evolMC demo

Grady Weyenberg

November 8, 2013

evolMC is a framework for doing Monte-Carlo simulations.

0.1 Univariate and multivariate sampling

We wish to use a metropolis sampler to draw from a distribution with density

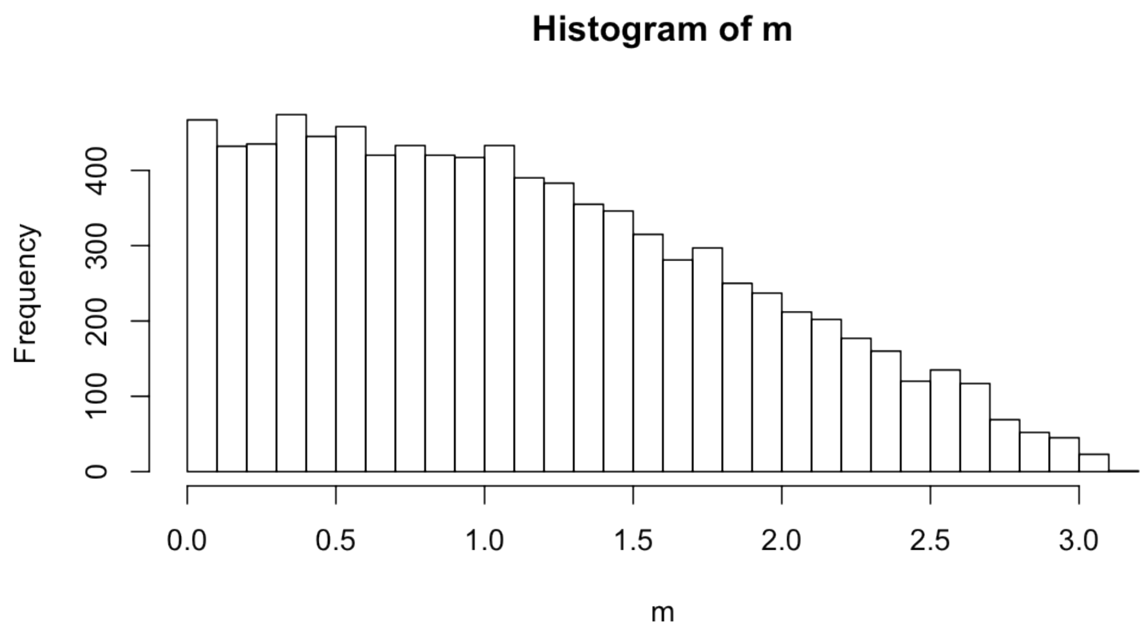
$$f(x) \propto \frac{\sin(x)}{x} \cdot 1_{(0,\pi)}(x).$$

We can use a uniform distribution on $(-1, 1)$ to propose distances to jump from the current location.
Since the proposal distribution is symmetric, this is enough information to implement a Metropolis updater.

```
fn <- function(x) sin(x)/x * (0 < x) * (x < pi)
propose <- function(x) x + runif(length(x), -1, 1)
updater <- metropolis(fn, propose)
```

A Markov chain is formed by iteratively calling the updating function starting with some initial value.

```
chain <- iterate(n = 10000, fn = updater, init = 1)
summary(chain)

## Discarding first 1000 states.
##    mean      se    2.5%   97.5%
## 1.11712 0.73918 0.04531 2.66390

hist(chain, breaks = "fd")

## Discarding first 1000 states.
```

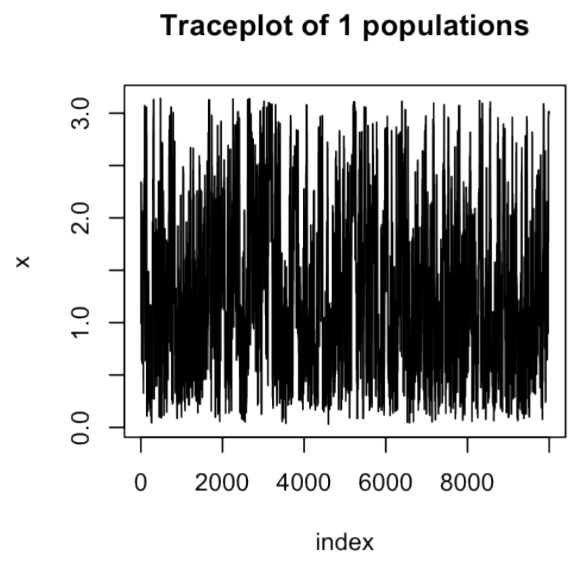


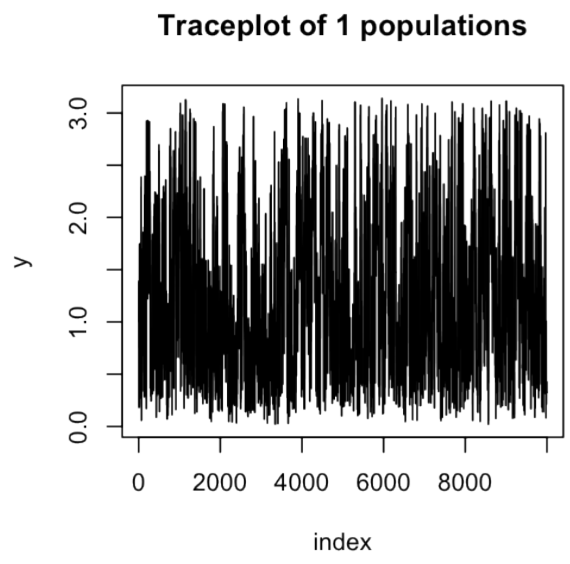
Of course, multivariate distributions may also be sampled.

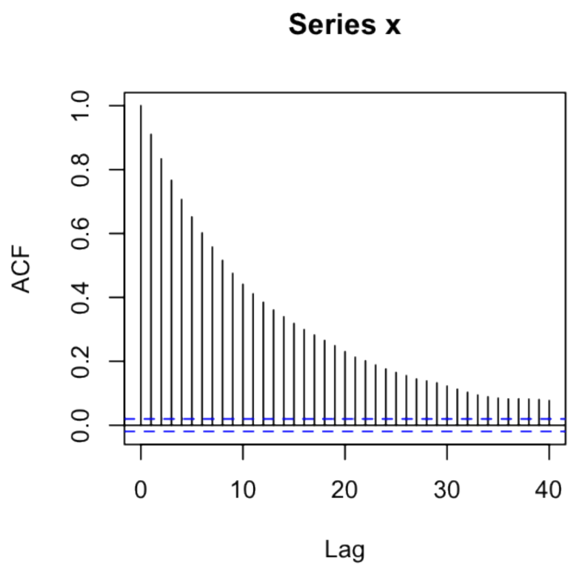
```
mvtarget <- function(x) prod(sin(prod(x))/x) * all(x > 0, x < pi, prod(x) <
    pi)
mvupdate <- metropolis(mvtarget, propose)
chain2 <- iterate(10000, mvupdate, cbind(x = 1, y = 1))
summary(chain2)
```

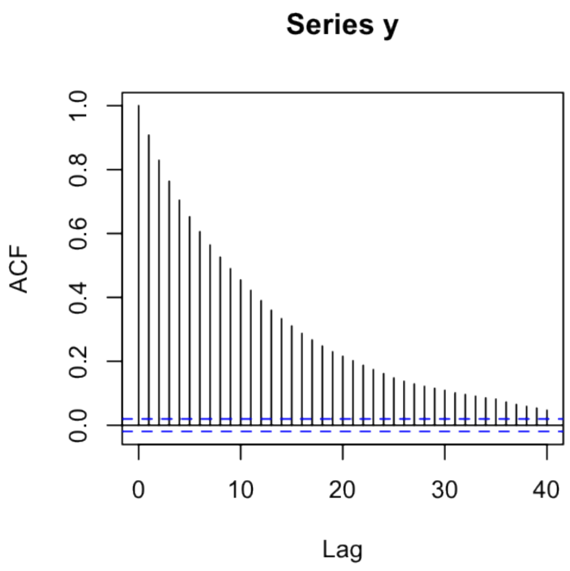
```
## Discarding first 1000 states.
##             x      y
## mean  1.2974 1.2333
## se    0.8146 0.8110
## 2.5%  0.1614 0.1618
## 97.5% 3.0129 2.9627
```

```
plot(chain2)
```

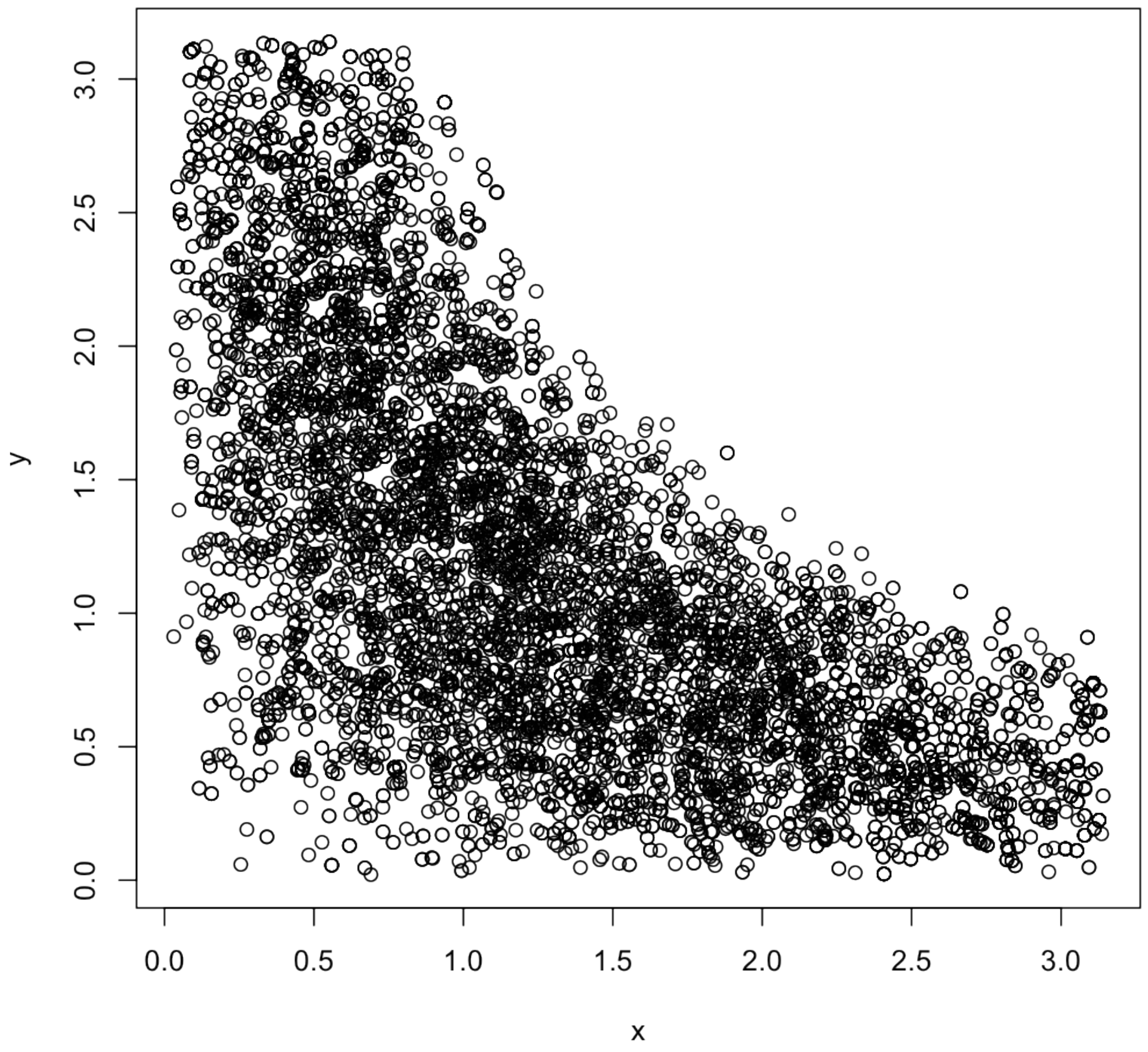








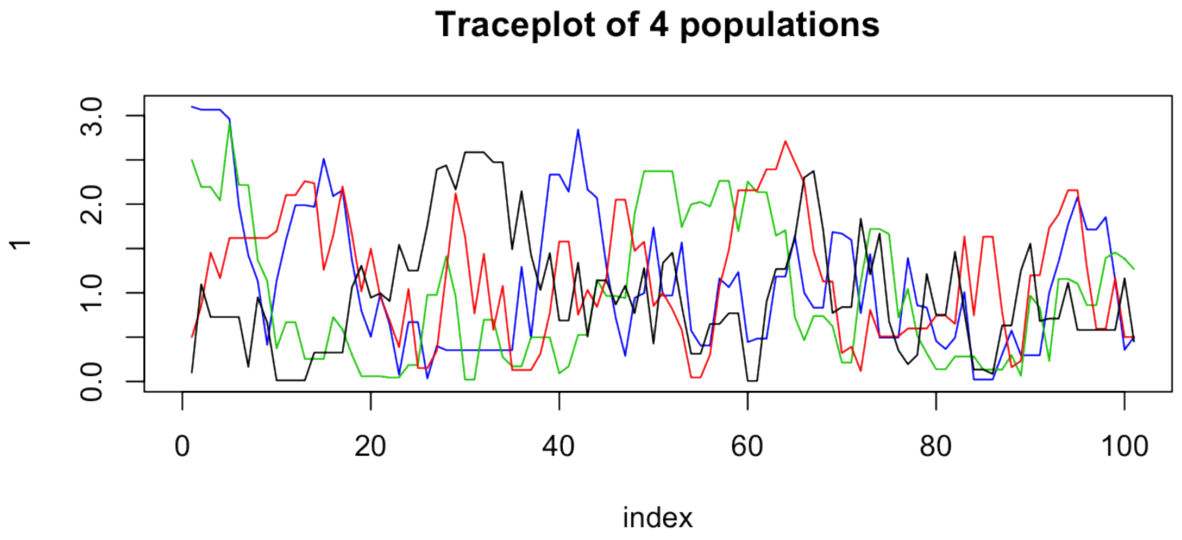
```r
plot(t(simplify2array(chain2)[1, , ]))
```

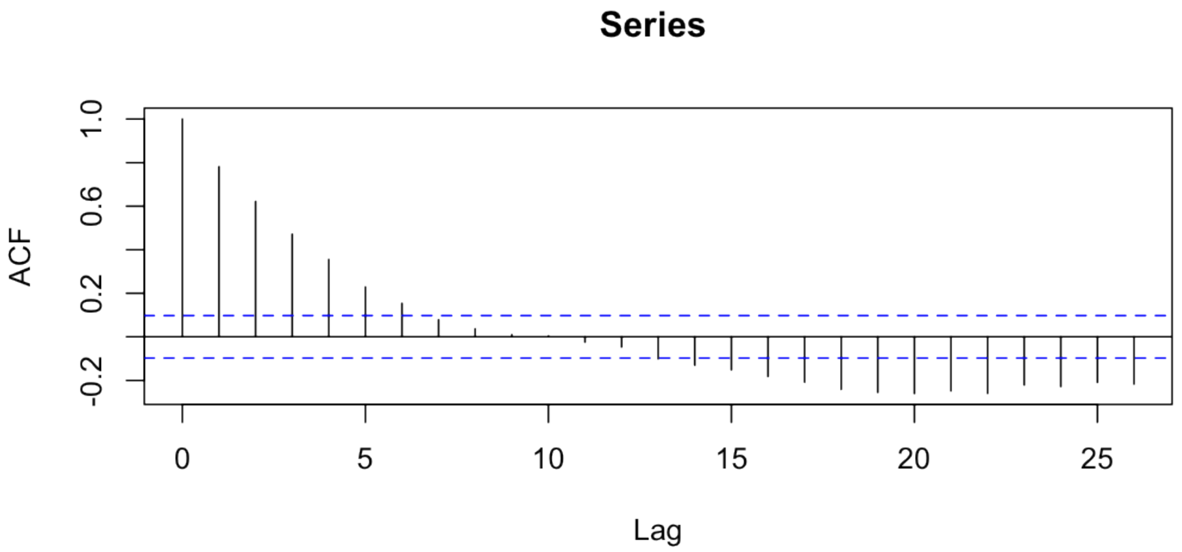


0.2 Multiple chains in parallel

If the target density returns a vector instead of a scalar, we can run multiple chains in parallel. (In this case the state object is a matrix with individuals in rows.) In this situation, the proposal updater should update the entire population.

```r
chain3 <- iterate(100, updater, init = rbind(0.1, 0.5, 2.5, 3.1))
plot(chain3)
```

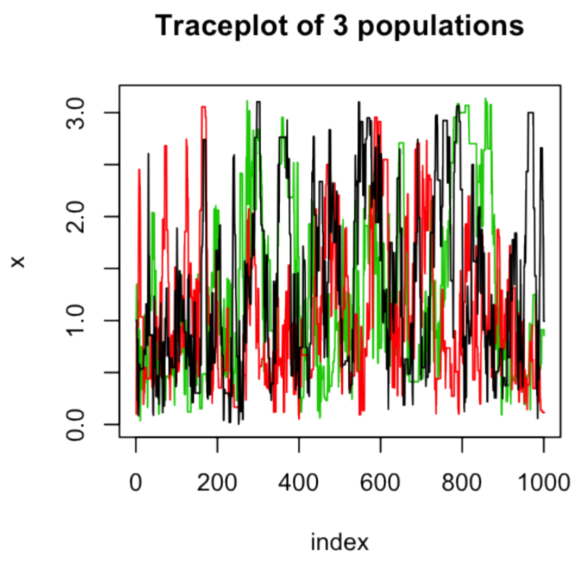


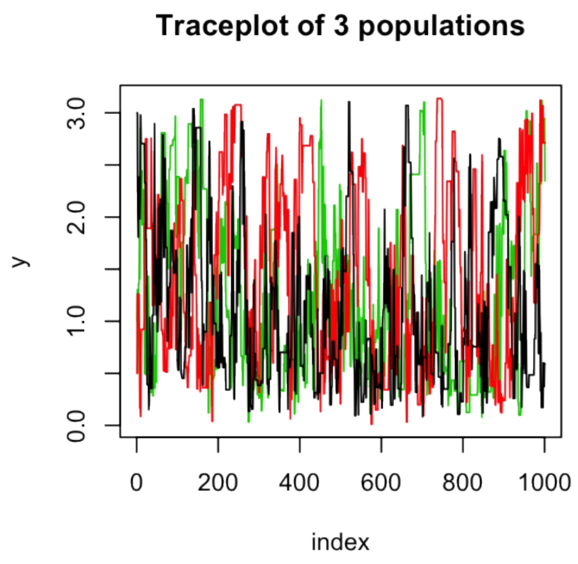


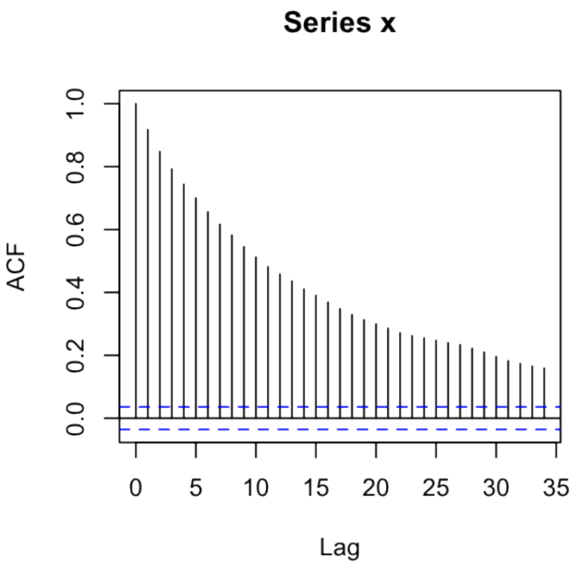
Multiple chains of multivariate samples are also possible, as long as a logical vector of the same length as that returned by the density function will "correctly" subset individuals from the population state object. [1]

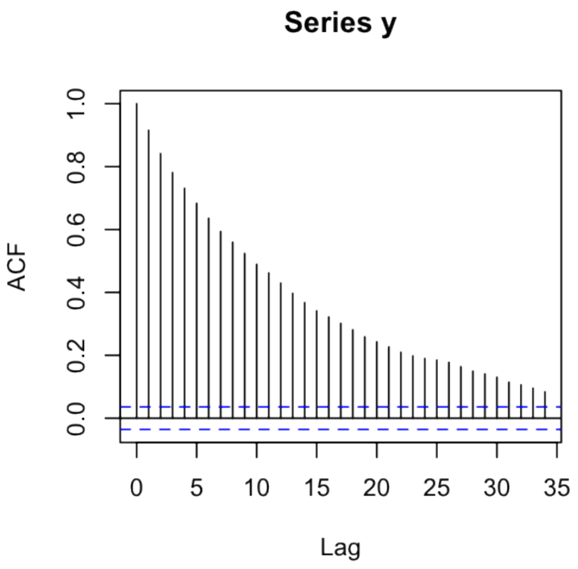
```
mvt2 <- function(x) apply(x, 1, mvtarget)
mvup2 <- metropolis(mvt2, propose)
init <- rbind(c(x = 1, y = 3), c(0.1, 0.5), c(1, 1))
chain4 <- iterate(1000, mvup2, init)
plot(chain4)
```

[1]Thus, a population can be a vector, a matrix with individuals in rows, or a list. Some convenience methods such as print, summary, plot, hist, etc., assume a population is a matrix, but this is not strictly required.









0.3 Gibbs sampling

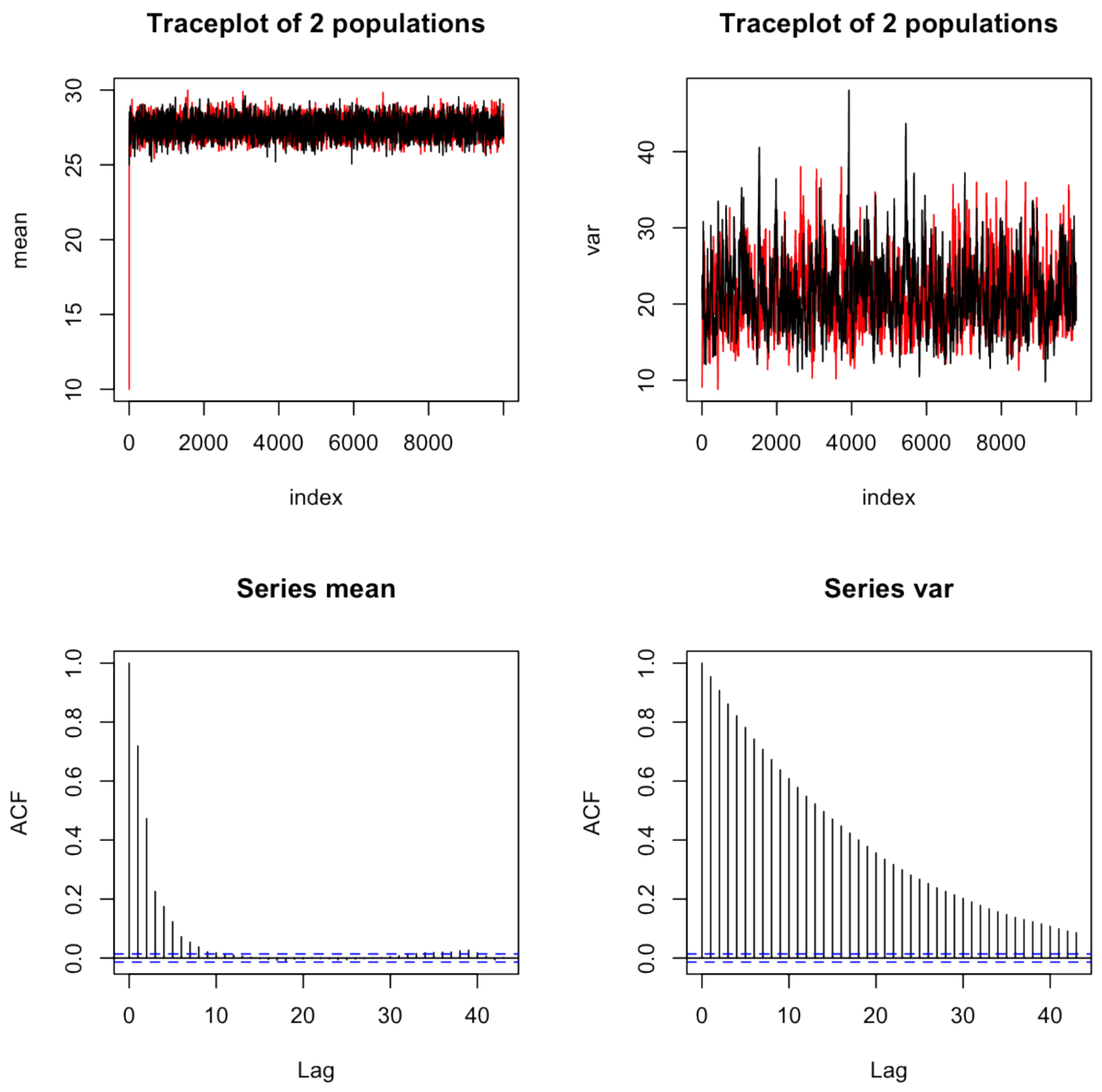
A gibbs updater calls several updating functions sequentially. Here we find Baysean location-scale parameter estimates for a t-distribution likelihood with known degrees of freedom 5/2. The model specification in this case is $y|\mu, w \sim N(\mu, w^{-1})$ and $w|\sigma^2, \nu \sim G(\nu/2, \sigma^2\nu/2)$, with priors $\mu \sim N$ and $\sigma^2 \sim G$.

```
newcomb <- c(28, -44, 29, 30, 26, 27, 22, 23, 33, 16, 24, 29, 24, 40, 21, 31,
    34, -2, 25, 19, 24, 28, 37, 32, 20, 25, 25, 36, 36, 21, 28, 26, 32, 28,
    26, 30, 36, 29, 30, 22, 36, 27, 26, 28, 29, 23, 31, 32, 24, 27, 27, 27,
    32, 25, 28, 27, 26, 24, 32, 29, 28, 33, 39, 25, 16, 23)
##' Full conditional updater for mu
```

```r
f.m <- function(state, ...) {
    k <- seq_len(ncol(state) - length(newcomb))
    w <- rowSums(state[, -k])
    mu <- (state[, -k] %*% newcomb)/(1e-04 + w)
    sd <- 1/sqrt(1e-04 + w)
    n <- nrow(state)
    state[, 1] <- rnorm(n, mu, sd)
    state
}
##' Full conditional updater for sigma^2
f.v <- function(state, df = FALSE) {
    nu <- if (df)
        1/state[, 3] else 5
    k <- seq_len(ncol(state) - length(newcomb))
    n <- ncol(state[, -k])
    rates <- 0.1 + rowSums(state[, -k]) * nu/2
    state[, 2] <- rgamma(nrow(state), 0.1 + n * nu/2, rates)
    state
}
##' Full conditional updater for w
f.w <- function(state, df = FALSE) {
    nu <- if (df)
        1/state[, 3] else 5
    k <- seq_len(ncol(state) - length(newcomb))
    rate <- (nu * state[, 2] + t(apply(state, 1, function(row) (newcomb - row[1])^2)))/2
    state[, -k] <- rgamma(length(state[, -k]), (nu + 1)/2, rate)
    state
}
```

The gibbs function is used to create a function which acts as the Gibbs updater.

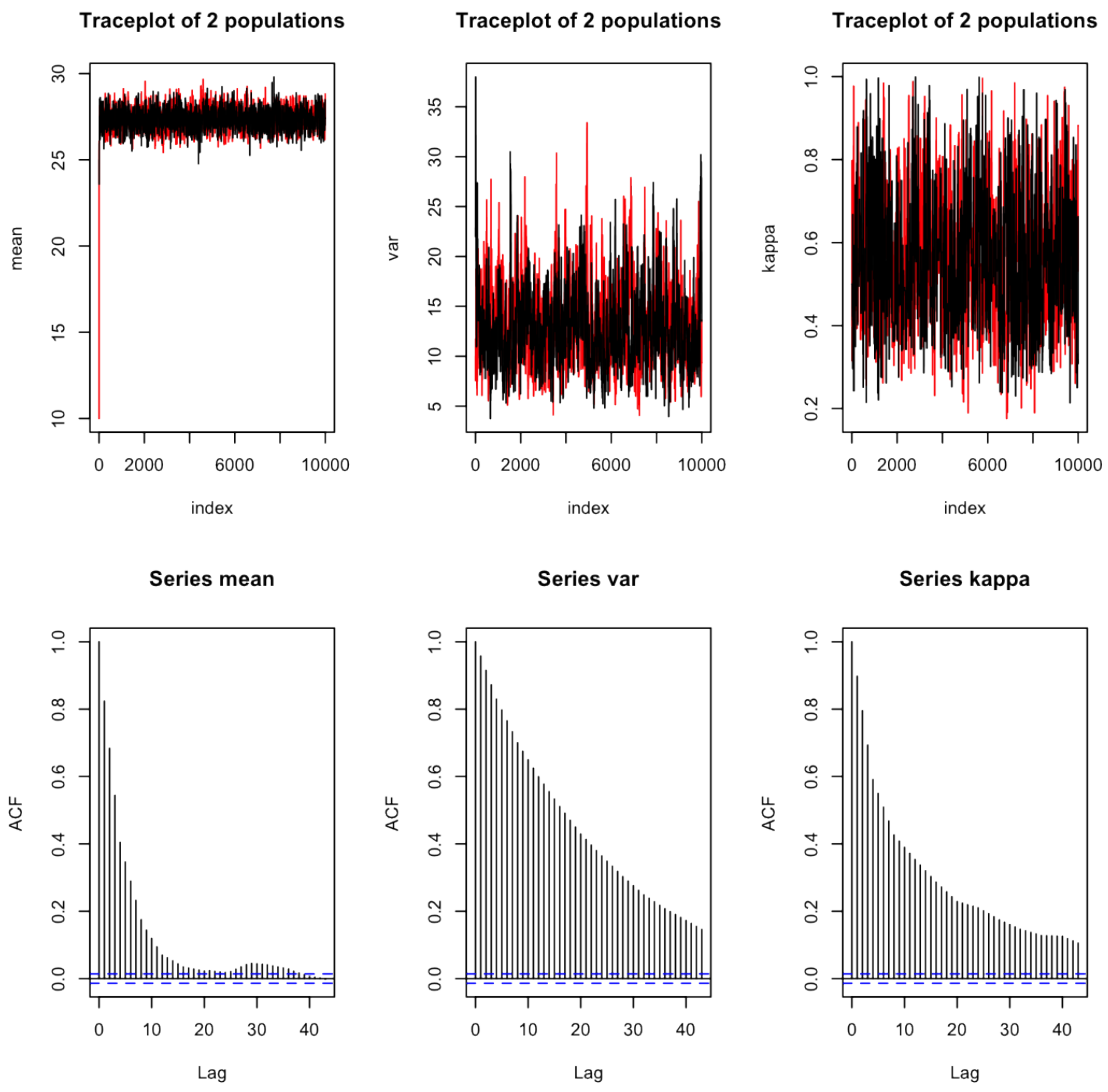
```r
init <- rbind(c(mean = 25, var = 22, rgamma(length(newcomb), 5/2, 22 * 5/2)),
    c(10, 10, rgamma(length(newcomb), 5/2, 10 * 5/2)))
newcomb.gibbs <- gibbs(f.m, f.v, f.w)
chain5 <- iterate(10000, newcomb.gibbs, init)
chain5 <- prune(chain5, TRUE, 1:2)
plot(chain5)
```



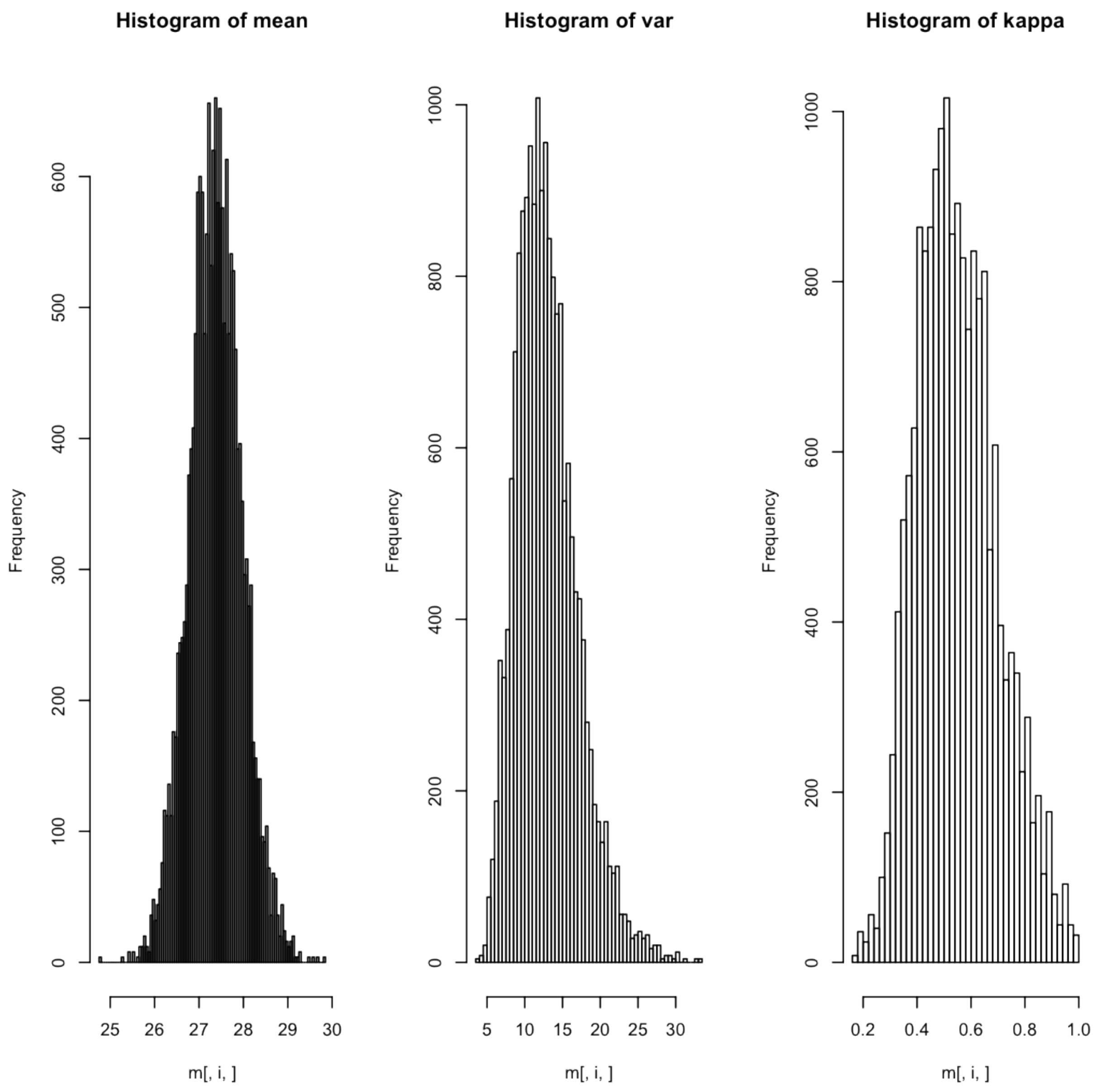
```
summary(chain5)

## Discarding first 1000 states.
##           mean    var
## mean  27.5091 21.219
## se     0.6419  4.511
## 2.5%  26.2784 13.794
## 97.5% 28.7828 31.273
```

Of course, you can use a metropolis updater in place of a full conditional. Here we put a prior on $\nu^{-1} \sim E(1)T(0,1)$

```r
dtexp <- function(x, rate = 1, min = 0, max = Inf, log = FALSE) {
    d <- dexp(x, rate) * (x >= min) * (x <= max)
    c <- pexp(max, rate) - pexp(min, rate)
    if (log)
        log(d/c) else d/c
}
posterior <- function(state) {
    if (!is.matrix(state))
        state <- matrix(state, 1)
    f <- function(x) sum(dt((newcomb - x[1])/sqrt(x[2]), 1/x[3], log = TRUE) -
        log(x[2])/2)
    exp(apply(state, 1, f) + dnorm(state[, 1], sd = 100, log = TRUE) + dgamma(state[,
        2], 0.1, 0.1, log = TRUE) + dtexp(state[, 3], max = 1, log = TRUE))
}
kappa.prop <- function(state) {
    state[, 3] <- runif(nrow(state))
    state
}
init <- rbind(c(mean = 25, var = 22, kappa = 0.5, rgamma(length(newcomb), 5/2,
    22 * 5/2)), c(10, 10, 0.4, rgamma(length(newcomb), 5/2, 10 * 5/2)))
newcomb.chain3 <- gibbs(f.m, f.v, f.w, metropolis(posterior, kappa.prop))
chain4 <- iterate(10000, newcomb.chain3, init, df = TRUE)
chain4 <- prune(chain4, TRUE, 1:3)
summary(chain4)

## Discarding first 1000 states.
##           mean    var  kappa
## mean  27.3737 12.950 0.5509
## se     0.5899  4.063 0.1495
## 2.5%  26.2344  6.557 0.3040
## 97.5% 28.5677 22.392 0.8824

plot(chain4)
```



```
hist(chain4, breaks = "fd")

## Discarding first 1000 states.
```



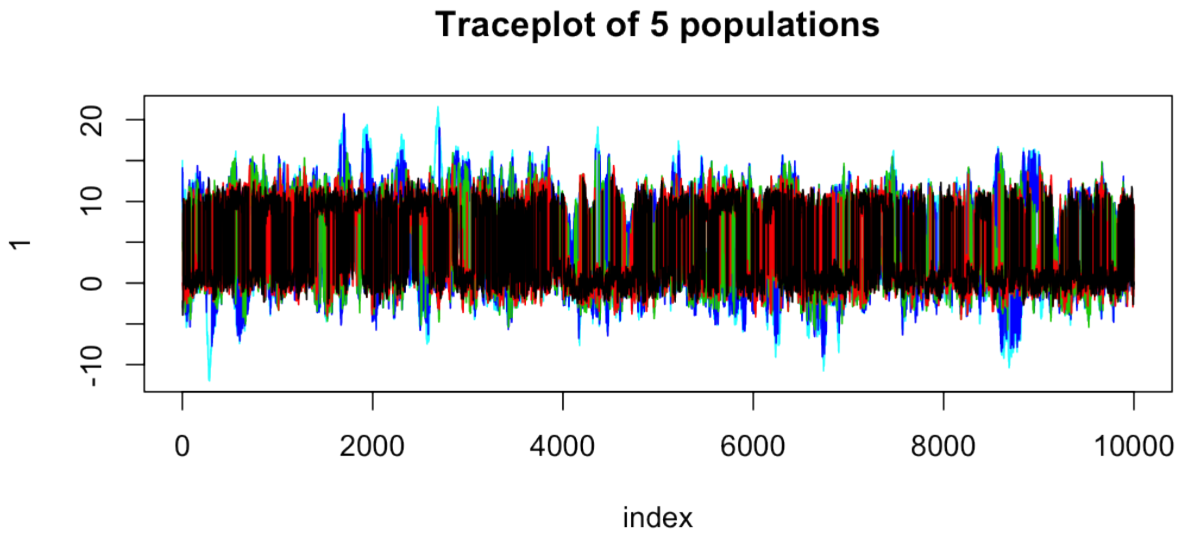
0.4 Parallel tempering

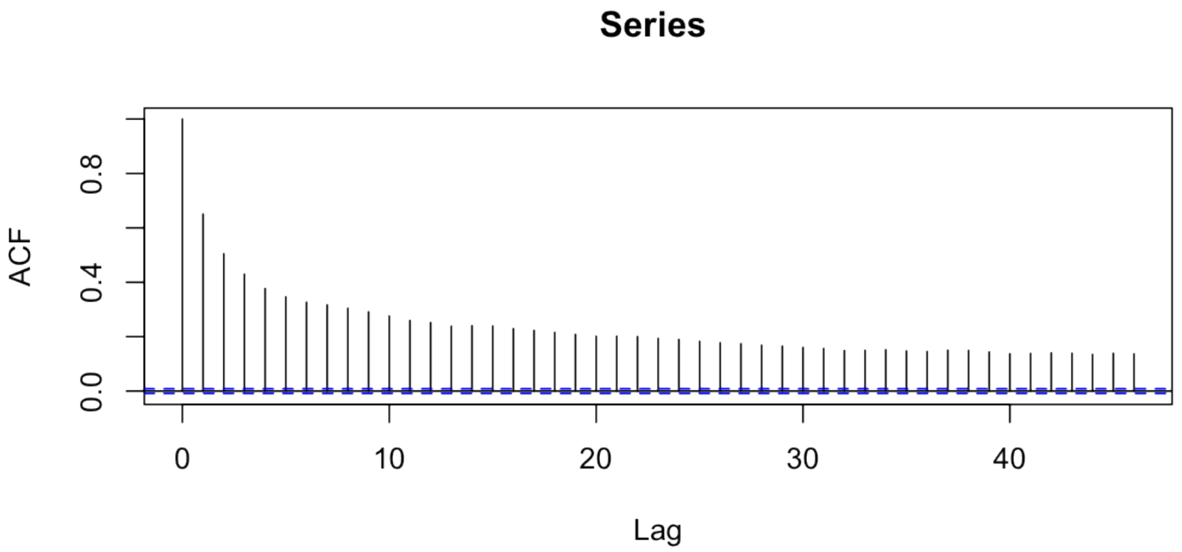
We want to sample from a distribution with separated modes. First we set up the target distribution function. This function should take a single individual and return the density of that individual.

```
f <- function(state) exp(sum(log(dnorm(state) + dnorm(state, 10))))
```

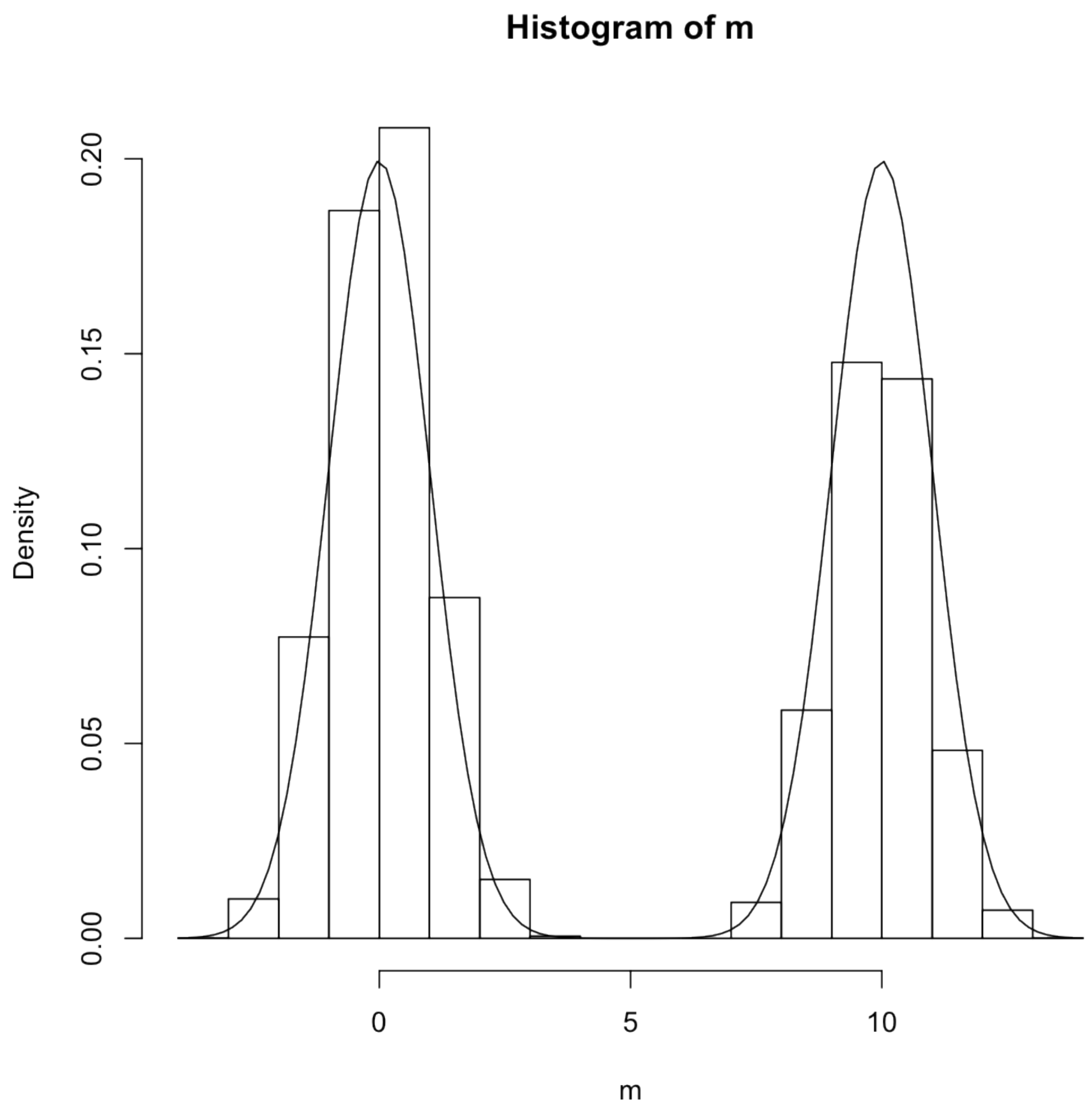
Now we can set up the parallel tempering updater. Function temper is similar to metropolis, but you must also provide a list of temperatures. (It is assumed that the number of individuals in the population will be the same as the number of temperature levels.)

```
temps <- c(1, 2, 4, 8, 15)
updater <- temper(f, temps, propose)
init <- rbind(-3, 0, 5, 10, 15)
gch <- iterate(10000, updater, init)
plot(gch)
```



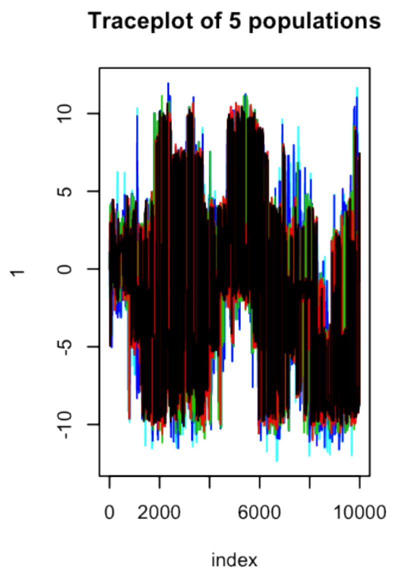


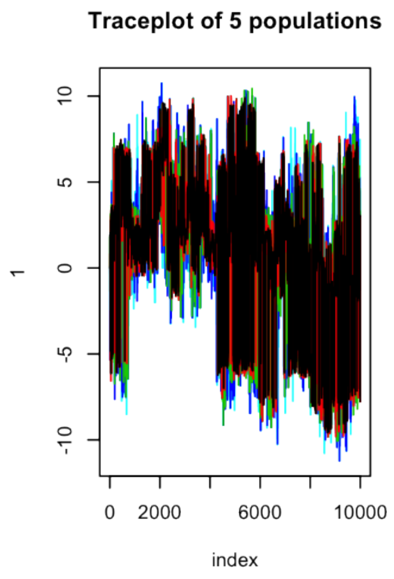
```
xx <- prune(gch, 1, 1)  #extract chain with correct distribution
hist(xx, freq = FALSE, breaks = "fd")

## Discarding first 1000 states.

curve((dnorm(x) + dnorm(x, 10))/2, add = TRUE)
```

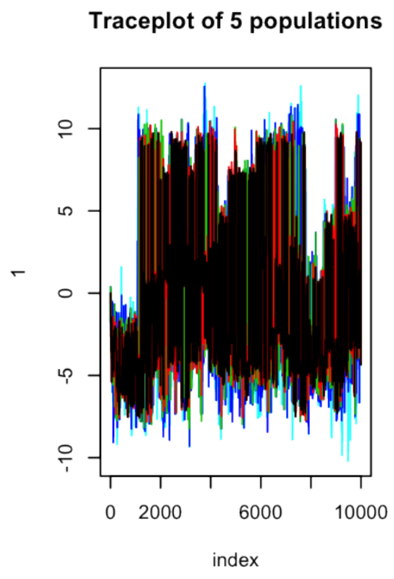


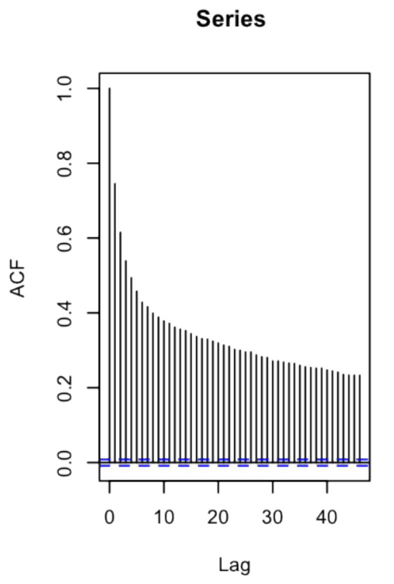
Here we go with the real-deal.

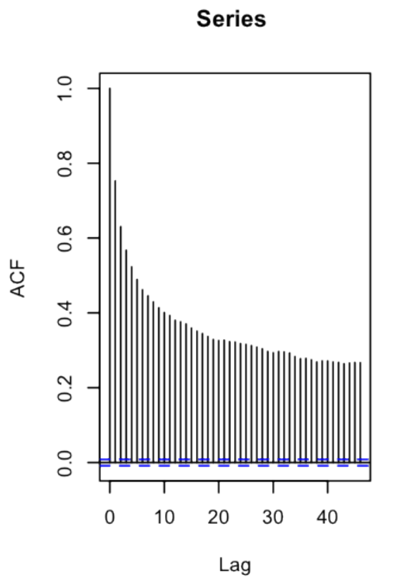
```r
init <- matrix(0, 5, 3)
means <- t(replicate(20, runif(3, -10, 10)))
ff <- function(x) target(x, means)
e <- seq(0.2, 3, length = 5)
pf <- function(x) x + runif(length(x), -e, e)
chainz <- iterate(10000, gibbs(temper(ff, temps, pf), crossover(ff, temps)),
    init, bar = TRUE)

## ===========================================================================

plot(chainz)
```

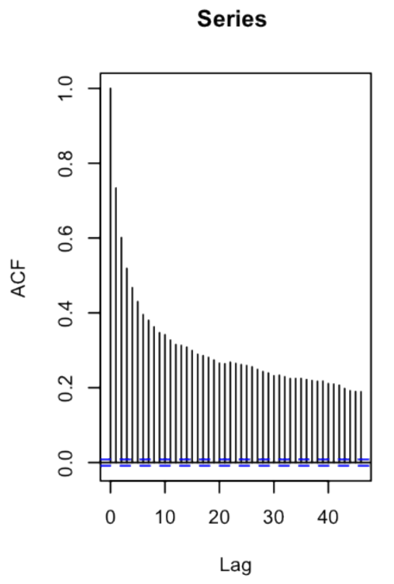










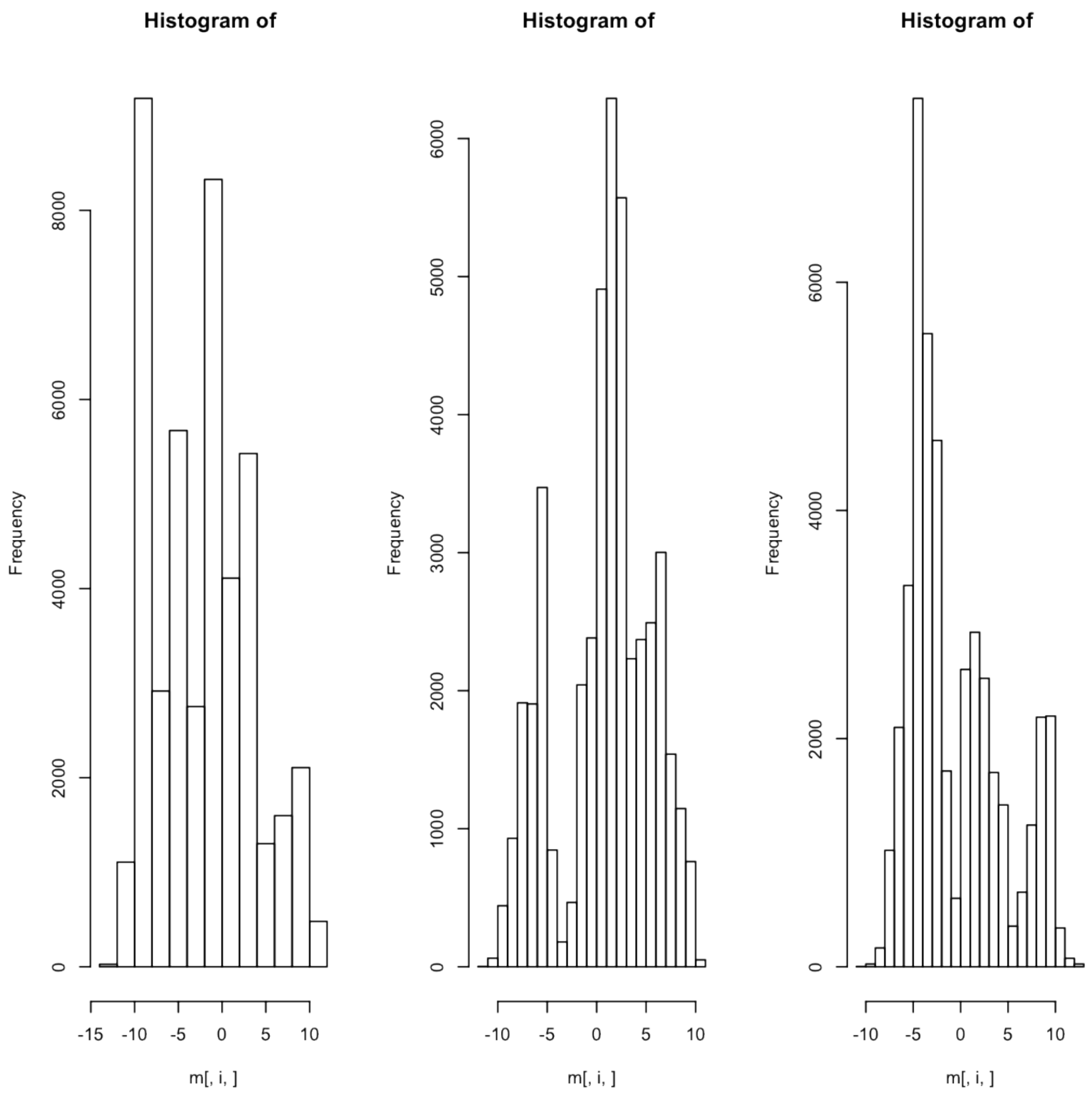


```
means
```

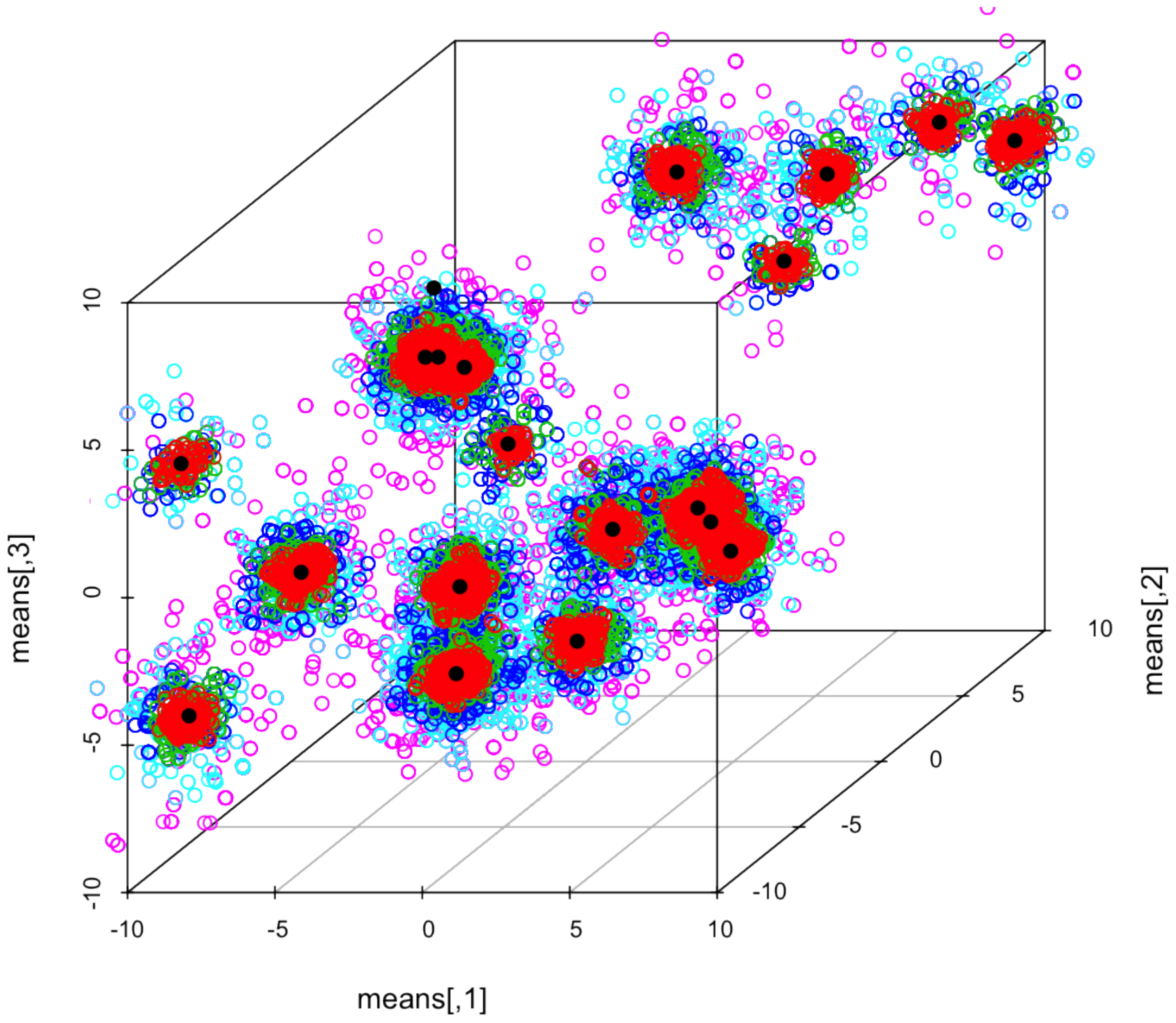
```
##           [,1]    [,2]    [,3]
##  [1,] -0.6754 -8.0863  9.6540
##  [2,] -4.4817  0.3633 -4.2299
##  [3,] -7.7012  4.8362  1.5675
##  [4,] -9.0343 -1.1390 -3.0753
##  [5,]  9.5861  8.9156  7.0958
##  [6,] -9.2719  6.8737  0.6592
##  [7,] -1.3293 -5.5340 -4.5626
##  [8,]  3.7028  0.9410 -2.2912
##  [9,]  1.2551  6.5646 -5.7794
## [10,]  1.3375 -7.1755  3.9643
## [11,]  6.9805  2.1556  8.9668
## [12,] -5.0253  1.6122  2.6546
## [13,]  2.4300  2.4501 -2.4892
## [14,]  8.7057  5.8989  9.0599
## [15,] -0.9179  3.2738 -3.5773
## [16,]  2.0126  1.9202  9.1413
## [17,] -8.7031 -9.0644  4.1327
## [18,]  3.6590  5.4958  4.5327
## [19,] -9.3013 -7.4997 -5.1163
## [20,] -1.4603  2.0843 -6.8479
```

```
hist(chainz)
```

```
## Discarding first 1000 states.
```



```
m <- aperm(simplify2array(chainz))
library(scatterplot3d)
pp <- scatterplot3d(means)
sapply(5:1, function(i) pp$points3d(m[, , i], col = i + 1))

## [[1]]
## NULL
##
## [[2]]
## NULL
##
## [[3]]
## NULL
##
## [[4]]
## NULL
##
## [[5]]
## NULL

pp$points3d(means, col = 1, pch = 19)
```



```
## library(rgl) plot3d(m[,,1]) plot3d(means,pch=19,size=15)
## sapply(1:5,function(i) points3d(m[,,i],col=i+1))
```