
# 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 numerous 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, or approximates, 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, Metropolis-within-Gibbs, Optimization, R, Random Walk, Random-Walk, Resilient Backpropagation, 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.

[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.

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 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 2011) 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, and discusses large data sets and speed.

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:

[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.

```
> library(LaplacesDemon)
```

## 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 $\mathbf{X}$.

```
> 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","sigma")
> parm.names <- as.parm.names(list(beta=rep(0,J), log.sigma=0))
> 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 $\mathbf{X}$. However, there are 11 parameters, since the residual variance, $\sigma^2$, must be included as well. The reason why it is called `log.sigma` 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. This is using a function called `as.parm.names`. 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. Specifying a Model

Laplace's Demon is capable of estimating 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:

$$\mathbf{y} \sim \mathcal{N}(\mu, \sigma^2)$$

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

[4]Examples of more than 70 Bayesian models may be found in the "Examples" vignette that comes with the **LaplacesDemon** package. Likelihood-free estimation is also possible by approximating the likelihood, such as in Approximate Bayesian Computation (ABC).

$$\mu = \mathbf{X}\beta$$

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

For a Bayesian model, the notation for the residual variance, $\sigma^2$, has often been replaced with the inverse of the residual precision, $\tau^{-1}$. Here, $\sigma^2$ will be used. Prior probabilities are specified for $\beta$ and $\sigma$ (the standard deviation, rather than the variance):

$$\beta_j \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J$$
$$\sigma \sim \mathcal{HC}(25)$$

Each of the $J$ $\beta$ parameters is assigned an uninformative[5] prior probability distribution that is normally-distributed according to $\mu = 0$ and $\sigma^2 = 1000$. The large variance or small precision indicates a lot of uncertainty about each $\beta$, and is hence an uninformative distribution. The residual standard deviation $\sigma$ is half-Cauchy-distributed according to its hyperparameter, scale=25.

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)
+      {
+      ### Parameters
+      beta <- parm[1:Data$J]
+      sigma <- exp(parm[Data$J+1])
+      ### Log(Prior Densities)
+      beta.prior <- dnormv(beta, 0, 1000, log=TRUE)
+      sigma.prior <- dhalfcauchy(sigma, 25, log=TRUE)
+      ### Log-Likelihood
+      mu <- tcrossprod(beta, Data$X)
+      LL <- sum(dnorm(Data$y, mu, sigma, log=TRUE))
+      ### Log-Posterior
+      LP <- LL + sum(beta.prior) + sigma.prior
+      Modelout <- list(LP=LP, Dev=-2*LL, Monitor=c(LP,sigma), yhat=mu,
+           parm=parm)
+      return(Modelout)
+      }
```

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:

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

[5] 'Non-informative' may be more widely used than 'uninformative', but here that is considered poor English, such as saying something is 'non-correct' when there's a word for that ... 'incorrect'.

where $\Theta$ is a set of parameters, $\mathbf{y}$ is the data, $\propto$ means 'proportional to'[6], $p(\Theta|\mathbf{y})$ is the joint posterior density, $p(\mathbf{y}|\Theta)$ 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, $\mathbf{y}^{rep}$ (`yhat`) or replicates of $\mathbf{y}$, 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) + sigma.prior
Modelout <- list(LP=LP, Dev=-2*LL, Monitor=c(LP,sigma),
    yhat=mu, parm=parm)
return(Modelout)
```

The rest of the function specifies the parameters, log of the prior densities, and calculates the log-likelihood. Since design matrix $\mathbf{X}$ has J=10 column vectors (including the intercept), there are 10 `beta` parameters and a `sigma` parameter for the residual standard deviation.

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 `sigma` is exponentiated will, again, be explained later.

```
beta <- parm[1:Data$J]
sigma <- exp(parm[Data$J+1])
```

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

```
beta.prior <- dnormv(beta, 0, 1000, log=TRUE)
sigma.prior <- dhalfcauchy(sigma, 25, 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 $\infty$, and be transformed in the `Model` function with the `exp` function, which will force it to positive values. A parameter $\theta$ 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, such as with the `interval` 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

[6]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'.

above example, the residual standard deviation `sigma` receives a half-Cauchy distributed prior of the form:

$$\sigma \sim \mathcal{HC}(25)$$

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

```
sigma <- exp(parm[Data$J+1])
```

Laplace's Demon will increase or decrease `parm[Data$J+1]`, which is effectively `log(sigma)`. Now it is possible for Laplace's Demon to decrease `log(sigma)` below zero without causing an error or violating its half-Cauchy 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 transpose of the design matrix, `Data$X`. Expectation vector `mu`, vector `Data$y`, and scalar `sigma` are used to estimate the sum of the log-likelihoods, where:

$$\mathbf{y} \sim \mathcal{N}(\mu, \sigma^2)$$

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

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

```
mu <- tcrossprod(beta, Data$X)
LL <- sum(dnorm(Data$y, mu, sigma, log=TRUE)
LP <- LL + sum(beta.prior) + sigma.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 are also easy to estimate (see the "Examples" vignette).

## 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 resilient backpropagation 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, such as with the `GIV` function (which stands for "generate initial values"), 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.sigma`, has been set equal to `log(1)`, which is equal to zero. This visually reminds me that I am working with the log of `sigma`, rather than `sigma`, 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[7]:

```
> 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=2,
+      Covar=NULL, DR=0, Gibbs=TRUE, Initial.Values, Iterations=2000,
+      Periodicity=10, Status=100, Thinning=2)
```

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 numerous 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 Adaptive Metropolis-within-Gibbs (AMWG) 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=2` argument indicates that a non-adaptive MCMC algorithm will begin, and that it will become adaptive at the 2nd iteration. Beginning with the 2nd iteration, the MCMC algorithm will estimate the proposal variance or covariance based on the history of the chains.

[7]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'.

- 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=0` argument indicates that delayed rejection will not occur. Delayed rejection means 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 `Gibbs` argument allows Metropolis-within-Gibbs and an adaptive version of the MCMC algorithm. In this case the AMWG algorithm is selected.
- The `Iterations=2000` argument indicates that the `LaplacesDemon` function will update 2,000 times before completion.
- The `Periodicity=10` argument indicates that once adaptation begins, the algorithm will adapt every 10 iterations.
- The `Status=100` argument indicates that a status message will be printed to the R console every 100 iterations.
- Finally, the `Thinning=2` argument indicates that only every `n`th iteration will be retained in the output, and in this case, every 2nd iteration will be retained.

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

```
> Fit <- LaplacesDemon(Model, Data=MyData, Adaptive=2,
+      Covar=NULL, DR=0, Gibbs=TRUE, Initial.Values, Iterations=2000,
+      Periodicity=10, Status=100, Thinning=2)

Laplace's Demon was called on Mon Jan  2 10:27:53 2012

Performing initial checks...
Algorithm: Adaptive Metropolis-within-Gibbs

Laplace's Demon is beginning to update...
Iteration: 100,   Proposal: Componentwise
Iteration: 200,   Proposal: Componentwise
Iteration: 300,   Proposal: Componentwise
Iteration: 400,   Proposal: Componentwise
Iteration: 500,   Proposal: Componentwise
Iteration: 600,   Proposal: Componentwise
Iteration: 700,   Proposal: Componentwise
Iteration: 800,   Proposal: Componentwise
Iteration: 900,   Proposal: Componentwise
Iteration: 1000,   Proposal: Componentwise
Iteration: 1100,   Proposal: Componentwise
Iteration: 1200,   Proposal: Componentwise
Iteration: 1300,   Proposal: Componentwise
```

```
Iteration: 1400,   Proposal: Componentwise
Iteration: 1500,   Proposal: Componentwise
Iteration: 1600,   Proposal: Componentwise
Iteration: 1700,   Proposal: Componentwise
Iteration: 1800,   Proposal: Componentwise
Iteration: 1900,   Proposal: Componentwise
Iteration: 2000,   Proposal: Componentwise

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 100 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] 0.4406364
```

## 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 = 2, Covar = NULL,
    DR = 0, Gibbs = TRUE, Initial.Values = Initial.Values, Iterations = 2000,
    Periodicity = 10, Status = 100, Thinning = 2)
```

```
Acceptance Rate: 0.44064
Adaptive: 2
Algorithm: Adaptive Metropolis-within-Gibbs
Covariance Matrix: (NOT SHOWN HERE; diagonal shown instead)
 [1] 0.11470712 0.12249620 1.36079085 0.69123544 0.29018869 0.08787558
 [7] 0.32307519 0.20677011 0.31359374 0.69709716 0.03421850

Covariance (Diagonal) History: (NOT SHOWN HERE)
Deviance Information Criterion (DIC):
         All Stationary
Dbar  91.110     86.696
pD   752.335     80.389
DIC  843.446    167.085

Delayed Rejection (DR): 0
Initial Values:
 [1] 0 0 0 0 0 0 0 0 0 0 0

Iterations: 2000
Log(Marginal Likelihood): NA
Minutes of run-time: 0.12
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: 602
Recommended Thinning: 34
Status is displayed every 100 iterations
Summary1: (SHOWN BELOW)
Summary2: (SHOWN BELOW)
Thinned Samples: 1000
Thinning: 2


Summary of All Samples
                 Mean         SD       MCSE       ESS            LB
beta[1]     5.01333926  0.3485819 0.04342861 123.28099    4.76118097
beta[2]    -0.46085623  0.3540413 0.04339323 162.05891   -1.15394353
beta[3]    -0.30385661  1.1685517 0.34375550  11.54950   -2.61611565
beta[4]    -0.31832145  0.8266525 0.22483936  18.58697   -2.38624906
beta[5]    -0.48271087  0.5369613 0.13853477  19.41756   -1.62145652
beta[6]    -0.47286420  0.2949916 0.04035201  99.20047   -0.98156572
beta[7]     2.20373840  0.5720556 0.10474617  52.98047    0.93795773
```

```
beta[8]     0.50982398  0.4549001 0.08549895  48.16421   -0.72035581
beta[9]     0.03776386  0.5591081 0.13142492  36.48884   -0.93934968
beta[10]    1.73527669  0.8376247 0.23866312  11.20771    0.08866425
log.sigma  -0.33478862  0.1863913 0.02840774  68.01568   -0.59065929
Deviance   91.11032920 38.7900862 3.66583547 206.97686   74.59419602
LP        -92.97403533 19.3948044 1.83303533 206.93889 -114.26029399
sigma       0.76078296  0.3358749 0.03516219 149.65478    0.39776576
                Median            UB
beta[1]     5.043242360   5.283911359
beta[2]    -0.456945375   0.278934130
beta[3]    -0.344515807   2.912109702
beta[4]    -0.232351687   1.011476676
beta[5]    -0.433388552   0.444862550
beta[6]    -0.487614292   0.122267380
beta[7]     2.254415102   3.154144701
beta[8]     0.563059029   1.249619795
beta[9]    -0.002924302   1.248272270
beta[10]    1.669790179   3.661925411
log.sigma  -0.350451254  -0.003628516
Deviance   85.297321118 133.689535115
LP        -90.066380372 -84.716891299
sigma       0.702365369   1.467313475


Summary of Stationary Samples
                  Mean         SD        MCSE       ESS            LB
beta[1]     5.05138850  0.1188242 0.009815896 252.65205    4.83271146
beta[2]    -0.48876968  0.3204360 0.040466365 124.27956   -1.15545494
beta[3]    -0.72031344  0.8647957 0.265173771  18.83299   -2.99390638
beta[4]    -0.06767355  0.6185672 0.169350843  21.17608   -1.18953278
beta[5]    -0.65955324  0.5012936 0.140973814  17.93399   -1.84178086
beta[6]    -0.51415349  0.2626651 0.035795446 124.98430   -1.01324605
beta[7]     2.27508708  0.4551870 0.078934662  50.56391    1.36866171
beta[8]     0.57399833  0.3398663 0.048858758  88.52791   -0.04520375
beta[9]    -0.03307469  0.5247923 0.134121218  27.74488   -0.94914524
beta[10]    2.04448163  0.7178201 0.221814102  13.29186    0.90311422
log.sigma  -0.36168447  0.1180212 0.011238542 200.57022   -0.58892643
Deviance   86.69618239 12.6798486 0.874729783 371.30375   74.10781068
LP        -90.76723686  6.3410014 0.437838835 370.90815 -104.82277676
sigma       0.71511490  0.2045537 0.013582220 318.80022    0.40933863
                Median           UB
beta[1]     5.04594662   5.28698508
beta[2]    -0.46015548   0.10554755
beta[3]    -0.57417352   0.53604143
beta[4]    -0.12814615   1.12753625
beta[5]    -0.62732289   0.26513828
beta[6]    -0.51021503  -0.01776986
```

```
beta[7]      2.27862675   3.13713212
beta[8]      0.57904477   1.30257470
beta[9]     -0.05837423   1.09966968
beta[10]     1.95597395   3.75706104
log.sigma   -0.35495218  -0.12305837
Deviance    84.10755633 114.81504553
LP         -89.47308409 -84.47088496
sigma        0.68614813   1.18304940
```

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 2000 iterations of 11 variables in 0.12 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 = 2, Covar = NULL,
    DR = 0, Gibbs = TRUE, Initial.Values = Initial.Values, Iterations = 2000,
    Periodicity = 10, Status = 100, Thinning = 2)

Acceptance Rate: 0.44064
Adaptive: 2
Algorithm: Adaptive Metropolis-within-Gibbs
Covariance Matrix: (NOT SHOWN HERE; diagonal shown instead)
```

```
 [1] 0.11470712 0.12249620 1.36079085 0.69123544 0.29018869 0.08787558
 [7] 0.32307519 0.20677011 0.31359374 0.69709716 0.03421850

Covariance (Diagonal) History: (NOT SHOWN HERE)
Deviance Information Criterion (DIC):
         All Stationary
Dbar  91.110     86.696
pD   752.335     80.389
DIC  843.446    167.085

Delayed Rejection (DR): 0
Initial Values:
 [1] 0 0 0 0 0 0 0 0 0 0 0

Iterations: 2000
Log(Marginal Likelihood): NA
Minutes of run-time: 0.12
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: 602
Recommended Thinning: 34
Status is displayed every 100 iterations
Summary1: (SHOWN BELOW)
Summary2: (SHOWN BELOW)
Thinned Samples: 1000
Thinning: 2


Summary of All Samples
                 Mean         SD       MCSE       ESS           LB
beta[1]     5.01333926  0.3485819 0.04342861 123.28099    4.76118097
beta[2]    -0.46085623  0.3540413 0.04339323 162.05891   -1.15394353
beta[3]    -0.30385661  1.1685517 0.34375550  11.54950   -2.61611565
beta[4]    -0.31832145  0.8266525 0.22483936  18.58697   -2.38624906
beta[5]    -0.48271087  0.5369613 0.13853477  19.41756   -1.62145652
beta[6]    -0.47286420  0.2949916 0.04035201  99.20047   -0.98156572
beta[7]     2.20373840  0.5720556 0.10474617  52.98047    0.93795773
beta[8]     0.50982398  0.4549001 0.08549895  48.16421   -0.72035581
beta[9]     0.03776386  0.5591081 0.13142492  36.48884   -0.93934968
beta[10]    1.73527669  0.8376247 0.23866312  11.20771    0.08866425
log.sigma  -0.33478862  0.1863913 0.02840774  68.01568   -0.59065929
Deviance   91.11032920 38.7900862 3.66583547 206.97686   74.59419602
```

```
LP        -92.97403533 19.3948044 1.83303533 206.93889 -114.26029399
sigma       0.76078296  0.3358749 0.03516219 149.65478    0.39776576
                 Median            UB
beta[1]     5.043242360   5.283911359
beta[2]    -0.456945375   0.278934130
beta[3]    -0.344515807   2.912109702
beta[4]    -0.232351687   1.011476676
beta[5]    -0.433388552   0.444862550
beta[6]    -0.487614292   0.122267380
beta[7]     2.254415102   3.154144701
beta[8]     0.563059029   1.249619795
beta[9]    -0.002924302   1.248272270
beta[10]    1.669790179   3.661925411
log.sigma  -0.350451254  -0.003628516
Deviance   85.297321118 133.689535115
LP        -90.066380372 -84.716891299
sigma       0.702365369   1.467313475


Summary of Stationary Samples
                  Mean         SD        MCSE       ESS             LB
beta[1]     5.05138850  0.1188242 0.009815896 252.65205    4.83271146
beta[2]    -0.48876968  0.3204360 0.040466365 124.27956   -1.15545494
beta[3]    -0.72031344  0.8647957 0.265173771  18.83299   -2.99390638
beta[4]    -0.06767355  0.6185672 0.169350843  21.17608   -1.18953278
beta[5]    -0.65955324  0.5012936 0.140973814  17.93399   -1.84178086
beta[6]    -0.51415349  0.2626651 0.035795446 124.98430   -1.01324605
beta[7]     2.27508708  0.4551870 0.078934662  50.56391    1.36866171
beta[8]     0.57399833  0.3398663 0.048858758  88.52791   -0.04520375
beta[9]    -0.03307469  0.5247923 0.134121218  27.74488   -0.94914524
beta[10]    2.04448163  0.7178201 0.221814102  13.29186    0.90311422
log.sigma  -0.36168447  0.1180212 0.011238542 200.57022   -0.58892643
Deviance   86.69618239 12.6798486 0.874729783 371.30375   74.10781068
LP        -90.76723686  6.3410014 0.437838835 370.90815 -104.82277676
sigma       0.71511490  0.2045537 0.013582220 318.80022    0.40933863
                Median           UB
beta[1]     5.04594662   5.28698508
beta[2]    -0.46015548   0.10554755
beta[3]    -0.57417352   0.53604143
beta[4]    -0.12814615   1.12753625
beta[5]    -0.62732289   0.26513828
beta[6]    -0.51021503  -0.01776986
beta[7]     2.27862675   3.13713212
beta[8]     0.57904477   1.30257470
beta[9]    -0.05837423   1.09966968
beta[10]    1.95597395   3.75706104
log.sigma  -0.35495218  -0.12305837
```

```
Deviance   84.10755633 114.81504553
LP        -89.47308409 -84.47088496
sigma       0.68614813   1.18304940

Demonic Suggestion

Due to the combination of the following conditions,

1. Adaptive Metropolis-within-Gibbs
2. The acceptance rate (0.4406364) is within the interval [0.15,0.5].
3. At least one target MCSE is >= 6.27% of its marginal posterior
   standard deviation.
4. At least one target distribution has an effective sample size
   (ESS) less than 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 <- as.initial.values(Fit)
Fit <- LaplacesDemon(Model, Data=MyData, Adaptive=2,
     Covar=Fit$Covar, DR=0, Gibbs=TRUE, Initial.Values, Iterations=34000,
     Mixture=FALSE, Periodicity=77, Status=16666, Thinning=34)

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. 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 harder to estimate for some adaptive algorithms that calculate the observed covariance matrix for proposals, such as AM and DRAM (explained later). 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 consid-

ered satisfactory if it is within the interval [15%,50%][8]. 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.

In an effort to save several pages in this tutorial due to output, the following will be run instead:

```
> Initial.Values <- as.initial.values(Fit)
> Fit <- LaplacesDemon(Model, Data=MyData, Adaptive=0,
+      Covar=Fit$Covar, DR=0, Gibbs=TRUE, Initial.Values, Iterations=70000,
+      Periodicity=0, Status=10000, Thinning=70)

Laplace's Demon was called on Mon Jan  2 10:28:00 2012

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

Laplace's Demon is beginning to update...
Iteration: 10000,   Proposal: Componentwise
Iteration: 20000,   Proposal: Componentwise
Iteration: 30000,   Proposal: Componentwise
Iteration: 40000,   Proposal: Componentwise
Iteration: 50000,   Proposal: Componentwise
Iteration: 60000,   Proposal: Componentwise
Iteration: 70000,   Proposal: Componentwise

Assessing Stationarity
Assessing Thinning and ESS
Creating Summaries
Estimating Log of the Marginal Likelihood
Creating Output

Laplace's Demon has finished.
```

Next, the user consorts with Laplace's Demon:

```
> Consort(Fit)

#############################################################
# Consort with Laplace's Demon                              #
```

[8]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%].

```
############################################################
Call:
LaplacesDemon(Model = Model, Data = MyData, Adaptive = 0, Covar = Fit$Covar,
    DR = 0, Gibbs = TRUE, Initial.Values = Initial.Values, Iterations = 70000,
    Periodicity = 0, Status = 10000, Thinning = 70)

Acceptance Rate: 0.43583
Adaptive: 70001
Algorithm: Metropolis-within-Gibbs
Covariance Matrix: (NOT SHOWN HERE; diagonal shown instead)
 [1] 0.11470712 0.12249620 1.36079085 0.69123544 0.29018869 0.08787558
 [7] 0.32307519 0.20677011 0.31359374 0.69709716 0.03421850

Covariance (Diagonal) History: (NOT SHOWN HERE)
Deviance Information Criterion (DIC):
         All Stationary
Dbar  84.703     84.703
pD    31.768     31.768
DIC  116.470    116.470

Delayed Rejection (DR): 0
Initial Values:
 [1]  5.0735946 -0.3742266 -1.7611446  0.3301105 -1.8785324 -0.4095678
 [7]  2.1972976  0.1917235  0.2985861  3.4574798 -0.6298484

Iterations: 70000
Log(Marginal Likelihood): -41.68301
Minutes of run-time: 2.53
Model: (NOT SHOWN HERE)
Monitor: (NOT SHOWN HERE)
Parameters (Number of): 11
Periodicity: 70001
Posterior1: (NOT SHOWN HERE)
Posterior2: (NOT SHOWN HERE)
Recommended Burn-In of Thinned Samples: 1
Recommended Burn-In of Un-thinned Samples: 70
Recommended Thinning: 210
Status is displayed every 10000 iterations
Summary1: (SHOWN BELOW)
Summary2: (SHOWN BELOW)
Thinned Samples: 1000
Thinning: 70


Summary of All Samples
                  Mean        SD        MCSE       ESS          LB       Median
beta[1]     5.04351639 0.1128707 0.003524512 1000.0000   4.8236801   5.04783872
```

```
beta[2]    -0.42797081 0.3621088 0.011910929 1000.0000  -1.1358425  -0.43503172
beta[3]    -0.39195740 0.9109941 0.054511932  443.0769  -2.1572129  -0.41273640
beta[4]    -0.08204913 0.6817360 0.031460631  659.4674  -1.4246879  -0.08824405
beta[5]    -0.37925542 0.5100626 0.024898196  617.9939  -1.4420904  -0.39265828
beta[6]    -0.47928014 0.2878761 0.010145029  748.4238  -1.0543989  -0.48106506
beta[7]     2.24187327 0.5303255 0.019293292  660.3233   1.1944379   2.23778639
beta[8]     0.59563692 0.4221134 0.016590344  797.3048  -0.2584061   0.61625235
beta[9]    -0.22120068 0.5629598 0.021316945  848.8130  -1.3256004  -0.20721067
beta[10]    1.58559749 0.7508782 0.040624703  508.0161   0.1664417   1.56867578
log.sigma  -0.36387600 0.1305048 0.003930595 1000.0000  -0.5960592  -0.36972945
Deviance   84.70267533 7.9708939 0.256802025 1000.0000  74.2432287  83.13175947
LP        -89.76947212 3.9862357 0.128439917 1000.0000 -98.7227440 -88.98387376
sigma       0.71007324 0.1655599 0.005060293 1000.0000   0.4545922   0.69351154
                   UB
beta[1]     5.26098699
beta[2]     0.24633178
beta[3]     1.46238452
beta[4]     1.19991923
beta[5]     0.60975482
beta[6]     0.07545473
beta[7]     3.25485109
beta[8]     1.39719718
beta[9]     0.87834530
beta[10]    3.04187803
log.sigma  -0.09954816
Deviance  102.60201396
LP        -84.53895655
sigma       1.10063565


Summary of Stationary Samples
                  Mean        SD        MCSE       ESS          LB       Median
beta[1]     5.04351639 0.1128707 0.003524512 1000.0000   4.8236801   5.04783872
beta[2]    -0.42797081 0.3621088 0.011910929 1000.0000  -1.1358425  -0.43503172
beta[3]    -0.39195740 0.9109941 0.054511932  443.0769  -2.1572129  -0.41273640
beta[4]    -0.08204913 0.6817360 0.031460631  659.4674  -1.4246879  -0.08824405
beta[5]    -0.37925542 0.5100626 0.024898196  617.9939  -1.4420904  -0.39265828
beta[6]    -0.47928014 0.2878761 0.010145029  748.4238  -1.0543989  -0.48106506
beta[7]     2.24187327 0.5303255 0.019293292  660.3233   1.1944379   2.23778639
beta[8]     0.59563692 0.4221134 0.016590344  797.3048  -0.2584061   0.61625235
beta[9]    -0.22120068 0.5629598 0.021316945  848.8130  -1.3256004  -0.20721067
beta[10]    1.58559749 0.7508782 0.040624703  508.0161   0.1664417   1.56867578
log.sigma  -0.36387600 0.1305048 0.003930595 1000.0000  -0.5960592  -0.36972945
Deviance   84.70267533 7.9708939 0.256802025 1000.0000  74.2432287  83.13175947
LP        -89.76947212 3.9862357 0.128439917 1000.0000 -98.7227440 -88.98387376
sigma       0.71007324 0.1655599 0.005060293 1000.0000   0.4545922   0.69351154
                   UB
```

```
beta[1]     5.26098699
beta[2]     0.24633178
beta[3]     1.46238452
beta[4]     1.19991923
beta[5]     0.60975482
beta[6]     0.07545473
beta[7]     3.25485109
beta[8]     1.39719718
beta[9]     0.87834530
beta[10]    3.04187803
log.sigma  -0.09954816
Deviance  102.60201396
LP        -84.53895655
sigma       1.10063565

Demonic Suggestion

Due to the combination of the following conditions,

1. Metropolis-within-Gibbs
2. The acceptance rate (0.4358325) 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
   1 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 2.53 minutes, Laplace's Demon updated 70000 iterations, retaining every 70th iteration due to thinning, and reported an acceptance rate of 0.436. Notice that all criteria have been met: MCSEs are sufficiently small, ESSs are sufficiently large, and all parameters were estimated to be stationary. Since the algorithm was the non-adaptive Metropolis-within-Gibbs (MWG), 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 own customized plots with `demonoid` objects. The variable `BurnIn` (below) may be left as it is so it will show only the stationary samples (samples that are no longer trending), or set equal to one so that all

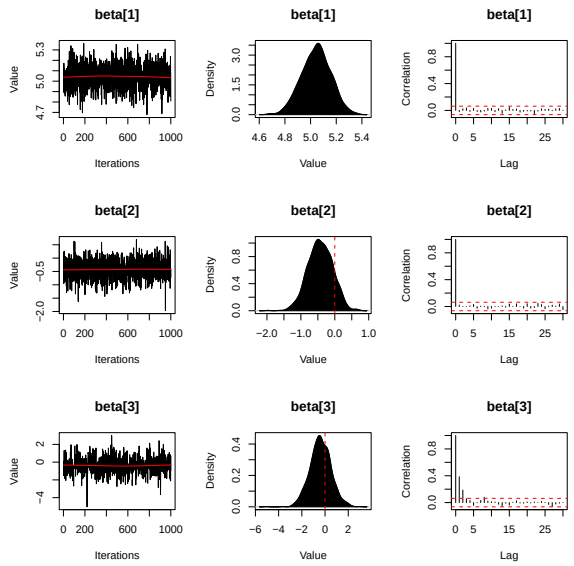


Figure 1: Plots of Marginal Posterior Samples

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 `sigma` 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),

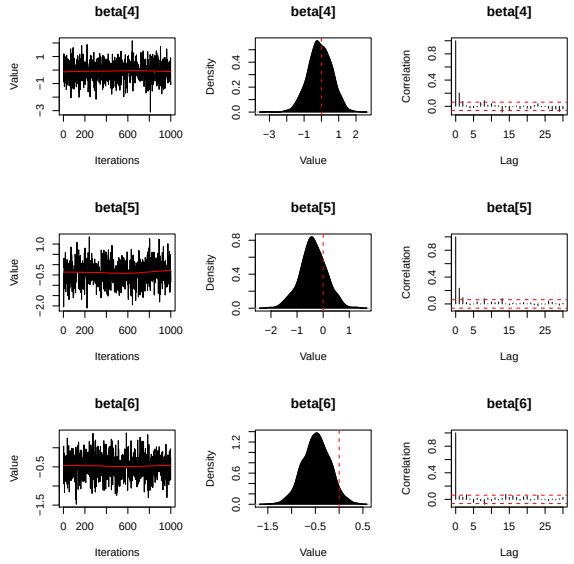


Figure 2: Plots of Marginal Posterior Samples

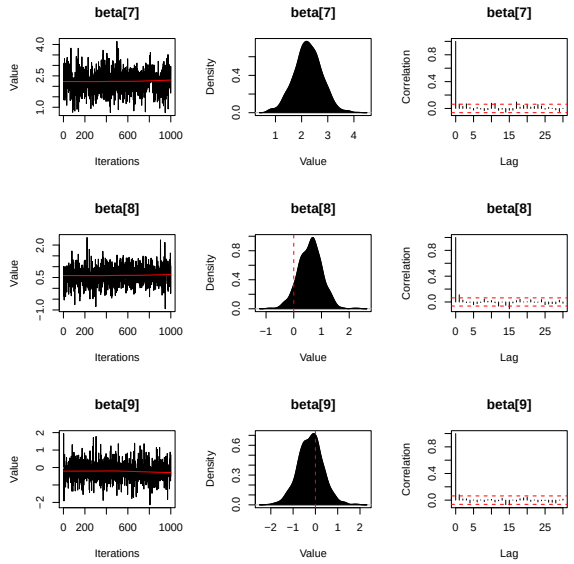


Figure 3: Plots of Marginal Posterior Samples

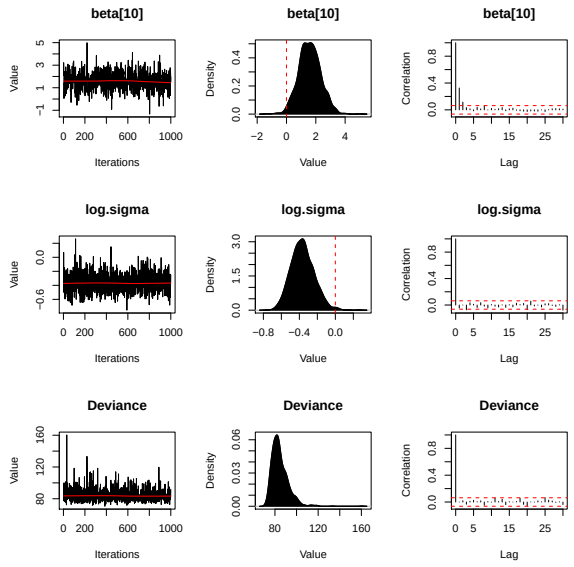


Figure 4: Plots of Marginal Posterior Samples

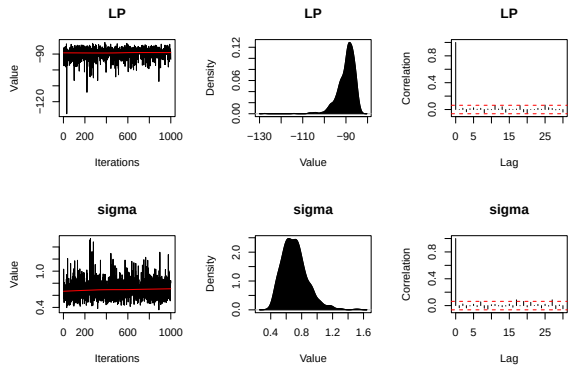


Figure 5: Plots of Marginal Posterior Samples

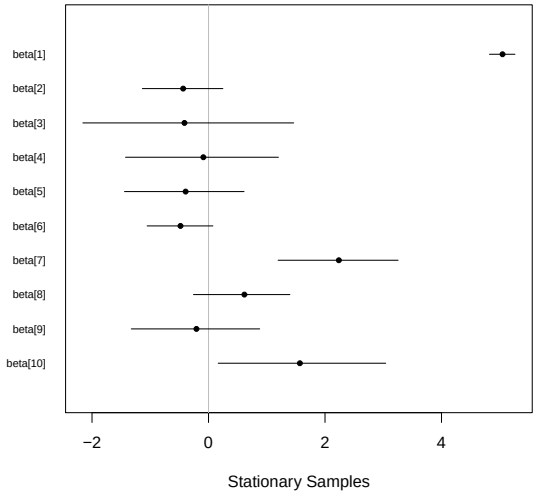


Figure 6: Caterpillar Plot

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 (such as visual inspections with the `CSF` or Cumulative Sample Function, introduced in the **LaplacesDemon** package), and finally, further assessments of model fit should be estimated with posterior predictive checks, showing how well (or poorly) the model fits the data. When the user is satisfied, 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 $\mathbf{y}^{rep}$) are estimated. If a new data set is supplied in `predict`, then new, unobserved instances of `y` (called $\mathbf{y}^{new}$) 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 $\mathbf{y}^{rep}$, 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, Discrep="Chi-Square")

Concordance:  0.7948718
Discrepancy Statistic:  305.872
L-criterion: 22.099, S.L: 0.393
Records:
          y  Mean    SD    LB Median    UB    PQ Discrep
1  4.174387 4.177 0.201 3.794  4.176 4.574 0.503   0.000
2  5.361292 5.269 0.407 4.505  5.267 6.063 0.405   0.051
3  6.089045 5.248 0.528 4.198  5.255 6.254 0.056   2.536
4  5.298317 5.138 0.335 4.526  5.137 5.817 0.319   0.231
5  4.406719 4.085 0.238 3.598  4.094 4.528 0.095   1.822
6  2.197225 3.809 0.198 3.398  3.812 4.191 1.000  66.216
7  5.010635 4.537 0.179 4.192  4.539 4.883 0.003   6.978
8  1.609438 3.865 0.196 3.469  3.866 4.221 1.000 132.880
9  4.343805 4.237 0.241 3.788  4.237 4.707 0.320   0.196
10 4.812184 4.702 0.216 4.270  4.713 5.135 0.291   0.263
11 4.189655 4.398 0.190 4.020  4.398 4.774 0.875   1.200
12 4.919981 4.530 0.171 4.181  4.527 4.854 0.008   5.214
13 4.753590 4.377 0.177 4.021  4.378 4.706 0.012   4.542
14 4.127134 4.164 0.175 3.831  4.165 4.503 0.583   0.045
15 3.713572 4.091 0.191 3.699  4.089 4.445 0.974   3.921
16 4.672829 4.400 0.234 3.970  4.398 4.862 0.117   1.362
17 6.930495 7.136 0.513 6.157  7.119 8.142 0.657   0.160
18 5.068904 4.786 0.240 4.313  4.794 5.237 0.111   1.381
19 6.775366 6.327 0.467 5.397  6.315 7.229 0.167   0.921
20 6.553933 7.187 0.462 6.292  7.179 8.065 0.912   1.878
21 4.890349 5.362 0.349 4.693  5.363 6.045 0.912   1.822
22 4.442651 4.267 0.279 3.745  4.258 4.821 0.257   0.399
23 2.833213 3.124 0.474 2.170  3.128 4.047 0.747   0.377
24 4.787492 4.926 0.247 4.438  4.922 5.419 0.712   0.314
25 6.933423 7.253 0.606 6.067  7.238 8.577 0.695   0.278
26 6.180017 6.067 0.583 4.873  6.077 7.249 0.430   0.038
27 5.652489 5.328 0.312 4.713  5.321 5.975 0.149   1.078
28 5.429346 4.462 0.205 4.049  4.461 4.867 0.000  22.172
29 5.634790 5.553 0.687 4.126  5.567 6.851 0.459   0.014
30 4.262680 4.064 0.204 3.631  4.065 4.444 0.164   0.945
31 3.891820 4.078 0.255 3.581  4.075 4.571 0.771   0.534
32 6.613384 6.603 0.377 5.854  6.612 7.339 0.497   0.001
33 4.919981 4.408 0.180 4.048  4.412 4.752 0.001   8.082
34 6.541030 6.457 0.505 5.460  6.461 7.424 0.433   0.028
35 6.345636 6.434 0.479 5.484  6.430 7.372 0.561   0.034
```

```
36 3.737670 4.070 0.246 3.572  4.074 4.542 0.911   1.832
37 7.356280 7.889 0.626 6.693  7.893 9.164 0.814   0.723
38 5.739793 4.762 0.170 4.440  4.755 5.111 0.000  33.029
39 5.517453 5.127 0.253 4.617  5.134 5.600 0.055   2.375
```

The `summary.demonoid.ppc` function returns a list with 4 components when `y` is continuous (different output occurs for categorical dependent variables when given the argument `Categorical=TRUE`):

- `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, the specified discrepancy measure was the $\chi^2$ test in Gelman *et˜al.* (2004, p. 175), and higher values indicate a worse fit.

- `L-criterion` is a posterior predictive check for model and variable selection that measures the distance between $\mathbf{y}$ and $\mathbf{y}^{rep}$, providing a criterion to be minimized (Laud and Ibrahim 1995).

- 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 $\mathbf{y}_1^{rep}$, given the model and data, is 4.177. Most importantly, `PQ[1]` is 0.503, indicating that 50.3% 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.

There are also a variety of plots for posterior predictive checks, and the type of plot is controlled with the `Style` argument. Many styles exist, such as producing plots of covariates and residuals. The last component of this summary may be viewed graphically as posterior densities. Rather than observing plots for each of 39 records or rows, only the first 9 densities will be shown here:

```
> plot(Pred, Style="Density", Rows=1:9)
```

The `Importance` function is not presented here in detail, but is a useful way to assess variable importance, which is defined here as the impact of each variable on $\mathbf{y}^{rep}$, when the variable is removed (or set to zero). Variable importance consists of differences in discrepancy statistics,

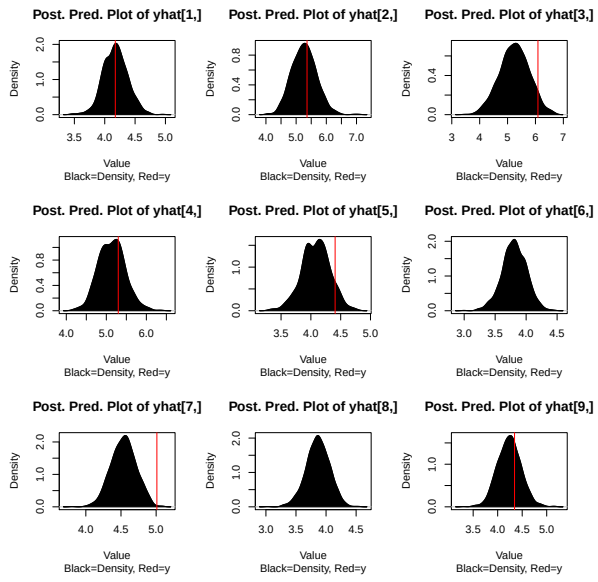


Figure 7: Posterior Predictive Plots

showing how well the model fits the data with each variable removed. This information may be used for model revision, or presenting the relative importance of variables.

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 2nd iteration (`Adaptive=2`), 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 with the `GIV` function 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.

- When speed is a concern, such as with complex models, there may be things in the `Model` function that can be commented out, such as sometimes calculating `yhat`. The

model can be updated without some features, that can be un-commented and used for posterior predictive checks. By commenting out things that are strictly unnecessary to updating, the model will update more quickly.

- 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 an adaptive algorithm (usually AMWG, AMM, AM, or DRAM) 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) See the `ESS` function for more information.

- 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 (simultaneously, with different sessions), each beginning with different initial values, until multiple output objects of class `demonoid` exist with stationary samples, if time allows, and the `Gelman.Diagnostic` function may be used to compare multiple chains.

## 10. Independence and Observability

Laplace's Demon 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 requires only base R. The variety of packages makes R extremely attractive. However, depending on multiple packages can be problematic when a change is made in one package, but other packages do not keep pace, and the user is dependent on packages being in synch. By avoiding dependencies on other packages, Laplace's Demon is attempting to be consistent and dependable for the user.

For example, common MCMC diagnostics and probability distributions (such as Dirichlet, multivariate normal, Wishart, and many others, as well as truncated forms of distributions) in Bayesian inference have been included in **LaplacesDemon** so the user does not have to load numerous R packages.

By observability, it is meant that Laplace's Demon is written entirely in R. Certain functions could be sped up in another language, but this may prevent some R users from understanding the code. Laplace's Demon is intended to be open and accessible. If a user desires speed and is familiar with a faster language, then the user is encouraged to program the model specification function in the faster language. See the documentation for the `Model.Spec.Time` function for more information. Moreover, it is demonstrated in the section 13 that Laplace's Demon is often significantly faster than other MCMC software that was programmed in faster languages, and users are encouraged to time comparisons, especially with large samples.

Observability also enables users to investigate or customize functions in Laplace's Demon. To access any function, simply enter the function name and press enter. For example, to print the code for **LaplacesDemon** to the R console, simply enter:

```
> LaplacesDemon
```

To access undocumented, internal-only functions, use the `:::` operator, such as:

```
> LaplacesDemon:::RWM
```

Laplace's Demon seeks to provide a complete, Bayesian environment within R. Independence from other software facilitates dependability, and its open code makes it easier for a user to investigate and customize.

# 11. Details

The **LaplacesDemon** package uses two broad types of numerical approximation algorithms: Laplace Approximation and Markov chain Monte Carlo (MCMC), and Approximate Bayesian Computation (ABC) may be estimated within each. Each is described below, but MCMC is emphasized.

## 11.1. Approximate Bayesian Computation

Approximate Bayesian Computation (ABC), also called likelihood-free estimation, is a family of numerical approximation techniques in Bayesian inference. ABC is especially useful when evaluation of the likelihood, $p(\mathbf{y}|\Theta)$ is computationally prohibitive, or when suitable likelihoods are unavailable. As such, ABC algorithms estimate likelihood-free approximations. ABC is usually faster than a similar likelihood-based numerical approximation technique, because the likelihood is not evaluated directly, but replaced with an approximation that is usually easier to calculate. The approximation of a likelihood is usually estimated with a measure of distance between the observed sample, $\mathbf{y}$, and its replicate given the model, $\mathbf{y}^{rep}$, or with summary statistics of the observed and replicated samples. See the accompanying vignette entitled "Examples" for an example.

## 11.2. 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).

The speed of Laplace Approximation depends on the optimization algorithm selected, and typically involves many evaluations of the objective function per iteration (where the AMM MCMC algorithm evaluates once per iteration), making most of the MCMC algorithms faster per iteration. The attractiveness of Laplace Approximation is that it typically improves the objective function better than MCMC when the parameters are in low-probability regions (in which MCMC algorithms may suffer unreasonably low acceptance rates) until an adaptive MCMC has "learned" how to move better. Laplace Approximation is also typically faster because it is seeking point-estimates, rather than attempting to represent the target distribution with enough simulation draws. 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.

`LaplaceApproximation` seeks a global maximum of the logarithm of the unnormalized joint posterior density. The approach differs by `Method`. The `LaplacesDemon` function uses the `LaplaceApproximation` algorithm to optimize initial values, estimate covariance, and save time for the user.

Most optimization algorithms assume 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.

### *Adaptive Gradient Ascent*

With adaptive gradient ascent, 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-Raphson, an advantage of the gradient ascent is that it works in infinite dimensions, requiring only sufficient computer memory. Although Newton-Raphson 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..

*Limited-Memory BFGS*

The limited-memory BFGS (Broyden-Fletcher-Goldfarb-Shanno) algorithm is a quasi-Newton optimization algorithm that compactly approximates the Hessian matrix. Rather than storing the dense Hessian matrix, L-BFGS stores only a few vectors that represent the approximation. This algorithm is better suited for large-scale models than the BFGS algorithm. This is the default algorithm (`method="LBFGS"`) for `LaplaceApproximation`, which calls `method="L-BFGS-B"` in the `optim` function of base R.

*Resilient Backpropagation*

"Rprop" stands for resilient backpropagation. In Rprop, the approximate gradient is taken for each parameter in each iteration, and its sign is compared to the approximate gradient in the previous iteration. A weight element in a weight vector is associated with each approximate gradient. A weight element is multiplied by 1.2 when the sign does not change, or by 0.5 if the sign changes. The weight vector is the step size, and is constrained to the interval [0.001, 50], and initial weights are 0.0125. This is the resilient backpropagation algorithm, which is often denoted as the "Rprop-" algorithm of Riedmiller (1994).

*Afterward*

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, then a scaled identity matrix is returned, and each parameter's variance will be 1.

## 11.3. 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, such as Metropolis-within-Gibbs (Tierney 1994). 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 un-

normalized 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 sampling of conjugate distributions, so it is precluded from non-conjugate sampling in its purest form. Gibbs sampling also suffers under high correlations (Gilks and Roberts 1996). 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 (even in the Metropolis-within-Gibbs algorithms, where block updating is popular), 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[9] or candidate, of the joint posterior density is calculated,

[9]Laplace's Demon allows the user to constrain proposals in the `Model` function. Laplace's Demon generates

and a ratio of the proposed to the current joint posterior density, called $\alpha$, 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.

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. Most 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, or if the acceptance rate is less than 5%, 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%.

---

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.

*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. Considered as a non-adaptive algorithm, it is acceptable to conclude model updates with this algorithm, rather than following up with RWM.

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.

Each problem is different, and this can be a useful algorithm. In general, however, it is more likely that other algorithms are used.

*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[10]. 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 optimize the acceptance rate, 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 $2.4^2/d$[11] based on the asymptotic limit of infinite-dimensional Gaussian target distributions that are independent and identically-

[10] 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.

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

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. It has been tested in Laplace's Demon with as many as 2,600 parameters, so it is capable of large-scale Bayesian inference, but you would not want to depend on it in a timely manner. 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, or when the acceptance rate is less than 5%, it defaults to random-scan single-component adaptive proposals (Haario, Saksman, and Tamminen 2005). Although single-component adaptive proposals should take more iterations 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:

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

where $\phi_K$ 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:

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

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

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 when sample 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.

The AM algorithm performs very well in practice. The Adaptive-Mixture Metropolis (AMM) of Roberts and Rosenthal (2009) is an extension of the AM algorithm.

*Adaptive-Mixture Metropolis*

The Adaptive-Mixture Metropolis (AMM) algorithm is an extension by Roberts and Rosenthal (2009) of the AM algorithm of Haario *et˜al.* (2001). AMM differs from the AM algorithm in two respects. First, AMM updates a scatter matrix based on the cumulative current parameters and the cumulative associated outer-products, and these are used to generate a multivariate normal proposal. This is more efficient with large numbers of parameters adapting over many iterations, especially with frequent adaptations, and results in a much faster algorithm. The second (and main) difference, is that the proposal is a mixture. The two mixture components are adaptive multivariate and static/symmetric univariate proposals. The mixture is determined at each iteration with a mixture weight. The mixture weight must be in the interval (0,1], and is set to 0.05, as in Roberts and Rosenthal (2009). A higher value of the mixture weight is associated with more static/symmetric univariate proposals, and a lower weight is associated with more adaptive multivariate proposals. The algorithm will be unable to include the multivariate mixture component until it has accumulated some history, and models with more parameters will take longer to be able to use adaptive multivariate proposals until later.

The AMM algorithm is the best alternative to Adaptive Metropolis-within-Gibbs (AMWG) as the general choice for an adaptive algorithm, though each problem is different. The advantage of AMM over AMWG is that it is much faster to update each iteration. The disadvantage is that more information must be learned in the covariance matrix to adapt properly (see AMWG for more). If AMM is used for adaptation, then the final, non-adaptive algorithm should be RWM.

*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.

The DRAM algorithm is useful to assist the AM algorithm when the acceptance rate is low. As an alternative, the Adaptive-Mixture Metropolis (AMM) is an extension of the AM algorithm that includes a mixture of proposals, and one mixture component has a small proposal standard deviation to assist in overcoming initially low acceptance rates.

*Metropolis-within-Gibbs*

Metropolis-within-Gibbs (MWG) is a hybrid algorithm, combining Metropolis-Hastings and Gibbs sampling, and was suggested in Tierney (1994). Also referred to as Metropolis within Gibbs or Metropolis-in-Gibbs, it is a componentwise algorithm in which the model specification function is evaluated a number of times equal to the number of parameters, per iteration. Parameters are updated sequentially, though other versions may update in a random order. MWG often uses blocks, but in **LaplacesDemon**, all blocks have dimension 1, meaning that each parameter is updated in turn. If parameters were grouped into blocks, then they would undesirably share a proposal standard deviation. MWG runs most efficiently when the acceptance rate of each parameter is 0.44, which is the optimal acceptance rate of a target distribution that is univariate and Gaussian.

The advantage of MWG over RWM is that it is more efficient with information per iteration, so convergence is faster in iterations. The disadvantage of MWG is that it is more time-consuming due to the evaluation of the model specification function for each parameter per iteration. As the number of parameters increases, and especially as model complexity increases, the run-time per iteration decreases. Since fewer iterations are completed in a given time-interval, the possible amount of thinning is also at a disadvantage. MWG may be used for simple models with few parameters, but is not recommended here for large and complex models.

*Adaptive Metropolis-within-Gibbs*

The Adaptive Metropolis-within-Gibbs (AMWG) algorithm is presented in (Roberts and Rosenthal 2009; Rosenthal 2007). The standard deviation of the proposal of each parameter is manipulated to optimize the associated acceptance rate toward 0.44. This is much simpler than other adaptive methods that adapt based on sample covariance in large dimensions. Large covariance matrices require a large number of elements to adapt, which takes exponentially longer to adapt as the dimension increases. Regardless of dimension, the AMWG optimizes each parameter to a univariate acceptance rate, and a sample covariance matrix does not need to be estimated for adaptation, which consumes time and memory.

Compared to other adaptive algorithms in **LaplacesDemon**, a disadvantage is the time to complete each iteration increases as a function of parameters and model complexity, as noted in MWG. For example, in a 100-parameter model, AMWG completes its first iteration as the AM algorithm completes its 100th. However, to adapt accurately, the AM algorithm must correctly estimate 5,050 elements of a sample covariance matrix, while AMWG must correctly estimate only 100 proposal standard deviations. Roberts and Rosenthal (2009) have shown an example model with 500 parameters that had a burn-in of around 25,000 iterations.

Overall, the AMWG algorithm currently seems to be the better, general choice for an adaptive algorithm, though each problem is different. If AMWG is used for adaptation, then the final, non-adaptive algorithm should be MWG rather than RWM.

*Afterward*

Once the model is updated with the `LaplacesDemon` function, the `Geweke.Diagnostic` function of Geweke (1992) is iteratively applied to successively smaller tail-sections of the thinned samples to assess stationarity (or lack of trend). When all parameters are estimated as sta-

tionary 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 (Gelman *et˜al.* 2004; Jones, Haran, Caffo, and Neath 2006). It is included in posterior summaries of `LaplacesDemon`, and is one of five main criteria as a stopping rule to appease Laplace's Demon. MCSE has been shown to be a better stopping rule than MCMC diagnostics (Jones *et˜al.* 2006). Laplace's Demon provides a `MCSE` function that allows three methods of estimation: sample variance, batch means (Jones *et˜al.* 2006), and Geyer's method (Geyer 1992).

## 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. 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* (Gilks and Roberts 1996). 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. BUGS 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.

Advantages of **LaplacesDemon** over JAGS and WinBUGS (not much experience with OpenBUGS) are: Bayes factors, confidence in results (correlations do not cause trouble like in Gibbs), elicitation, enivornment is part of R for data manipulation and posterior analysis, examples in documentation are more plentiful, faster with large data sets (when model specification avoids loops), Importance (Variable and Parameter), Laplace Approximation, log-posterior is available, likelihood-free estimation, marginal likelihood calculated automatically, modes (functions for multimodality), missing values do not require initial values (unless predicting predictors with missing predictors), model specification gives the user complete control on how everything is calculated (including the log-likelihood, posterior, etc., and "tricks" do not have to be used), more MCMC algorithms, posterior predictive checks and discrepancy statistics, predict function for posterior predictive checks or scoring new data sets, suggested code is provided at the end of each run, trap errors do not exist or occur, and weights can be applied easily (such as weighting records in the likelihood).

The 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[12]. 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 2009; Rosenthal 2007), and uses C language, which results in significantly faster sampling, but only when the model specification function is also programmed in C. This algorithm is included in `LaplacesDemon`, where it is referred to as AMWG, for adaptive Metropolis-within-Gibbs. The algorithm is excellent, except it is associated with long run-times for large and complex models.

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 2011) 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.

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.

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

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.

## 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' and `apply` functions wherever possible[13]. 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=500`, `DR=0`, `Mixture=TRUE`, `Periodicity=100`, and the initial value for each $\beta$ set to 0.1 to bypass Laplace Approximation. This took 0.35 minutes with Laplace's Demon, according to a simple, linear regression[14]. It was nowhere near convergence, but updating the same model with the same data for 1,000 iterations took 15.02 minutes in JAGS 3.1.0.

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[15].

However, with small data sets, other MCMC software (**AMCMC** is a good example) can be faster than Laplace's Demon, if it is programmed entirely (meaning, the model specification function as well) in a faster language such as Component Pascal, C, C++, or FORTRAN. 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 (entirely) in C as well.

## 14. Conclusion

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 `laplacesdemon@statisticat.com` with constructive criticism, reports of software bugs, or offers to contribute to Laplace's Demon.

[13] However, when 'for loops' or `apply` functions must be used, Laplace's Demon is typically slower than BUGS.

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

[15] To continue this example, JAGS may be *guessed* to take 20,000 iterations or 5.01 hours, and `LaplacesDemon` may take 400,000 iterations or 2.33 hours, and also have less autocorrelation in the chains due to more thinning. The slowest adaptive algorithm in Laplace's Demon is AMWG, which updated in 5.04 minutes, and should finish around 50,000 iterations or 4.2 hours.

## 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 Chances. 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://cran.r-project.org/web/packages/mcmc/index.html.

Gilks W, Roberts G (1996). "Strategies for Improving MCMC." In W˜Gilks, S˜Richardson, D˜Spiegelhalter (eds.), *Markov Chain Monte Carlo in Practice*, pp. 89–114. Chapman & Hall, Boca Raton, FL.

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 11.12.05, URL http://cran.r-project.org/web/packages/LaplacesDemon/index.html.

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

Jones G, Haran M, Caffo B, Neath R (2006). "Fixed-Width Output Analysis for Markov chain Monte Carlo." *Journal of the American Statistical Association*, **100**(1), 1537–1547.

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).

Laud P, Ibrahim J (1995). "Posterior Model Selection." *Journal of the Royal Statistical Society*, **B 57**, 247–262.

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 (2011). ***MCMCpack**: Markov chain Monte Carlo (MCMC) Package*. R package version 1.1-1, URL http://cran.r-project.org/web/packages/MCMCpack/index.html.

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 (2011). *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.

Riedmiller M (1994). "Advanced Supervised Learning in Multi-Layer Perceptrons - From Backpropagation to Adaptive Learning Algorithms." *Computer Standards and Interfaces*, **16**, 265–278.

Roberts G, Rosenthal J (2001). "Optimal Scaling for Various Metropolis-Hastings Algorithms." *Statistical Science*, **16**, 351–367.

Roberts G, Rosenthal J (2009). "Examples of Adaptive MCMC." *Computational Statistics and Data Analysis*, **18**, 349–367.

Rosenthal J (2007). "AMCMC: An R Interface for 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 (1994). "Markov Chains for Exploring Posterior Distributions." *The Annals of Statistics*, **22**(4), 1701–1762. With discussion and a rejoinder by the author.

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: laplacesdemon@statisticat.com
URL: http://www.statisticat.com/laplacesdemon.html