hw2

May 14, 2019

- Answer to Exercise 3.1, 3.2, part of 4.1 and 4.4 are handwritten and attached in the end.
- The code used are included in the end of the notebook.

1 Exercise 4.2

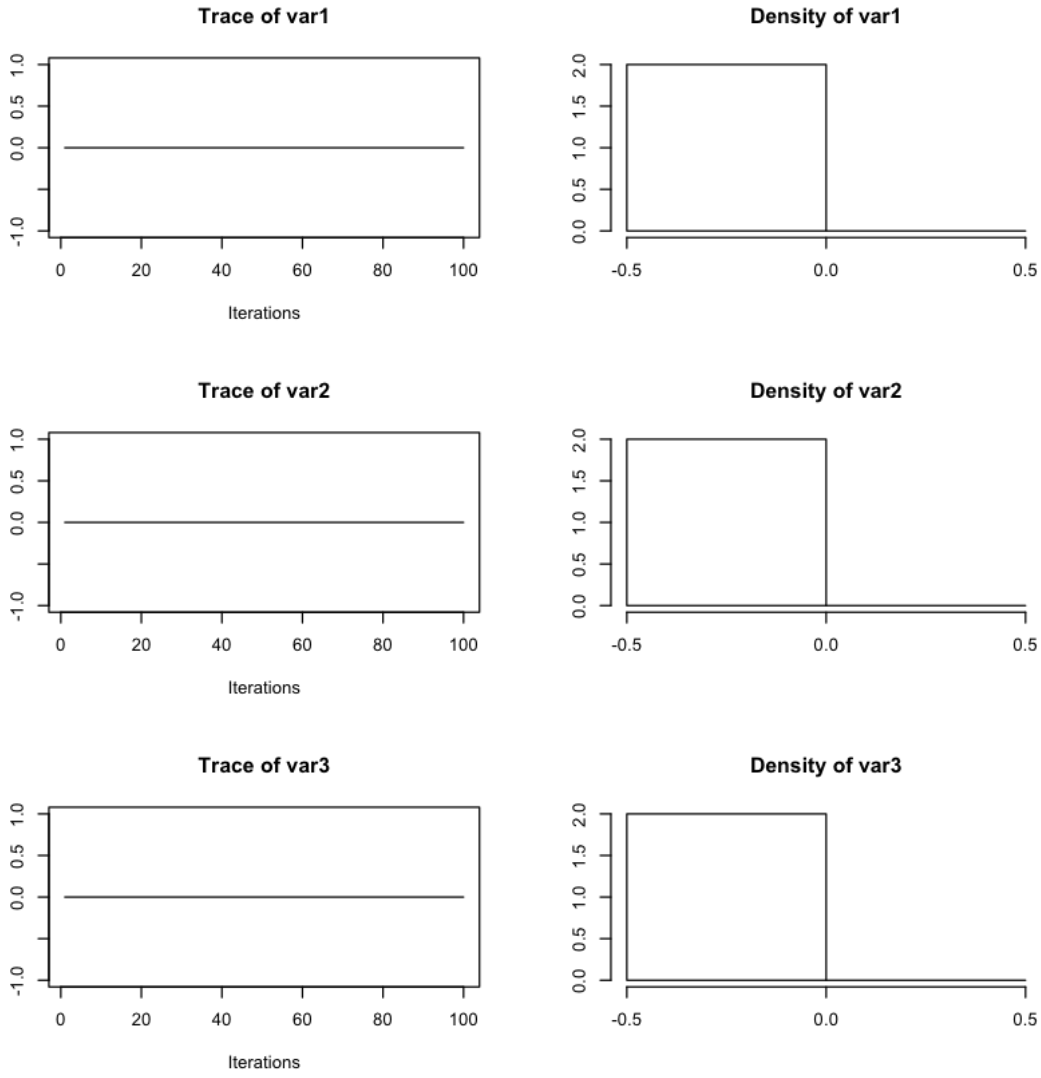
```
In [27]: rm(list = ls())
         source("mcmc_games.R")
```

1.1 4.2.1

Answer: it uses independence sampler. Not a good choice because the acceptance rate is too low.

1.2 4.2.2

```
In [44]: mc_out = mcmc_is(iterations = 100, burn_in = 0, sd = 10)
         mc_vis(mc_out)
```

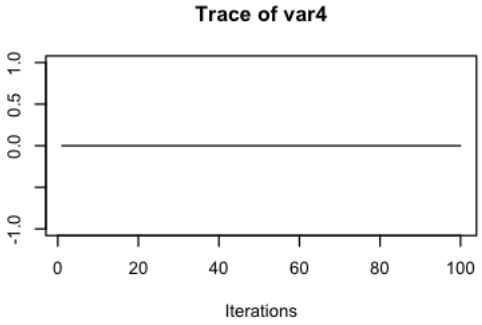


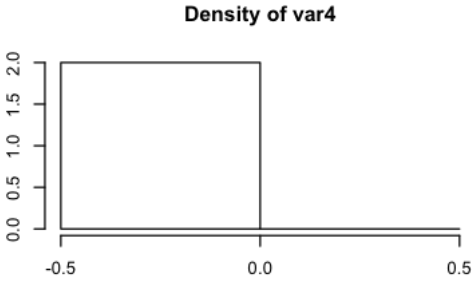

```
Iterations = 1:100
Thinning interval = 1
Number of chains = 1
Sample size per chain = 100

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

     Mean SD Naive SE Time-series SE
[1,]    0  0        0              0
[2,]    0  0        0              0
[3,]    0  0        0              0
```

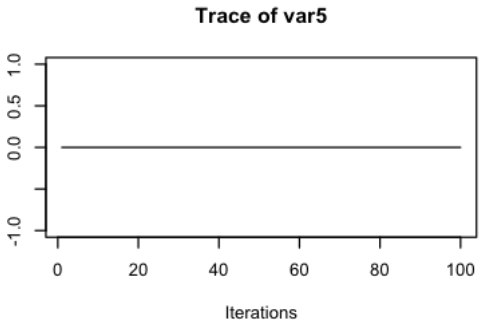
```
[4,]    0  0        0              0
[5,]    0  0        0              0
[6,]    0  0        0              0
```

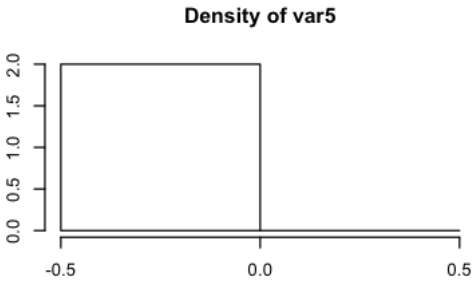
2. Quantiles for each variable:

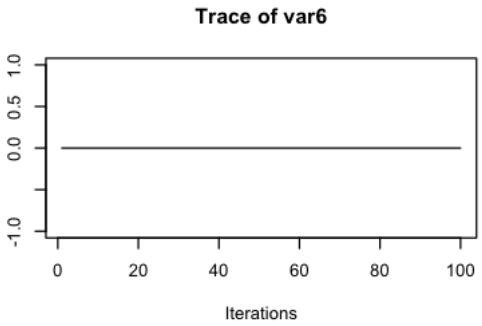
```
     2.5% 25% 50% 75% 97.5%
var1    0   0   0   0     0
var2    0   0   0   0     0
var3    0   0   0   0     0
var4    0   0   0   0     0
var5    0   0   0   0     0
var6    0   0   0   0     0
```

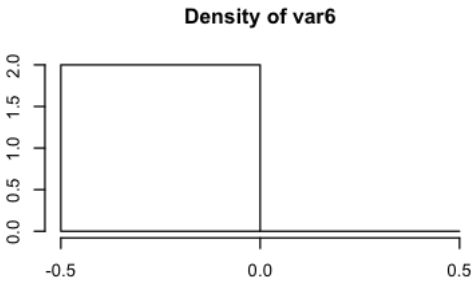








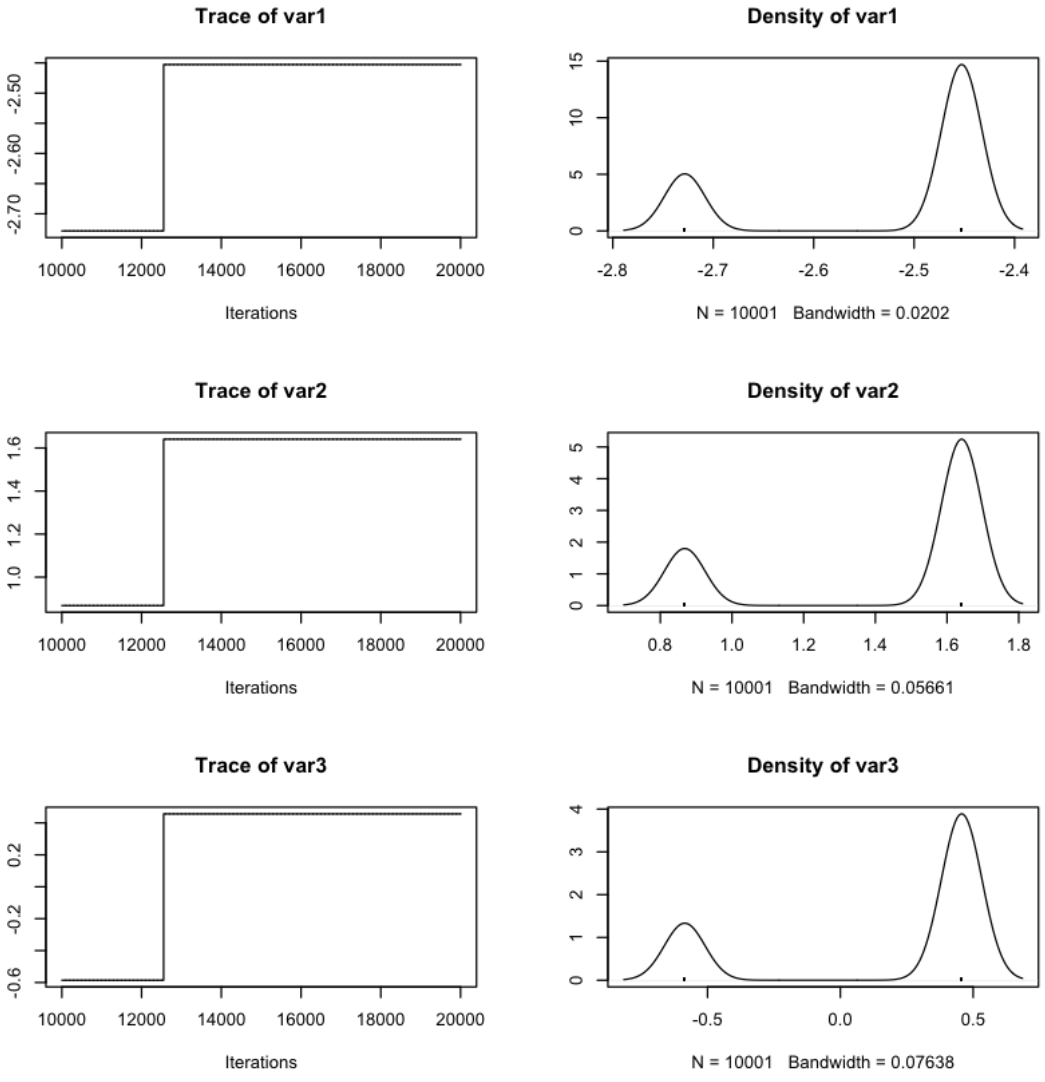




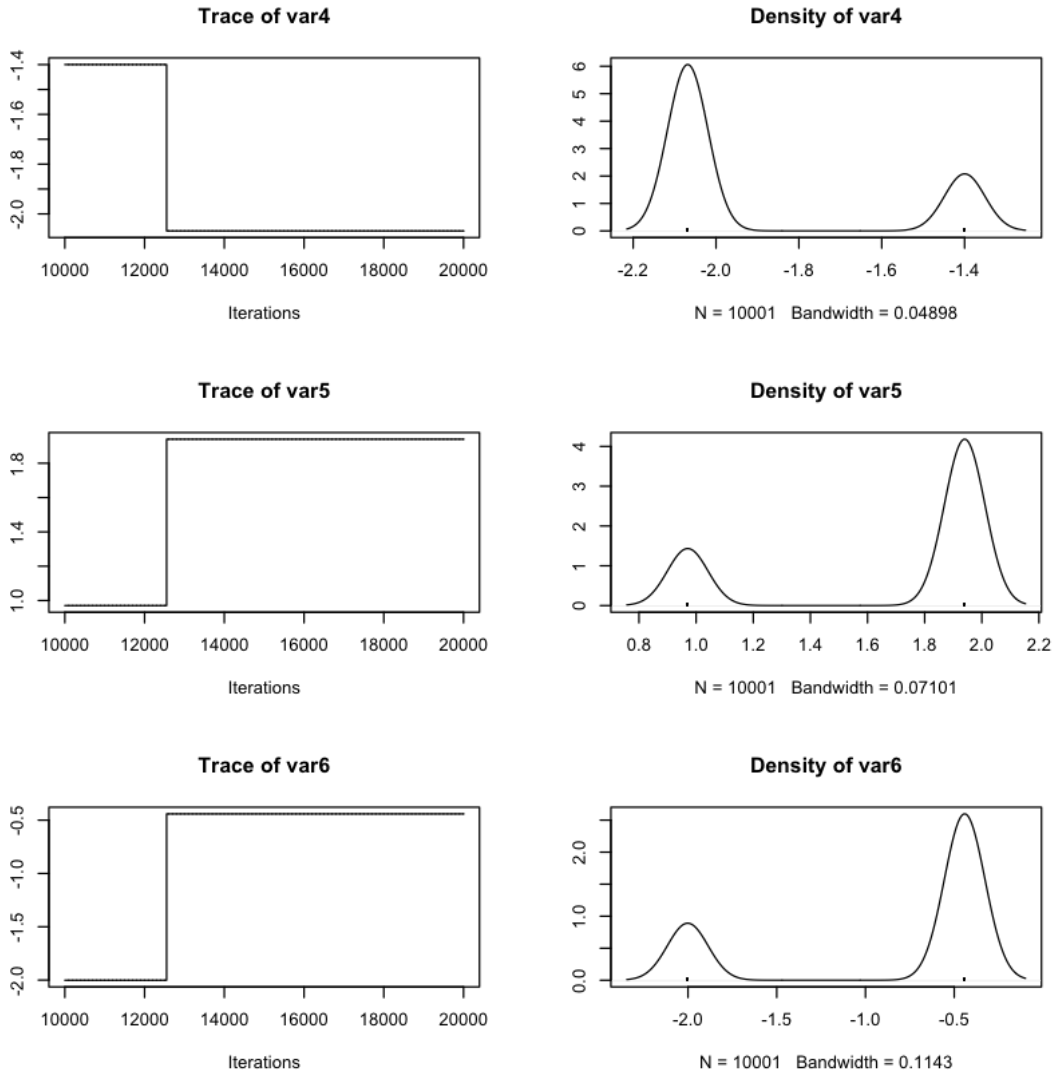
Answer: it has not reached equilibrium in 100 iterations. In fact, all smaples are the same as the initiali state (all zeros). Thus (almost) no proposals have been accepted in 100 iterations yet.

1.3 4.2.3

```
In [48]: burn_in = 1e+04
         iterations =  burn_in + 1e+04
         mc_out = mcmc_is(iterations = iterations, burn_in = burn_in, sd = 1)
         mc_vis(mc_out)
```



```
Iterations = 10000:20000
Thinning interval = 1
Number of chains = 1
Sample size per chain = 10001

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

        Mean     SD Naive SE Time-series SE
[1,] -2.5230 0.1202 0.001202        0.08238
[2,]  1.4436 0.3370 0.003370        0.23089
[3,]  0.1905 0.4546 0.004546        0.31149
[4,] -1.8977 0.2915 0.002915        0.19974
[5,]  1.6925 0.4227 0.004227        0.28961
[6,] -0.8395 0.6804 0.006804        0.46616

2. Quantiles for each variable:

        2.5%     25%     50%     75%   97.5%
var1 -2.7284 -2.7284 -2.4527 -2.4527 -2.4527
var2  0.8680  0.8680  1.6409  1.6409  1.6409
var3 -0.5860 -0.5860  0.4567  0.4567  0.4567
var4 -2.0684 -2.0684 -2.0684 -1.3998 -1.3998
var5  0.9706  0.9706  1.9400  1.9400  1.9400
var6 -2.0016 -2.0016 -0.4412 -0.4412 -0.4412
```

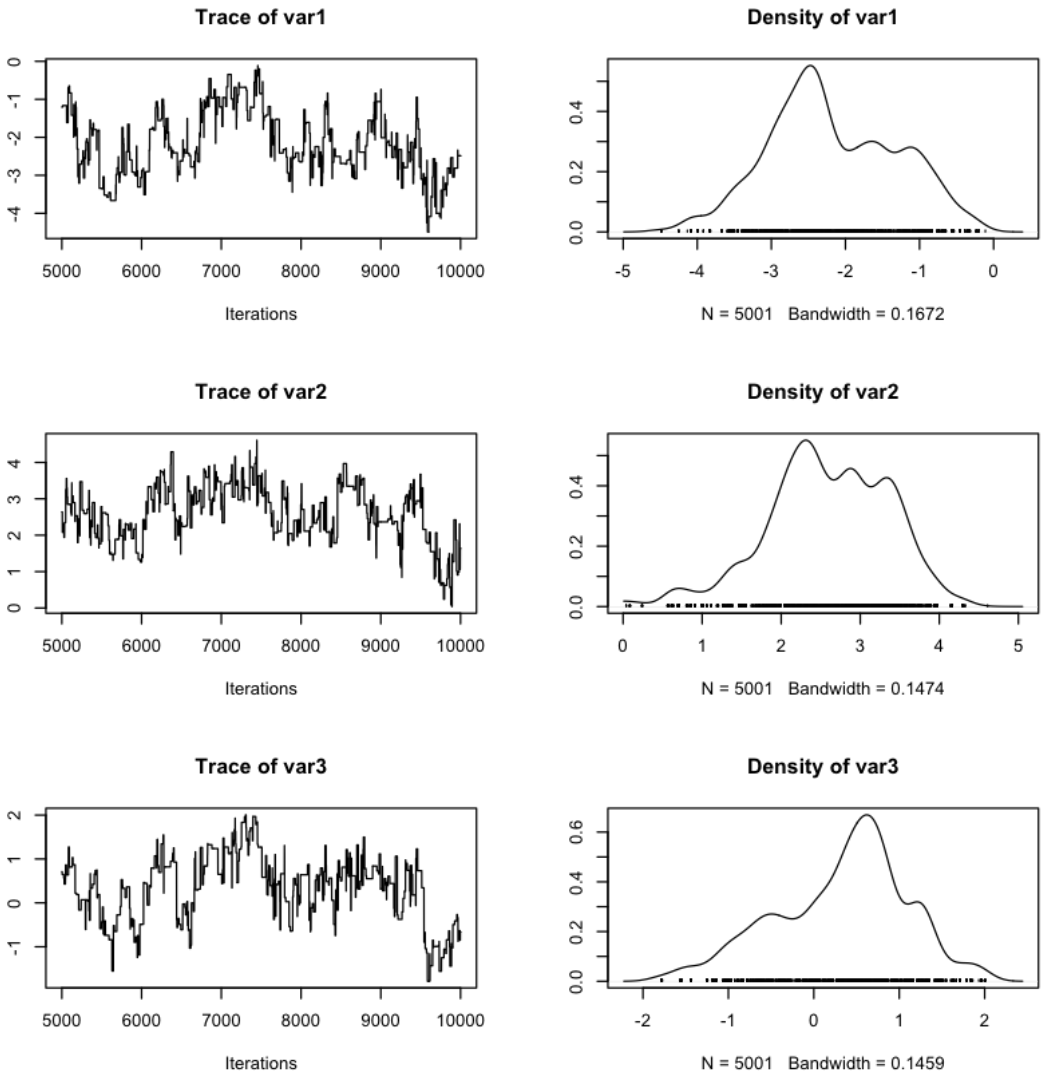


Answer: I changed the standard deviation of proposal to be 2, burn-in to be 10000, niterations to be 2000. the chain has not converged: each variables "jump" between two modes. The 95% percentile can be read off from the summary (from 2.5% to 97.5% percentile):

```
      2.5%        97.5%
var1 -2.7284  -2.4527
var2  0.8680   1.6409
var3 -0.5860   0.4567
var4 -2.0684  -1.3998
var5  0.9706    1.9400
var6 -2.0016  -0.4412
```

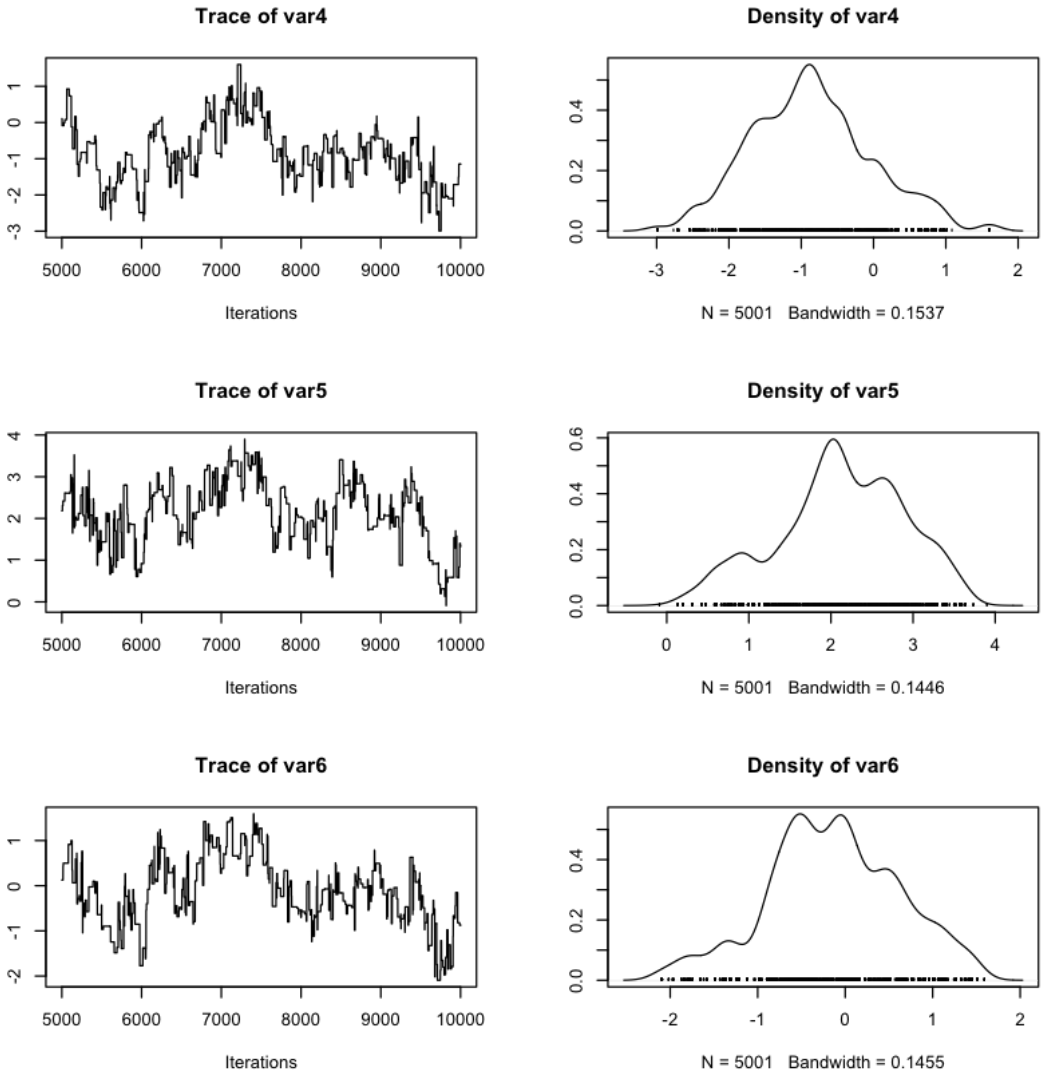
1.4 4.3.1

```
In [49]: iterations = 1e+04
         burn_in = iterations/2
         mc_out0 = mcmc_rw(iterations = iterations, burn_in = burn_in)
         mc_vis(mc_out0)
```




```
Iterations = 5000:10000
Thinning interval = 1
Number of chains = 1
Sample size per chain = 5001

1. Empirical mean and standard deviation for each variable,
```

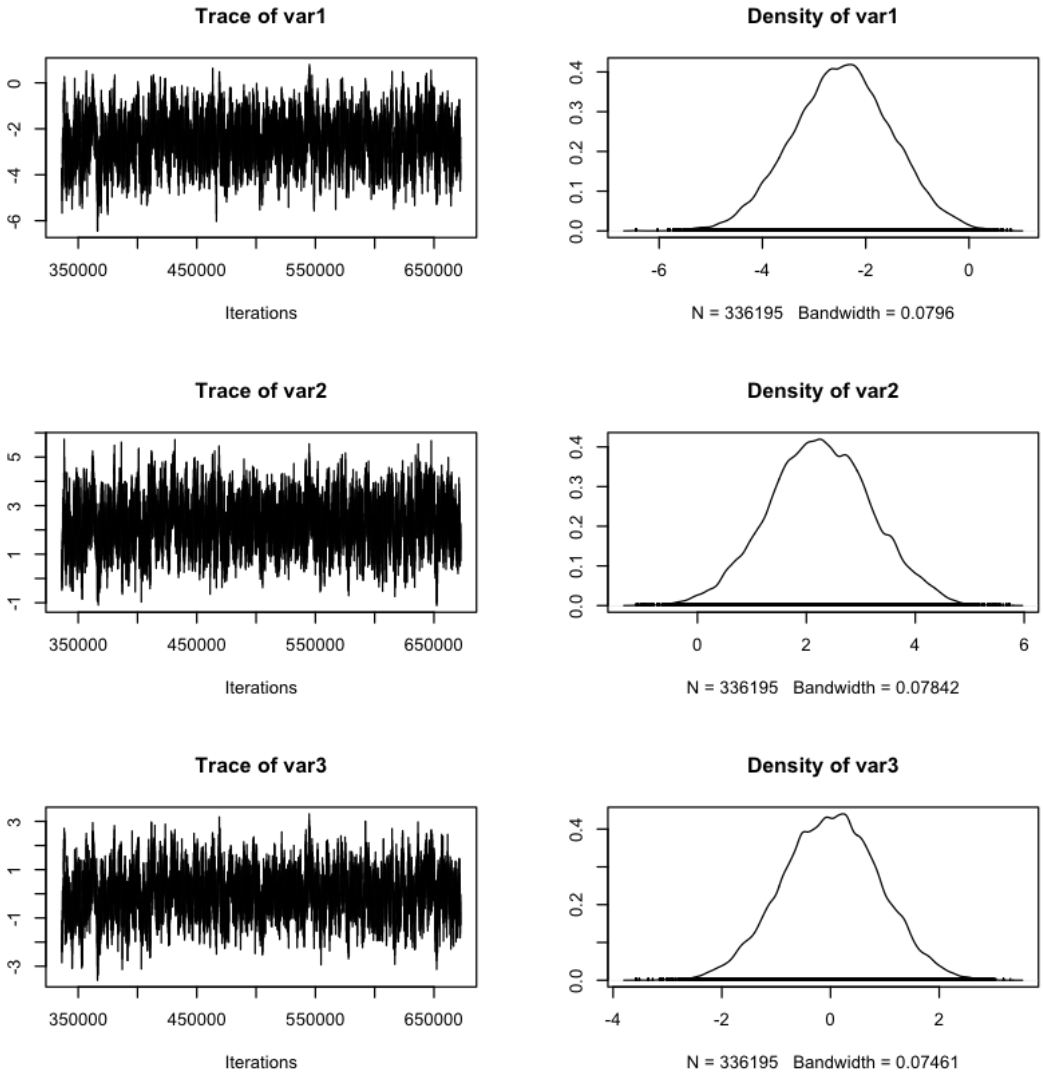
```
   plus standard error of the mean:

        Mean     SD Naive SE Time-series SE
[1,] -2.1735 0.8665  0.01225         0.1881
[2,]  2.5901 0.7637  0.01080         0.1207
[3,]  0.3263 0.7653  0.01082         0.1893
[4,] -0.8651 0.8347  0.01180         0.1628
[5,]  2.1261 0.7744  0.01095         0.1444
[6,] -0.1356 0.7541  0.01066         0.1591

2. Quantiles for each variable:

        2.5%     25%     50%     75%   97.5%
var1 -3.8234 -2.7896 -2.3372 -1.5271 -0.5460
var2  0.8094  2.1108  2.5966  3.1966  3.8961
var3 -1.4340 -0.1942  0.4413  0.8192  1.7070
var4 -2.4861 -1.4550 -0.8862 -0.3877  0.8658
var5  0.5855  1.6827  2.1171  2.6866  3.4059
var6 -1.7720 -0.6255 -0.1381  0.4098  1.3695
```



Answer: It is supposed to be more efficient, as the acceptance rate should be close to 0.5 for small d, whereas in independence sampler, we see that acceptance rate is high (it stays the same for many iterations before it takes a big "jump")

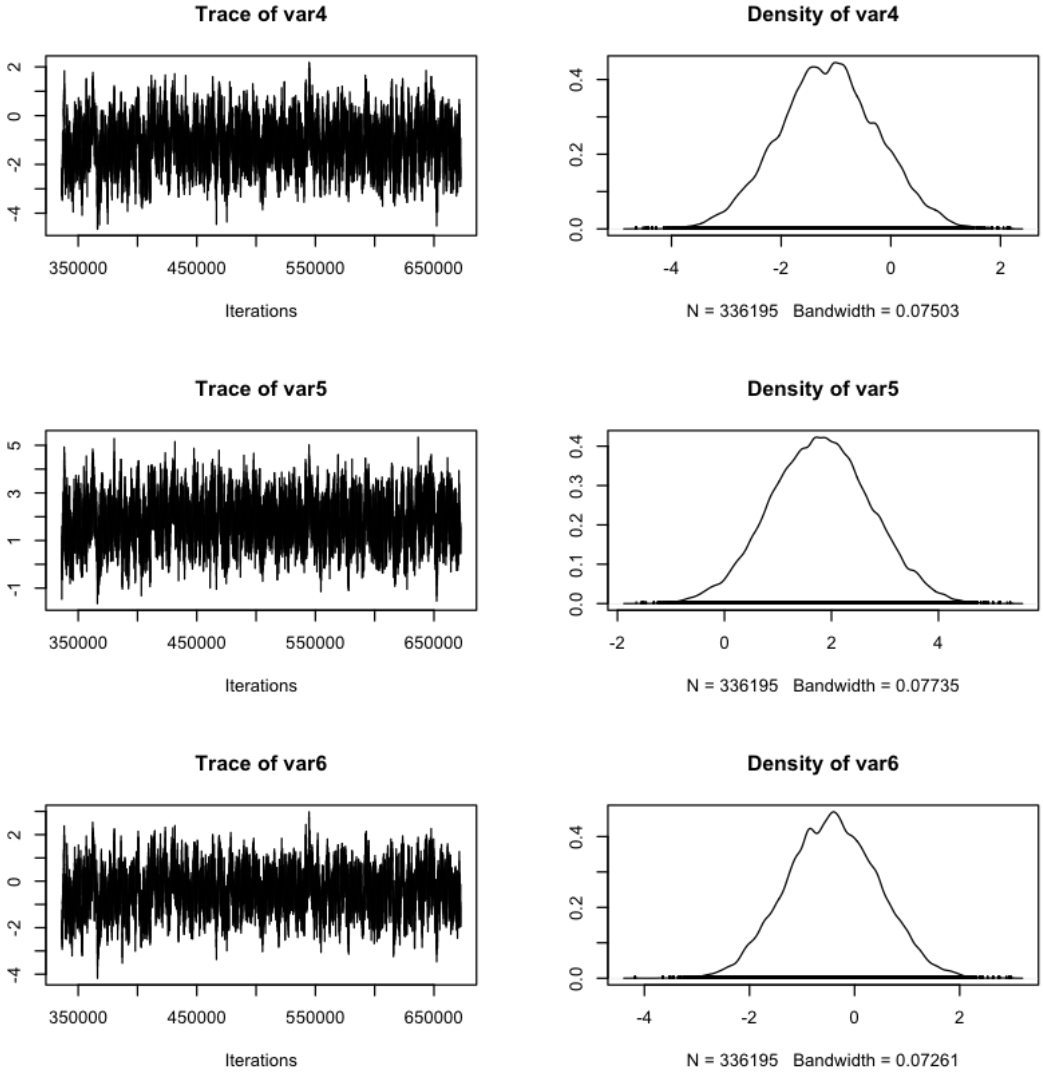
1.5 4.3.2

```
In [50]: raf = raftery.diag(mc_out0,q = 0.025)
         N = max(raf$resmatrix[,"N"]) ## use the largest N suggested by raftery
         #mc_out = mcmc_rw(iterations = N, burn_in = N/2)
         #saveRDS(mc_out, paste0("mc_p3_iter",N, ".mc"))
         mc_out = readRDS(paste0("mc_p3_iter",N, ".mc"))
         mc_vis(mc_out)
```




```
Iterations = 336193:672387
Thinning interval = 1
Number of chains = 1
Sample size per chain = 336195

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

          Mean     SD Naive SE Time-series SE
[1,] -2.452878 0.9570 0.001651        0.03547
[2,]  2.259840 0.9428 0.001626        0.03390
[3,]  0.006828 0.8970 0.001547        0.03607
```

```
[4,] -1.135750 0.9020 0.001556         0.03611
[5,]  1.800781 0.9300 0.001604         0.03503
[6,] -0.417216 0.8730 0.001506         0.03708

2. Quantiles for each variable:

         2.5%     25%      50%     75%   97.5%
var1 -4.34101 -3.0966 -2.44399 -1.8056 -0.5833
var2  0.43673  1.6224  2.25308  2.8995  4.1306
var3 -1.75182 -0.5974  0.01246  0.6204  1.7598
var4 -2.89419 -1.7387 -1.13519 -0.5297  0.6381
var5 -0.01126  1.1600  1.80146  2.4239  3.6327
var6 -2.09966 -1.0094 -0.42186  0.1810  1.2803
```



1.6 4.3.3

- I plot the sample path above (the Trace of var%d)
- We can read off the the 95% posterior from the summary above.

```
      2.5%        97.5%
var1 -4.34101  -0.5833
var2  0.43673    4.1306
var3 -1.75182   1.7598
var4 -2.89419   0.6381
var5 -0.01126   3.6327
var6 -2.09966   1.2803
```

1.7 4.3.4

```
In [59]: geweke.diag(mc_out, frac1 = 0.3, frac2 = 0.3)
```

```
Fraction in 1st window = 0.3
Fraction in 2nd window = 0.3

   var1    var2    var3    var4    var5    var6
-1.0698 -0.6877 -0.7329 -0.8712 -0.5352 -0.7589
```

Answer: The difference between the mean of the first 30% and that of the last 30% are quite big (especially var1).

1.8 4.3.5

```
In [83]: set.seed(12345)
         k = 6 ## number of variables
         n_exper = 10
         m = N ## niter
         starts = matrix(runif(k*n_exper, 0, 1), nrow = k)
         chains = c()
         for(i in 1:k){
             out <- mcmc_rw(iterations = m, burn_in = m/2, skill = starts[,i])
             chains[[i]] <- out
         }
         saveRDS(chains, "chains.mc")

In [84]: gelman.diag(mcmc.list(chains))
```

```
Potential scale reduction factors:

     Point est. Upper C.I.
[1,]          1          1
[2,]          1          1
[3,]          1          1
[4,]          1          1
[5,]          1          1
[6,]          1          1

Multivariate psrf

1
```

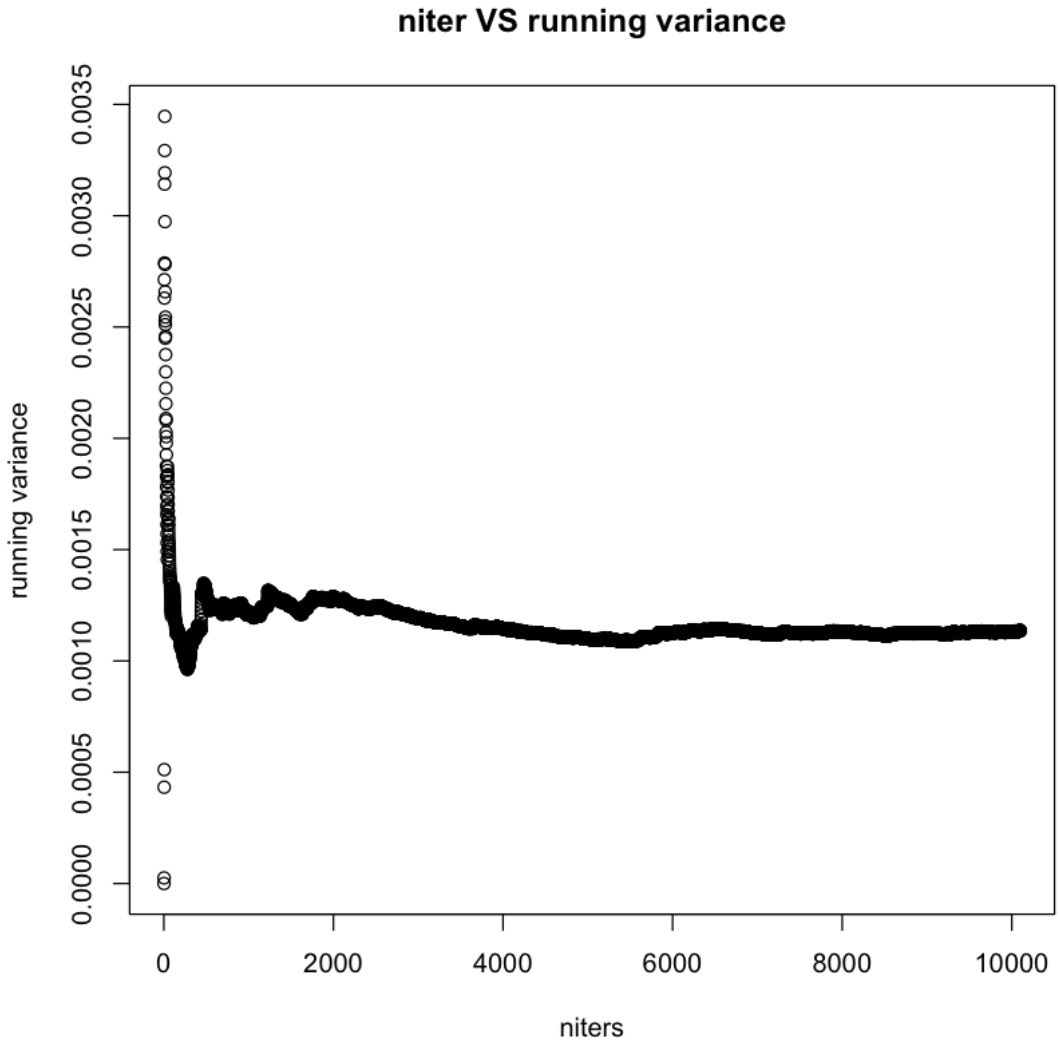
Answer: from Gelman diagnosis, the convergence is quite good.

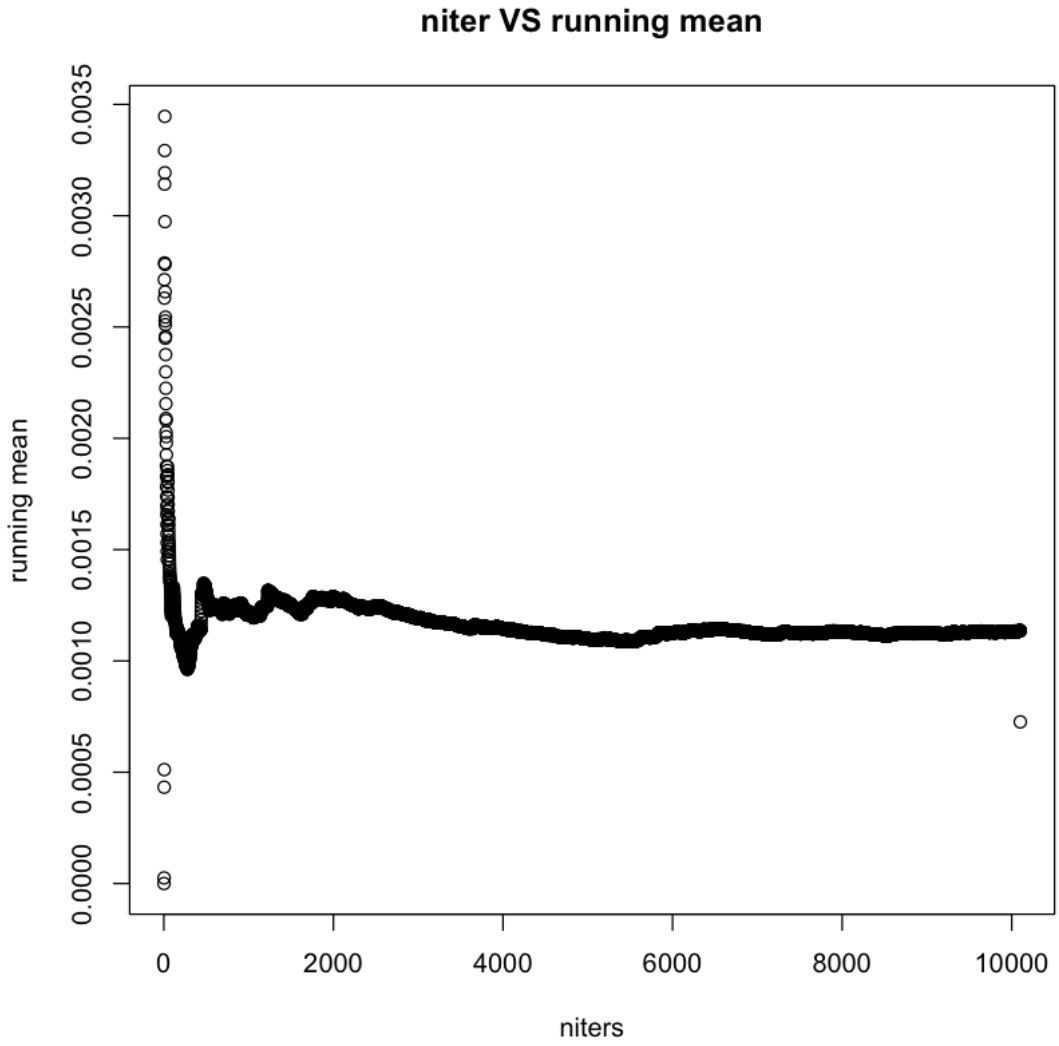
2 Exercise 4.4

```
In [3]: rm(list = ls())
        source("mcmc_cor.R")
        ## get data
        N = 1000
        X = rnorm(N, 0, 1)
        Y = rnorm(N, 0, 1)
        data = cbind(X,Y)

        burn_in = 10000
        iterations = 100 + burn_in
        ###I run mcmc and save result and model below
        start = proc.time()
        mc_out = mcmc_cor(data, iterations,burn_in = 0 , rho = 0.1, thinning= 1, size = 0.1)
        runtime = proc.time() - start
        print(paste0("runtime: ", runtime[[3]]))
```

```
[1] "runtime: 0.561999999999898"
```

```
In [4]: running = running_var_mean(mc_out)
        plot(running$var, xlab = "niters", ylab = "running variance", main = "niter VS running
        plot(running$mean, xlab = "niters", ylab = "running mean", main = "niter VS running mea
```


niter VS running variance
running variance
0.0000
0.0005
0.0010
0.0015
0.0020
0.0025
0.0030
0.0035
0
2000
4000
6000
8000
10000
niters



Answer: The running mean converges to around `0.001` and variance converges to be slightly higher than `0.001`. Probably they are close to the mean and variance of the posterior.

3 Appendix

3.1 code for part 4.2, 4.3

```
In [1]: ## mcmc importance sampling (adapted from Daniel's code)
        mcmc_is <- function(iterations,burn_in,sd = 10, skill = c(0,0,0,0,0,0), thinning= 1){
            set.seed(12345)
            teams=as.matrix(read.table("teams.txt"))
            outcomes=read.table("outcomes.txt")
            g<-function(x) { 1/(1+exp(-x)) }
```

```
    pi<-function(skill) {  #Calculate the prior*likelihood, up to a constant.
      x=rowSums(teams %*% skill)  #Calculate the parameter to the function g.
      exp(-sum(skill**2)/8)*prod(ifelse(outcomes==1,g(x),1-g(x)))
    }

    n0=max(burn_in %/% thinning,1)
    n1=iterations %/% thinning
    mc_output=matrix(nrow=n1-n0+1,ncol=6)       #Make space for the output

    for (i in 0:n1) {
        if (i>=n0) mc_output[i-n0+1,]=skill
        for (j in 1:thinning) {
            skill_=rnorm(6,0,sd)
            alpha=min(1,pi(skill_)/pi(skill)*exp(sum(skill_**2-skill**2)/(2*sd**2)))
            if ( runif(1) < alpha ) skill=skill_}}
    x<-mcmc(mc_output,start=n0*thinning,thin=thinning)
    return(x)
}

## mcmc random walk
mcmc_rw <- function(iterations,burn_in,skill = c(0,0,0,0,0,0), thinning= 1){
    set.seed(12345)
    teams=as.matrix(read.table("teams.txt"))
    outcomes=read.table("outcomes.txt")
    g<-function(x) { 1/(1+exp(-x)) }

    pi<-function(skill) {  #Calculate the prior*likelihood, up to a constant.
      x=rowSums(teams %*% skill)  #Calculate the parameter to the function g.
      exp(-sum(skill**2)/8)*prod(ifelse(outcomes==1,g(x),1-g(x)))
    }

    n0=max(burn_in %/% thinning,1)
    n1=iterations %/% thinning
    mc_output=matrix(nrow=n1-n0+1,ncol=6)       #Make space for the output

    for (i in 0:n1) {
    if (i>=n0) mc_output[i-n0+1,]=skill
    for (j in 1:thinning) {
        idx = sample(1:6,1)
        skill_= skill
        skill_[idx] = skill_[idx] + rnorm(1,0,1)

        #alpha=min(1,pi(skill_)/pi(skill))
        alpha=min(1,pi(skill_)/pi(skill))
        if ( runif(1) < alpha ) skill=skill_}}
    x<-mcmc(mc_output,start=n0*thinning,thin=thinning)
    return(x)
}
```

```
## visualize mcmc chain
mc_vis <- function(x){
    plot(x)
    summary(x)
}
```

3.2 code for part 4.4

```
In [ ]: ## mcmc random walk for estimating correlation in problem 4.4
mcmc_cor <- function(data, iterations,burn_in, rho = 0.1, thinning= 1, size = 0.1){
    set.seed(12345)

    ll <- function(rho, data){ ## compute loglikelihood of data given rho
        N = nrow(data)
        #lls <- apply(data, 1, ll_helper, rho)
        X = data[,1]
        Y = data[,2]
        lls = X^2 + Y^2 - 2*rho*X*Y
        ll <- -1/(2*(1-rho^2)) * sum(lls) - N * log(2*pi*sqrt(1-rho^2))
        return(ll)
    }

    prior <- function(rho){
        return(1/(pi*(1-rho^2)^(1/2)))
    }

    target_ratio <-function(rho_, rho, data) {  # compute ratio of target posterior(rh
        ll = ll(rho, data)
        ll_ = ll(rho_, data)
        ratio = exp(ll_ - ll) * prior(rho_)/prior(rho)
        return(ratio)
    }

    accept <- function(rho_, rho, data){ ## compute the acceptance probability
        return(min(1, target_ratio(rho_, rho, data)))
    }

    rw_sampler <- function(rho, size){ ## get new rho_ from old rho
        return(runif(1, rho-size, rho+size))
    }

    ## computation

    n0=max(burn_in %/% thinning,1)
    n1=iterations %/% thinning
    mc_output=matrix(nrow=n1-n0+1,ncol=1)       #Make space for the output
```

```
    for (i in 0:n1) {
        if (i>=n0) mc_output[i-n0+1,]=rho
        for (j in 1:thinning) {
            rho_ = rw_sampler(rho, size)
            alpha= accept(rho_, rho, data)
            if ( runif(1) < alpha ) rho=rho_}}
    x<-mcmc(mc_output,start=n0*thinning,thin=thinning)
    return(x)
}

## compute running variance an d running mean
running_var_mean <- function(mc_out){
  var_running = c()
  mean_running = c()
  for(i in 1:length(mc_out)){
    var_running <- c(var_running, var(mc_out[1:i]))
    mean_running <- c(var_running, mean(mc_out[1:i]))
  }
  return(list(var = var_running, mean = mean_running))
}
```