ThetaMater: Rapid and scalable Bayesian estimation of theta from genomic data

Rich Adams, Drew Schield, Daren Card, Andrew Corbin, and Todd Castoe

last updated: 2017-08-08

The population size parameter $\theta = 4N_e\mu$

The population size parameter $\theta = 4N_e\mu$ reflects the effects of genetic drift and mutation on patterns of genetic variation within a diploid population ($\theta = 2N_e\mu$ for a haploid population) with an effective size of N_e individuals and a mutation rate of μ per site per generation. If two homologous sequences are sampled at random from a population, θ describes the expected number of mutations between these two sequences. θ is a fundamental measure of genetic diversity in populations and is thus an informative parameter used in many population genetic models. The R package ThetaMater provides a Bayesian framework to estimate both θ and α (shape of among-locus rate variation) parameters from a variety of genetic datasets, including haploid or diploid genomic data from single or multiple individuals, reduced-representation genomic data (e.g., RADseq, sequence capture), and single or multilocus Sanger sequence data (and variations of these datasets). ThetaMater implements three different functions that can be used to estimate these parameters within a Bayesian framework:

* `ThetaMater.M1` : estimate θ without among-locus variation
* `ThetaMater.M2` : estimate θ with a fixed α parameter of rate variation and a user-defined number of locus rate classes
* `ThetaMater.M3` : estimate both θ and the shape parameter α and a user-defined number of rate classes

The likelihood function implemented by ThetaMater

The three functions (`ThetaMater.M1` , `ThetaMater.M2` , `ThetaMater.M3`) simulate posterior distributions of and/or parameters for a given dataset. These functions employ the likelihood function ($P(S = k|l, n; \theta)$) to compute the probability of observing *k* mutations in a sample size of *n* from a locus with length *l*. These methods compute the likelihood of a given dataset as a summation of the log-likelihood probabilities across all loci, each with respective lengths, mutation counts, and sample sizes. See the following publications for more information about this model, its derivation, applications, and similar models:

* Tavaré, Simon. “Line-of-descent and genealogical processes, and their applications in population genetics models.” Theoretical population biology 26.2 (1984): 119-164.
* Watterson,G.A. (1975) On the number of segregating sites in genetical models without recombination. Theor. Popul. Biol.
* Wakeley, John. “Coalescent theory.” Roberts & Company (2009).
* Hein, Jotun, Mikkel Schierup, and Carsten Wiuf. Gene genealogies, variation and evolution: a primer in coalescent theory. Oxford University Press, USA, 2004.
* Takahata, Naoyuki, and Yoko Satta. “Evolution of the primate lineage leading to modern humans: phylogenetic and demographic inferences from DNA sequences.” Proceedings of the National Academy of Sciences 94.9 (1997): 4811-4815.
* Takahata, Naoyuki, Yoko Satta, and Jan Klein. “Divergence time and population size in the lineage leading to modern humans.” Theoretical population biology 48.2 (1995): 198-221. * Yang, Ziheng. “On the estimation of ancestral population sizes of modern humans.” Genetical research 69.02 (1997): 111-116.

Below is the formula for the likelihood function described in these papers that is central to the three ThetaMater functions:

$$P(S = k|l, n; \theta) = \int_0^\infty P(S = k|t) f_T(t) dt$$
$$P(S = k|l, n; \theta) = (l\theta/2) \sum_{i=2}^{n} (-1)^i choose(n-1, i-1)(i-1)/2 \int_0^\infty (t^k e^(- l\theta - 1)/2/k! dt)$$
$$P(S = k|l, n; \theta) = (l\theta/2)^k \sum_{i=2}^{n} (-1)^i choose(n-1, i-1)(i-1)/2(2/(l\theta + i - 1)^(k + 1)$$
$$P(S = k|l, n; \theta) = \sum_{i=2}^{n} (-1)^i choose(n-1, i-1)((i-1)/(l\theta + i - 1))((l\theta)/(l\theta + i - 1))^k$$

For a dataset consisting of x loci, each with mutation count k_i , number of bases l_i, and number of sequences sampled n_i , we can sum the likelihoods of the individual loci to get the likelihood of the entire dataset under a given value of θ:

$$L(D|\theta) = \sum_{i=1}^{x} log(P(S = k_i|l_i, n_i; \theta)$$

Applications, Assumptions, and Limitations of ThetaMater

Understanding the assumptions of ThetaMater and the underlying coalescent model are critical to the appropriate use of ThetaMater. Importantly, it is assumed that there is no recombination within individual loci and free recombination between loci (i.e., no linkage). Furthermore, all loci are assumed to have evolved under strictly neutral evolution. These are fundamental assumptions of the coalescent model and the likelihood function implemented in ThetaMater. This can been seen in the form of the likelihood equation provided above: the likelihood of an entire dataset is a summation of the log-likelihoods across loci that are assumed to be genetically unlinked. In other words, the genealogy and number of mutations at each locus is assumed to be independently and identically distributed (i.i.d).

To explore the potential effects of one such model violation (unrecognized recombination) in datasets, we simulated loci using the software *msprime* under 6 different recombination rates: (2e-9, 2e-8, 2e-7, 2e-6, 2e-5, 2e-4), using a sample size of 5 gene copies per 10kb locus, and with each dataset consisting of 10k loci. See Step 8: “Recombination & ThetaMater” for a plot of these analyses for each recombination rate and a dataset without recombination. In general, ThetaMater appears largely unaffected by recombination, as the posterior distribution of each analysis is peaked at the true simulation value (θ = 0.008). Nonetheless, these are complex subjects, and we recommend users to explore all potential violations of the model (including selection and recombination/linkage) prior to using ThetaMater

As estimates from any one locus entail significant uncertainity, ThetaMater allows researchers to take full advantage of large, genomic datasets when estimating θ and provides a distribution of plausible values while accounting for uncertainty. Users can also use an estimate of the shape of among-locus rate variation (ThetaMater.M1) or estimate the shape of among-locus rate variaton (ThetaMater.M2) to account for among-locus rate variation when estimating θ, as well as characterize the genomic landscape of rate variation. TThe posterior predictive simulator included in ThetaMater allows users to identify potential outlier loci from the genomic distribution of genetic variation, whether due to issues of orthology (see Step 7), or other violations of model assumptions, such as selection (see GppFST R package, Adams 2015). ThetaMater also includes several functions for simulating datasets under the neutral coalescent model. Briefly, datasets are simulated under the infinite-sites model of mutation according to the protocol discribed in Wakeley 2008 (pg. 255).

Users can estimate locus-specific θs for each locus within a dataset to characterize among-locus estimates of , or leverage all loci to estimate a single, population-level estimate of θ. For single locus-based estimates, θ also reflects the time to the most common ancestor among a sample of sequences. This is because the average time for 2 copies to reach a common ancestor is equal to 2N generations (~4N generations for larger sample sizes). Thus, users can characterize differences in TRMCA (locus-specific θ) among loci for a number of different applications, such as understanding what evolutionary processes may be at work across the genome. For example, a short TMRCA (i.e., small effective population size) may indicate the effects of positive selection, while a older TMRCA (i.e., large effective population size) may indicate balancing selection (or other processes).

Step 0: Before running ThetaMater

ThetaMater takes as input a variety of data types in the form of nucleotide alignments (fasta, phylip, nexus; see Step 2 below) to estimate population parameters from genetic and genomic data. Large, multi-sample and multilocus datasets as well as individual loci can be used with ThetaMater. Importantly, ThetaMater requires that all loci are quality-filtered and processed prior to parameter estimation. It is important that users ensure that the input data are of high quality (although the PPS can potentially help identify paralogs, See Step 7 below) and are in one of the commonly used formats discussed below (Step 2, although email the author at radams@uta.edu (mailto:radams@uta.edu) if additional formats are required). ThetaMater is not a ‘magic box’ that will instantly provide an estimate of population parameters from raw sequence data, and it is therefore imperative the users carefully evaluate quality of the input alignment data, the assumptions of the model (recombination, selection, etc.; see above discussion and Step 8), as well as the prior beliefs concerning these parameters (Step 3).

Step 1: install R package ThetaMater from github

*** IMPORTANT: ThetaMater was written using R.3.3.3. We recommend using this version of R to ensure that the underlying c++ functions will operate correctly (this version of Rcpp works best with R.3.3.3). If you discover memory errors when attempting to run ThetaMater, please reinstall R.3.3.3 and this should correct any issues. Contact the author (radams@uta.edu (mailto:radams@uta.edu)) if any memory errors associated with c++ and Rcpp arise when using ThetaMater. R version 3.3.3 can be download here: https://cran.r-project.org/bin/ (https://cran.r-project.org/bin/) ***

The R package ThetaMater is freely available to download and distribute from github https://github.com/radamsRHA/ThetaMater/ (https://github.com/radamsRHA/ThetaMater/). To install and load ThetaMater, you must first install the R packages `devtools`, `MCMCpack`, `Rcpp`, `phangorn` and `ape`.

```
# download dependencies
install.packages("devtools")
install.packages("MCMCpack")
install.packages("ape")
install.packages("phangorn")
install.packages("Rcpp")
```

Now using devtools we can install `ThetaMater` from github:

```
library(devtools)
install_github("radamsRHA/ThetaMater")
```

Next, load the dependency packages for ThetaMater into the R working environment with the following code:

```
library(ThetaMater) # Load package
library(MCMCpack) # Load dependency phybase
library(ape) # Load dependency ape
library(phangorn) # Load dependency phangorn
```

Step 2: Functions to read input data formats and convert into infinite-sites data used by ThetaMater

ThetaMater currently includes a set of 5 functions to import the following widely-used data formats and convert these into the infinite sites format used by ThetaMater:
* Fasta alignments: a directory containing a set of fasta alignments. This can include any number of datatypes, provided they are in fasta format (i.e., Sequence capture, RADseq, Sanger sequenced, multilocus data, whole-genome alignments)
* Nexus alignments: a directory containing a set of nexus alignments. This can include any number of datatypes, provided they are in nexus format (i.e., Sequence capture, RADseq, Sanger sequenced, multilocus data, whole-genome alignments)
* pyRAD output alignments: a single, multilocus alleles file produced by the pyRAD pipeline
* Interleaved fasta alignments: a single, multilocus fasta file comprising multiple independent loci (i.e., similar to stacks output)
* Diploid genome fasta alignments: a single fasta file representing a diploid sequence alignment in which SNPs are coded as ambiquities

Please contact Rich Adams (radams@uta.edu (mailto:radams@uta.edu)) to request additional formats that are not currently supported (provide a short example file to be used for building a custom function). The format for the input conversion functions arguments are as follows:

`Read.FastaDir(fasta.dir)`
* `fasta.dir` : path to the directory of fasta alignments, each with a suffix of .fasta or .fa

`Read.NexusDir(nexus.dir)`
* `nexus.dir` : path to the directory of nexus alignments, each with a suffix of .nexus or .nex

`Read.AllelesFile(alleles.file)`
* `alleles.file` : path to the .alleles file provided by the pyRAD pipeline

`Read.InterleavedFasta(fasta.file)`
* `fasta.file` : path to an 'interleaved' fasta file (i.e., Stacks output)

`Read.DiploidFasta(genome.fasta.file)`
* `genome.fata.file` : path to a diploid genome fasta alignment (ambiquities code for SNPs)

Below we read one of the example datasets used in this tutorial:

```
# Load the example data provided with the package
data(example.dat,package= "ThetaMater")

# Let's look at the data
example.dat$k.vec # mutation counts
```

```
##  [1] 0 1 2 3 4 0 1 2 3 6
```

```
example.dat$l.vec # locus lengths
```

```
##  [1] 100 100 100 100 100 100 100 100 100 100
```

```
example.dat$n.vec # number of samples
```

```
##  [1] 5 5 5 5 5 7 7 7 7 7
```

```
example.dat$c.vec # number of observations (i.e., sum(example.dat$c.vec) = number of loci)
```

```
##  [1] 350 121  27  12   2 318 126  38   5   1
```

For the remainder of the manual, we use the following notation in R:
* `k.vec` : vector containing the number of mutations observed per locus (collapsed into unique data patterns)
* `l.vec` : vector containing the number of sites per locus (i.e., locus length, collapsed into unique data patterns)
* `n.vec` : vector containing the number of samples per locus (i.e., number of gene copies per locus, collapsed into unique data patterns)
* `c.vec` : vector containing the unique data pattern counts (i.e., number of observations consisting of the same mutation count, locus length, and number of samples)

```
library(ThetaMater)
# To use the function 'Read.AllelesFile' you can try loading the raw file included with this package
file.loc <- system.file("example.alleles", package="ThetaMater")
example.dat <- Read.AllelesFile(alleles.file = file.loc)
```

```
## [1] "Reading /Library/Frameworks/R.framework/Versions/3.3/Resources/library/ThetaMater/example.alleles for all
1000 loci..."
## [1] "ALL DONE! returning a list of k.vec, n.vec, l.vec, locus.nums for 1000 loci"
```

Here, the object 'example.dat' returns a list with the following vectors:
* k.vec = vector of mutation counts
* l.vec = vector of locus lengths
* n.vec = vector of sample counts
* c.vec = vector of unique pattern counts

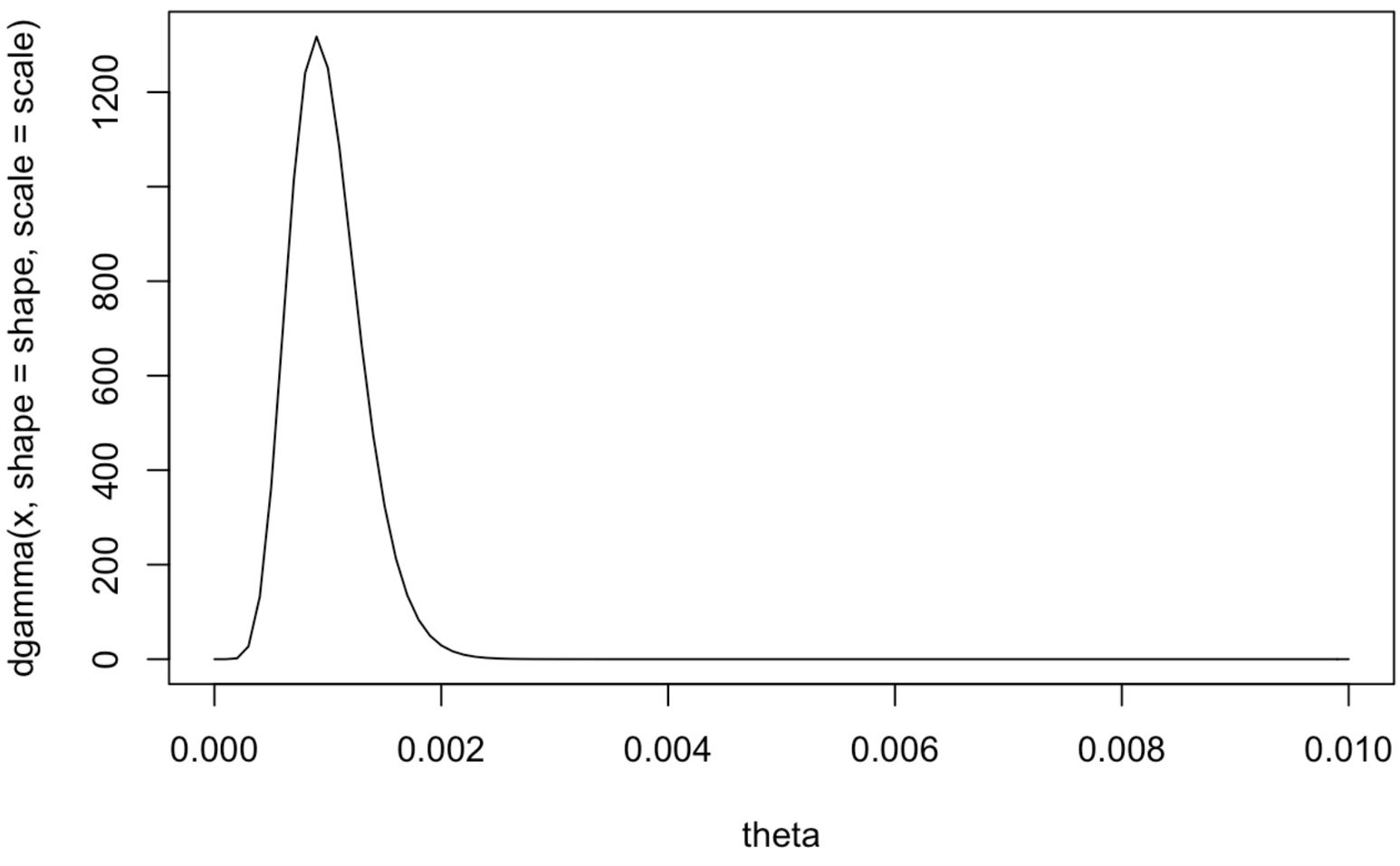
Step 3: Setting the prior distributions for θ and/or α

ThetaMater uses gamma distributions to model the prior probabilty distributions for both θ and α. The prior gamma distribution for θ and α are described by two parameters (shape and scale, with expectation[parameter] = shape*scale). These parameters should be set to reflect prior knowledge about θ and α before analyzing the observed dataset. In practice, we find that most theta values are within the range 0.00001-0.01. You can use the code below to view the gamma distribution prior to running ThetaMater:

```
# Lets see some gamma distribution settings for theta.

# Here we have a relatively peaked prior with the expectation (shape * scale) = 0.001
shape = 10
scale = 0.0001

# See expected theta
E.theta = shape*scale
E.theta
```

```
## [1] 0.001
```

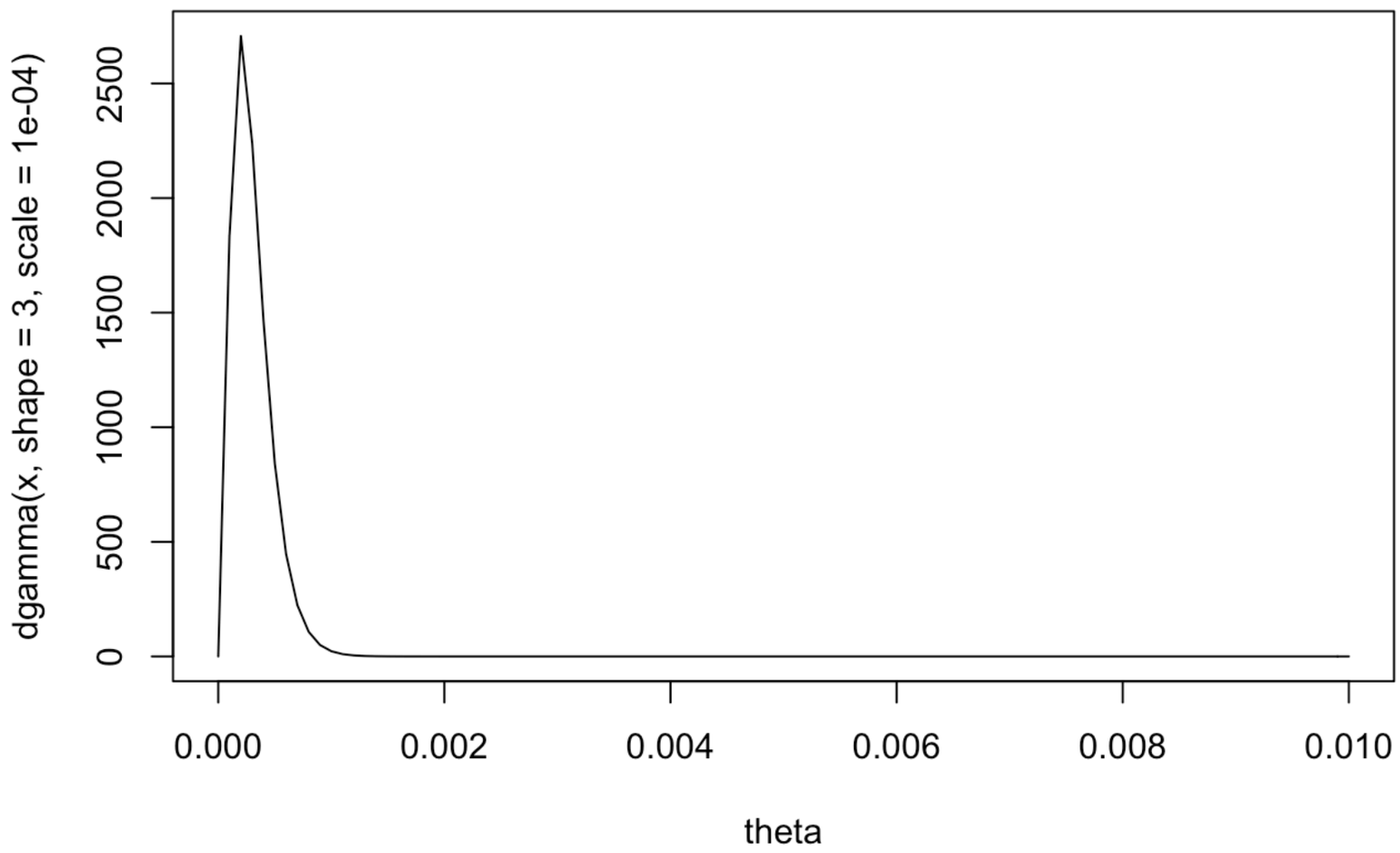
```
# Now let's plot this distribution
curve( dgamma(x,shape = shape,scale = scale), xlim=c(0,.01), xlab = "theta")
```



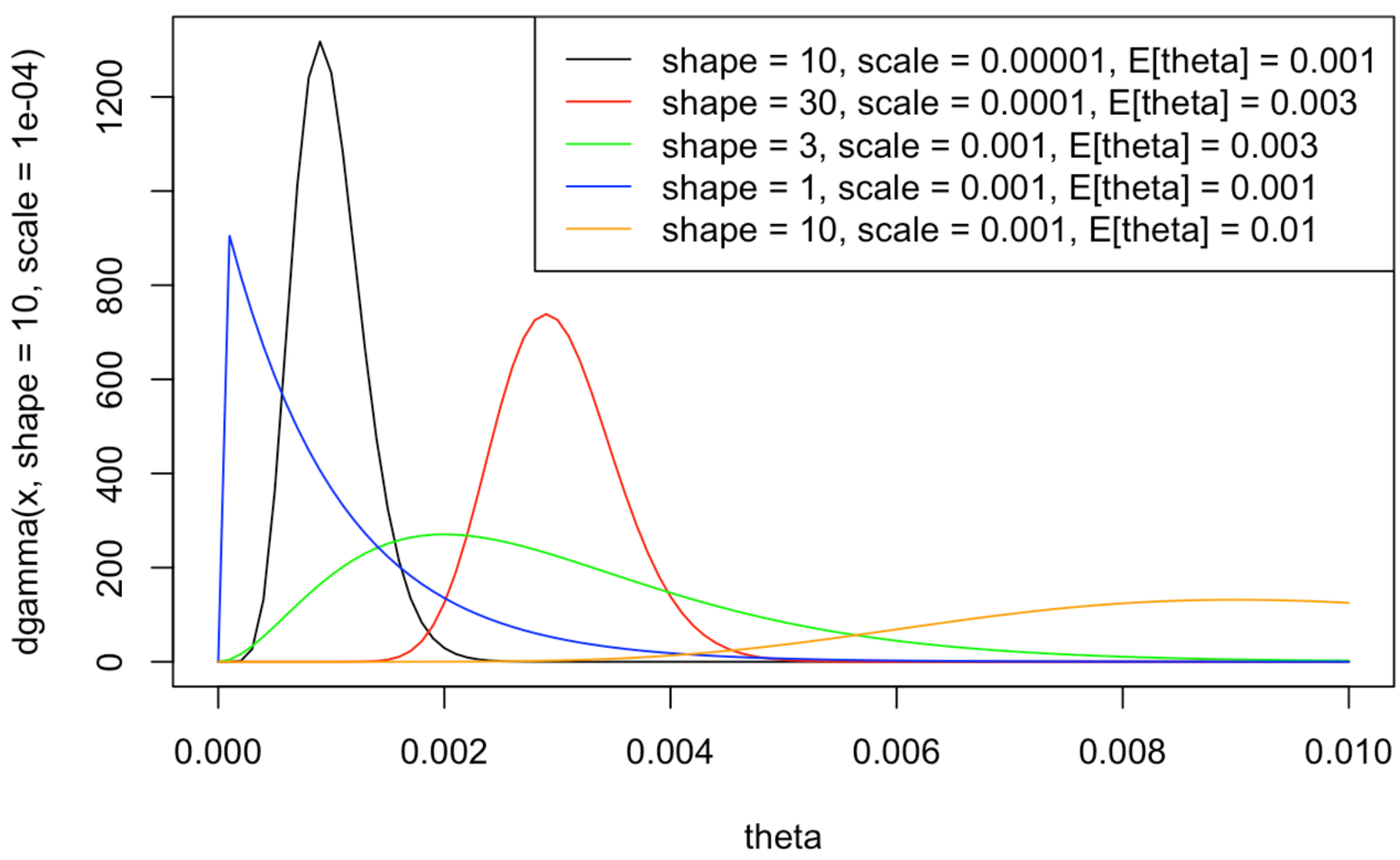
```
# Here's a prior for ~order of magnitude smaller than before
shape = 3
scale = 0.0001

# See expected theta
E.theta = shape*scale
E.theta
```

```
## [1] 3e-04
```

```
curve( dgamma(x,shape = 3,scale = 0.0001), xlim=c(0,.01), xlab = "theta")
```




```
# Finally, here's an array of different settings for the prior
curve( dgamma(x,shape = 10,scale = 0.0001), xlim=c(0,0.01), xlab = "theta")
curve( dgamma(x,shape = 30, scale =0.0001), add=T, col='red' )
curve( dgamma(x,shape = 3,scale = 0.001), add=T, col='green' )
curve( dgamma(x,shape = 1,scale = 0.001), add=T, col='blue' )
curve( dgamma(x,shape = 10, scale = 0.001), add=T, col='orange' )
title(main="Gamma probability distribution function")
legend(par('usr')[2], par('usr')[4], xjust=1,
            c('shape = 10, scale = 0.00001, E[theta] = 0.001', 'shape = 30, scale = 0.0001, E[theta] = 0.003',
               'shape = 3, scale = 0.001, E[theta] = 0.003', 'shape = 1, scale = 0.001, E[theta] = 0.001', 'shap
e = 10, scale = 0.001, E[theta] = 0.01'),
            lwd=1, lty=1,
            col=c(par('fg'), 'red', 'green', 'blue', 'orange') )
```



The shape and scale parameters can be set for α in a similar manner to reflect prior knowledge about the distributon of among-locus rate variation in your dataset.

IMPORTANT: The prior distribution is designed to reflect prior knowledge about the parameters before viewing the dataset. For example, one can set the shape and scale parameters of the prior to reflect an expected value provided from previous datasets. It is reasonable to try different sets of prior values to determine the sensitivty of the posterior to the prior and to evaluate the results under different settings.

Step 4.1: `ThetaMater.M1`: function to simulate a posterior distribution of theta without among-locus rate variation

Here we will estimate Theta for the given dataset using ThetaMater.M1. The input arguments for the function `ThetaMater.M1` are as follows:

```
ThetaMater.M1(k.vec, l.vec, n.vec, c.vec, ngens, burnin, thin, theta.shape, theta.scale)
```

* `k.vec` : vector of mutation counts
* `l.vec` : vector of locus lengths
* `n.vec` : vector of sample counts
* `c.vec` : vector of unique pattern counts
* `ngens` : number of iterations to run the MCMC simulation
* `burnin` : number of iterations to discard as burnin
* `thin` : number of iterations between recorded MCMC samples
* `theta.shape` : shape parameter of the prior gamma distribution on theta (See Step 2)
* `theta.scale` : scale parameter of the prior gamma distribution on theta (See Step 2)

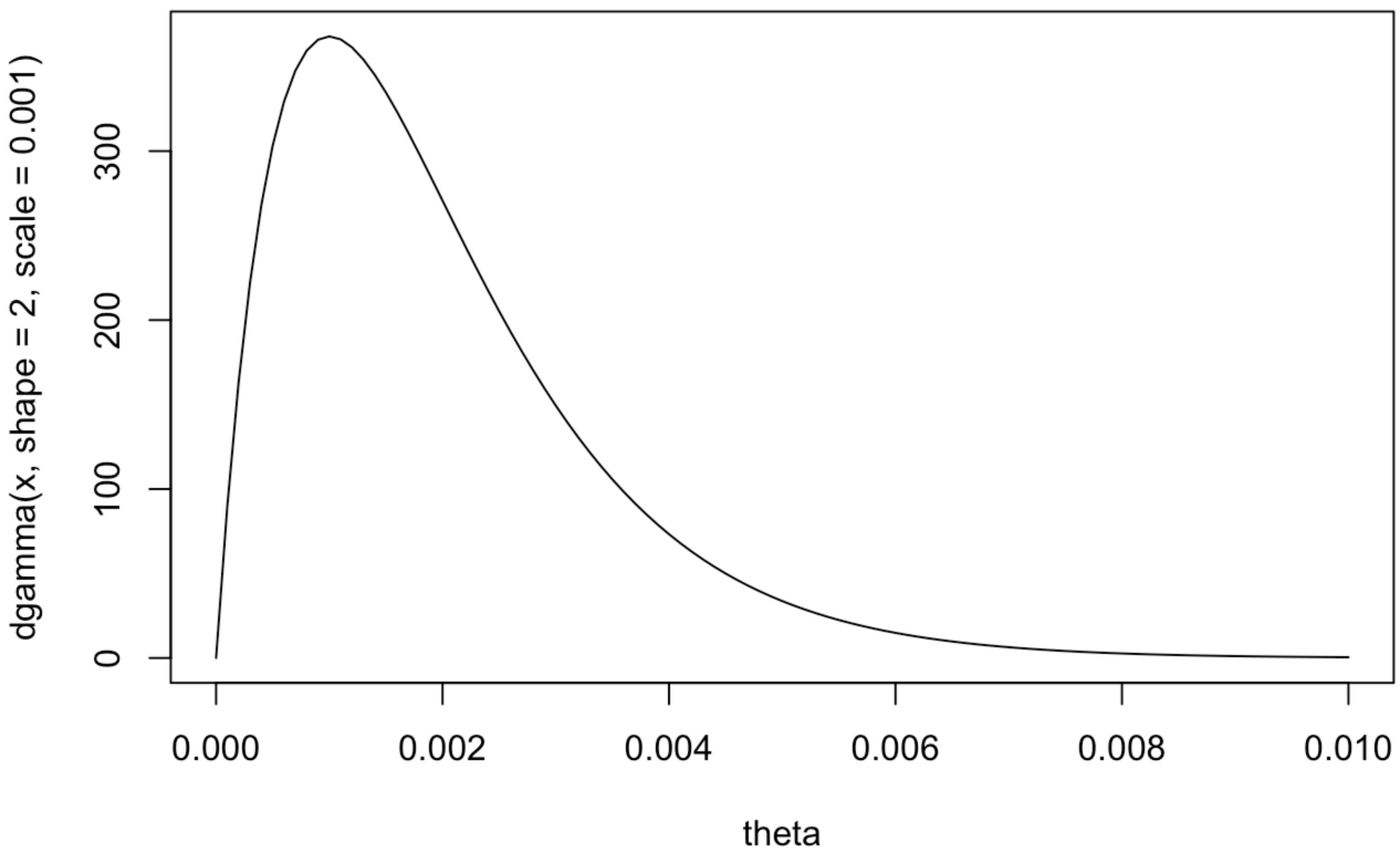
Before estimating θ with ThetaMater.M1, let's first view and set the prior distribution for θ that we will use in this example analysis. These data were simulated using theta = 0.002, and thus we set a prior distribution with an expectation of 0.002 for this example population. For empirical analyses, the values of these parameters (theta.shape, theta.scale) will be ideally set to reflect prior knowledge about a given population under study. For this analysis will set `theta.shape` and `theta.scale` as the following:

```
shape = 2
scale = 0.001
E.theta = shape*scale
E.theta
```

```
## [1] 0.002
```

Let's go ahead and plot this distribution to visualize our prior (See Step 3 for more information on setting priors):

```
curve( dgamma(x,shape = 2,scale = 0.001), xlim=c(0,.01), xlab = "theta")
```



Now we are ready to estimate θ We can load the example data and run the MCMC for ThetaMater.M1 using the code below:

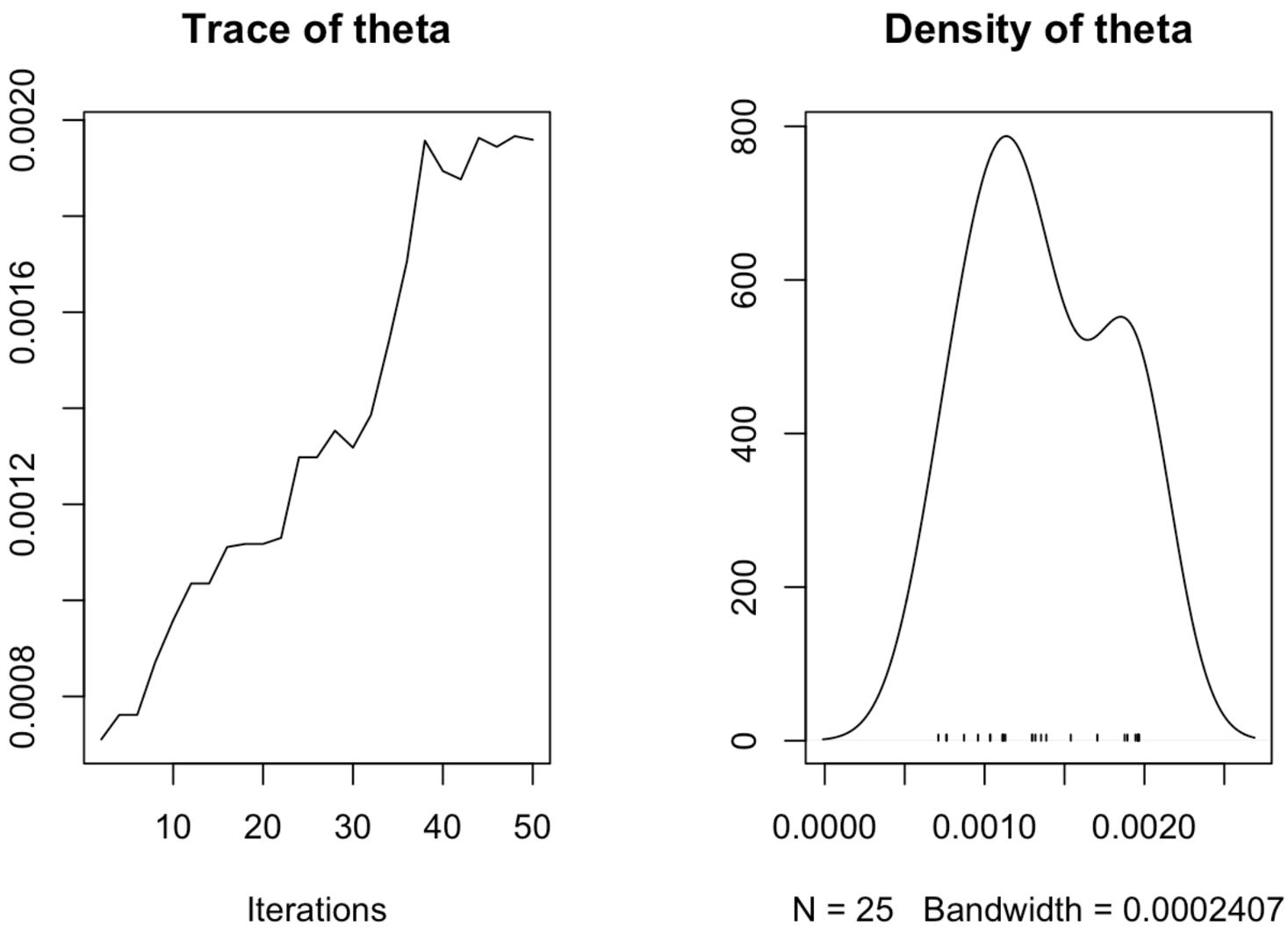
```
data(example.dat,package= "ThetaMater")

example.MCMC <- ThetaMater.M1(k.vec = example.dat$k.vec, l.vec = example.dat$l.vec, n.vec = example.dat$n.vec, c.
vec = example.dat$c.vec, ngens = 50, burnin = 1, theta.shape = shape, theta.scale = scale, thin = 2)
```

```
## MCMCmetrop1R iteration 1 of 51
## function value = -1097.41402
## theta =
##    0.00057
## Metropolis acceptance rate = 1.00000
##
##
##
## @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
## The Metropolis acceptance rate was 0.68627
## @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

*** IMPORTANT: ThetaMater was written using R.3.3.3. We recommend using this version of R to ensure that the underlying c++ functions will operate correctly (this version of Rcpp works best with R.3.3.3). If you discover memory errors (Rstudio unexpectedly closeles, segmental fault errors, etc.) when attempting to run ThetaMater, please reinstall R.3.3.3 and this should correct any issues. Contact the author (radams@uta.edu (mailto:radams@uta.edu)) if any memory errors associated with c++ and Rcpp arise when using ThetaMater. R version 3.3.3 can be download here: https://cran.r-project.org/bin/ (https://cran.r-project.org/bin/). These packages will be updated in the future for further support in updated R versions. ***

We didn't run the mcmc very long, and thus the MCMC output does not appear at stationarity. See trace and density plot of the posterior below:

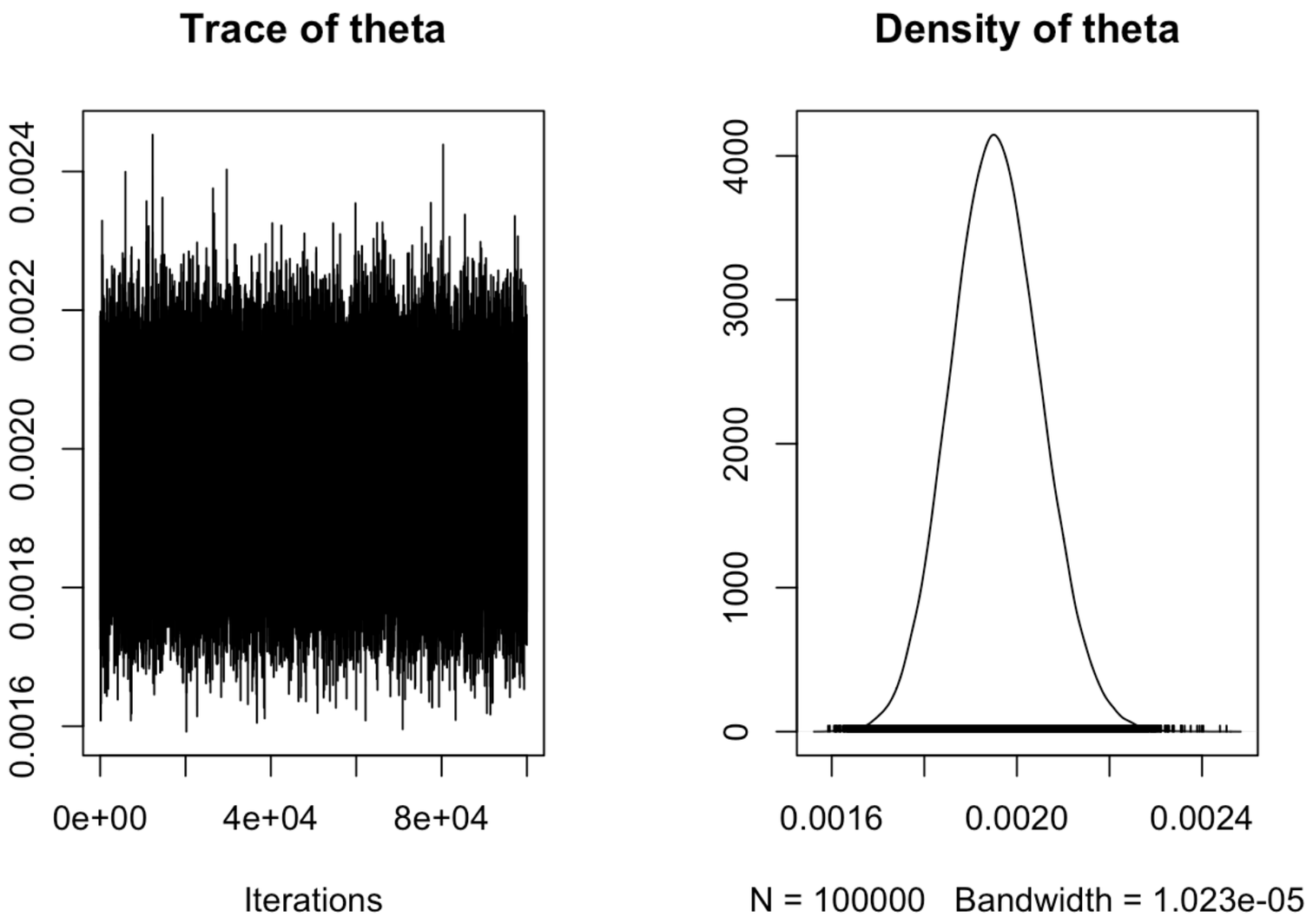
```
varnames(example.MCMC) <- "theta"
plot(example.MCMC)
```



Let’s look at the results from a much longer run that was executed using the below command:

```
example.MCMC <- ThetaMater.M1(k.vec = example.dat$k.vec, l.vec = example.dat$l.vec, n.vec = example.dat$n.vec, c.
vec = example.dat$c.vec,ngens = 1000000, burnin = 1000, thin = 10, theta.shape = shape, theta.scale = scale)
```

```
file.loc <- system.file("example.MCMC.M1.csv", package="ThetaMater")
plot(as.mcmc(read.csv(file = file.loc)))
```



As you can see from the trace and density plot, the MCMC has reached stationarity (represented by the classic “fuzzy caterpillar” shape). Also, we can look at the mean and variance of the posterior distribution of θ using this code below:

```
file.loc <- system.file("example.MCMC.M1.csv", package="ThetaMater")
mean(as.mcmc(read.csv(file = file.loc))) # close to the simulated value of 0.002
```

```
## [1] 0.00195588
```

```
sd(as.mcmc(read.csv(file = file.loc)))
```

```
## [1] 9.646917e-05
```

```
summary(as.mcmc(read.csv(file = file.loc)))
```

```
##
## Iterations = 1:1e+05
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 1e+05
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean             SD       Naive SE Time-series SE
##      1.956e-03      9.647e-05      3.051e-07      4.375e-07
##
## 2. Quantiles for each variable:
##
##     2.5%      25%      50%      75%    97.5%
## 0.001772 0.001890 0.001954 0.002020 0.002150
```

IMPORTANT: Don't panic if you see this error when running ThetaMater: "initial value in 'vmmin' is not finite". This just means that the likelihood of the data is very small under the current prior settings and thus 'infinite' likelihood values may arise. Thus, the likelihood function used by ThetaMater may react poorly to badly specified prior values for θ. If you see this error when running ThetaMater, try different prior settings and multiple runs. See below commands for a demonstration. As always, contact the author (radams@uta.edu (mailto:radams@uta.edu)) if you need further guidance with setting priors for your dataset.

```
# Here's a poorly specified prior that is far from the true value (this will give the error)
example.MCMC <- ThetaMater.M1(k.vec = example.dat$k.vec, l.vec = example.dat$l.vec, n.vec = example.dat$n.vec, c.
vec = example.dat$c.vec,ngens = 500, burnin = 1, thin = 1, theta.shape = 10, theta.scale = 10)
```

```
## Error in optim(theta.init.0, maxfun, control = optim.control, lower = optim.lower, : initial value in 'vmmin'
is not finite
```

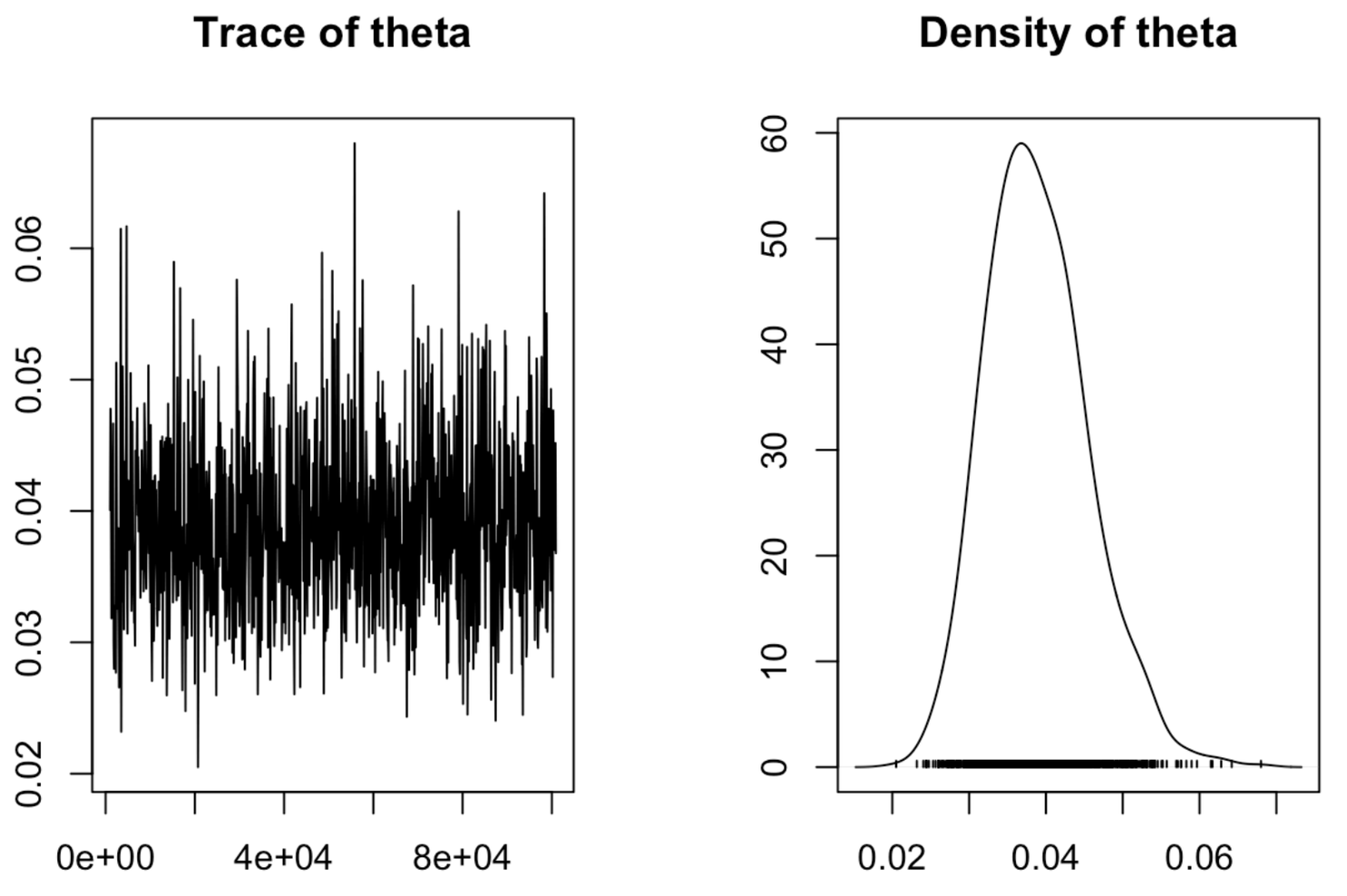
```
# Let's try another prior setting that is closer to the true value of theta
example.MCMC <- ThetaMater.M1(k.vec = example.dat$k.vec, l.vec = example.dat$l.vec, n.vec = example.dat$n.vec, c.
vec = example.dat$c.vec,ngens = 500, burnin = 1, thin = 1, theta.shape = 1, theta.scale = 1)
```

```
## MCMCmetrop1R iteration 1 of 501
## function value = -6270.41259
## theta =
##    0.07333
## Metropolis acceptance rate = 0.00000
##
## MCMCmetrop1R iteration 501 of 501
## function value = -5811.45863
## theta =
##    0.06406
## Metropolis acceptance rate = 0.62874
##
##
##
## @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
## The Metropolis acceptance rate was 0.62874
## @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

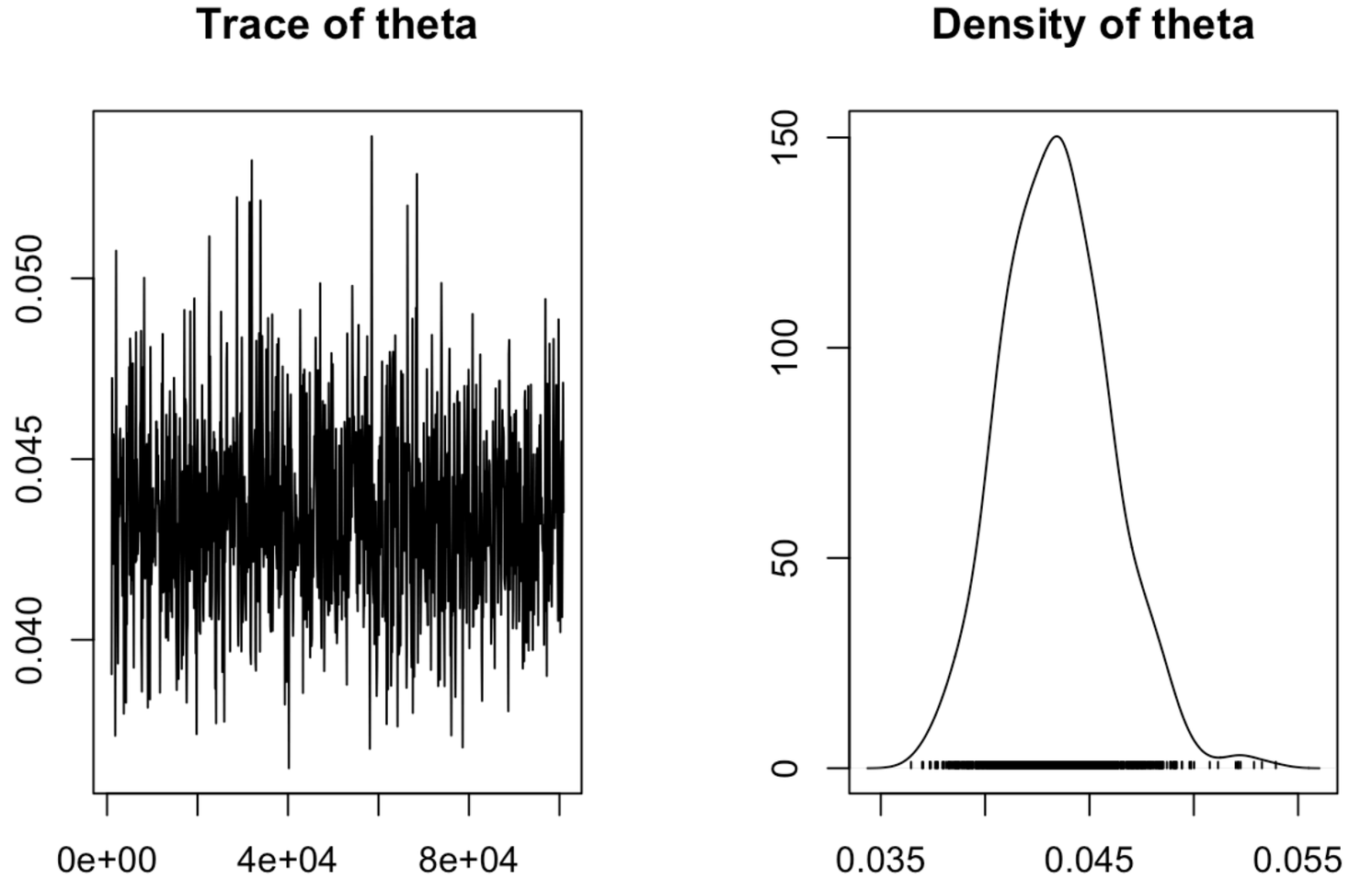
It works! No error this time. With better prior settings that are close to the true value, this likelihood function will not misbehave and the error will not occur. As always, contact the author Rich Adams (radams@uta.edu (mailto:radams@uta.edu)) if you have any questions and/or receive this error message.

Below are some examples of running `ThetaMater.M1` on various dataset sizes: 10, 100, 1000 and 10000 loci. Notice how the posterior becomes more 'peaked' at the true simulated value (θ = 0.04) as the number of loci increases:

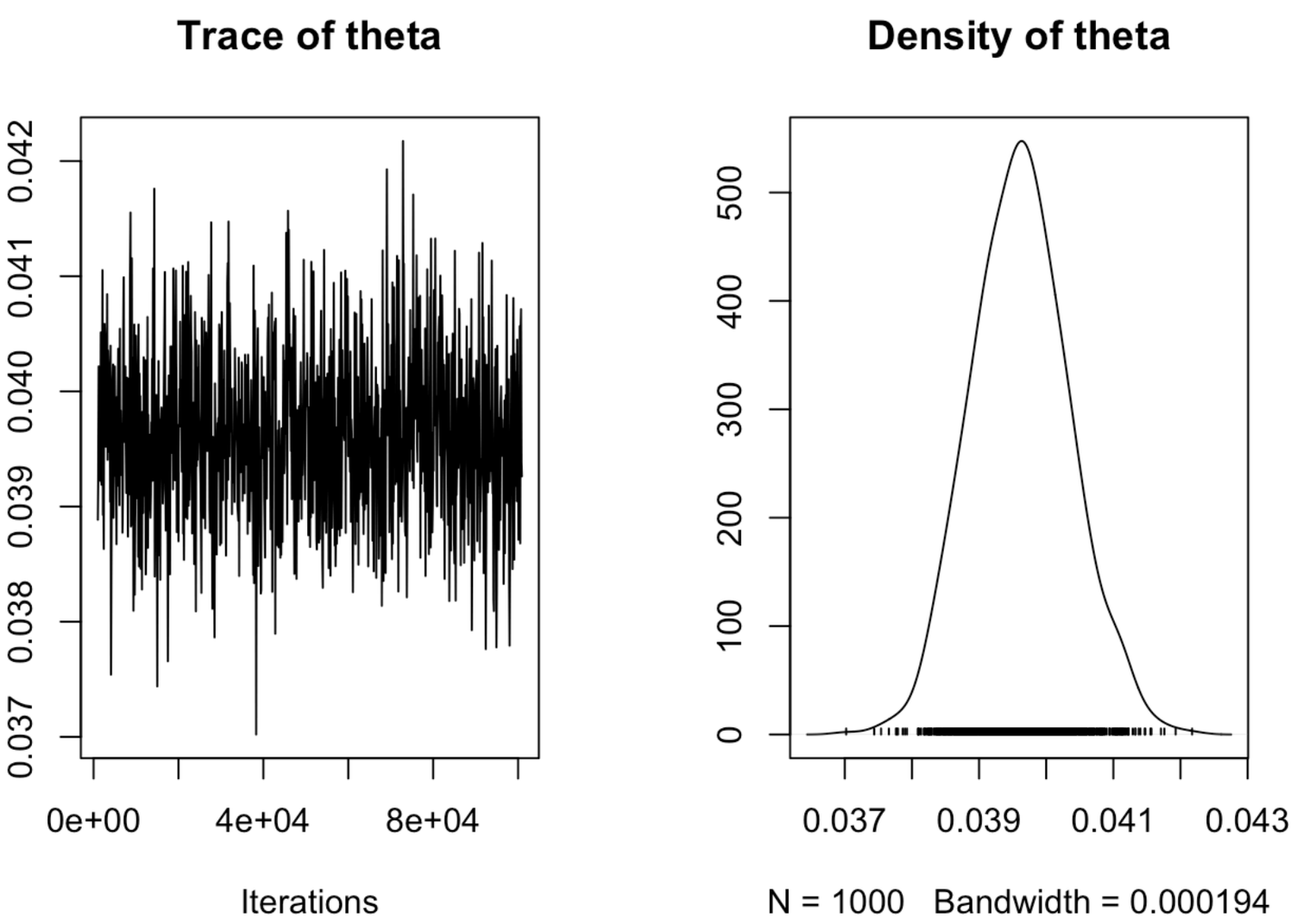
```
# Load results from package
data(example.locus10.mcmc, package = "ThetaMater")
data(example.locus100.mcmc, package = "ThetaMater")
data(example.locus1000.mcmc, package = "ThetaMater")
data(example.locus10000.mcmc, package = "ThetaMater")
# Rename variabel to Theta
varnames(example.locus10.mcmc) <- "theta"
varnames(example.locus100.mcmc) <- "theta"
varnames(example.locus1000.mcmc) <- "theta"
varnames(example.locus10000.mcmc) <- "theta"

# plot results, notice how the posterior becomes more peaked at the true simulated value with larger sample sizes
:
plot(as.mcmc(example.locus10.mcmc)) # 10 loci
```



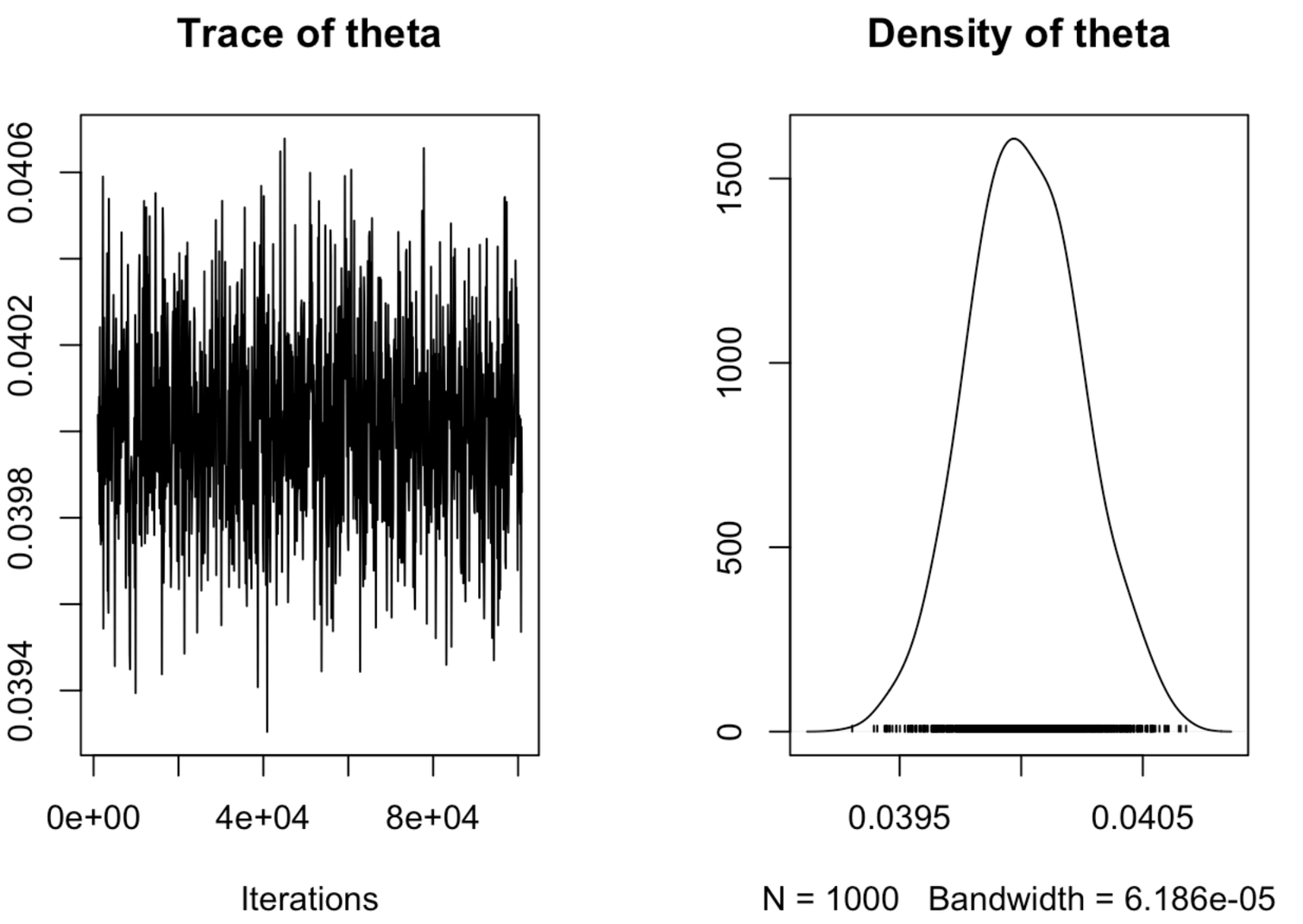

```
plot(as.mcmc(example.locus100.mcmc)) # 100 loci
```



```
plot(as.mcmc(example.locus1000.mcmc)) # 1000 loci
```




```
plot(as.mcmc(example.locus10000.mcmc)) # 10000 loci
```



Step 4.2: `ThetaMater.M2` : function to simulate a posterior distribution of θ with a fixed shape parameter α of among-locus rate variation

Here we will estimate the posterior distribution of θ using a fixed α parameter describing the distribution of among-locus rate variation within our dataset. The input arguments for the function `ThetaMater.M2` are as follows:

`ThetaMater.M2(k.vec, l.vec, n.vec, c.vec, alpha, K.classes, ngens, burnin, thin, theta.shape, theta.scale)` * `k.vec` : vector of mutation counts
* `l.vec` : vector of locus lengths
* `n.vec` : vector of sample counts

* `c.vec` : vector of unique pattern counts
* `ngens` : number of generations to run the MCMC simulation
* `alpha` : fixed alpha parameter describing the shape of the distribution of among-locus rate variation
* `K.classes` : number of disctinct classes to approximate the gamma distribution (4-20 are commonly used for datasets) * `burnin` : number of generations to discard as burnin
* `thin` : number of generations between recorded MCMC samples
* `theta.shape` : shape parameter of the prior gamma distribution on θ (See Step 2)
* `theta.scale` : scale parameter of the prior gamma distribution on θ (See Step 2)

The following example data were simulated using θ = 0.002 and α = 0.1 (using 4 rate classes to approximate the gamma distribution)

```
data(example.M2.dat, package= "ThetaMater")
# Let's look at the data
example.M2.dat$k.vec # mutation counts
example.M2.dat$l.vec # locus lengths
example.M2.dat$n.vec # number of samples
example.M2.dat$c.vec # number of observations
```

Failure to account for among-locus rate variation can affect parameter estimates, such as θ. So, let's see what happens when we do not account for among-locus rate variation (i.e., model misspecification). Here the data were generated under ThetaMater.M2, but we will first simulate posterior distributions of θ using ThetaMater.M1, which does not account for rate variation.

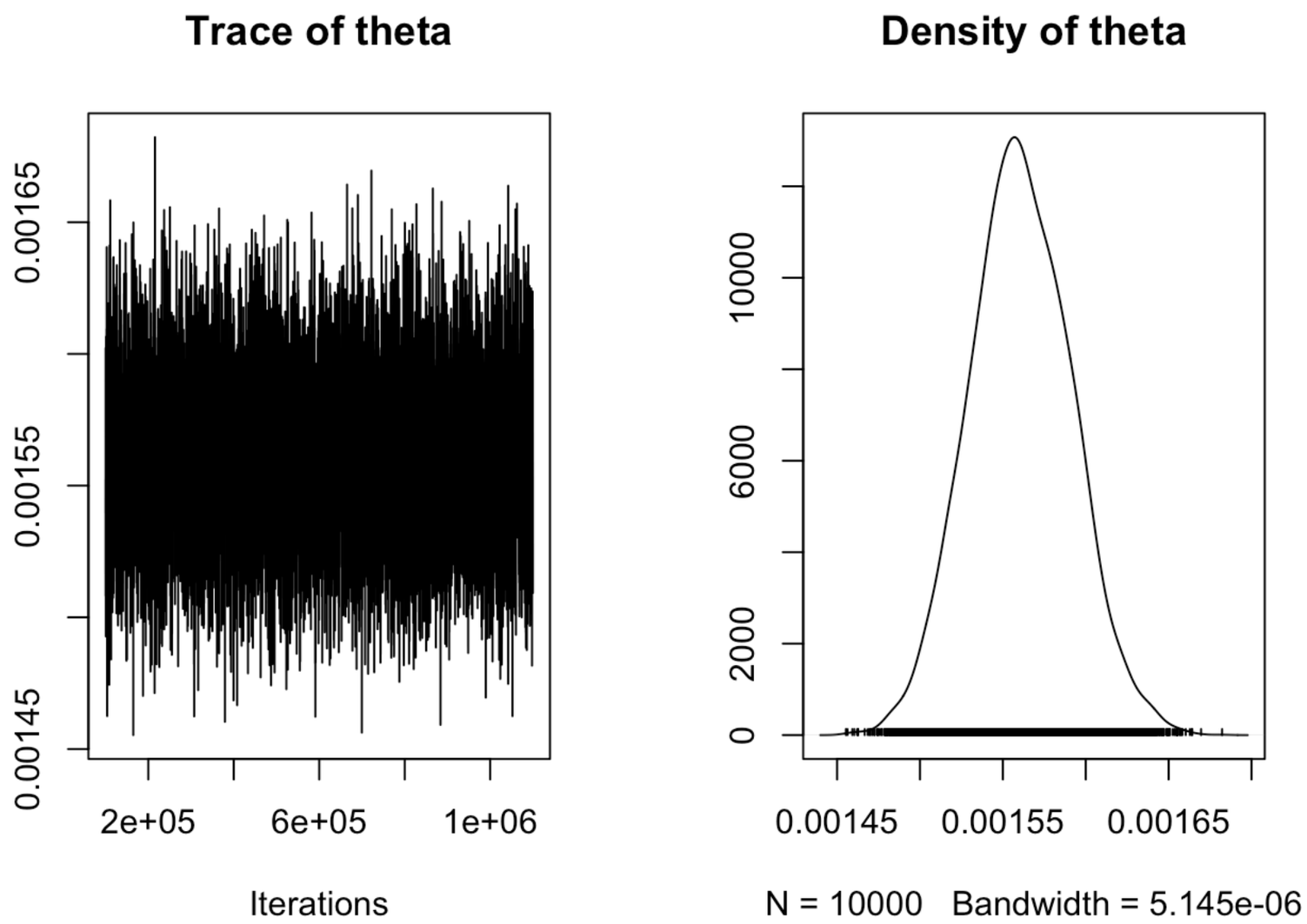
```
shape = 2
scale = 0.001

example.MCMC.M1.M2 <- ThetaMater.M1(k.vec = example.M2.dat$k.vec, l.vec = example.M2.dat$l.vec, n.vec = example.M
2.dat$n.vec, c.vec = example.M2.dat$c.vec, theta.shape = shape, theta.scale = scale, ngens = 1000000, burnin = 10
0000, thin = 100)
```

The results from this 'model-misspecification' analysis are shown below:

```
data(example.MCMC.M2,package= "ThetaMater")
mean(example.MCMC.M2)
```

```
## [1] 0.001560757
```

```
varnames(example.MCMC.M2) <- "theta"
plot(as.mcmc(example.MCMC.M2))
```



In the above case, using `ThetaMater.M1` instead of the correct `ThetaMater.M2` (or `ThetaMater.M3` , see below) function lead to substantially lower estimates θ of with a posterior distribution that is tightly peaked at θ = 0.0015. So, let's use `ThetaMater.M2` to infer the posterior distribution of θ given α = 0.10 and k = 4.

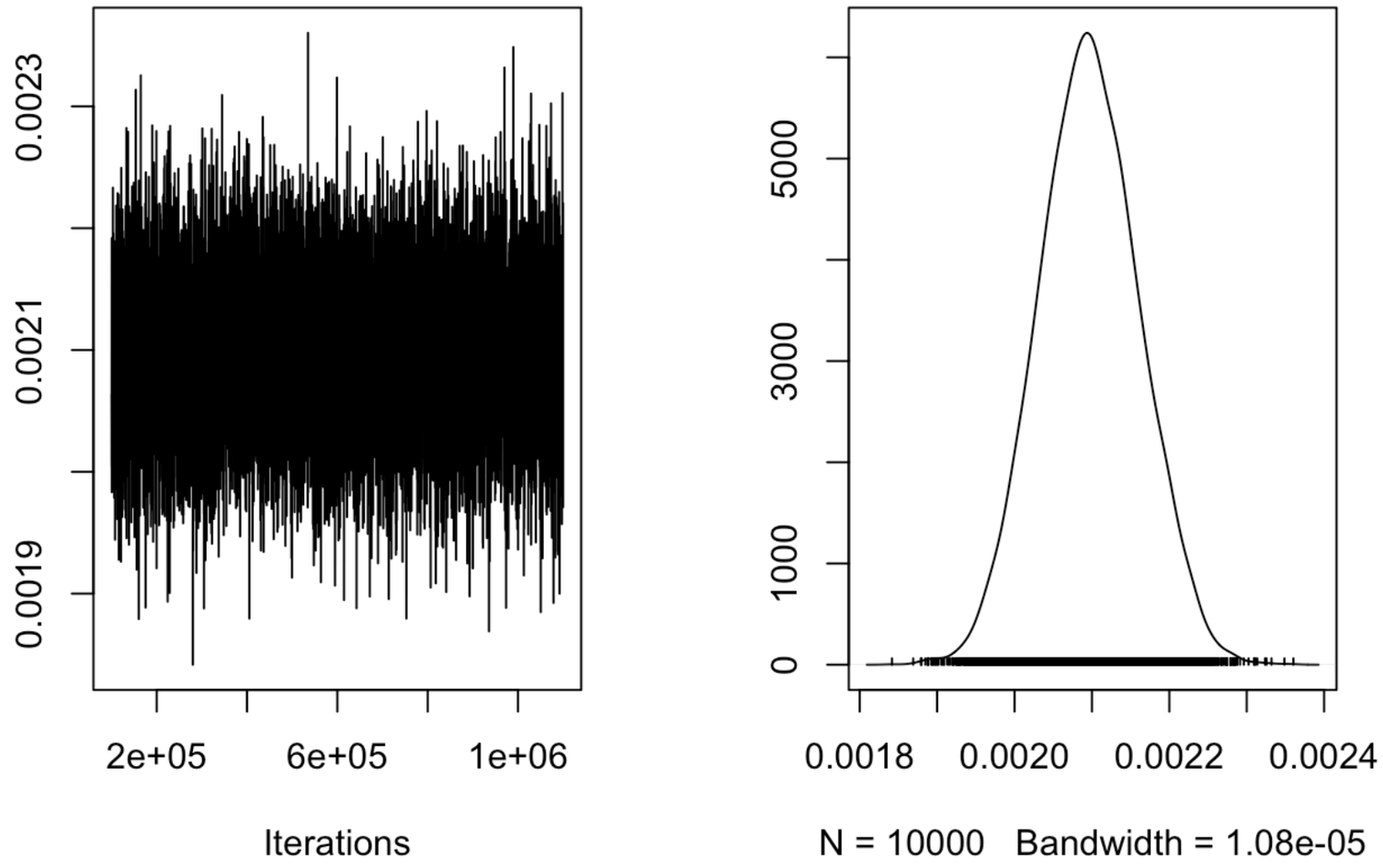
```
shape = 2
scale = 0.001

example.MCMC.M2 <- ThetaMater.M2(k.vec = example.M2.dat$k.vec, l.vec = example.M2.dat$l.vec, n.vec = example.M2.d
at$n.vec, c.vec = example.M2.dat$c.vec, theta.shape = shape, theta.scale = scale, ngens = 1000000, burnin = 10000
0, thin = 100, K = 4, alpha.param = 0.1)
```

And here are the results from this analysis:

```
data(example.MCMC.M2.M2,package= "ThetaMater")
mean(example.MCMC.M2.M2)
```

```
## [1] 0.002096711
```

```
varnames(example.MCMC.M2.M2) <- "theta"
plot(as.mcmc(example.MCMC.M2.M2))
```



Step 4.3: `ThetaMater.M3`: function to simulate a posterior distribution of θ and α

Here we use `ThetaMater.M3` to estimate the joint posterior distribution of θ and α for our dataset. The input arguments for the function `ThetaMater.M3` are as follows:

```
ThetaMater.M3(k.vec, l.vec, n.vec, c.vec, alpha, K.classes, ngens, burnin, thin, theta.shape, theta.scale, alpha.shape, alpha
```

* `k.vec` : vector of mutation counts
* `l.vec` : vector of locus lengths
* `n.vec` : vector of sample counts
* `c.vec` : vector of unique pattern counts
* `ngens` : number of generations to run the MCMC simulation
* `K.classes` : number of diiscrete classes to approximate the gamma distribution (4-20 are commonly used)
* `burnin` : number of generations to discard as burnin
* `thin` : number of generations between recorded MCMC samples
* `theta.shape` : shape parameter of the prior gamma distribution on θ (See Step 2)
* `theta.scale` : scale parameter of the prior gamma distribution on θ (See Step 2)
* `alpha.shape` : shape parameter of the prior gamma distribution on α (See Step 2)
* `alpha.scale` : scale parameter of the prior gamma distribution on α (See Step 2)

Notice: here we will set the prior distributions for both θ and α (theta.shape, theta.scale, alpha.shape, alpha.scale). Let's load the data from the previous analyses (Step 4.2, M2)

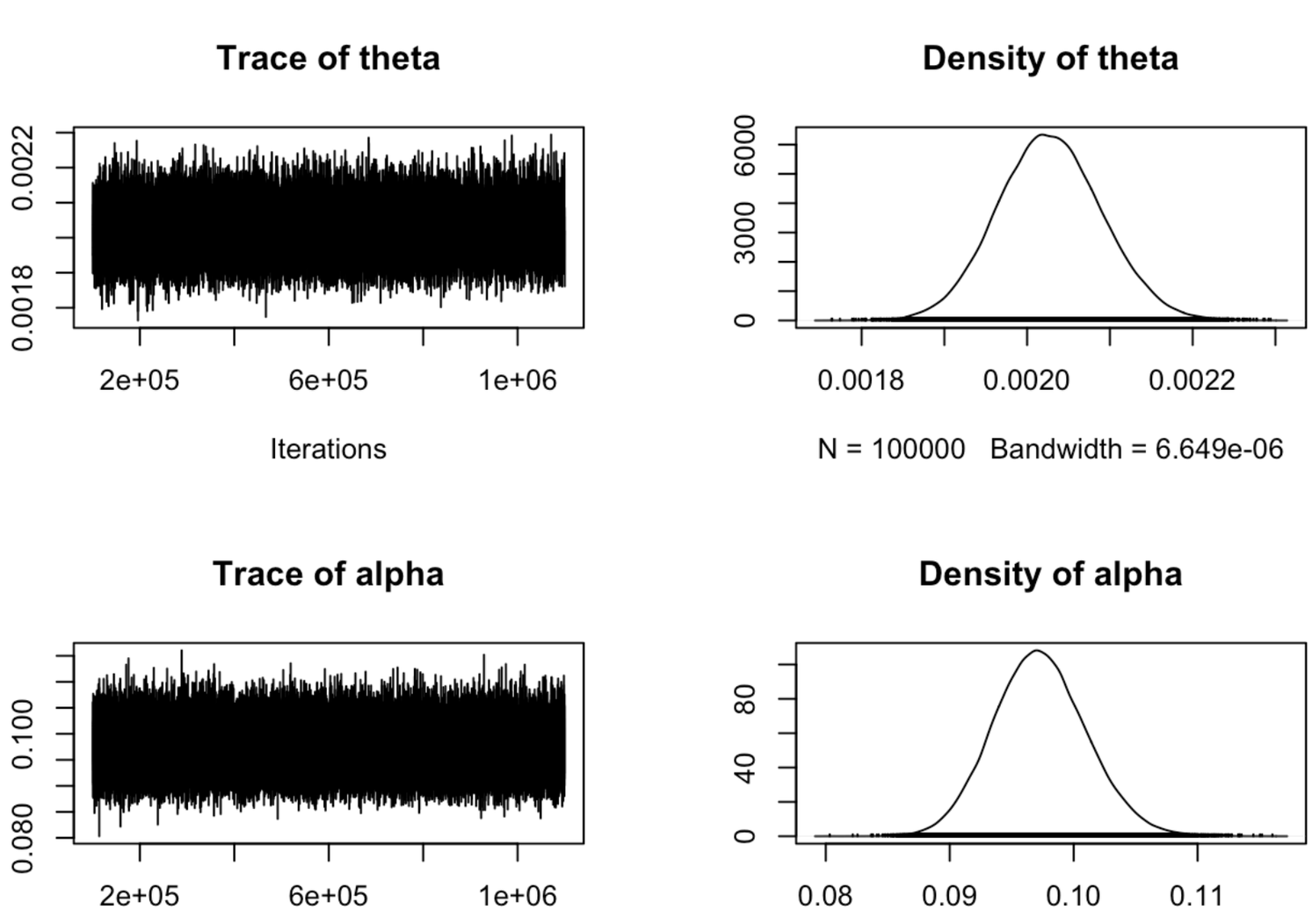
```
data(example.M2.dat, package= "ThetaMater")
# Let's look at the data
example.M2.dat$k.vec # mutation counts
example.M2.dat$l.vec # locus lengths
example.M2.dat$n.vec # number of samples
example.M2.dat$c.vec # number of observations
```

Let's run these analysis using the command below:

```
theta.shape = 2
theta.scale = 0.001
alpha.shape = 5
alpha.scale = 0.01
example.MCMC.M3 <- ThetaMater.M3(k.vec = example.M2.dat$k.vec, l.vec = example.M2.dat$l.vec, n.vec = example.M2.d
at$n.vec, c.vec = example.M2.dat$c.vec, K = 4, ngens = 1000000, burnin = 100000, thin = 10, theta.shape = theta.s
hape, theta.scale = theta.scale, alpha.shape = alpha.shape, alpha.scale = alpha.scale)
```

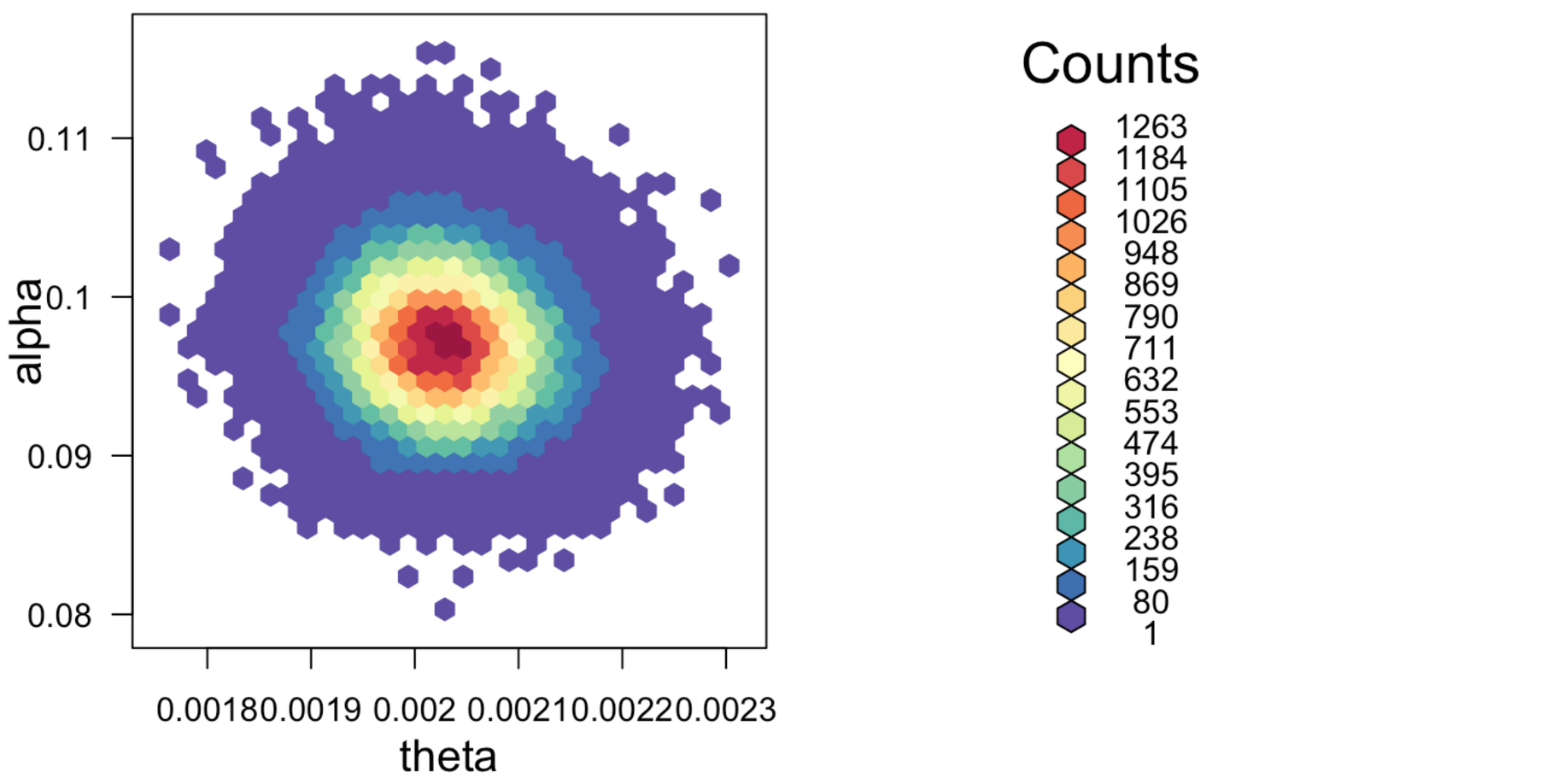
And here are the results from this run, with the posterior of θ on top and α on bottom

```
data(example.MCMC.M3,package= "ThetaMater")
varnames(example.MCMC.M3) <- c("theta", "alpha")
plot(as.mcmc(example.MCMC.M3))
```



We can also make a nice 3D hexbin plot with colors indicating the number of MCMC steps in that state (i.e., warmer colors showing higher posterior probability):

```
# See instructions at http://www.everydayanalytics.ca/2014/09/5-ways-to-do-2d-histograms-in-r.html
library(hexbin)
library(RColorBrewer)
rf <- colorRampPalette(rev(brewer.pal(11,'Spectral')))
h <- hexbin(example.MCMC.M3)
plot(h, colramp=rf, xlab = "theta", ylab = "alpha")
```



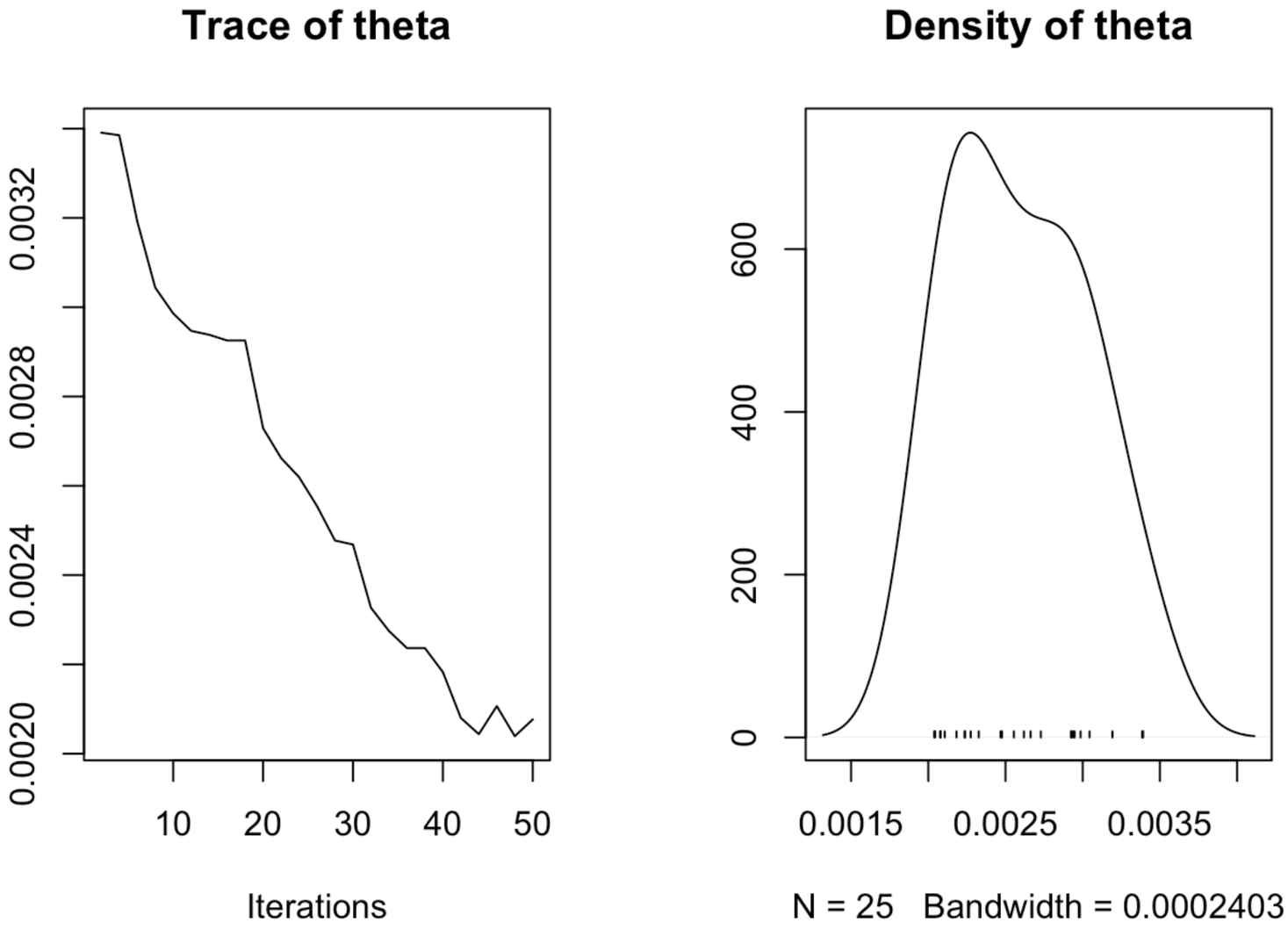
Step 5: Evaluating the results of a ThetaMater analysis

We can visualize/assess the results of MCMC runs under each model using the plotting function provided by MCMCpack. As discussed earlier, it is important to assess the MCMC mixing behavior to convergence on the posterior distribution. In this sense, we want to make sure the posterior sampled sufficient steps to reach the posterior distribution. Roughly speaking, we are looking for a ‘fuzzy caterpillar’ shape of the MCMC trace shown in the plots. These plots can be used to decide how many generations should be discarded as burnin; these are steps that are correlated with the initial state, and may not be accurate approximations to the true posterior. For example, the below MCMC analysis has yet to reach convergence and does not show the “fuzzy caterpillar” shape that is indicative of convergence to the posterior distribution.

```
data(example.dat,package= "ThetaMater")

example.MCMC <- ThetaMater.M1(k.vec = example.dat$k.vec, l.vec = example.dat$l.vec, n.vec = example.dat$n.vec, c.
vec = example.dat$c.vec, ngens = 50, burnin = 1, theta.shape = shape, theta.scale = scale, thin = 2)
```

```
## MCMCmetrop1R iteration 1 of 51
## function value = -945.92621
## theta =
##    0.00336
## Metropolis acceptance rate = 0.00000
##
##
## @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
## The Metropolis acceptance rate was 0.64706
## @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

```
varnames(example.MCMC) = "theta"
plot(example.MCMC)
```



Here's a much better mcmc run with many more steps sampled and adequate burnin, demonstrating the "fuzzy caterpillar" shape:

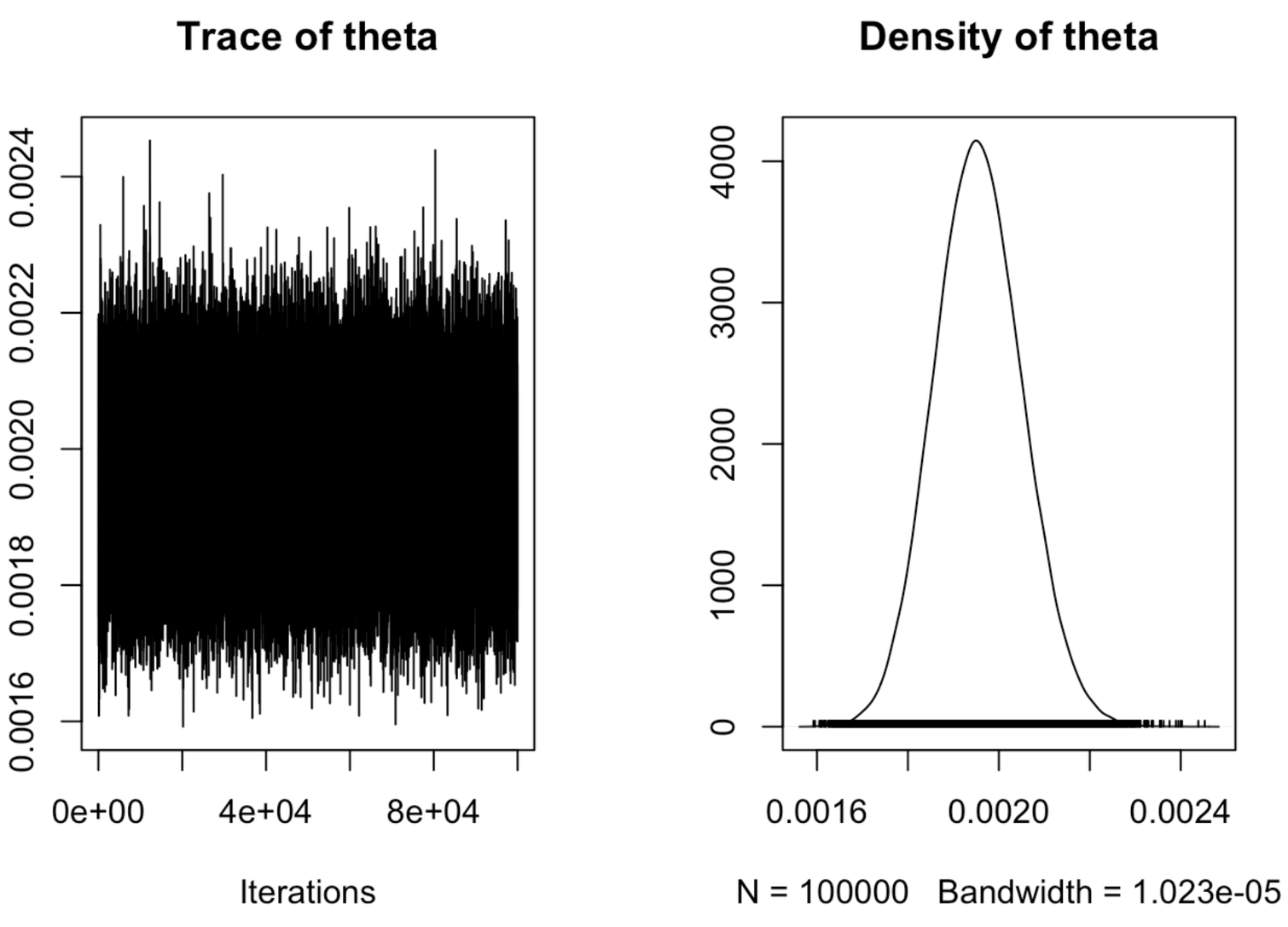
```
file.loc <- system.file("example.MCMC.M1.csv", package="ThetaMater")
mean(as.mcmc(read.csv(file = file.loc))) # close to the simulated value of 0.002
```

```
## [1] 0.00195588
```

```
sd(as.mcmc(read.csv(file = file.loc)))
```

```
## [1] 9.646917e-05
```

```
plot(as.mcmc(read.csv(file = file.loc)))
```



Use the argument `ngens` to run the MCMC chain longer if your analysis has not yet converged to stationarity.

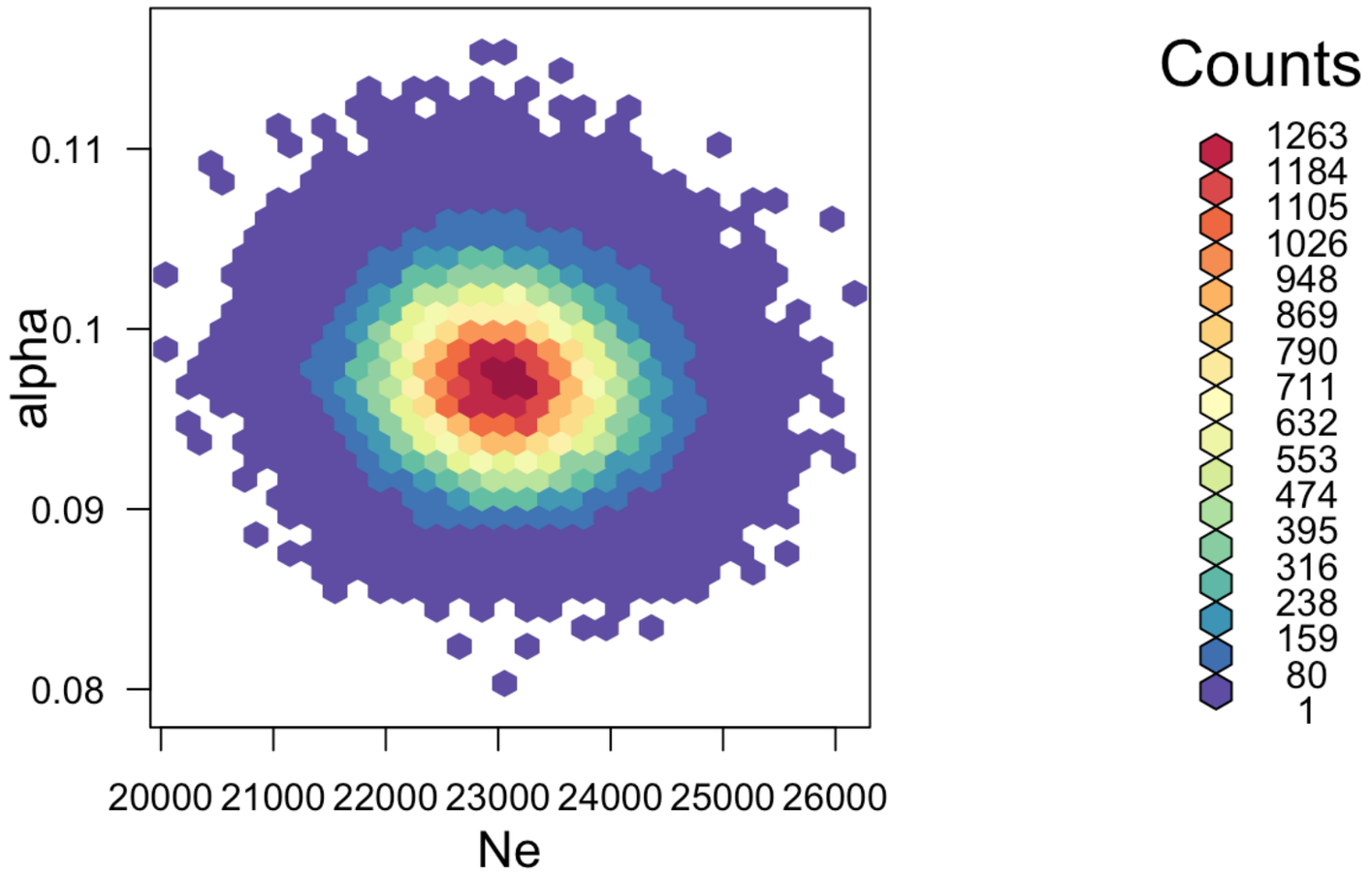
Step 6: (Optional) Convert θ estimates into estimates of effective population size N_e

It is often desirable to convert estimates of θ into estimates of the true effective population size N_e. Giving an estimate of the mutation rate μ, we can convert the posterior distriburion of θ into a posterior distribution of population N_e

$N_e = \theta/4\mu$

For example, let's assume our given population evolved under a mutation rate μ of 2.2*10^-8 (similar to human estimates). We simply take the results from ThetaMater and divide the vector of θ by this mutation rate multiplied by factor of 4 (or 2 for haploid data).

```
# load the results from a ThetaMater analysis
data(example.MCMC.M3,package= "ThetaMater")
mutation.rate = 2.2*10^-8
example.MCMC.M3.Ne <- example.MCMC.M3
example.MCMC.M3.Ne[,1] = example.MCMC.M3.Ne[,1]/(mutation.rate*4)
h <- hexbin(example.MCMC.M3.Ne)
plot(h, colramp=rf, xlab = "Ne", ylab = "alpha")
```



Step 7: (Optional) conduct posterior predictive simulation to remove loci with evidence of unlikely mutation counts (i.e., potential paralogs)

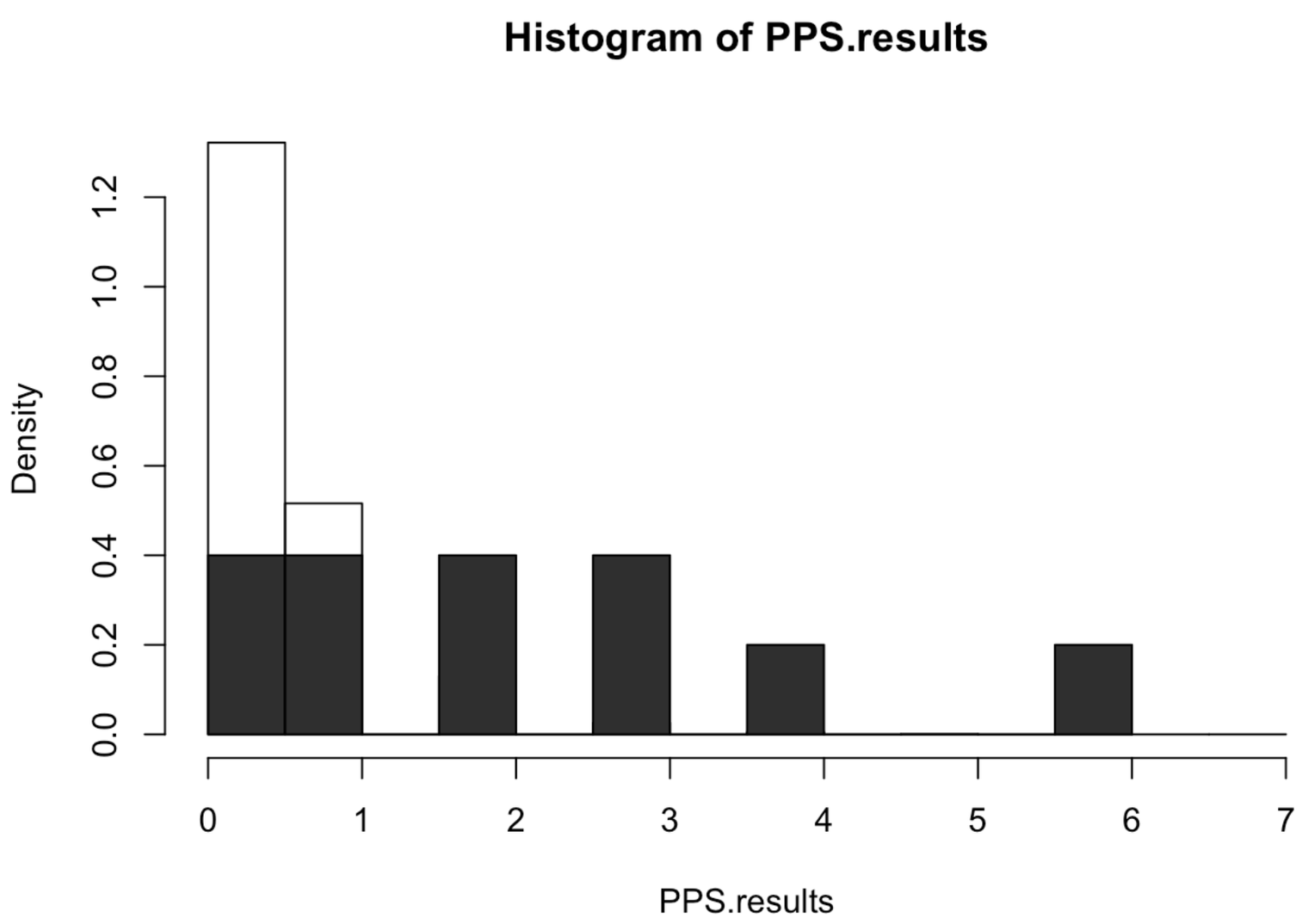
Finally, we can leverage the posterior distribution of θ that is estimated by ThetaMater to simulate posterior predictive simulated (PPS) distributions of mutation counts (k.vec) using the function ThetaMater.PPS. We can leaverage this PPS distribution filter out loci with unexpected mutation counts, such as incorrectly assembled paralogous loci, which will result in greater than expected numbers of single-locus 'mutations'. Additionally, PPS can be leveraged to identify outlier loci with unexpected mutation counts due to other evolutionary processes, such as selection. The commands for using the function ThetaMater.PPS are as follows:

`ThetaMater.PPS(theta.MCMC, l.vec, n.vec)`

* `l.vec` : vector of locus lengths
* `n.vec` : vector of sample counts
* `theta.MCMC` : Posterior distribution of theta inferred via ThetaMater

In this example, the PPS (white) and the observed (gray) distribution overlap considerably, and thus there is no need to filter out loci based on mutation counts alone

```
data(example.dat,package= "ThetaMater")
file.loc <- system.file("example.MCMC.M1.csv", package="ThetaMater")
mcmc.results <- read.csv(file = file.loc) # close to the simulated value of
PPS.results <- ThetaMater.PPS(theta.MCMC = mcmc.results[,1], l.vec = example.dat$l.vec, n.vec = example.dat$n.vec
)
hist(PPS.results, freq = F, breaks = 10)
hist(example.dat$k.vec, col="gray20",add=T, freq = F, breaks = 10)
```



Now, in this next sample we have a simulated dataset in which 4 paralogous loci (out of 1000) have been erronously placed into the same alignment, because they were assumed to be homologous. Using the code below, we will load the 'example.SeqError.alles' data into R and first estimate using this unfiltered dataset.

```
# Let's look at the mutation count vector
data(example.SeqError.alles,package= "ThetaMater")
example.SeqError.alles$k.vec
```

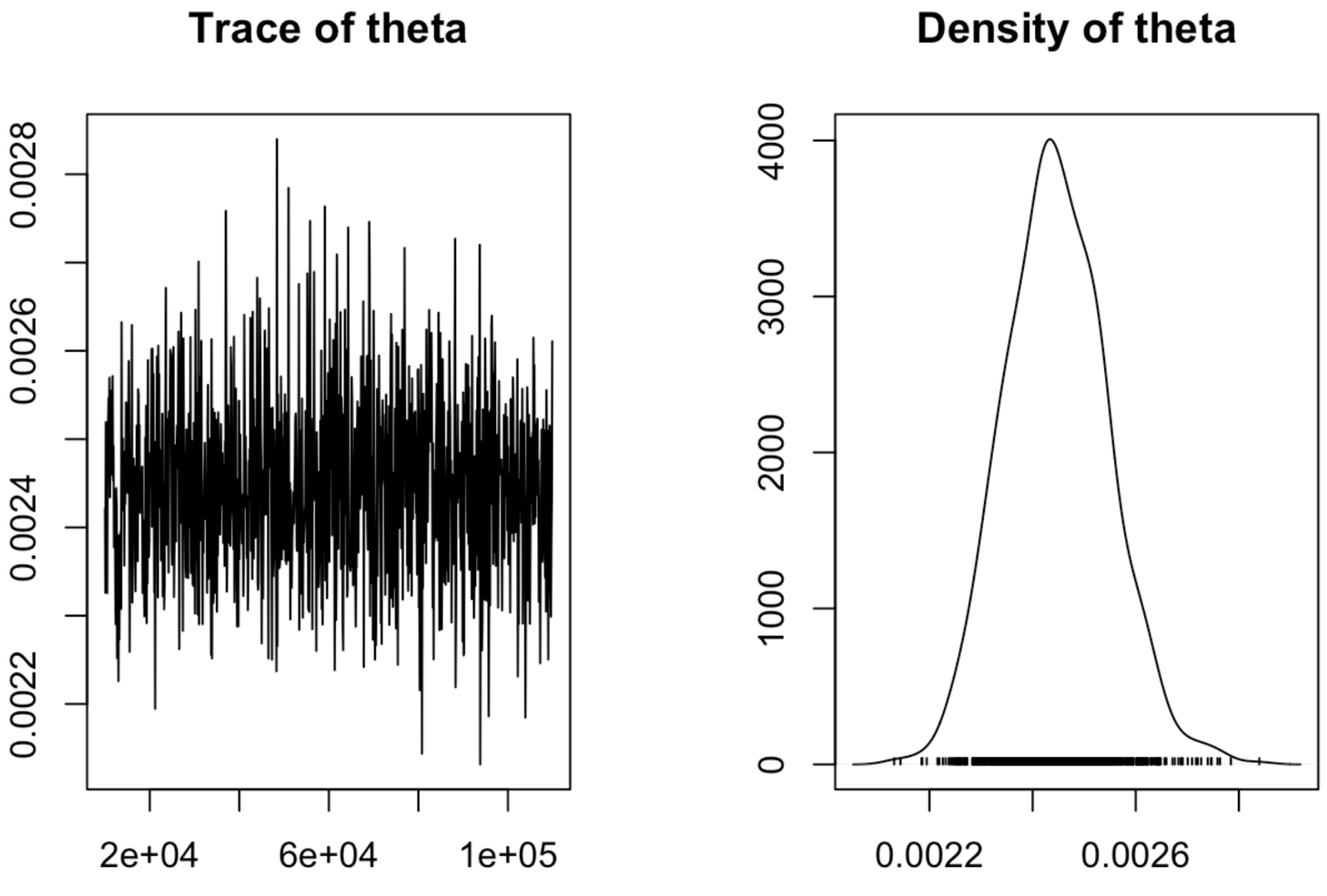
```
##  [1]  0  1  2 23 28  3 31 38  4  5
```

Next, we estimated θ using ThetaMater.M1 on this unfiltered dataset:

```
mcmc.seq.error <- ThetaMater.M1(k.vec= example.SeqError.alles$k.vec, l.vec = example.SeqError.alles$l.vec, n.vec
= example.SeqError.alles$n.vec, c.vec = example.SeqError.alles$c.vec, ngens = 100000, burnin = 10000, thin = 100,
theta.shape = 2, theta.scale = 0.001)
```

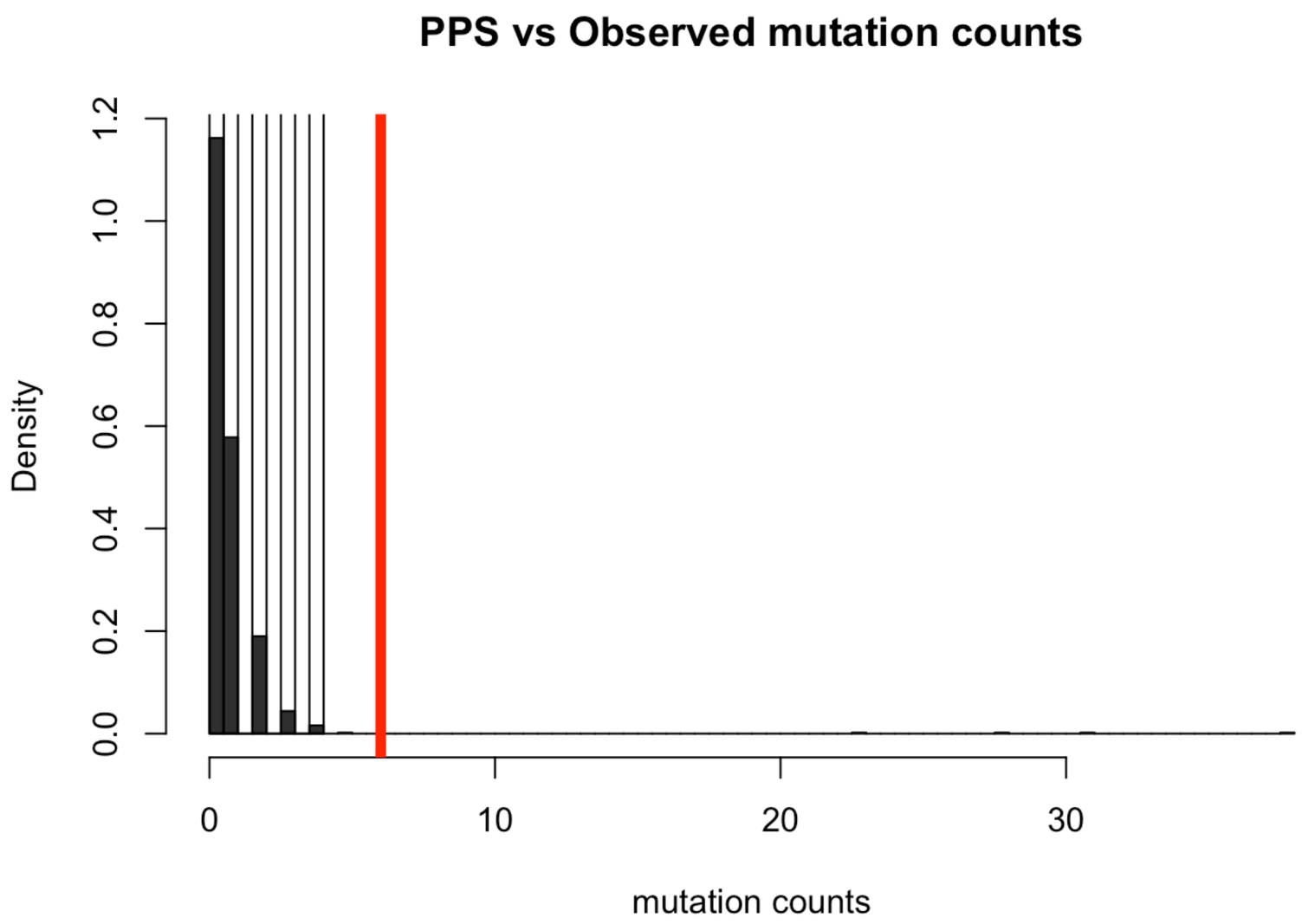
And here are the results:

```
data(example.MCMC.SequenceError, package = "ThetaMater")
varnames(example.MCMC.SequenceError) = "theta"
plot(as.mcmc(example.MCMC.SequenceError))
```



Now let's conduct PPS using ThetaMater.PPS and overlay the distributions to see if we can identiy the 4 outlier loci:

```
# run PPS analysis using this command:
example.PPS.results <- ThetaMater.PPS(theta.MCMC = example.MCMC.SequenceError[,1], l.vec = example.SeqError.alles
$l.vec, n.vec = example.SeqError.alles$n.vec)
hist(rep(example.SeqError.alles$k.vec, example.SeqError.alles$c.vec), col="gray20",add=F, freq = F, breaks = 100,
main = "PPS vs Observed mutation counts", xlab = "mutation counts")
hist(example.PPS.results, freq = T, breaks = 10, add = T)
abline(v=6, col="red", lwd = 5)
```



Here the red line shows the maximum number of mutations observed in the PPS data. So, let's remove the four extreme loci that are beyond this value and reestimate θ after filtering using this PPS distribution

```
max(example.PPS.results)
```

```
## [1] 4
```

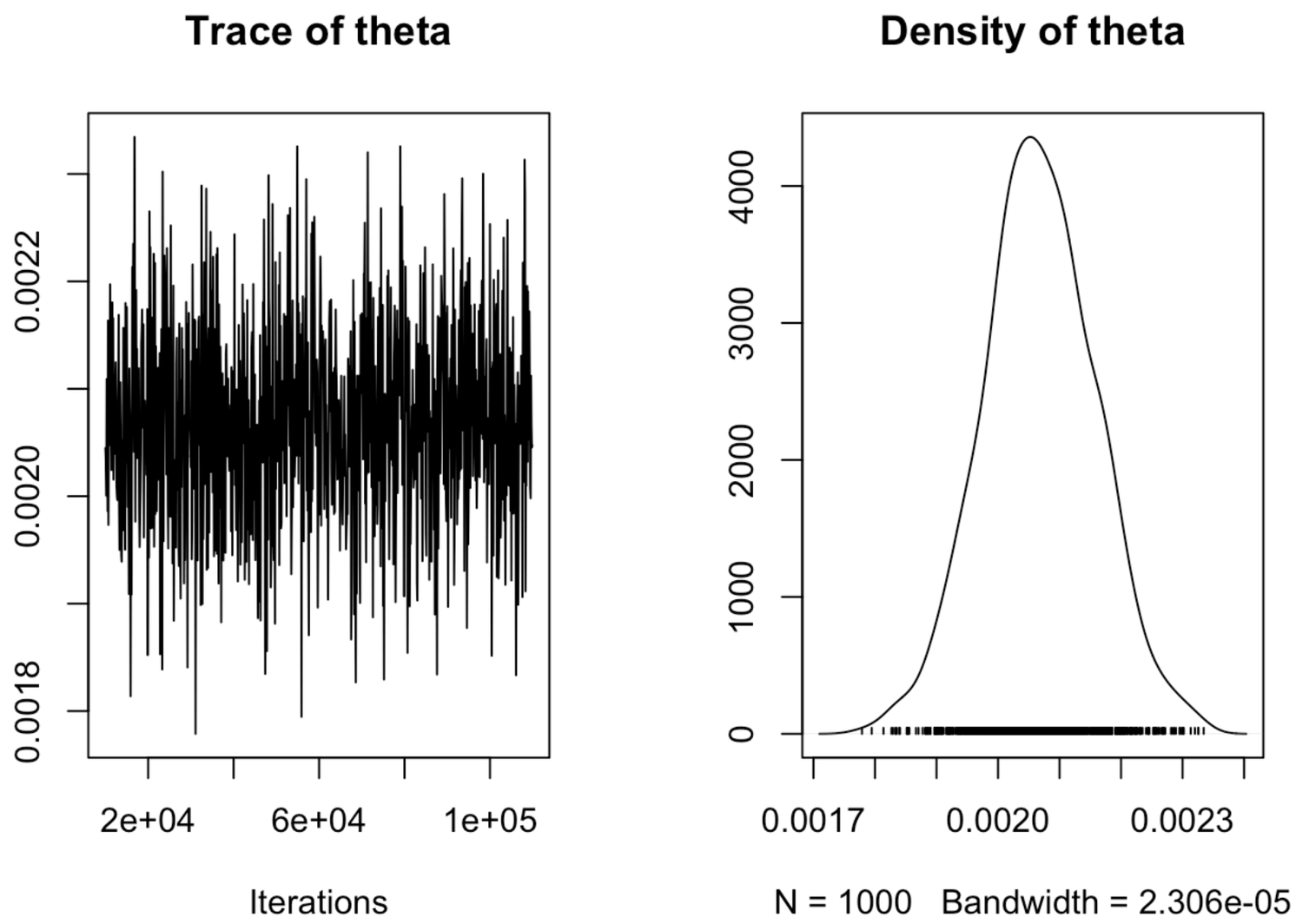
```
example.Filtered <- FilterData.PPS(dataset = example.SeqError.alles, threshold = 6)
```

Now, let's estimate θ after PPS filtering using the following commands:

```
# Now let's estimate theta after PPS filtering
example.MCMC.PostFilter <- ThetaMater.M1(k.vec = example.Filtered$k.vec, l.vec = example.Filtered$l.vec, n.vec =
example.Filtered$n.vec, c.vec = example.Filtered$c.vec, ngens = 100000, burnin = 10000, thin = 100, theta.shape =
2, theta.scale = 0.001)
```

And let's load these results below:

```
data(example.MCMC.PostFilter, package = "ThetaMater")
varnames(example.MCMC.PostFilter) = "theta"
plot(as.mcmc(example.MCMC.PostFilter))
```



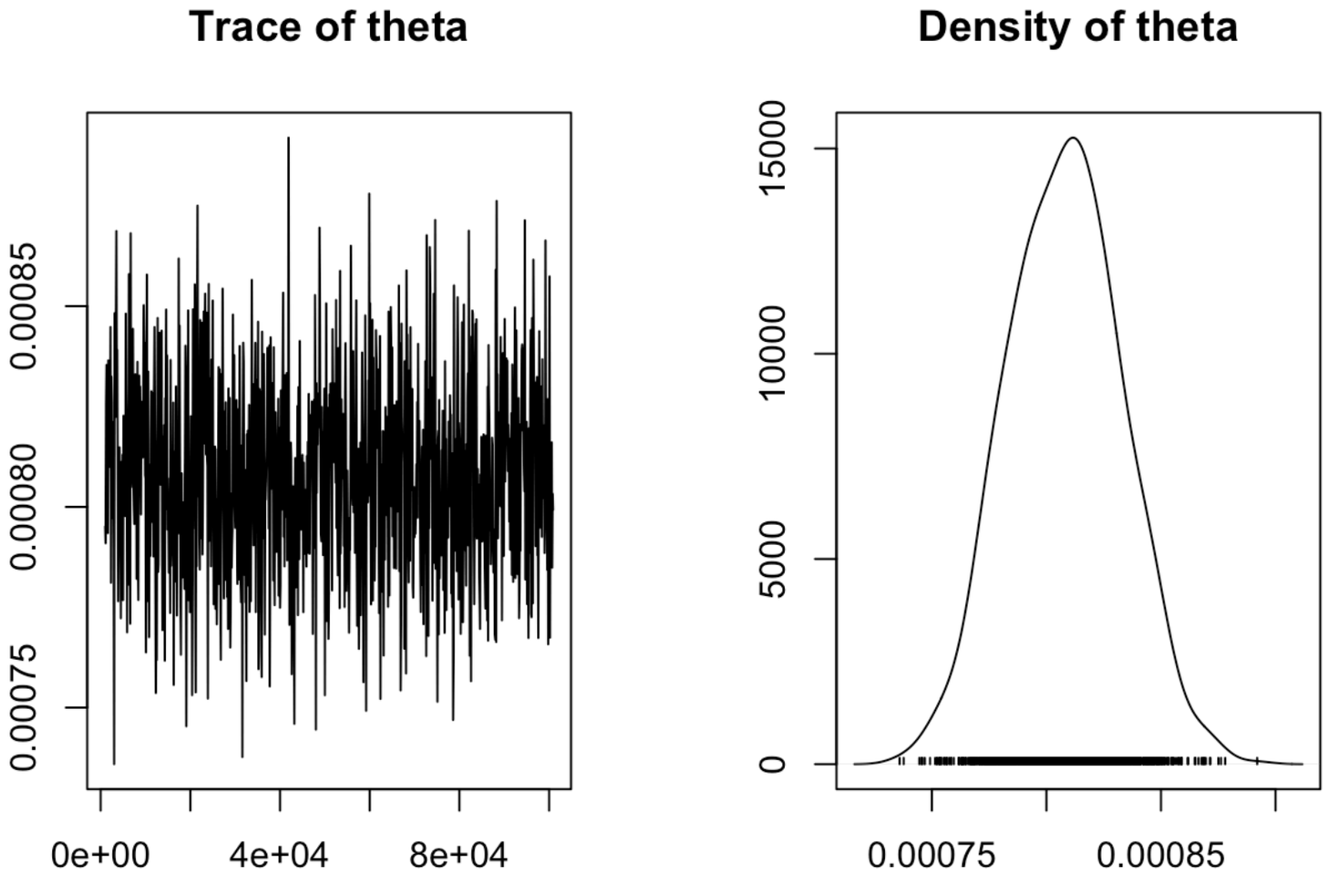
Step 8: Recombination & ThetaMater

We conducted a simple simulation analyses to evaluate the behavior of ThetaMater in the presence of recombination. We simulated 6 datasets using the program msprime and varied the recombination rate for each dataset (2e-4, 2e-5, 2e-6, 2e-7, 2e-8, 2e-9). We used the following command in msprime to simulate these data (10,000 loci each):

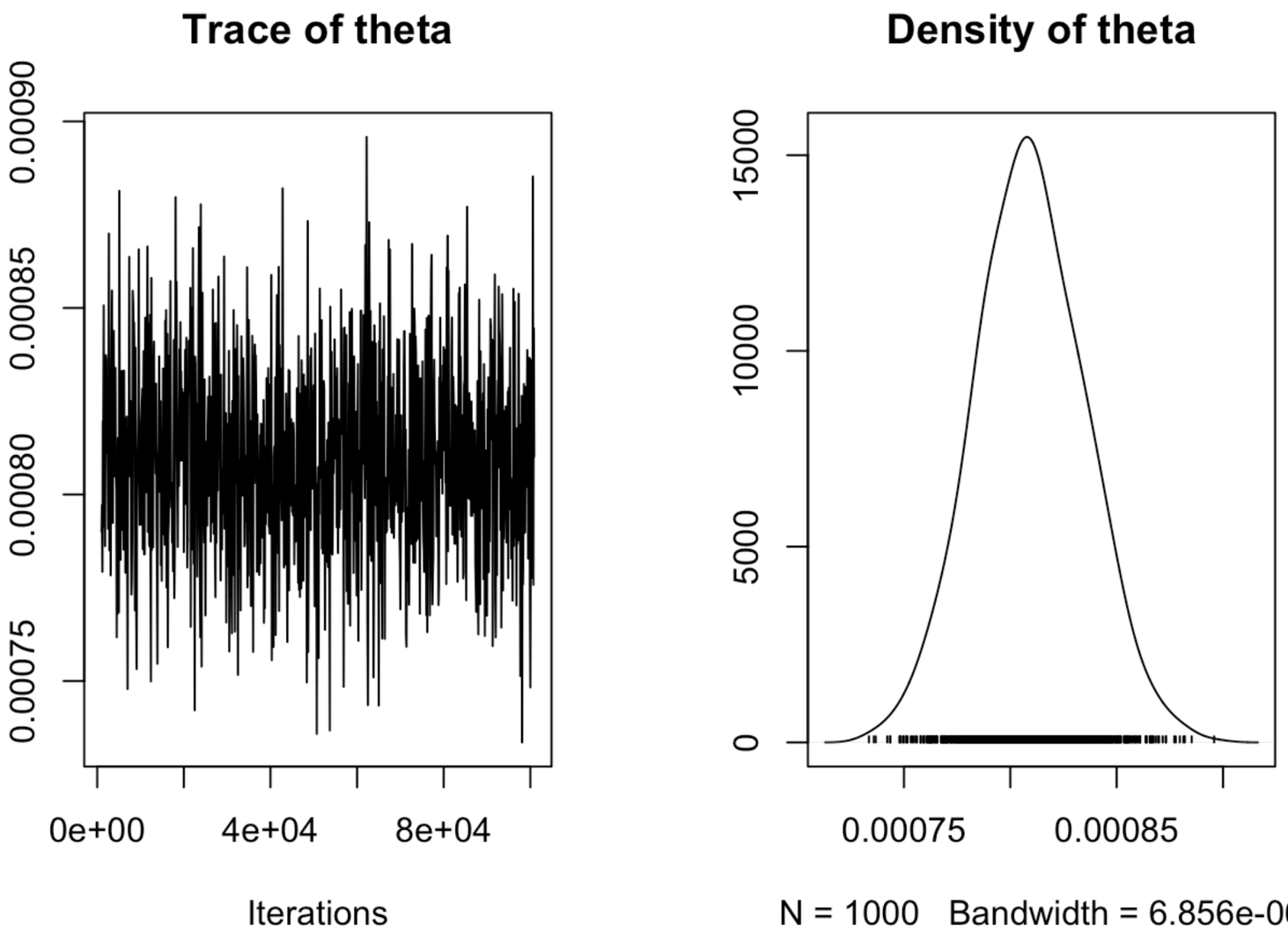
```
msprime.simulate(sample_size=5, Ne=10000, length=1000, recombination_rate=recombination_rate, mutation_rate=2e-8)
```

The results of these analyses are plotted below for each recombination rate. As you can see, in all cases, the posterior distribution of θ was tightly peaked at the true simulated value (θ = 0.0008). Nonetheless, we encourage users to explore all potential violations of the coalescent model prior to using ThetaMater (i.e., recombination, linkage, selection). If there is some concern for model violations, one can simulate datasets (as we have done with msprime) to explore other potential violations, including linkage and selection.

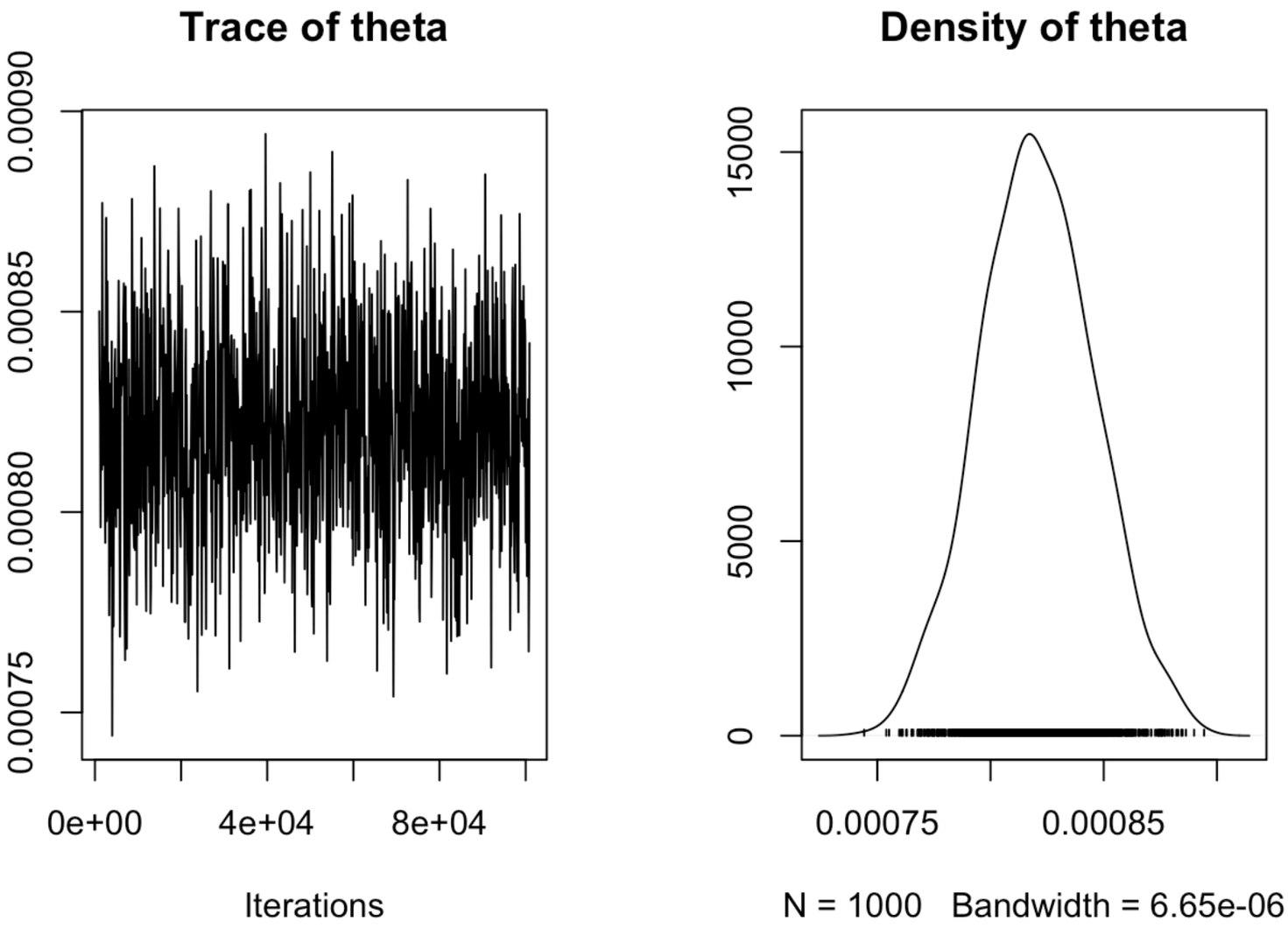
```
# load the data
data(example.r4.MCMC, package = "ThetaMater") # 2e-4
data(example.r5.MCMC, package = "ThetaMater") # 2e-5
data(example.r6.MCMC, package = "ThetaMater") # 2e-6
data(example.r7.MCMC, package = "ThetaMater") # 2e-7
data(example.r8.MCMC, package = "ThetaMater") # 2e-8
data(example.r9.MCMC, package = "ThetaMater") # 2e-9

varnames(example.r4.MCMC) <- "theta"
varnames(example.r5.MCMC) <- "theta"
varnames(example.r6.MCMC) <- "theta"
varnames(example.r7.MCMC) <- "theta"
varnames(example.r8.MCMC) <- "theta"
varnames(example.r9.MCMC) <- "theta"

plot(as.mcmc(example.r4.MCMC))# 2e-4
```



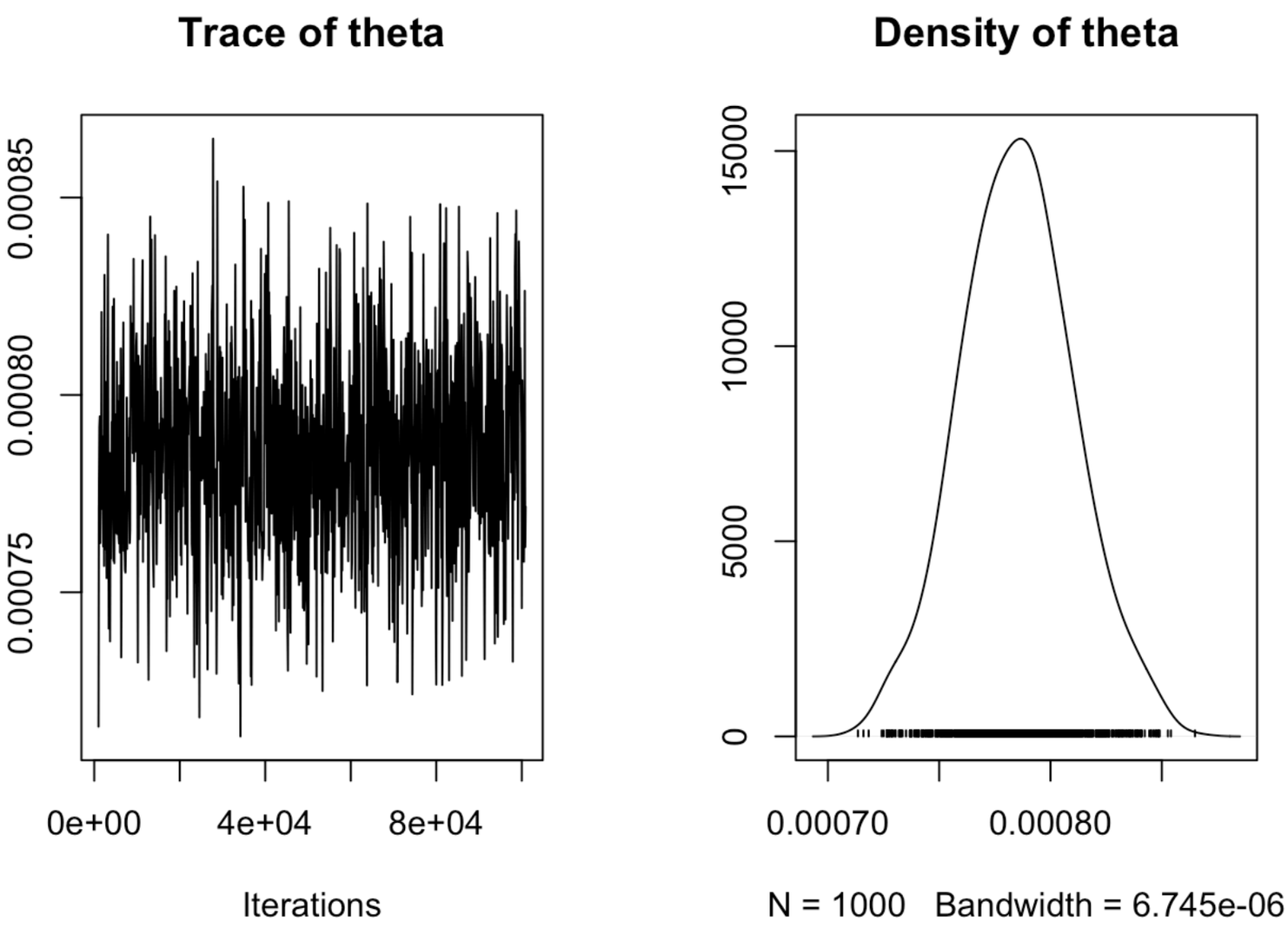

```
plot(as.mcmc(example.r5.MCMC))# 2e-5
```



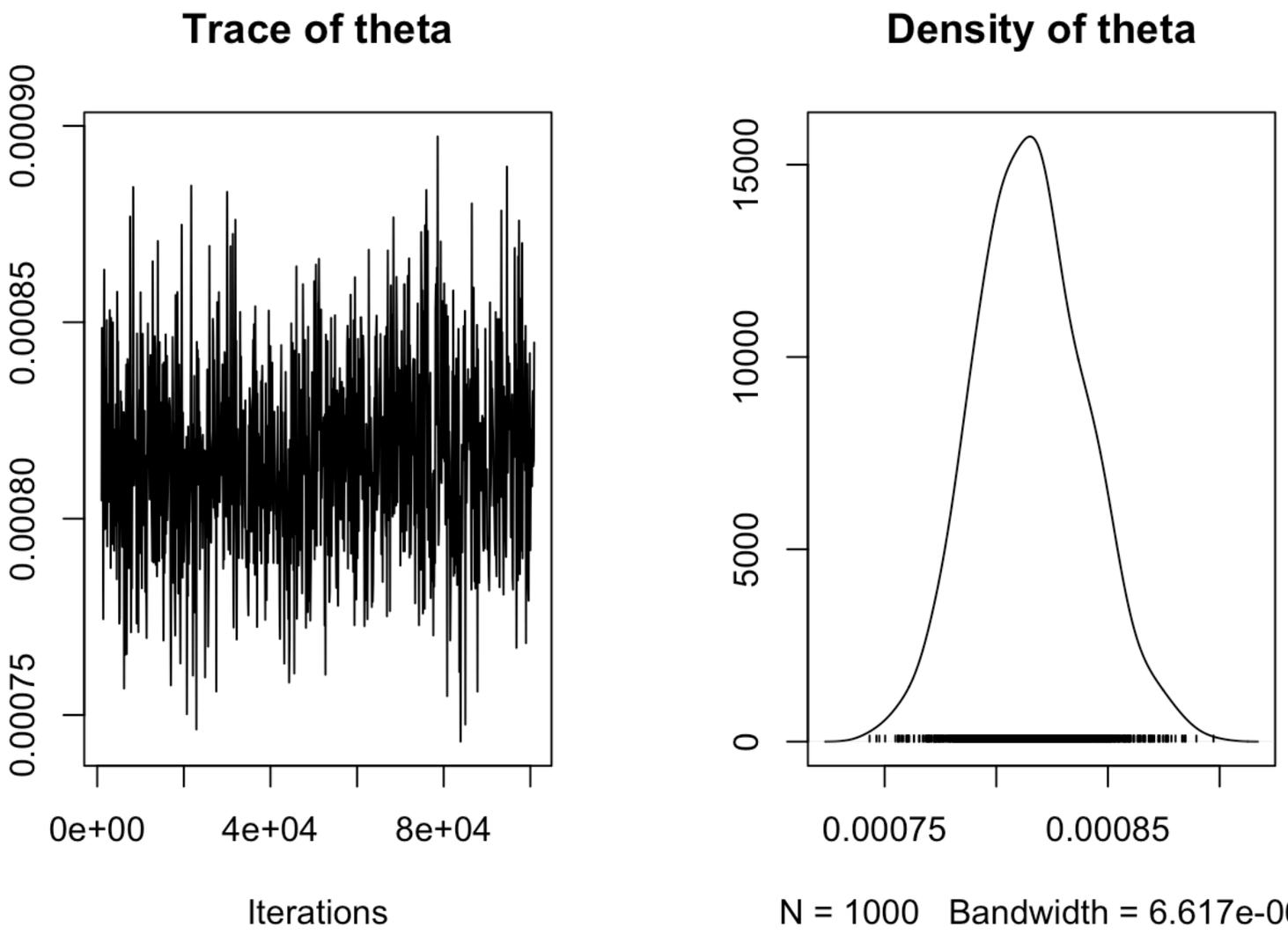

```
plot(as.mcmc(example.r6.MCMC))# 2e-6
```




```
plot(as.mcmc(example.r7.MCMC))# 2e-7
```




```
plot(as.mcmc(example.r8.MCMC))# 2e-8
```




```
plot(as.mcmc(example.r9.MCMC))# 2e-9
```