0.21079 3.4557 3.8417 4.2577 1.0000000      NA
9  4.3438 4.2150 0.23151 3.7378 4.2169 4.6569 0.2866667      NA
10 4.8122 4.7045 0.22569 4.2526 4.7014 5.1578 0.3077778      NA
11 4.1897 4.3951 0.19999 4.0063 4.3952 4.7953 0.8622222      NA
12 4.9200 4.5176 0.18663 4.1535 4.5271 4.9179 0.0255556      NA
13 4.7536 4.3610 0.19303 3.9823 4.3652 4.7610 0.0266667      NA
14 4.1271 4.1479 0.17417 3.7763 4.1481 4.4803 0.5455556      NA
15 3.7136 4.0779 0.20372 3.6884 4.0845 4.4967 0.9644444      NA
```

```
16 4.6728 4.3840 0.22698 3.9375 4.3917 4.8290 0.0966667      NA
17 6.9305 7.1724 0.53716 6.0722 7.1803 8.2126 0.6733333      NA
18 5.0689 4.7904 0.25144 4.3268 4.7895 5.2797 0.1311111      NA
19 6.7754 6.3287 0.49707 5.3762 6.3324 7.2899 0.1766667      NA
20 6.5539 7.2313 0.48924 6.1942 7.2261 8.2012 0.9233333      NA
21 4.8903 5.3742 0.36537 4.6564 5.3688 6.0934 0.9122222      NA
22 4.4427 4.2431 0.27006 3.7171 4.2416 4.7853 0.2111111      NA
23 2.8332 3.0834 0.51024 2.0786 3.0808 4.0732 0.6900000      NA
24 4.7875 4.9290 0.25896 4.4336 4.9261 5.4385 0.7077778      NA
25 6.9334 7.2306 0.64160 5.8709 7.2599 8.5254 0.6866667      NA
26 6.1800 6.0758 0.60984 4.8684 6.1053 7.2282 0.4444444      NA
27 5.6525 5.3365 0.30624 4.7078 5.3379 5.9275 0.1455556      NA
28 5.4293 4.4533 0.21510 4.0434 4.4475 4.8982 0.0000000      NA
29 5.6348 5.5007 0.70196 4.0060 5.5188 6.8680 0.4355556      NA
30 4.2627 4.0472 0.21568 3.6271 4.0483 4.4821 0.1488889      NA
31 3.8918 4.0495 0.25002 3.5462 4.0448 4.5420 0.7533333      NA
32 6.6134 6.6235 0.40237 5.8314 6.6219 7.4616 0.5066667      NA
33 4.9200 4.3900 0.19699 4.0219 4.3994 4.7850 0.0066667      NA
34 6.5410 6.4554 0.49434 5.5089 6.4732 7.4336 0.4333333      NA
35 6.3456 6.4213 0.48730 5.4675 6.4209 7.3654 0.5677778      NA
36 3.7377 4.0511 0.25659 3.5367 4.0524 4.5672 0.8900000      NA
37 7.3563 7.9271 0.66413 6.6515 7.9456 9.2677 0.7966667      NA
38 5.7398 4.7552 0.17873 4.4015 4.7497 5.0852 0.0000000      NA
39 5.5175 5.1305 0.26659 4.6350 5.1266 5.6582 0.0733333      NA
```

The `summary.demonoid.ppc` function returns a list with 3 components:

- `Concordance` is the predictive concordance of Gelfand (1996), that indicates the percentage of times that `y` that was within the 95% probability interval of `yhat`. A goal is to have 95% predictive concordance. For more information, see the accompanying vignette entitled "Bayesian Inference". In this case, roughly 1% of the time, `y` is within the 95% probability interval of `yhat`. These results suggest that the model should be attempted again under different conditions, such as using different predictors, or specifying a different form to the model.

- `Discrepancy.Statistic` is a summary of a specified discrepancy measure. There are many options for discrepancy measures that may be specified in the `Discrep` argument. In this example, a discrepancy measure was not specified.

- The last part of the summarized output reports `y`, information about the distribution of `yhat`, and the predictive quantile (`PQ`). The mean prediction of `y[1]`, or y_1^{rep}, given the model and data, is 4.155. Most importantly, `PQ[1]` is 0.463, indicating that 46.3% of the time, `yhat[1,]` was greater than `y[1]`, or that `y[1]` is close to the mean of `yhat[1,]`. Contrast this with the 6th record, where `y[6]`=2.197 and `PQ[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[1]` is within the 95% probability interval of `yhat[1,]`, the 95% probability interval of `yhat[6,]` is above `y[6]` 100% of the time, indicating a strong discrepancy between the model and data, in this case.

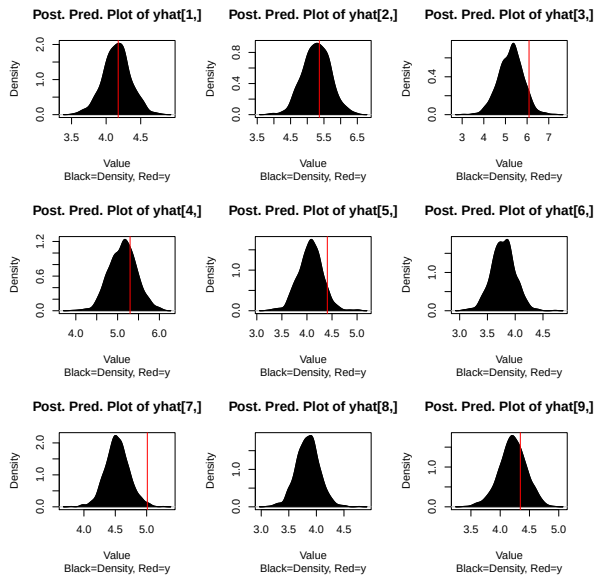

Figure 7: Posterior Predictive Plots

The last component 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.

9. 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 `x`: `x.cs <- (x - mean(x)) / (2*sd(x))`. However, it is instead easier to use the `CenterScale` function provided in **LaplacesDemon**.
- Do not forget to reparameterize any bounded parameters in the `Model` function to be real-valued in the `parm` vector.
- 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 the `Model` function, it may be tempting to specify a large number of iterations and thinning in the `LaplacesDemon` function, 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.

- As model complexity increases, the number of parameters increases, and as initial values are further from high-probability regions, the initial acceptance rate may be very low. If the previous general suggestion was successful, but the aceptance rate was zero, then update the model again, but for more iterations. The goal here is to verify that proposals are accepted without problems before attempting an "actual" model update.

- After studying updates with few iterations, the first "actual" update should be long enough that proposals are accepted (the acceptance rate is not zero), 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 optimizing the initial values with the `LaplaceApproximation` function, changing the initial values, or setting all initial values equal to zero so the `LaplacesDemon` function will use the `LaplaceApproximation` function. In MCMC, initial values are most effective when the starting points are close to the target distributions (though, if the target distributions were known *a priori*, then there would be little point in much of this). When initial values are far enough away from the target distributions to be in low-probability regions, the algorithms (both Laplace Approximation and MCMC) may take longer than usual. The MCMC algorithms herein will struggle more as the proposal covariance matrix approaches near-singularity. In extreme examples, it is possible for the proposal covariance matrix to become singular, which will stop Laplace's Demon. If there is no information available to make a better selection, then randomize the initial values and use `LaplaceApproximation`. Centered and scaled predictors also help by essentially standardizing the possible range of the target distributions.

- If Laplace's Demon exhibits an unreasonably low acceptance rate (say, arbitrarily, lower than 15%, but greater than 0%) and is having a hard time exploring (but is still able to explore) after significant iterations, then investigate the latest proposal covariance matrix by entering `Fit$Covar`. Chances are that the elements of 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 update, 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-plots of the deviance, or better yet, the logarithm of the unnormalized joint posterior density. If the deviance is decreasing and the joint posterior is increasing run after run, then the model is continuously fitting better and better, and one possible sign of convergence will be when the deviance and the joint posterior seem to become stationary or no longer show a trend.

- If Laplace's Demon is exploring areas of the state space that the user knows *a priori* should not be explored, then the parameters may be constrained in the `Model` function before being passed back to the `LaplacesDemon` function. Simply change the parameter of interest as appropriate and place the constrained value back in the `parm` vector.

- `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 (Gelman *et˜al.* 2004, p. 295), where the ESS is at least 100 (and more is desirable).

- Disagreement exists in MCMC literature as to whether to update one, long chain (Geyer 1992), or multiple, long chains with different, randomized initial values (Gelman and Rubin 1992). Laplace's Demon is not designed to simultaneously update multiple chains. 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.

10. 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 dependence on other software. Laplace's Demon is performed completely within base R (though of course the **LaplacesDemon** package is required). A goal is to provide a complete, Bayesian environment. 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.

Common Bayesian probability distributions (such as Dirichlet, multivariate normal, Wishart, and others, as well as truncated forms of distributions) have been included in **LaplacesDemon** so the user does not have to load numerous R packages. 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
```

11. Details

The **LaplacesDemon** package uses two broad types of numerical approximation algorithms: Laplace Approximation and Markov chain Monte Carlo (MCMC). Each are described below, but MCMC is emphasized.

11.1. Laplace Approximation

The Laplace Approximation or Laplace Method is a family of asymptotic techniques used to approximate integrals. Laplace's method seems to accurately approximate uni-modal posterior moments and marginal posterior distributions in many cases. Since it is not applicable in all cases, it is recommended here that Laplace Approximation is used cautiously in its own right, or preferably, it is used before MCMC.

After introducing the Laplace Approximation (Laplace 1774, p. 366–367), a proof was published later (Laplace 1814) as part of a mathematical system of inductive reasoning based on probability. Laplace used this method to approximate posterior moments.

Since its introduction, the Laplace Approximation has been applied successfully in many disciplines. In the 1980s, the Laplace Approximation experienced renewed interest, especially in statistics, and some improvements in its implementation were introduced (Tierney and Kadane 1986; Tierney, Kass, and Kadane 1989). Only since the 1980s has the Laplace Approximation been seriously considered by statisticians in practical applications.

There are many variations of Laplace Approximation, with an effort toward replacing Markov chain Monte Carlo (MCMC) algorithms as the dominant form of numerical approximation in Bayesian inference. The run-time of Laplace Approximation is a little longer than Maximum Likelihood Estimation (MLE), and much shorter than MCMC (Azevedo-Filho and Shachter 1994). In the **LaplacesDemon** package, Laplace Approximation may iterate faster or slower than MCMC, so this is not the fastest possible implementation of Laplace Approximation. Laplace Approximation extends MLE, but shares similar limitations, such as its asymptotic nature with respect to sample size. Bernardo and Smith (2000) note that Laplace Approximation is an attractive numerical approximation algorithm, and will continue to develop, though it currently works best with few parameters.

The `LaplaceApproximation` function may be called by the user before using `LaplacesDemon`, or `LaplacesDemon` may call this function if all initial values are zero. Chasing convergence with `LaplaceApproximation` may be time-consuming and unimportant. The goal, instead, is to improve the logarithm of the unnormalized joint posterior density so that it is easier for the `LaplacesDemon` function to begin updating the parameters in search of the target distributions. This can be difficult when the initial values are in low-probability regions, and can cause unreasonably low acceptance rates.

`LaplaceApproximation` seeks a global maximum of the logarithm of the unnormalized joint posterior density by taking steps proportional to an adaptive scale of the approximate gradient. This portion of the `LaplaceApproximation` function uses a gradient ascent algorithm, and is called a gradient descent or steepest descent algorithm elsewhere for minimization problems. Laplace's Demon uses the `LaplaceApproximation` algorithm to optimize initial

values, estimate covariance, and save time for the user, though it is used only when sample size is at least five times the number of parameters or initial values.

This algorithm assumes that the logarithm of the unnormalized joint posterior density is defined and differentiable. An approximate gradient is taken for each initial value as the difference in the logarithm of the unnormalized joint posterior density due to a slight increase versus decrease in the parameter.

At 10 evenly-space times, `LaplaceApproximation` attempts several step sizes, which are also called rate parameters in other literature, and selects the best step size from a set of 10 fixed options. Thereafter, each iteration in which an improvement does not occur, the step size shrinks, being multiplied by 0.999.

Gradient ascent is criticized for sometimes being relatively slow when close to the maximum, and its asymptotic rate of convergence is inferior to other methods. However, compared to other popular optimization algorithms such as Newton-Rhapson, an advantage of the gradient ascent is that it works in infinite dimensions, requiring only sufficient computer memory. Although Newton-Rhapson converges in fewer iterations, calculating the inverse of the negative Hessian matrix of second-derivatives is more computationally expensive and subject to singularities. Therefore, gradient ascent takes longer to converge, but is more generalizable.

After `LaplaceApproximation` finishes, due either to early convergence or completing the number of specified iterations, it approximates the Hessian matrix of second derivatives, and attempts to calculate the covariance matrix by taking the inverse of the negative of this matrix. If successful, then this covariance matrix may be passed to `LaplacesDemon`, and the diagonal of this matrix is the variance of the parameters. If unsuccessful, a scaled identity matrix is returned, and each parameter's variance will be 1.

11.2. Markov Chain Monte Carlo

Although the `LaplacesDemon` function may be assisted by Laplace Approximation, Laplace's Demon mainly 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. RWM was developed first (Metropolis, Rosenbluth, M.N., and Teller 1953), and Metropolis-Hastings was a generalization of RWM (Hastings 1970). All MCMC algorithms are known as special cases of the Metropolis-Hastings algorithm. Regardless of the algorithm, the goal in Bayesian inference is to maximize the unnormalized 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. The most generalizable MCMC algorithm is the Metropolis-Hastings (MH) generalization of the RWM algorithm. The MH algorithm extended RWM to include asymmetric proposal distributions. Having no need of asymmetric proposals, Laplace's Demon uses variations of the original RWM algorithm, which use symmetric proposal distributions, specifically Gaussian proposals. For years, the main disadvantage of the RWM and MH algorithms was that the proposal variance (see below) had to be tuned manually, and therefore other MCMC algorithms have become popular because they do not need to be tuned.

Gibbs sampling became popular for Bayesian inference, though it requires conditional sam-

pling of conjugate distributions, so it is precluded from non-conjugate sampling in its purest form. Gibbs sampling also suffers under high correlations. Due to these limitations, Gibbs sampling is less generalizable than RWM. Slice sampling samples a distribution by sampling uniformly from the region under the plot of its density function, and 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. A brief discussion follows of RWM and its adaptive variants.

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, though this increases the importance that the initial values are not in low-probability regions, and may cause Laplace's Demon to have chains that are slow to begin moving.

Random-Walk Metropolis

In MCMC algorithms, each iterative estimate of a parameter is part of a changing state. The succession of states or iterations constitutes a Markov chain when the current state is influenced only by the previous state. In random-walk Metropolis (RWM), a proposed future estimate, called a proposal[8] or candidate, of the joint posterior density is calculated, 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 logarithm of the unnormalized 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, and it is possible for the algorithm to backtrack while it explores.

[8]Laplace's Demon allows the user to constrain proposals in the `Model` function. Laplace's Demon generates a proposal vector, which is passed to the `Model` function in the `parm` vector. In the `Model` function, the user may constrain the proposal to prevent the sampler from exploring certain areas of the state space by altering the proposed values and placing them back into the `parm` vector, which will be passed back to Laplace's Demon.

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 of a target distribution 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 single-component proposals by updating one randomly-selected parameter, 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%.

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. However, DRM does not violate the Markov property (see below), because the proposal is based on the current state. 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 by considering previous states in the Markov chain, but merely makes

more attempts from the current state. DRM is rarely suggested by Laplace's Demon, though the combination of DRM and AM, called DRAM (see below), is suggested frequently.

Laplace's Demon also temporarily shrinks the proposal covariance arbitrarily by 50% for delayed rejection. A smaller proposal covariance is more likely to be accepted, and the goal of delayed rejection is to increase acceptance. In the long-term, a proposal covariance that is too small is undesirable, and so it is only used in this case to assist acceptance.

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[9]. 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 *et˜al.* (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 $2.4^2/d$[10] based on the asymptotic limit of infinite-dimensional Gaussian target distributions that are independent and identically-distributed (Gelman, Roberts, and Gilks 1996b). 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: $2.381204^2/d$. 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 single-component adaptive proposals (Haario, Saksman, and Tamminen 2005). Although single-component 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:

$$\Sigma^* = [\phi_K cov(\Theta_{1:t,1:K})] + (\phi_K C \mathbf{I}_K)$$

[9] Haario, Saksman, and Tamminen (2001) assert that the chains remain ergodic in the limit as the amount of change in the adaptations should decrease to zero as the chains approach the target distributions.

[10] The optimal proposal standard deviation in this case is approximately $2.4/\sqrt{d}$.

where ϕ_K is the scale according to `K` parameters, C is a small (1.0E-5) 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 single-component adaptive tuning, the formula across `K` parameters and `t` iterations is:

$$\sigma_k^{*2} = \phi_k var(\Theta_{1:t,k}) + \phi_k C$$

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 single-component 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 the `LaplacesDemon` function. 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 (and will instead use Laplace Approximation whensample size permits), it uses the second suggested trick of shrinking the proposal as long as the acceptance rate is less than 5%, and there have been at least five acceptances. 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 (1 - 1/Iterations). 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.

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.

11.3. Afterward

Once the model is updated with the `LaplacesDemon` function, the `Geweke.Diagnostic` func-

tion 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` function is applied by beginning with each cumulative group.

The importance of Monte Carlo Standard Error (MCSE) is debated. Here, it is considered important enough to be one of five main criteria to appease Laplace's Demon. It is often recommended that one of several competing batch methods should be used to estimate MCSE, arguing that the simple method (`MCSE =` $\sigma/\sqrt{m}$) is biased and reports less error (where m is the ESS). 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.

12. 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 (Spiegelhalter *et˜al.* 2003), 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, whereas the RWM algorithm backtracks due to its random-walk behavior. 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. Laplace's Demon limits the user to one chain per parameter per update, where BUGS can update multiple chains per parameter simultaneously. 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 disadvantages are that convergence is slower and RWM may get stuck in regions of low probability. 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[11]. 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 (Geyer 2010) 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 (Martin, Quinn, and Park 2010) 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, the **UMACS** package (Kerman 2007) has been removed from CRAN. It became outdated due to lack of interest, but did include an adaptive MCMC algorithm as well as Gibbs sampling.

In SAS 9.2 (SAS Institute Inc. 2008), 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.

[11] **AMCMC** is available from J. S. Rosenthal's website at `http://www.probability.ca/amcmc/`

13. 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[12]. It was nowhere near convergence, but updating the same model with the same data for 1,000 iterations took 45.55 minutes in JAGS.

However, the speed with which an iteration is estimated is not a good, overall criterion of performance. For example, a Gibbs sampling algorithm with uncorrelated target distributions should converge in fewer iterations than a random-walk algorithm, such as those used in Laplace's Demon. Depending on circumstances, Laplace's Demon may handle larger data sets better, and it may estimate each iteration faster, but it may also take more iterations to converge.

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.

14. Laplace's Demon Predicts Its Own Future

Following are some predictions of the future of Laplace's Demon:

1. Additional MCMC algorithms will be explored and considered for inclusion.

2. The MCMC algorithms will be attempted to be coded in C++ for faster sampling, but will remain available in R code. The user will have the option to run the algorithm in R or C++, probably with an additional command such as `C=TRUE` in the `LaplacesDemon` function.

3. The "Examples" vignette will grow as numerous examples of methods are included.

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

4. Posterior predictive checks will be attempted to be expanded to accommodate a wider variety of methods, and more checks and plotting options will be included.

The **LaplacesDemon** package is a significant contribution toward Bayesian inference in R. In turn, contributions toward the development of Laplace's Demon are welcome. Please send an email to statisticat@gmail.com with constructive criticism, reports of software bugs, or offers to contribute to Laplace's Demon.

References

Azevedo-Filho A, Shachter R (1994). "Laplace's Method Approximations for Probabilistic Inference in Belief Networks with Continuous Variables." In R˜Mantaras, D˜Poole (eds.), *Uncertainty in Artificial Intelligence*, pp. 28–36. Morgan Kauffman, San Francisco, CA.

Bayes T, 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, MA. and F.R.S." *Philosophical Transactions of the Royal Society of London*, **53**, 370–418.

Bernardo J, Smith A (2000). *Bayesian Theory*. John Wiley & Sons, West Sussex, England.

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

Gelfand A (1996). "Model Determination Using Sampling Based Methods." In W˜Gilks, S˜Richardson, D˜Spiegelhalter (eds.), *Markov Chain Monte Carlo in Practice*, pp. 145–161. Chapman & Hall, Boca Raton, FL.

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

Gelman A, Carlin J, Stern H, Rubin D (2004). *Bayesian Data Analysis*. 2nd edition. Chapman & Hall, Boca Raton, FL.

Gelman A, Meng X, Stern H (1996a). "Posterior Predictive Assessment of Model Fitness via Realized Discrepancies." *Statistica Sinica*, **6**, 773–807.

Gelman A, Roberts G, Gilks W (1996b). "Efficient Metropolis Jumping Rules." *Bayesian Statistics*, **5**, 599–608.

Gelman A, Rubin D (1992). "Inference from Iterative Simulation Using Multiple Sequences." *Statistical Science*, **7**(4), 457–472.

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

Geyer C (1992). "Practical Markov Chain Monte Carlo (with Discussion)." *Statistical Science*, **7**(4), 473–511.

Geyer C (2010). ***mcmc**: Markov Chain Monte Carlo*. R package version 0.8, URL http://www.R-project.org/package=mcmc.

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

Haario H, Saksman E, Tamminen J (2001). "An Adaptive Metropolis Algorithm." *Bernoulli*, **7**(2), 223–242.

Haario H, Saksman E, Tamminen J (2005). "Componentwise Adaptation for High Dimensional MCMC." *Computational Statistics*, **20**(2), 265–274.

Hall B (2011). ***LaplacesDemon**: Software for Bayesian Inference*. R package version 11.03.06, URL http://www.R-project.org/package=LaplacesDemon.

Hastings W (1970). "Monte Carlo Sampling Methods Using Markov Chains and Their Applications." *Biometrika*, **57**(1), 97–109.

Kerman J (2007). ***UMACS**: Universal Markov Chain Sampler*. R package version 0.924, URL http://www.R-project.org/package=UMACS.

Laplace P (1774). "Memoire sur la Probabilite des Causes par les Evenements." *l'Academie Royale des Sciences*, **6**, 621–656. English translation by S.M. Stigler in 1986 as "Memoir on the Probability of the Causes of Events" in *Statistical Science*, **1**(3), 359–378.

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

Laplace P (1814). "Essai Philosophique sur les Probabilites." English translation in Truscott, F.W. and Emory, F.L. (2007) from (1902) as "A Philosophical Essay on Probabilities". ISBN 1602063281, translated from the French 6th ed. (1840).

Lunn D, Spiegelhalter D, Thomas A, Best N (2009). "The BUGS Project: Evolution, Critique, and Future Directions." *Statistics in Medicine*, **28**, 3049–3067.

Martin A, Quinn K, Park J (2010). ***MCMCpack**: Markov chain Monte Carlo (MCMC) Package*. R package version 1.0-8, URL http://www.R-project.org/package=MCMCpack.

Metropolis N, Rosenbluth A, MN R, Teller E (1953). "Equation of State Calculations by Fast Computing Machines." *Journal of Chemical Physics*, **21**, 1087–1092.

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

Plummer M (2003). "JAGS: A Program for Analysis of Bayesian Graphical Models Using Gibbs Sampling." In *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, Rosenthal J (2001). "Optimal Scaling for Various Metropolis-Hastings Algorithms." *Statistical Science*, **16**, 351–367.

Roberts G, Rosenthal J (2007). "Examples of Adaptive MCMC." *Computational Statistics and Data Analysis*, **51**, 5467–5470.

SAS Institute Inc (2008). *SAS/STAT 9.2 User's Guide*. Cary, NC: SAS Institute Inc.

Spiegelhalter D, Thomas A, Best N, Lunn D (2003). *WinBUGS User Manual, Version 1.4*. MRC Biostatistics Unit, Institute of Public Health and Department of Epidemiology and Public Health, Imperial College School of Medicine, UK. http://www.mrc-bsu.cam.ac.uk/bugs.

Tierney L, Kadane J (1986). "Accurate Approximations for Posterior Moments and Marginal Densities." *Journal of the American Statistical Association*, **81**(393), 82–86.

Tierney L, Kass R, Kadane J (1989). "Fully Exponential Laplace Approximations to Expectations and Variances of Nonpositive Functions." *Journal of the American Statistical Association*, **84**(407), 710–716.

Affiliation:

Byron Hall
STATISTICAT, LLC
Farmington, CT
E-mail: statisticat@gmail.com
URL: http://www.statisticat.com/laplacesdemon.html