Morphometric MCMC Example (mcmc Version 0.9)

Leif T. Johnson Charles J. Geyer

June 15, 2012

1 Overview

This is an example how to use morphometric Markov chains as implemented in the `mcmc` package in R.

Let X be a $\mathbb{R}^k$ valued random variable with probability density function, f_X. Let g be a diffeomorphism, and $Y = g(X)$. Then the probability density function of Y, f_Y is given by

$$f_Y(y) = f_X\big(g^{-1}(y)\big) \det\big(\nabla g^{-1}(y)\big). \tag{1}$$

Since g is a diffeomorphism, we can draw inference about X from information about Y (and vice versa).

It is not unusual for f_X to either be known only up to a normalizing constant, or to be analytically intractable in other ways — such as being high dimensional. A common solution to this problem is to use Markov chain Monte Carlo (MCMC) methods to learn about f_X.

When using MCMC, a primary concern of the practitioner should be the question "Does the Markov chain converge fast enough to be useful?" One very useful convergence rate is called *geometrically ergodic* (Johnson, 2011, Chapter 1).

The `mcmc` package implements the Metropolis random-walk algorithm for arbitrary log unnormalized probability densities. But the Metropolis random-walk algorithm does not always perform well. As is demonstrated in Johnson and Geyer (submitted), for f_X and f_Y related by diffeomorphism as in (1), a Metropolis random-walk for f_Y can be geometrically ergodic even though a Metropolis random-walk for f_X is not. Since the transformation is one-to-one, inference about f_X can be drawn from the Markov chain for f_Y.

The `morph.metrop` and `morph` functions in the `mcmc` package provide this functionality, and this vignette gives a demonstration on how to use them.

2 T Distribution

We start with a univariate example, which is a Student t distribution with three degrees of freedom. Of course, one doesn't need MCMC to simulate this distribution (the R function `rt` does that), so this is just a toy problem. But it does illustrate some aspects of using variable transformation.

A necessary condition for geometric ergodicity of a random-walk Metropolis algorithm is that the target density π have a moment generating function (Jarner and Tweedie, 2003). For a univariate target density, which we have in this section, a sufficient condition for geometric ergodicity of a random-walk Metropolis algorithm is that the target density π be exponentially light Mengersen and Tweedie (1996). Thus if we do not use variable transformation, the Markov chain simulated by the `metrop` function will not be geometrically ergodic. Johnson and Geyer (submitted, Example 4.2) show that a t distribution is sub-exponentially light. Hence using the transformations described in their Corollaries 1 and 2 will induce a target density π_γ for which a Metropolis random-walk will be geometrically ergodic. using the transformation described as h_2 in Johnson and Geyer (submitted, Corollary 2) will induce a target density for which a Metropolis random-walk will be geometrically ergodic.

Passing a positive value for `b` to `morph` function will create the aforementioned transformation, h_2. It's as simple as

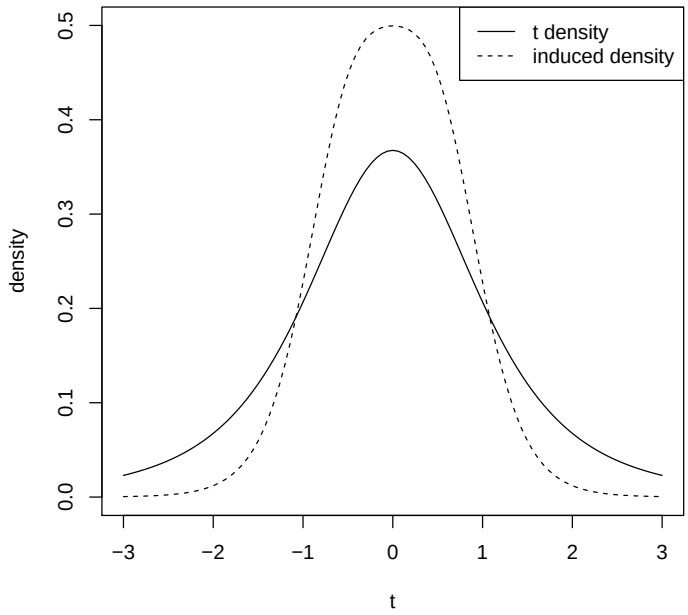
```
> library(mcmc)
> h2 <- morph(b=1)
```

We can now see the induced density. Note that `morph` works for log unnormalized densities, so we need exponentiate the induced density to plot it on the usual scale.

```
> lud <- function(x) dt(x, df=3, log=TRUE)
> lud.induced <- h2$lud(lud)
```

We can plot the two densities,

```
> curve(exp(Vectorize(lud.induced)(x)), from = -3, to = 3, lty = 2,
+     xlab = "t", ylab = "density")
> curve(exp(lud(x)), add = TRUE)
> legend("topright", c("t density", "induced density"), lty=1:2)
```



The `Vectorize` in this example is necessary because the function `lud.induced` is not vectorized. Instead, it treats any vector passed as a single input, which is rescaled (using the specified diffeomorphism) and passed to `lud`. Compare the behavior of `lud` and `lud.induced` in the following example.

```
> lud(1:4)

[1] -1.576253 -2.695485 -3.773478 -4.692542

> lud(1)

[1] -1.576253

> foo <- try(lud.induced(1:4))
> class(foo)

[1] "try-error"

> cat(foo, "\n")

Error in lud.induced(1:4) :
  log unnormalized density function returned vector not scalar

> lud.induced(1)
```

```
[1] -1.479686
```

Because the function `dt` is vectorized, the function `lud` is also vectorized, mapping vectors to vectors, whereas the function `lud.induced` is not vectorized, mapping vectors to scalars.

Before we start using random numbers, we set the see of the random number generator so this document always produces the same results.

```
> set.seed(42)
```

To change the results, change the seed or delete the `set.seed` statement.

Running a Markov chain for the induced density is done with `morph.metrop`.

```
> out <- morph.metrop(lud, 0, blen=100, nbatch=100, morph=morph(b=1))
```

The content of `out$batch` is on the scale of used by `lud`. Once the transformation has been set, no adjustment is needed (unless you want to change transformations). We start by adjusting the scale.

```
> # adjust scale to find a roughly 20% acceptance rate
> out$accept
```

```
[1] 0.6309
```

An acceptance rate of 63.1% is probably too high. By increasing the scale of the proposal distribution we can bring it down towards 20%.

```
> out <- morph.metrop(out, scale=4)
> out$accept
```

```
[1] 0.2339
```

We now use this Markov chain to estimate the expectation of the target distribution. But first we need to check whether our batch length is good. The following code

```
> acf(out$batch)
```

makes the autocorrelation plot (Figure 1). It looks like there is no significant autocorrelation among the batches so the following produces a valid confidence interval for the true unknown mean of the target distribution (since this is a toy problem we actually know the true "unknown" mean is zero, but we pretend we don't know that for the purposes of the toy problem)

```
> t.test(out$batch)
```

```
        One Sample t-test

data:  out$batch
t = 1.3684, df = 99, p-value = 0.1743
```

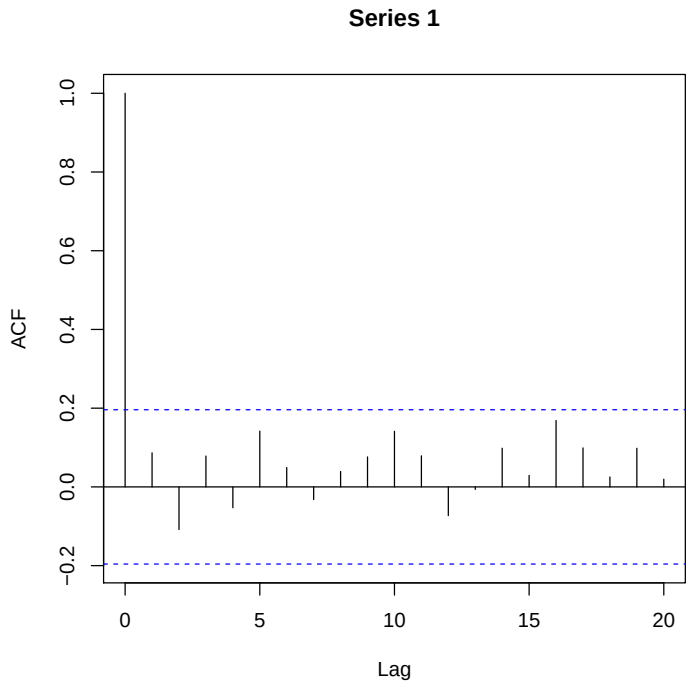


Figure 1: Autocorrelation plot for the sequence of batch means.

```
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 -0.02565067  0.13963647
sample estimates:
mean of x
0.0569929
```

If we want a point estimate and a Monte Carlo standard error, those are

```
> colMeans(out$batch)

[1] 0.0569929

> apply(out$batch, 2, sd) / sqrt(out$nbatch)

[1] 0.04165047
```

If a shorter confidence interval is desired, the Markov chain can be run longer (increase either the number of batches or the batch length, or both).

Note that when calculating our estimate and the Monte Carlo standard error we are not concerned with what was happening on the transformed scale. The `morph.metrop` function seamlessly does this for us.

2.1 Comparison of Morphed and Unmorphed

To show the utility of the transformation, we will study the behavior of the Markov chain with and without the transformation for the same problem as in the preceding section. We will consider two different estimation methods.

1. Estimate the mean of the target distribution using a random-walk Metropolis algorithm implemented by the `metrop` function. Jarner and Roberts (2007) demonstrate that a central limit theorem does not hold for these estimates.

2. Estimate the mean of the target distribution using a random-walk Metropolis algorithm implemented by the `morph.metrop` function with argument `morph = morph(b=1)`. Johnson and Geyer (submitted) demonstrate that a central limit does hold for these estimates.

We do the latter first, since we are already set up for it. But we use an unbatched and longer run.

```
> out.morph <- morph.metrop(out, blen = 1, nbatch = 1e5)
> out.morph$accept

[1] 0.22922
```

For the former, we need to adjust the scale.

```
> out <- metrop(lud, 0, blen=1000, nbatch=1)
> out$accept
```

```
[1] 0.719
```

```
> out <- metrop(out, scale=4)
> out$accept
```

```
[1] 0.346
```

```
> out <- metrop(out, scale=6)
> out$accept
```

```
[1] 0.258
```

A scale of 6 appears to be about right. Now we do a similar long run for this sampler.

```
> out.unmorph <- metrop(out, blen = 1, nbatch = 1e5)
> out.unmorph$accept
```

```
[1] 0.2577
```

The following code

```
> oldpar <- par(mar = c(1, 4, 1, 1), mfrow = c(2, 1))
> plot(as.vector(out.morph$batch), xlab = "", axes = FALSE,
+     ylab = "morphed chain", pch = ".")
> axis(side = 2)
> box()
> plot(as.vector(out.unmorph$batch), xlab = "", axes = FALSE,
+     ylab = "unmorphed chain", pch = ".")
> axis(side = 2)
> box()
> par(oldpar)
```

makes time series plots for the two chains (Figure 2). Both appear stationary and it is hard to tell the difference from these plots. The range of the unmorphed plot appears wider but that is only due to a few points and so may not be correct.

The following code

```
> oldpar <- par(mar = c(5, 4, 1, 1), mfrow = c(2, 1))
> acf(as.vector(out.morph$batch), type = "covariance",
+     ylab = "morphed chain")
> acf(as.vector(out.unmorph$batch), type = "covariance",
+     ylab = "unmorphed chain")
> par(oldpar)
```

makes autocovariance plots for the two chains (Figure 3). Again, not much apparent difference.

The following code compares Monte Carlo standard errors

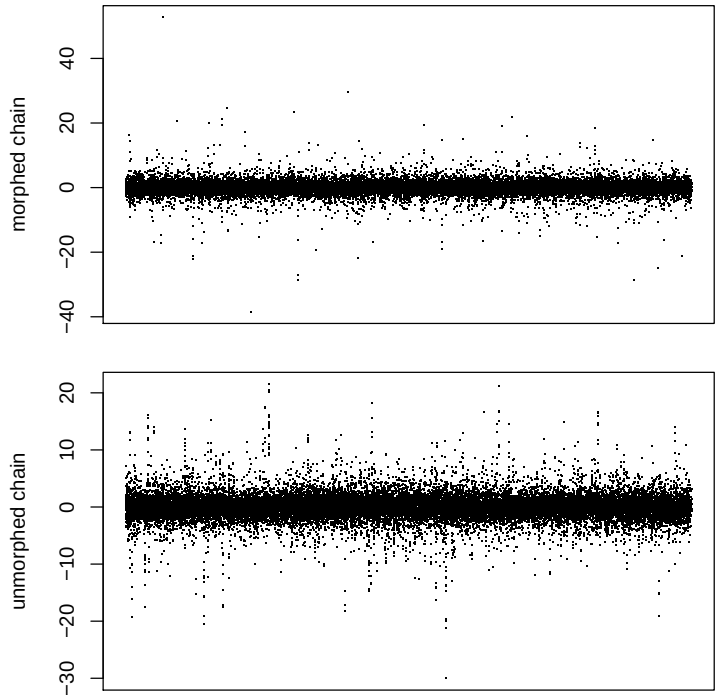


Figure 2: Time series plots for morphed and unmorphed chains.

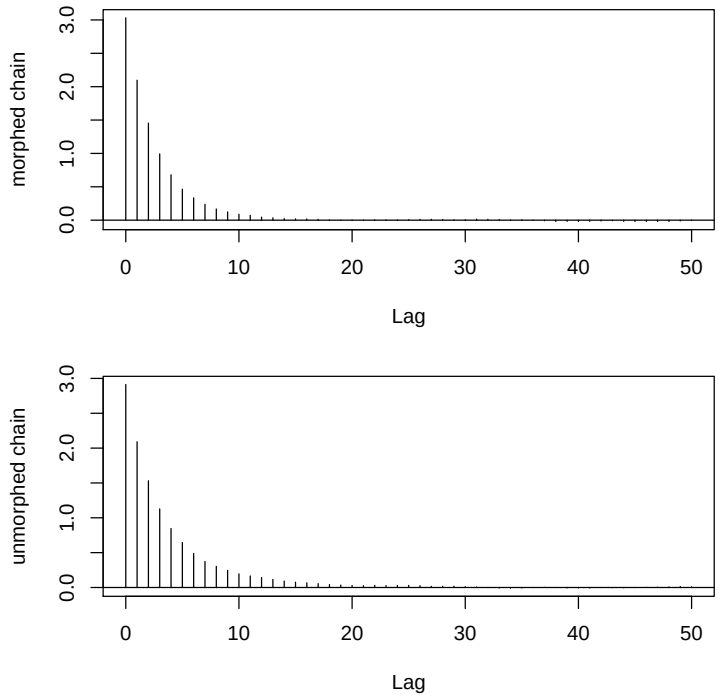


Figure 3: Autocovariance plots for morphed and unmorphed chains.

```
> sqrt(initseq(as.vector(out.morph$batch))$var.con / out.morph$nbatch)

[1] 0.01294456

> sqrt(initseq(as.vector(out.unmorph$batch))$var.con / out.morph$nbatch)

[1] 0.01434341

> initseq(as.vector(out.morph$batch))$Gamma.con

 [1] 5.128840e+00 2.448004e+00 1.143790e+00 5.722395e-01
 [5] 2.904791e-01 1.614450e-01 8.058423e-02 4.243122e-02
 [9] 2.284201e-02 3.252797e-03 1.301043e-18

> initseq(as.vector(out.unmorph$batch))$Gamma.con

 [1] 5.004351e+00 2.659592e+00 1.492292e+00 8.621914e-01
 [5] 5.489926e-01 3.613866e-01 2.592166e-01 1.692117e-01
 [9] 1.229689e-01 7.672616e-02 5.321689e-02 4.434741e-02
[13] 3.547793e-02 2.660845e-02 1.773896e-02 8.869482e-03
[17] 2.890923e-18
```

Again, not much difference. Although we know the unmorphed chain is much worse in theory, it seems to work in practice. (More study is needed here, perhaps looking at longer runs. But we don't want to take the time for really long runs in a vignette.)

Let's look at the distribution of batch means.

```
> blen <- 100
> foo <- as.vector(out.unmorph$batch)
> stopifnot(length(foo) %% blen == 0)
> foo <- matrix(foo, nrow = blen)
> foo <- colMeans(foo)
```

The following code

```
> qqnorm(foo)
> qqline(foo)
```

makes a Q-Q plot of the batch means (Figure 4). Finally we see bad behavior of the unmorphed chain. These batch means (or at least some batch means for sufficiently long batch length) should look normally distributed, and these don't. Not even close.

But on further thought, this is just heavy-tailedness of the target distribution. If one makes a Q-Q plot for means of sample size 100 for independent and identically distributed $t(3)$ random variables one sees more or less the same thing. One needs sample size 300 for approximate normality (as indicated by a Q-Q plot) for this distribution. For MCMC we need about 300 times the lag at which autocorrelations become insignificant, so we need batch length about 5000. Let's try that.

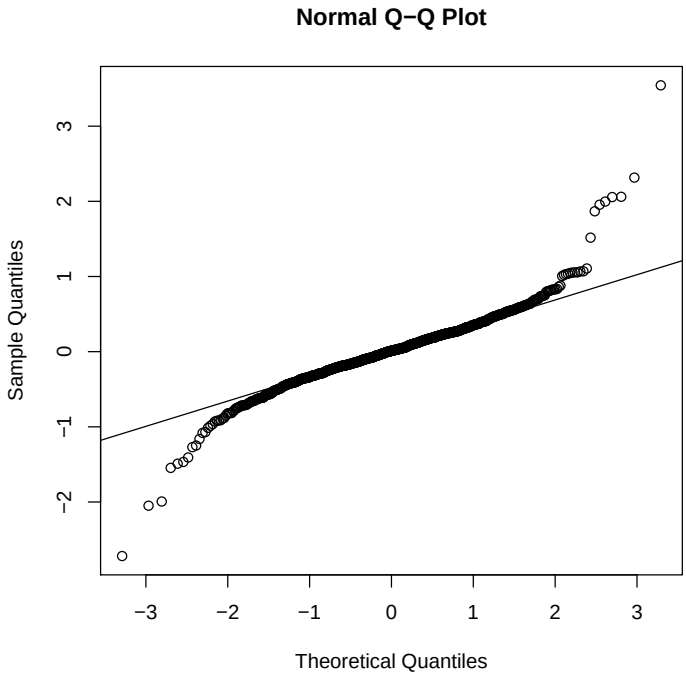


Figure 4: Q-Q plot of batch means (batch length 100) for the unmorphed chain.

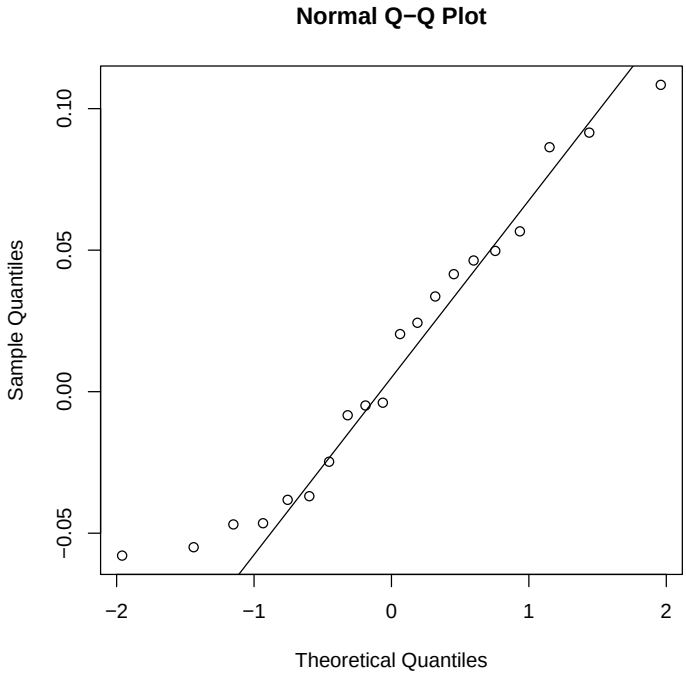


Figure 5: Q-Q plot of batch means (batch length 5000) for the unmorphed chain.

```
> blen <- 5000
> foo <- as.vector(out.unmorph$batch)
> stopifnot(length(foo) %% blen == 0)
> foo <- matrix(foo, nrow = blen)
> foo <- colMeans(foo)
```

The following code

```
> qqnorm(foo)
> qqline(foo)
```

makes a Q-Q plot of the batch means (Figure 5). Better, in fact

```
> shapiro.test(foo)

        Shapiro-Wilk normality test

data:  foo
W = 0.9415, p-value = 0.2561
```

not significantly different from normally distributed (although with such a small number of batch means, perhaps that is just low power).

Since last issue we encountered (non-normality of batch means for moderate but not really large batch length) is caused by the heavy-tailedness of the target distribution, the morphed chain will have all the same issues.

This suggests that we should perhaps use a more robust estimator of scale than the standard deviation of the batch means which using small batches (length 100), but since there is nothing in the literature about this (as far as we know), we will say no more about it.

3 Binomial Distribution with a Conjugate Prior

We demonstrate a morphometric Markov chain using the `UCBAdmisions` data set included in `R`, (use `help(UCBAdmissions)` to see details of this data set). We will model the probability of a student being admitted or rejected, using the sex of the student and the department that the student applied to as predictor variables. For our prior, we naively assume that 30% of all students are admitted, independent of sex or department. As this is a naive prior, we will only add 5 students to each gender-department combination. This will not give the prior much weight, most of the information in the posterior distribution will be from the data.

If we have L observations from a multinomial distribution, then using a multinomial logit-link, with model matrices $M^1, \ldots, M^L$, regression parameter β, observed counts $Y^1, \ldots, Y^N$ with observed sample sizes $N^1, \ldots, N^L$ and prior probabilities $\xi^1, \ldots, \xi^L$ and prior "sample sizes" $\nu^1, \ldots, \nu^L$ then the posterior distribution of β is given by (Johnson, 2011, Sec. 5.1.2)

$$\pi(\beta|y, n, \xi, \nu) \propto \exp\left\{ \sum_{l=1}^{L} \langle y^l + \xi^l \nu^l, M^l \beta \rangle - (n^l + \nu^l) \log\left(\sum_j e^{M_j \cdot \beta} \right) \right\} \qquad (2)$$

where $\langle a, b \rangle$ denotes the usual inner product between vectors a and b. For our application, we can simplify this in two ways.

First, we use the posterior counts instead of the sum of the prior and data counts, i.e. use $y^{*l} = y^l + \xi^l \nu^l$ and $n^{*l} = n^l + \nu^l$.

Second, to avoid having a direction of recession in $\pi(\beta|\cdot)$, we need to fix the elements of β that correspond with one of the response categories. Since we are going to fitting a binomial response, if we set these elements of β to be 0, we may then replace the sequence of model matrices with a single model matrix; M instead of $M^1, \ldots, M^L$. The l-th row of M will correspond to M^l. Label the two response categories A and B. Without loss of generality, we will fix the elements of β corresponding to category B to 0.

Let $x_1, \ldots, x_L$ represent the posterior counts of category A, and β^* represent the corresponding elements of β — these are the elements of β we did not fix as 0. The meaning of $n^{*1}, \ldots, n^{*L}$ is unchanged. Then our simplified unnormalized

posterior density is

$$\pi(\beta|x, n^*) \propto \exp\left\{ \langle x, M\beta^* \rangle - \sum_{l=1}^{L} n^{*l} \log\bigl(1 + e^{(M\beta^*)_l}\bigr) \right\}. \tag{3}$$

This can be computed with a very simple `R` function, we implement it in log form.

```
> lud.binom <- function(beta, M, x, n) {
+   MB <- M %*% beta
+   sum(x * MB) - sum(n * log(1 + exp(MB)))
+ }
```

Now that we have a function to calculate a log-unnormalized posterior density, we can run the Markov chain. To that, we need the model matrix. First we convert the `UCAdmissions` data to a `data.frame`.

```
> dat <- as.data.frame(UCBAdmissions)
> dat.split <- split(dat, dat$Admit)
> dat.split <- lapply(dat.split,
+                     function(d) {
+                       val <- as.character(d$Admit[1])
+                       d["Admit"] <- NULL
+                       names(d)[names(d) == "Freq"] <- val
+                       d
+                     })
> dat <- merge(dat.split[[1]], dat.split[[2]])
```

Next we build the model matrix. Our model specification allows for an interaction between gender and department, even though our prior assumes that they are independent.

```
> formula <- cbind(Admitted, Rejected) ~ (Gender + Dept)^2
> mf <- model.frame(formula, dat)
> M <- model.matrix(formula, mf)
```

As stated above, we will take $\nu = 5$ and $\xi = 0.30$. That is, we will add 5 students to each gender-department combination, where each combination has a 30% acceptance rate.

```
> xi <- 0.30
> nu <- 5

> lud.berkeley <- function(B)
+   lud.binom(B, M, dat$Admitted + xi * nu, dat$Admitted + dat$Rejected + nu)
```

This function is suitable for passing to `metrop` or `morph.metrop`. We know that using `morph.metrop` with `morph=morph(p=3)` will run a geometrically ergodic Markov chain (Johnson and Geyer, submitted).

```
> berkeley.out <- morph.metrop(lud.berkeley, rep(0, ncol(M)), blen=1000,
+                              nbatch=1, scale=0.1, morph=morph(p=3))
> berkeley.out$accept

[1] 0.033

> berkeley.out <- morph.metrop(berkeley.out, scale=0.05)
> berkeley.out$accept

[1] 0.01

> berkeley.out <- morph.metrop(berkeley.out, scale=0.02)
> berkeley.out$accept

[1] 0.204

> berkeley.out <- morph.metrop(berkeley.out, blen=10000)
> berkeley.out$accept

[1] 0.1986

> berkeley.out <- morph.metrop(berkeley.out, blen=1, nbatch=100000)
```

Estimate the posterior mean acceptance probabilities for each gender-department combination.

```
> beta <- setNames(colMeans(berkeley.out$batch), colnames(M))
> MB <- M %*% beta
> dat$p <- dat$Admitted / (dat$Admitted + dat$Rejected)
> dat$p.post <- exp(MB) / (1 + exp(MB))
> dat

   Gender Dept Admitted Rejected          p     p.post
1  Female    A       89       19 0.82407407 0.79805171
2  Female    B       17        8 0.68000000 0.61832325
3  Female    C      202      391 0.34064081 0.34044730
4  Female    D      131      244 0.34933333 0.34851703
5  Female    E       94      299 0.23918575 0.23996275
6  Female    F       24      317 0.07038123 0.07313925
7    Male    A      512      313 0.62060606 0.61975136
8    Male    B      353      207 0.63035714 0.62813305
9    Male    C      120      205 0.36923077 0.36733047
10   Male    D      138      279 0.33093525 0.32993706
11   Male    E       53      138 0.27748691 0.27550362
12   Male    F       22      351 0.05898123 0.06118500
```

The small difference between the data and posterior probabilities is expected, our prior was given very little weight. Using `morph.metrop` with the setting

`morph=morph(p=3)` in this setting is an efficient way of sampling from the posterior distribution.

We can also compare the posterior distribution of admittance probability for each gender-department combination. Table 1 gives the 5% and 95% quantiles for the posterior distribution of the admittance probabilities for each gender-department combination. Figure 6 gives the same quantiles, plus the mean posterior-probability for each gender-department combination. From these we can see that for each department, there is considerable overlap of the distributions of probabilities for males and females.

```
> posterior.probabilities <-
+   t(apply(berkeley.out$batch, 1,
+           function(r) {
+             eMB <- exp(M %*% r)
+             eMB / (1 + eMB)
+           }))
> quants <- apply(posterior.probabilities, 2, quantile, prob=c(0.05, 0.95))
> quants.str <- matrix(apply(quants, 2,
+                            function(r) sprintf("[%0.2f, %0.2f]", r[1], r[2])),
+                      nrow=2, byrow=TRUE)
>
```

Table 1: 5% and 95% posterior quantiles for admittance probability for each gender-department combination

Gender	Dept. A	Dept. B	Dept. C	Dept. D	Dept. E.	Dept. F
Female	[0.73, 0.86]	[0.46, 0.76]	[0.31, 0.37]	[0.31, 0.39]	[0.21, 0.28]	[0.05, 0.10]
Male	[0.59, 0.65]	[0.59, 0.66]	[0.32, 0.41]	[0.29, 0.37]	[0.22, 0.33]	[0.04, 0.08]

4 Cauchy Location-Scale Model

We are going to do a Cauchy location-scale family objective Bayesianly.

4.1 Data

First we generate some data.

```
> n <- 15
> mu0 <- 50
> sigma0 <- 10
> x <- rcauchy(n, mu0, sigma0)
> round(sort(x), 1)
```

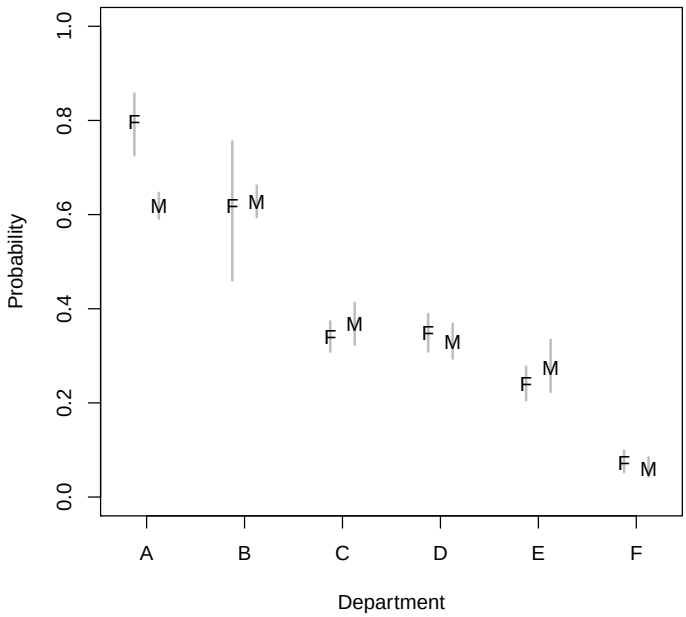


Figure 6: Posterior 5% and 95% quantiles and mean, by department and gender.

```
 [1] -48.1  22.5  35.1  37.2  42.5  44.9  47.8  48.5  53.0
[10]  53.2  57.7  80.8 106.5 118.2 224.2
```

`mu0` and `sigma0` are the true unknown parameter values (since the data are simulated we actually know these "unknown" parameter values, but we must pretend we don't know them and estimate them).

4.2 Prior

The standard objective prior distribution for this situation (insofar as any prior distribution can be said to be an objective standard) is the improper prior

$$g(\mu, \sigma) = \frac{1}{\sigma}$$

which is right Haar measure for the location-scale group, and is the standard prior that comes from the group invariance argument (Kass and Wasserman, 1996, Section 3.2).

4.3 Log Unnormalized Posterior

We need a function whose argument is a two-vector

```
> lup <- function(theta) {
+     if (any(is.na(theta)))
+         stop("NA or NaN in input to log unnormalized density function")
+     mu <- theta[1]
+     sigma <- theta[2]
+     if (sigma <= 0) return(-Inf)
+     if (any(! is.finite(theta))) return(-Inf)
+     result <- sum(dcauchy(x, mu, sigma, log = TRUE)) - log(sigma)
+     if (! is.finite(result)) {
+         warning(paste("Oops!  mu = ", mu, "and sigma =", sigma))
+     }
+     return(result)
+ }
```

4.4 Laplace Approximation

To have some idea what we are doing, we first maximize the log unnormalized posterior. To do it helps to have good starting points for the optimization. Robust estimators of location and scale are

```
> mu.twiddle <- median(x)
> sigma.twiddle <- IQR(x)
> c(mu.twiddle, sigma.twiddle)

[1] 48.45826 29.37265
```

The maximum likelihood estimator (MLE) is

```
> oout <- optim(c(mu.twiddle, sigma.twiddle), lup,
+     control = list(fnscale = -1), hessian = TRUE)
> stopifnot(oout$convergence == 0)
> mu.hat <- oout$par[1]
> sigma.hat <- oout$par[2]
> c(mu.hat, sigma.hat)
```

```
[1] 47.58766 10.30696
```

This is the posterior mode.

```
> oout$hessian
```

```
              [,1]          [,2]
[1,] -0.0724436404  0.0004741043
[2,]  0.0004741043 -0.0593415059
```

The hessian is nearly diagonal and one can check that theoretically is exactly diagonal. Thus approximate (asymptotic) posterior standard deviations are

```
> sqrt(- 1 / diag(oout$hessian))
```

```
[1] 3.715351 4.105071
```

4.5 Theory

To use the theory in Johnson and Geyer (submitted) we must verify that the target distribution (the unnormalized posterior) is everywhere positive, and it isn't (it is zero for $\sigma \leq 0$). We tried making $\log(\sigma)$ the parameter but this didn't work either because $\log(\sigma)$ goes to infinity so slowly that this stretches out the tails so much that the transformations introduced by Johnson and Geyer (submitted) can't pull them back in again. We do know (Johnson and Geyer, submitted, Example 3.4) that if we fix σ this is a sub-exponentially light target distribution. Letting σ vary can only make this worse. Thus, if we don't do anything and just use the `metrop` function, then performance will be very bad. So we are going to use the transformations and the `morph.metrop` function, even though the theory that motivates them does not hold.

4.6 Morph

We want to center the transformation at the posterior mode, and use a radius r that doesn't transform until several approximate standard deviations

```
> library(mcmc, lib.loc = "../package/mcmc.Rcheck")
> moo <- morph(b = 0.5, r = 7, center = c(mu.hat, sigma.hat))
> mout <- morph.metrop(lup, c(mu.hat, sigma.hat), 1e4,
+     scale = 3, morph = moo)
> mout$accept
```

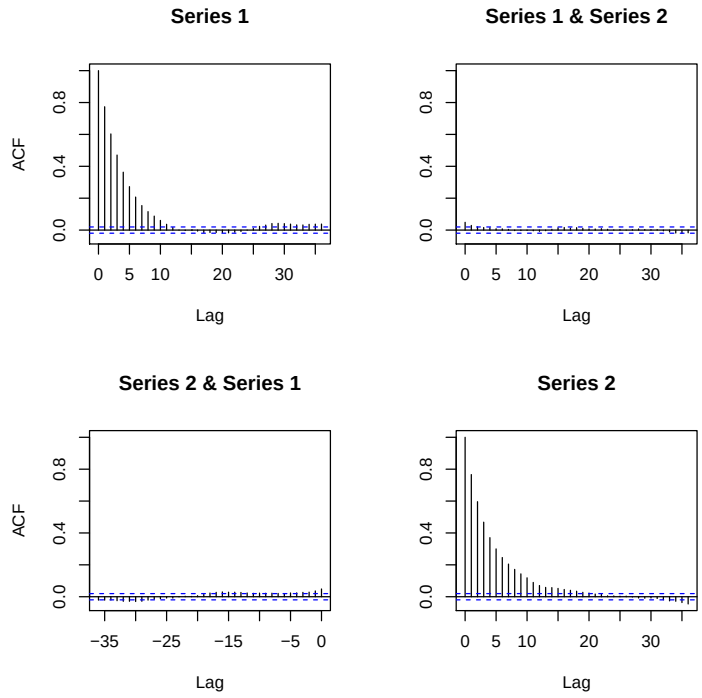


Figure 7: Autocorrelation plot. First component is μ, second is σ.

```
[1] 0.4525
```

```
> mout <- morph.metrop(mout)
```

Good enough. An attempt to increase the scale led to error when the transformation functions overflowed. Can't take steps too big with this stuff. The following code

```
> acf(mout$batch)
```

makes an autocorrelation plot (Figure 7). It looks like lag 10 to 15 is enough to get near independence.

Now we want to make marginal density plots. If we just feed our MCMC output to the R function `density` it undersmooths because it expects independent and identically distributed data rather than autocorrelated data. Thus we feed it subsampled, nearly uncorrelated data to select the bandwidth and then use that bandwidth on the full data. Here's how that works. The following code

```
> mu <- mout$batch[ , 1]
> i <- seq(1, mout$nbatch, by = 15)
```

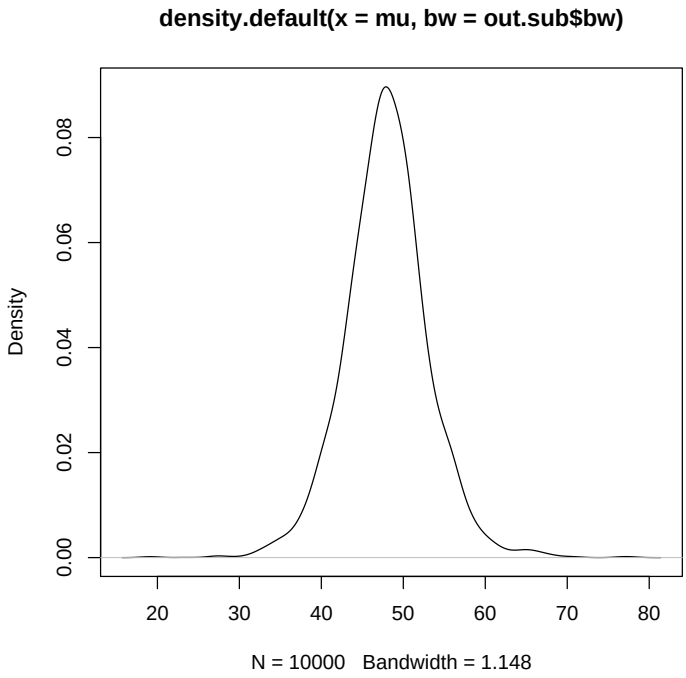


Figure 8: Density plot for the marginal posterior for μ.

```
> out.sub <- density(mu[i])
> out <- density(mu, bw = out.sub$bw)
> plot(out)
```

makes the density plot (Figure 8). And a similar plot for σ (Figure 9)

References

Jarner, S.F., and G.O. Roberts (2007). Convergence of heavy-tailed Monte Carlo Markov chain algorithms. *Scandinavian Journal of Statistics*, 34, 781–815.

Jarner, S. F., and Tweedie, R. L. (2003). Necessary conditions for geometric and polynomial ergodicity of random-walk-type Markov chains. *Bernoulli*, 9, 559–578.

Johnson, L. T. (2011). Geometric Ergodicity of a Random-Walk Metropolis Algorithm via Variable Transformation and Computer Aided Reasoning in Statistics. Ph. D. thesis. University of Minesota. `http://purl.umn.edu/113140`

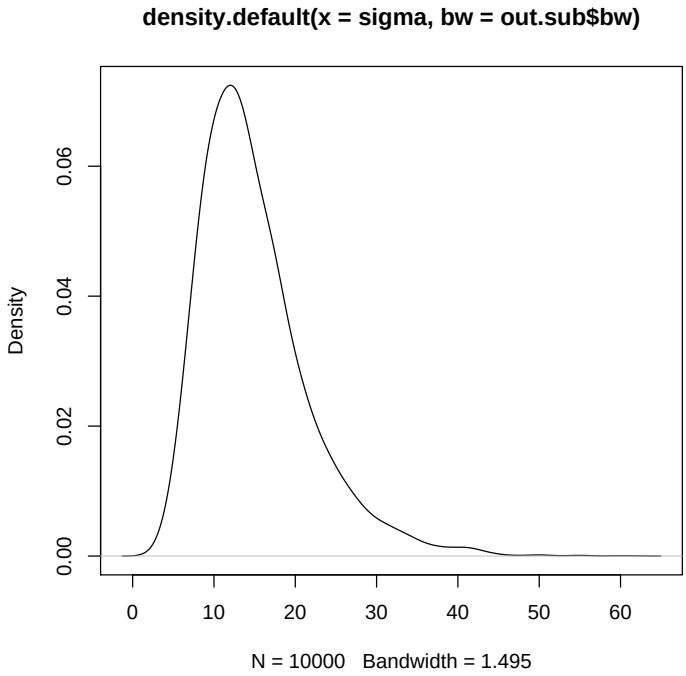


Figure 9: Density plot for the marginal posterior for σ.

Johnson, L. T., and C. J. Geyer (submitted). Variable Transformation to Obtain Geometric Ergodicity in the Random-walk Metropolis Algorithm. Revised and resubmitted to *Annals of Statistics*.

Kass, R. E., and Wasserman, L. (1996). Formal rules for selecting prior distributions: A review and annotated bibliography. *Journal of the American Statistical Association*, 435, 1343–1370.

Mengersen, K.L., ad R. L. Tweedie (1996). Rates of convergence of the Hastings and Metropolis algorithms. *Annals of Statistics*, 24, 101–121.