STAT 431 - HW 5

Sharvi Tomar

11/11/2022

Problem 1

Part 1(b)

```
library(rjags)  # automatically loads coda package
```

```
## Loading required package: coda
```

```
## Linked to JAGS 4.3.1
```

```
## Loaded modules: basemod,bugs
```

```
### Set up data, initializations, and model

d = read.table("airlinerdata.txt", header=TRUE)
head(d)
```

```
##    y     N
## 1  1  0.08
## 2  9  8.00
## 3  5  2.70
## 4  4  6.00
## 5 46 70.00
## 6 47 76.00
```

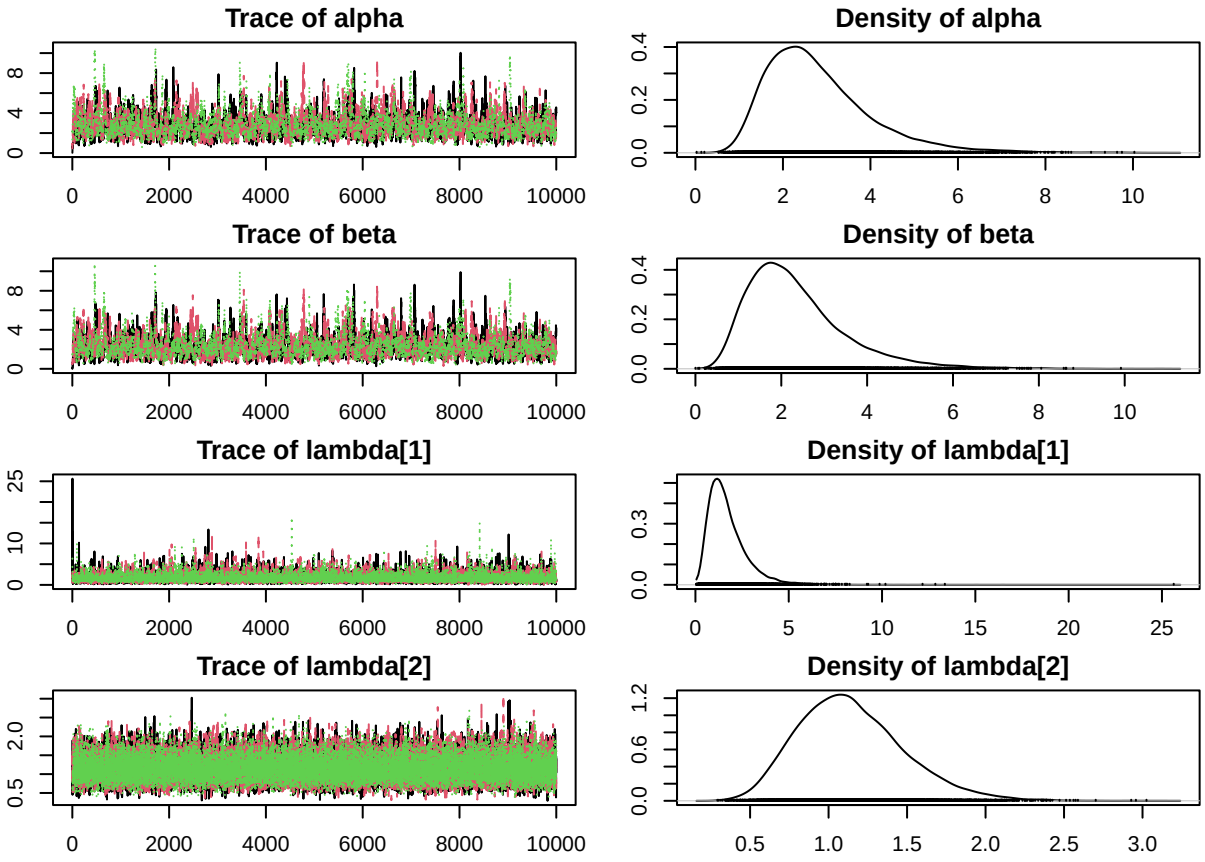
```
inits = list(list(alpha = 0.005, beta = 0.005),
             list(alpha = 0.05, beta = 0.05),
             list(alpha = 0.5, beta = 0.5))

model = jags.model("prob1model.bug", d, inits, n.chains=3 , n.adapt = 0)
```

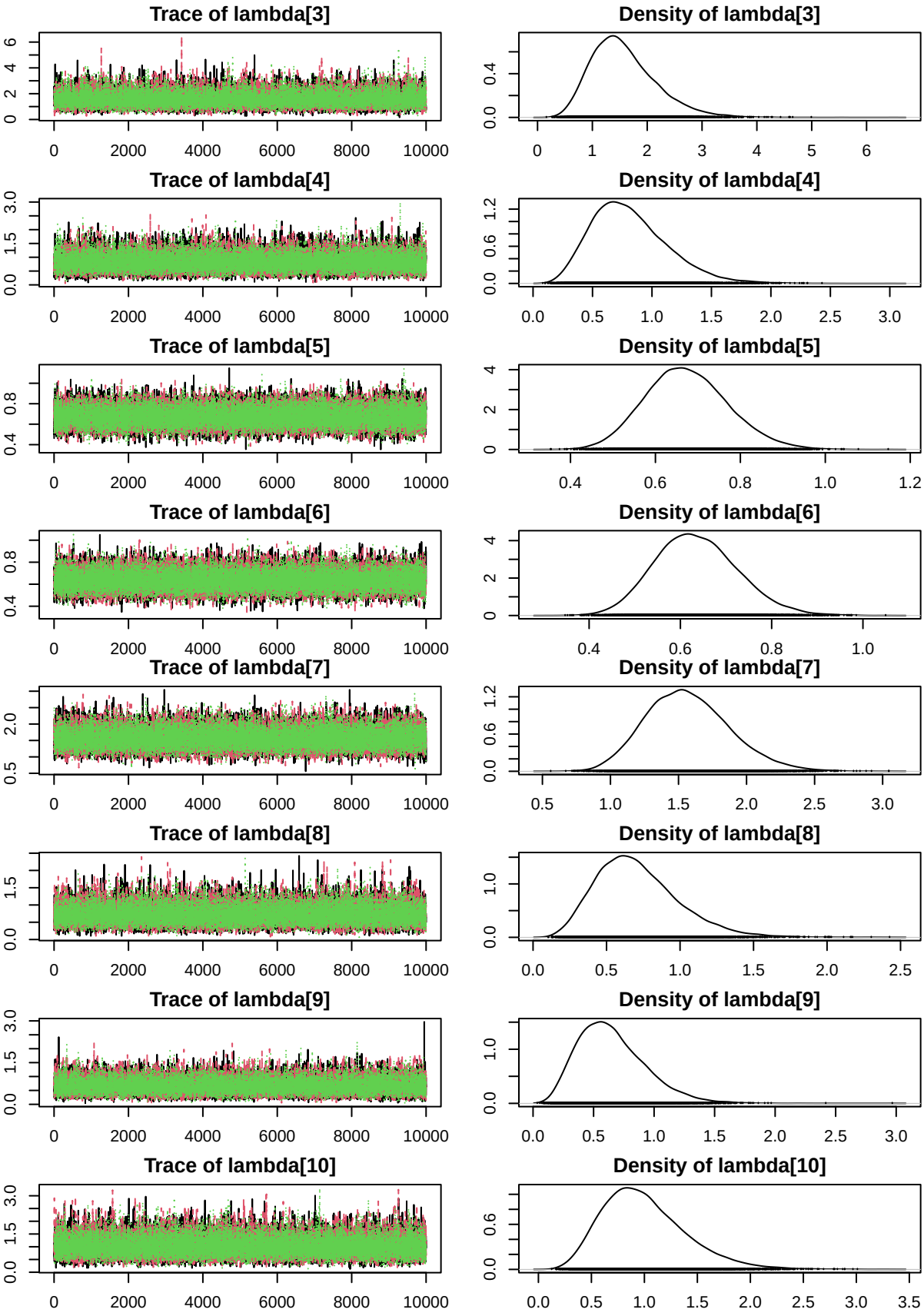
```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 20
##    Unobserved stochastic nodes: 22
##    Total graph size: 83
##
## Initializing model
```

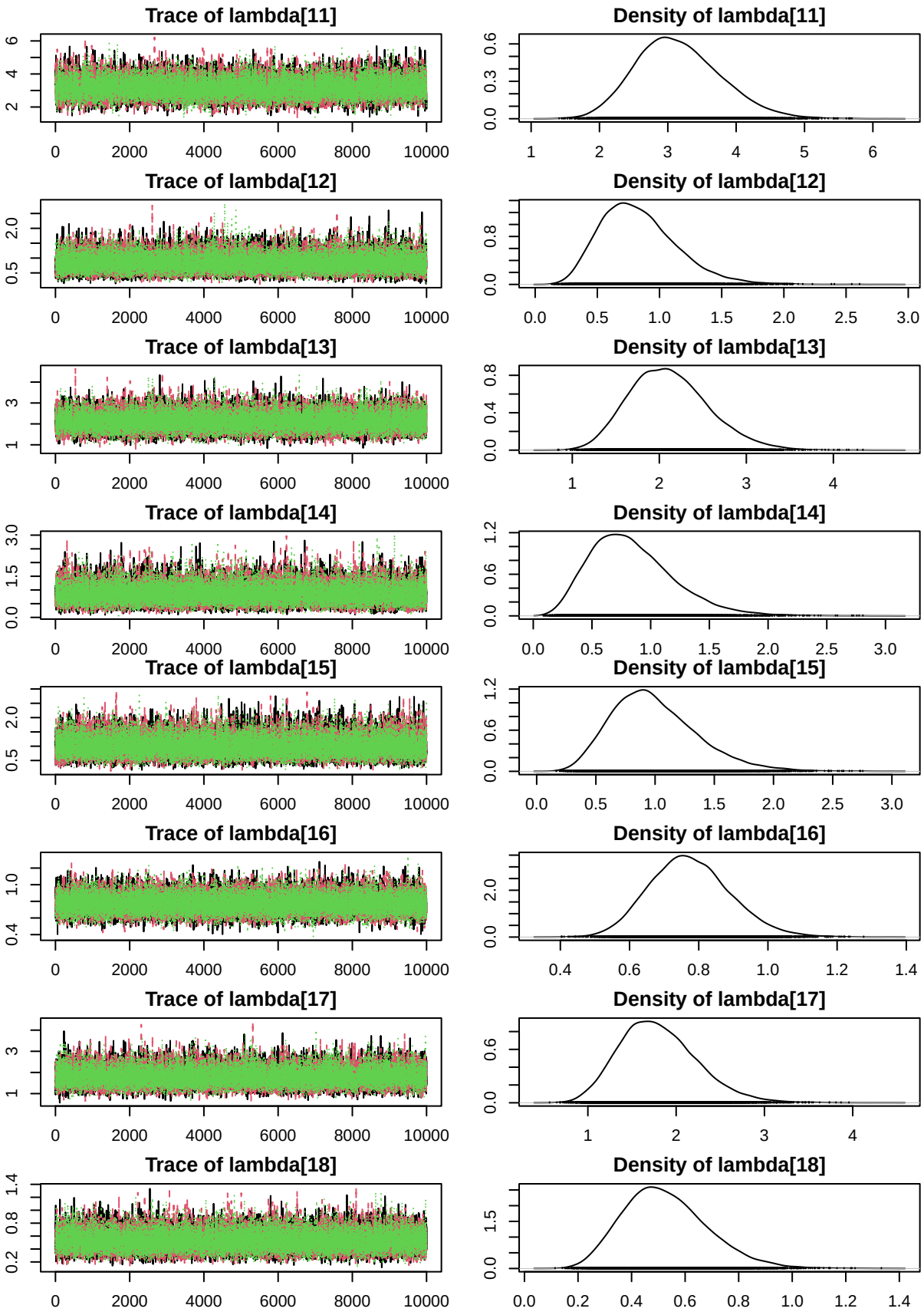
```
### Make a preliminary run of 1000 iterations
params= c("alpha" , "beta" , "lambda")
x = coda.samples(model,params, n.iter=10000, n.burnin = 1000)
```

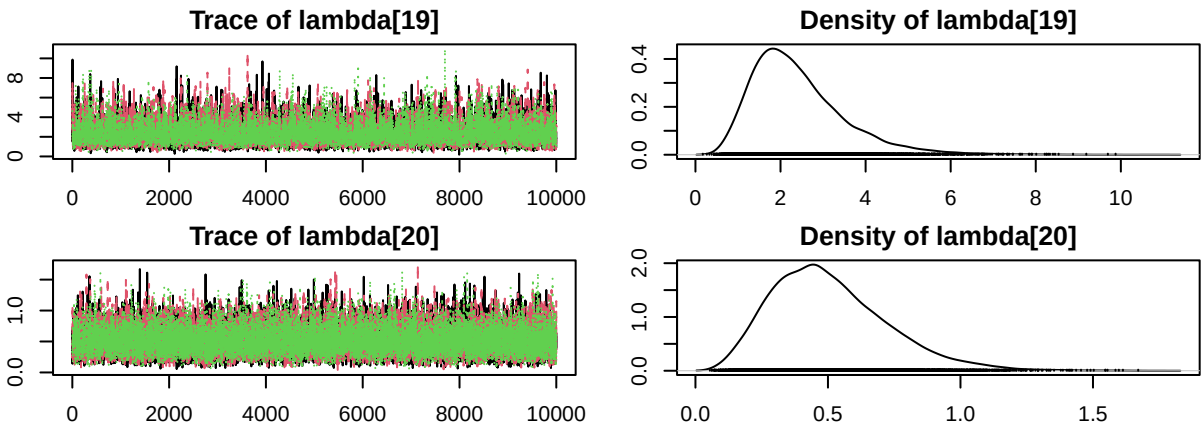
```
## NOTE: Stopping adaptation
```

```
par(mar=c(2,2,2,2))
plot(x, smooth=FALSE)
```

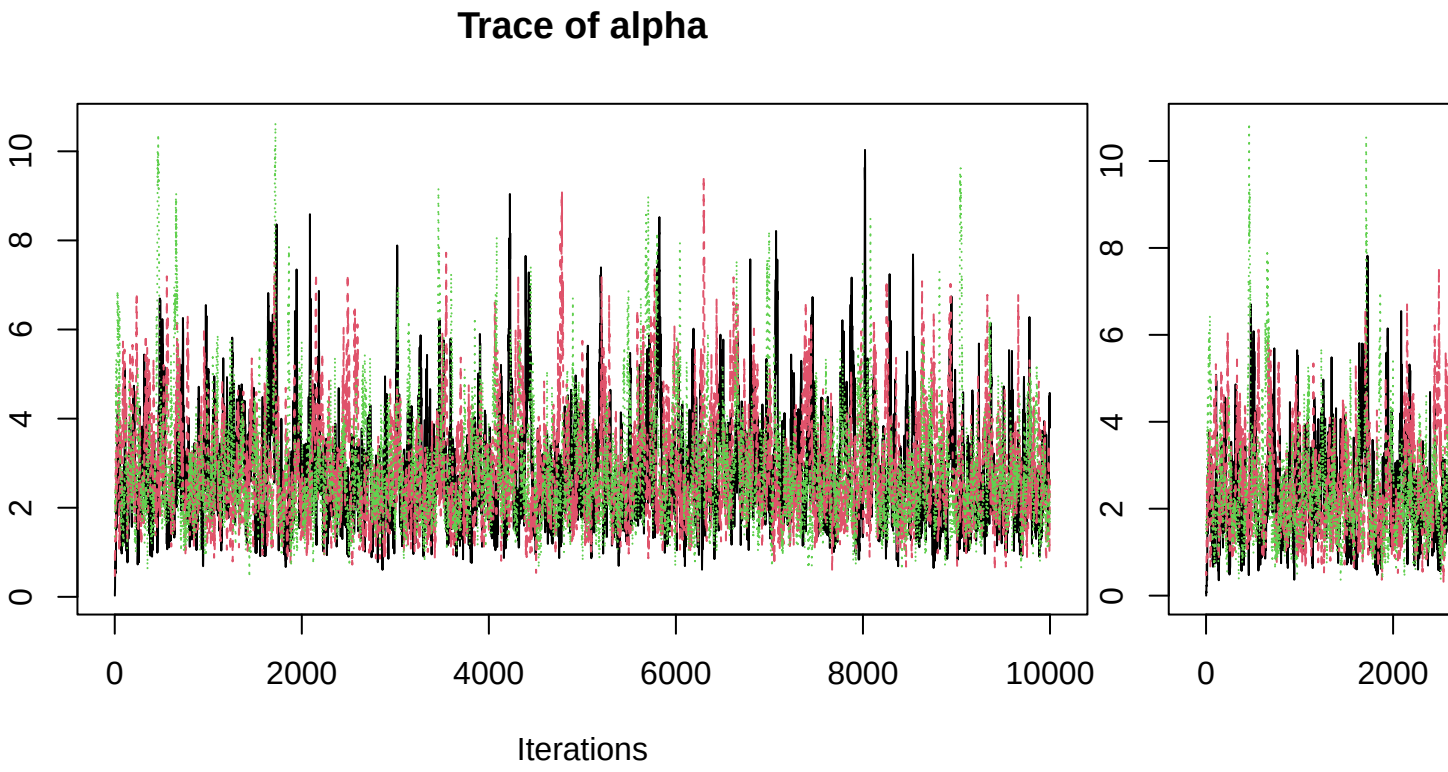


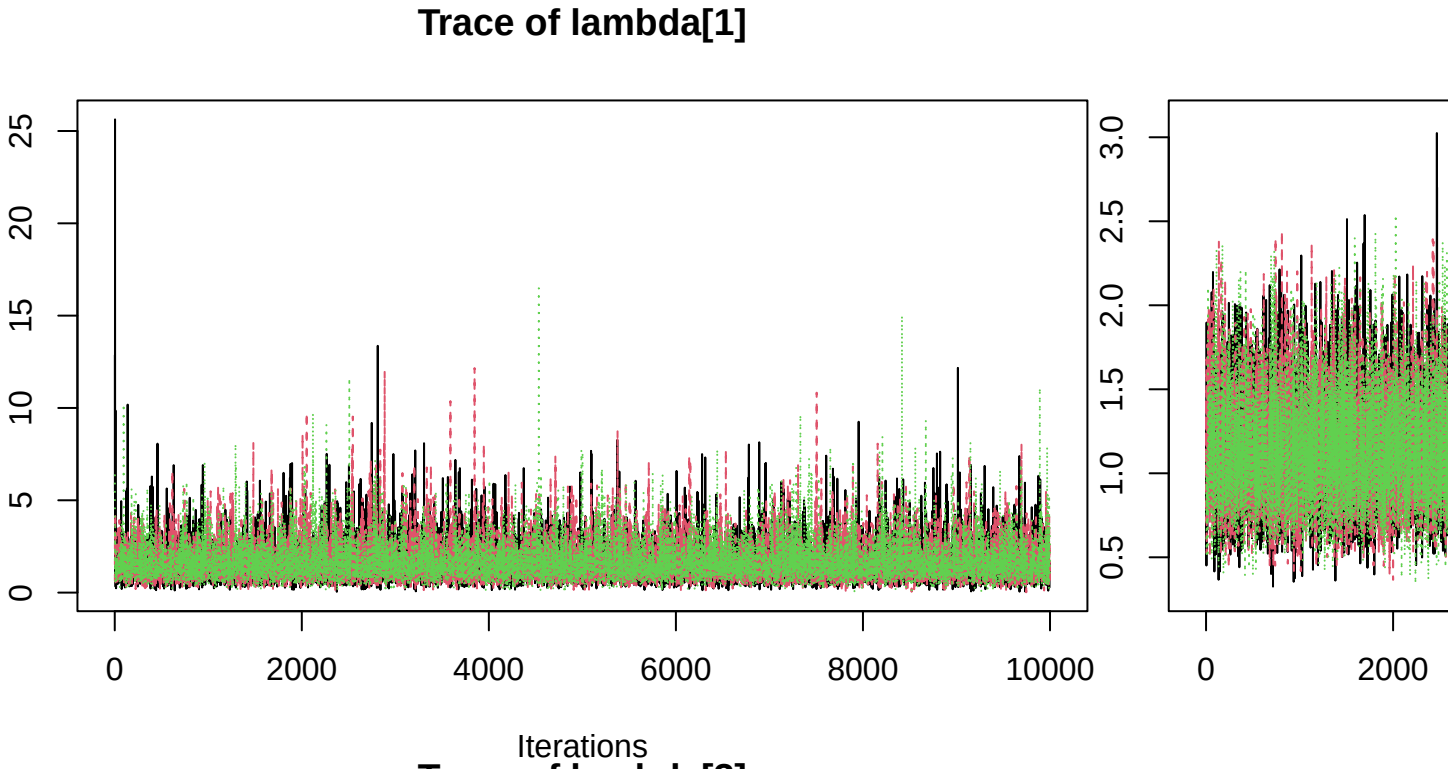

Trace of lambda[3]
Density of lambda[3]
Trace of lambda[4]
Density of lambda[4]
Trace of lambda[5]
Density of lambda[5]
Trace of lambda[6]
Density of lambda[6]
Trace of lambda[7]
Density of lambda[7]
Trace of lambda[8]
Density of lambda[8]
Trace of lambda[9]
Density of lambda[9]
Trace of lambda[10]
Density of lambda[10]

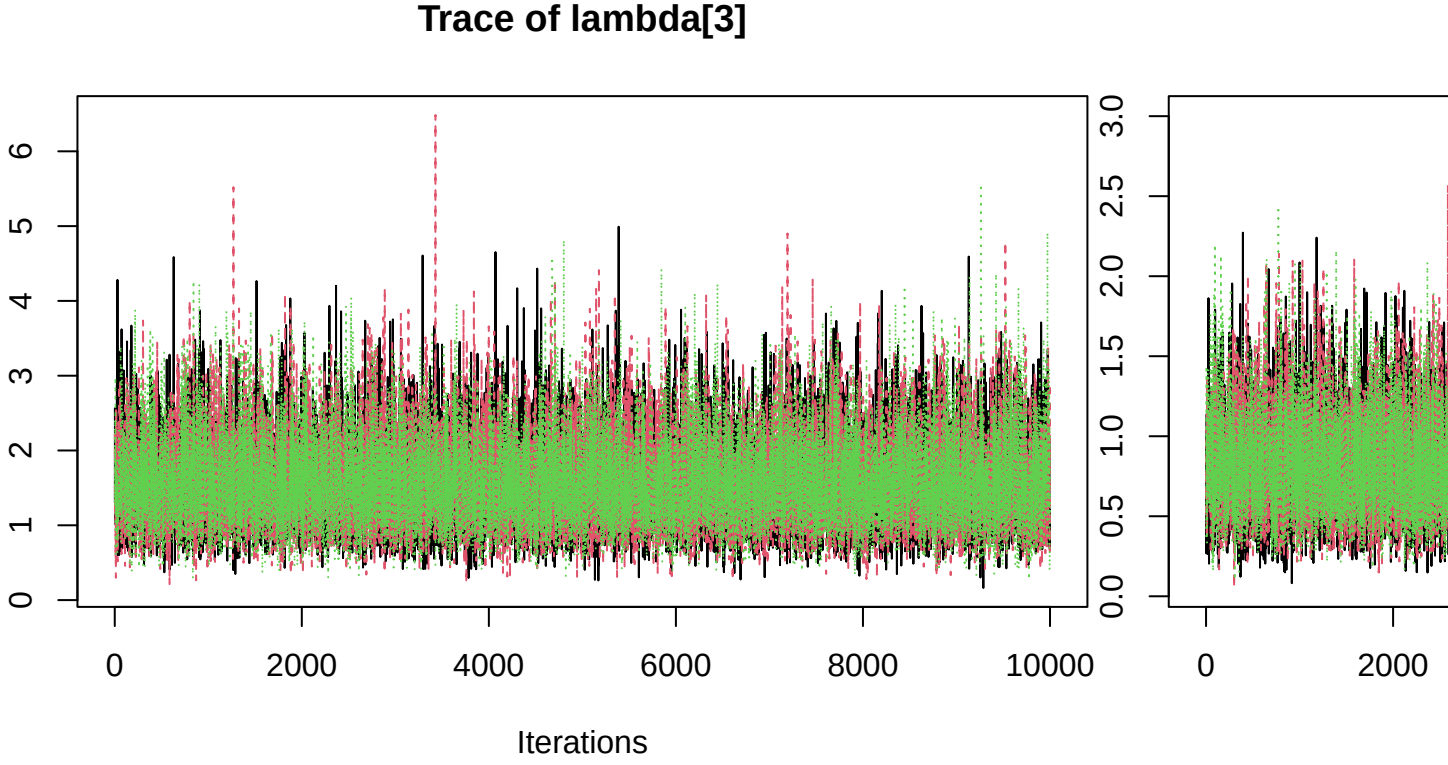

Trace of lambda[11]
Density of lambda[11]
Trace of lambda[12]
Density of lambda[12]
Trace of lambda[13]
Density of lambda[13]
Trace of lambda[14]
Density of lambda[14]
Trace of lambda[15]
Density of lambda[15]
Trace of lambda[16]
Density of lambda[16]
Trace of lambda[17]
Density of lambda[17]
Trace of lambda[18]
Density of lambda[18]

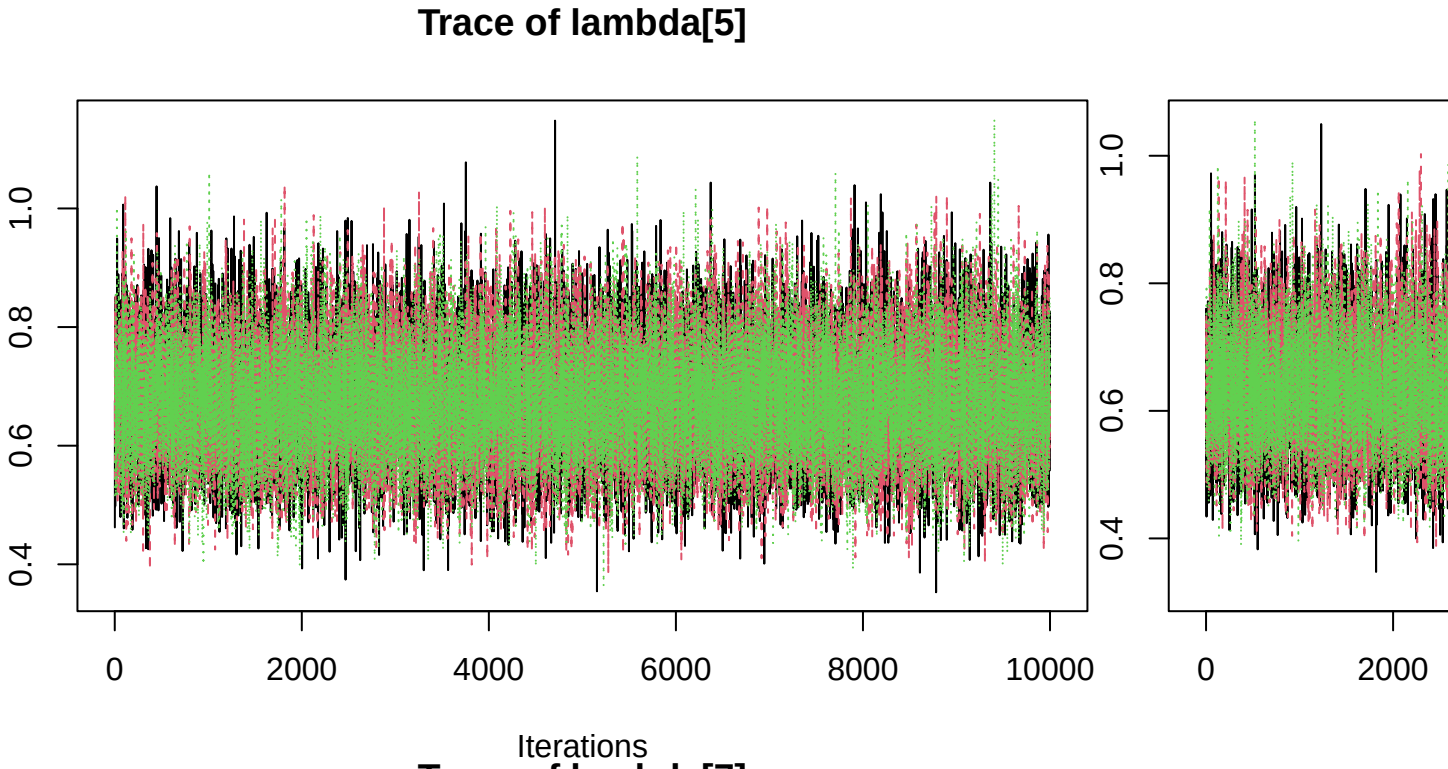


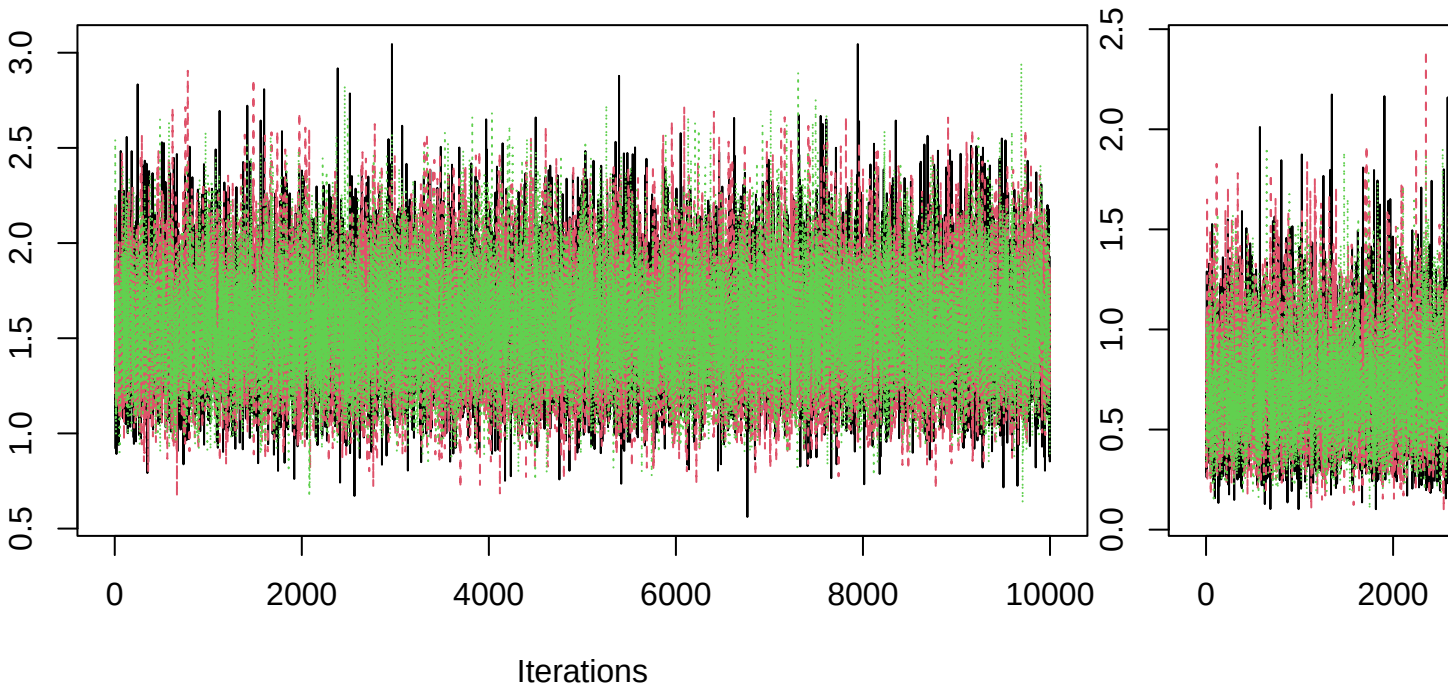

```
traceplot(x) # plot the chains
```

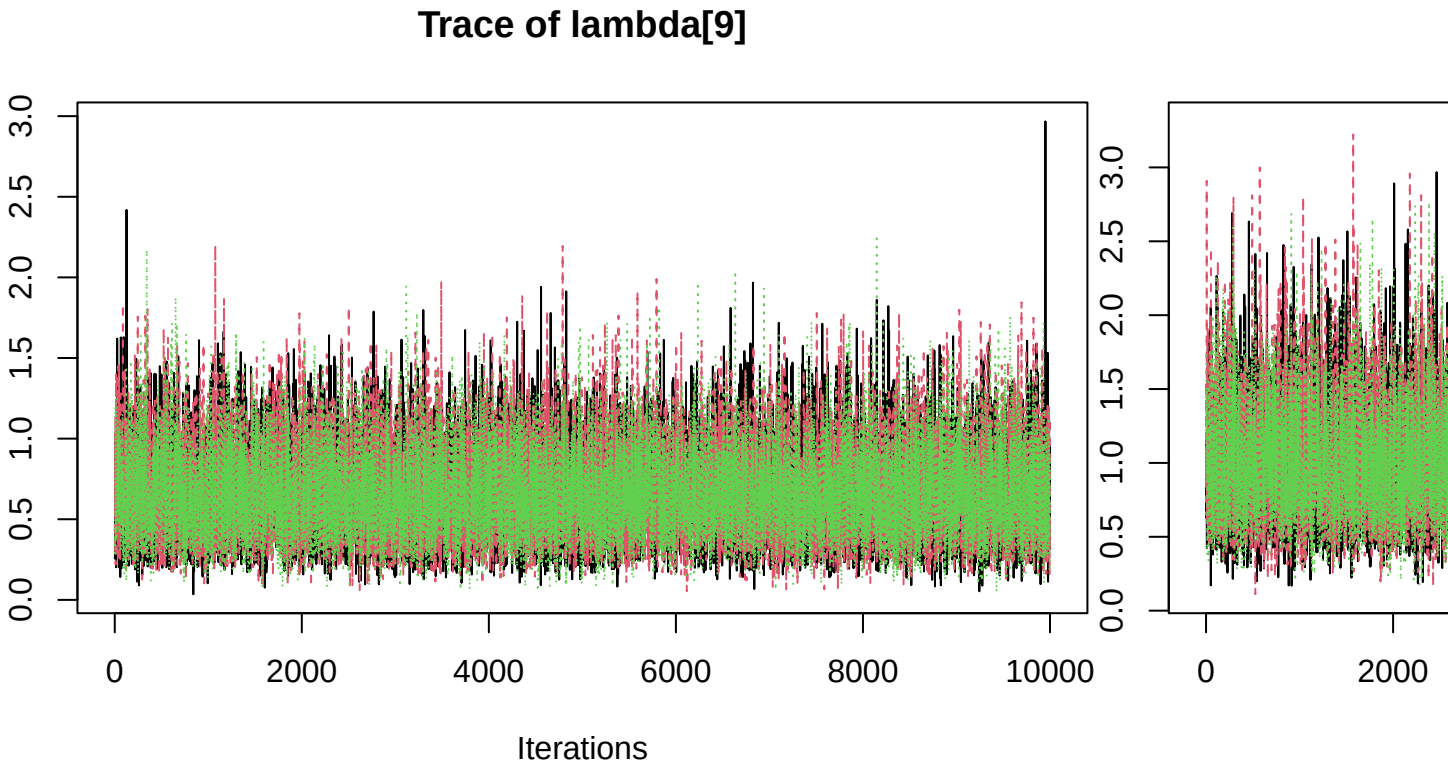


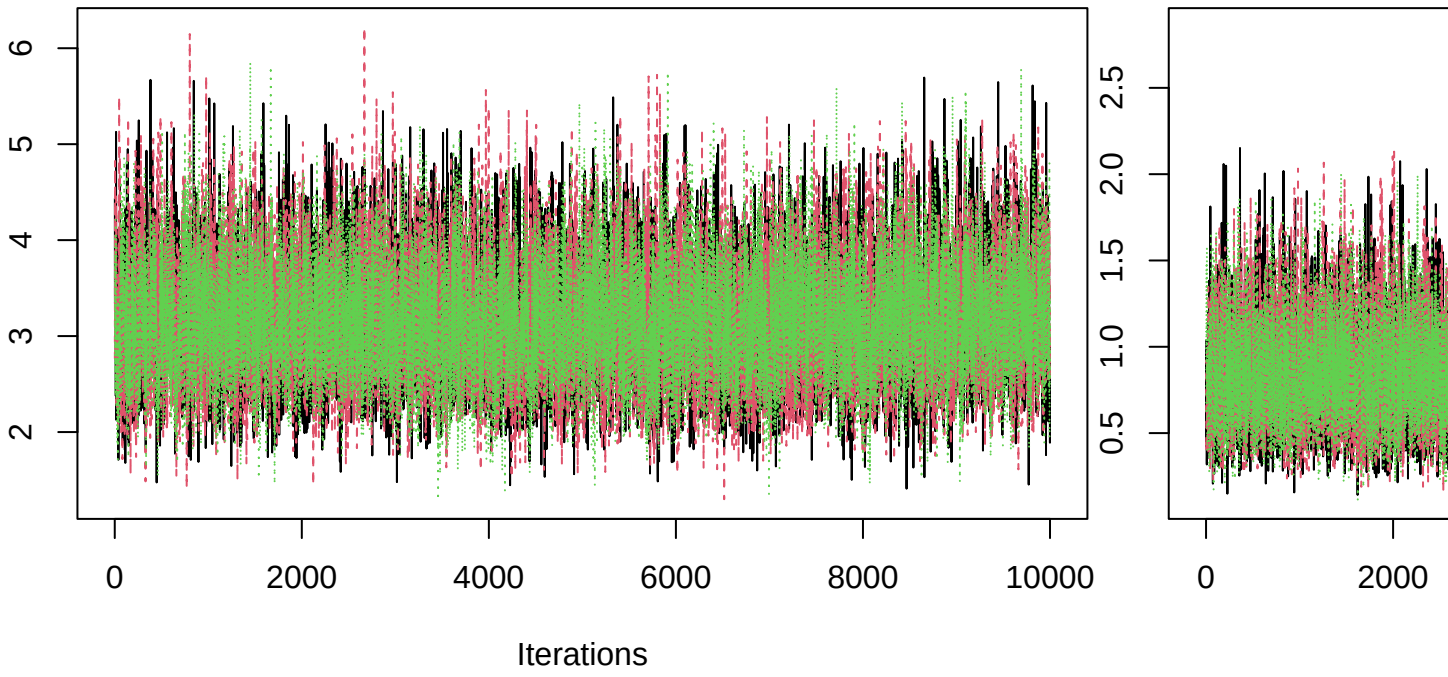

Trace of lambda[1]
0
5
10
15
20
25
0
2000
4000
6000
8000
10000
Iterations
0.5
1.0
1.5
2.0
2.5
3.0
0
2000

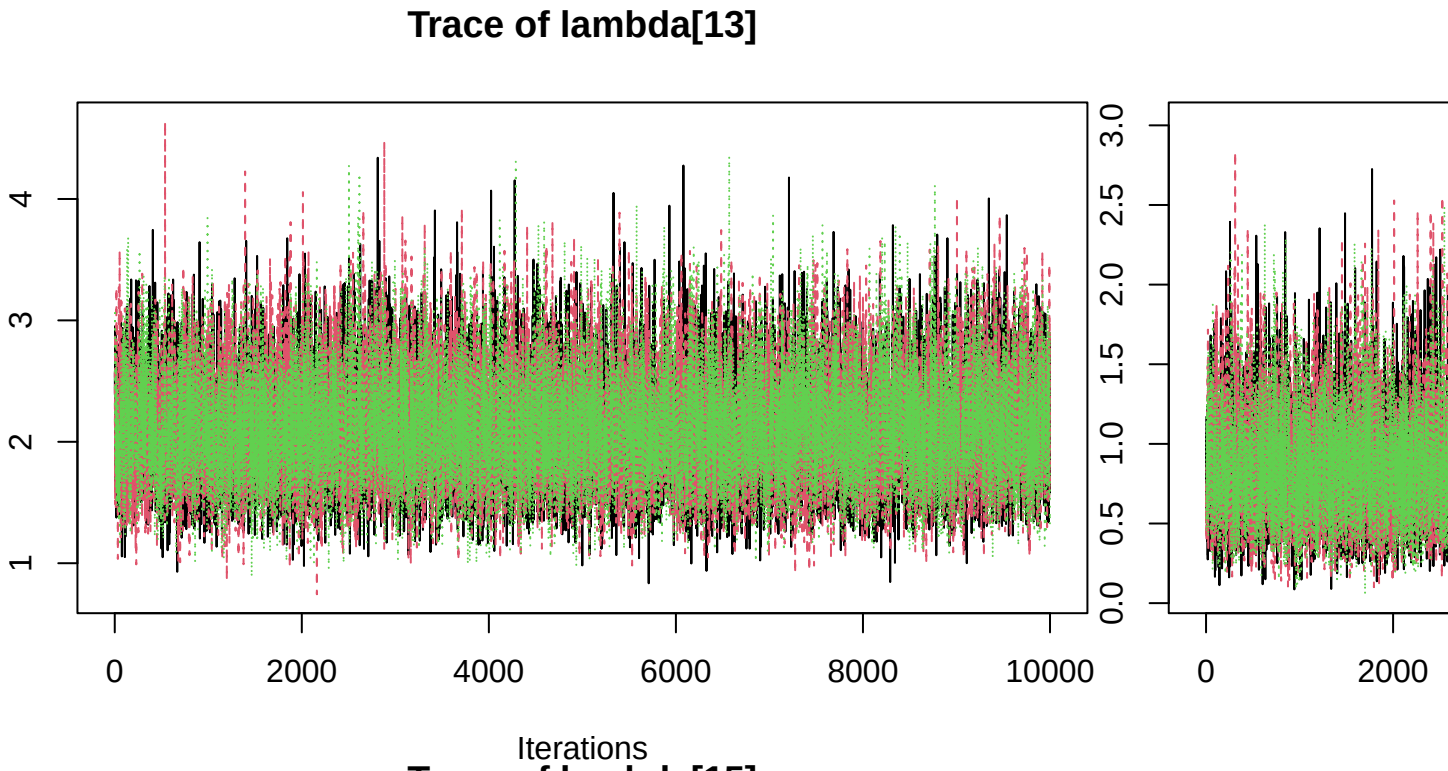

Trace of lambda[3]
0
1
2
3
4
5
6
0
2000
4000
6000
8000
10000
Iterations
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
2000

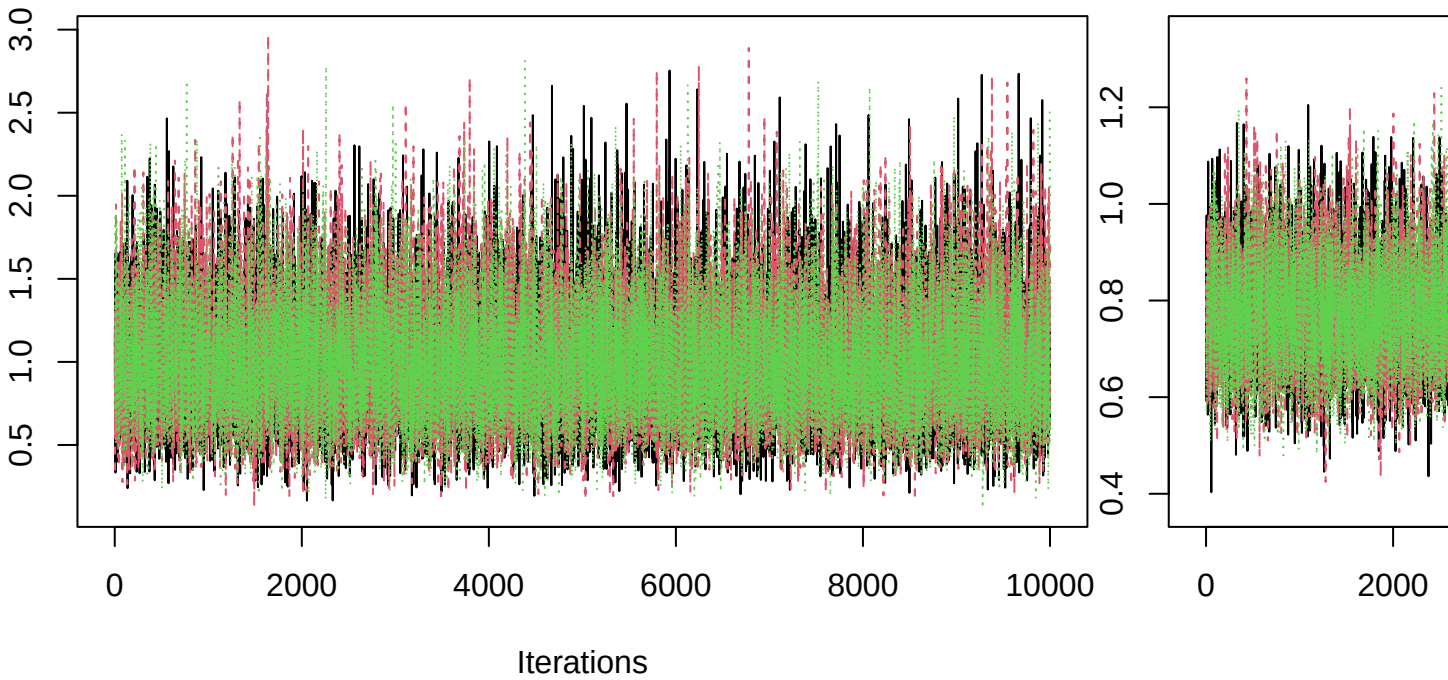

Trace of lambda[5]
0.4
0.6
0.8
1.0
0
2000
4000
6000
8000
10000
Iterations

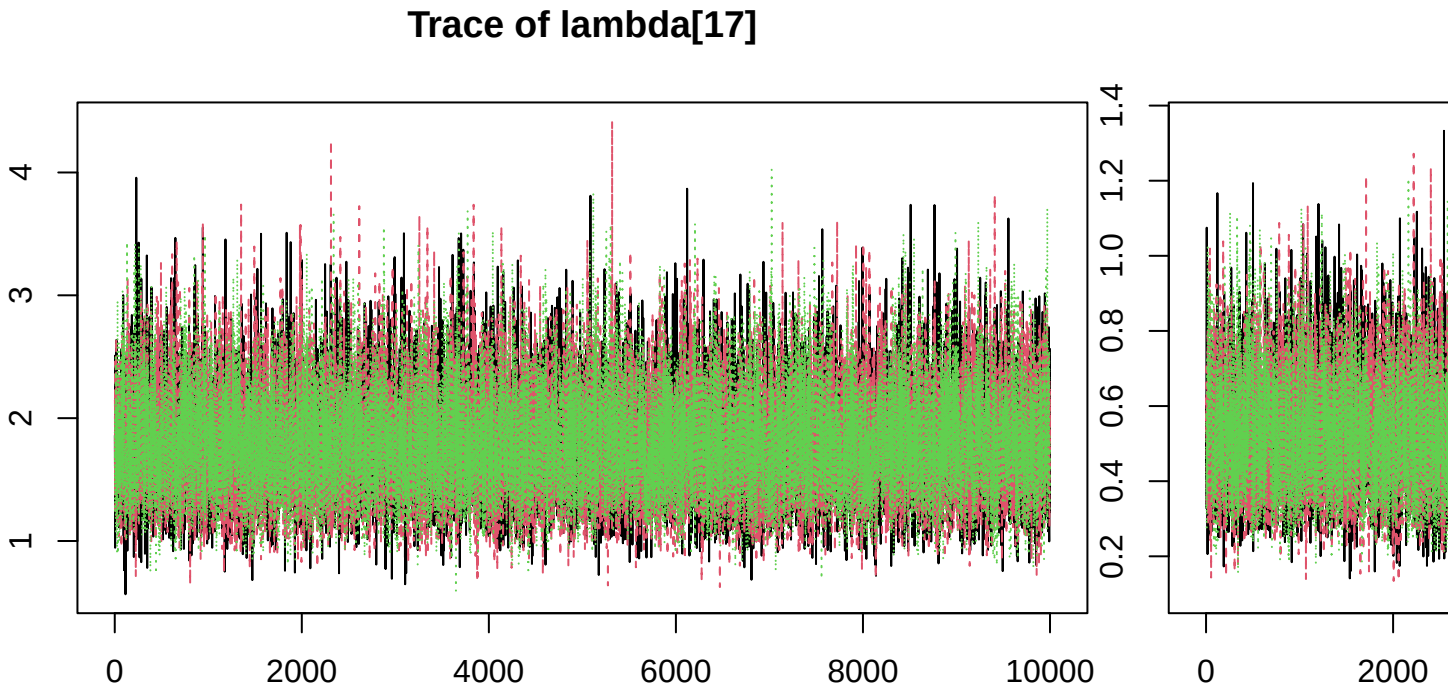

Trace of lambda[7]
0.5
1.0
1.5
2.0
2.5
3.0
0
2000
4000
6000
8000
10000
Iterations

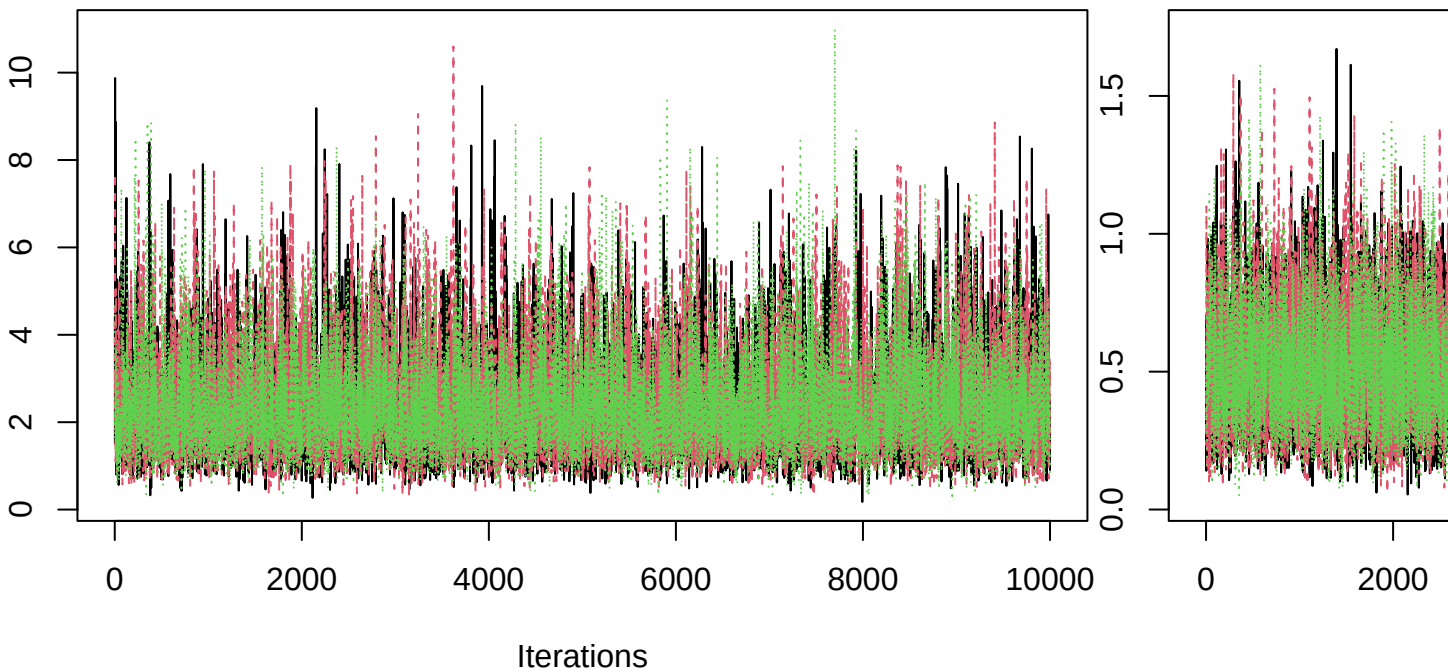

Trace of lambda[9]
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
2000
4000
6000
8000
10000
Iterations


Trace of lambda[11]
2
3
4
5
6
0
2000
4000
6000
8000
10000
Iterations

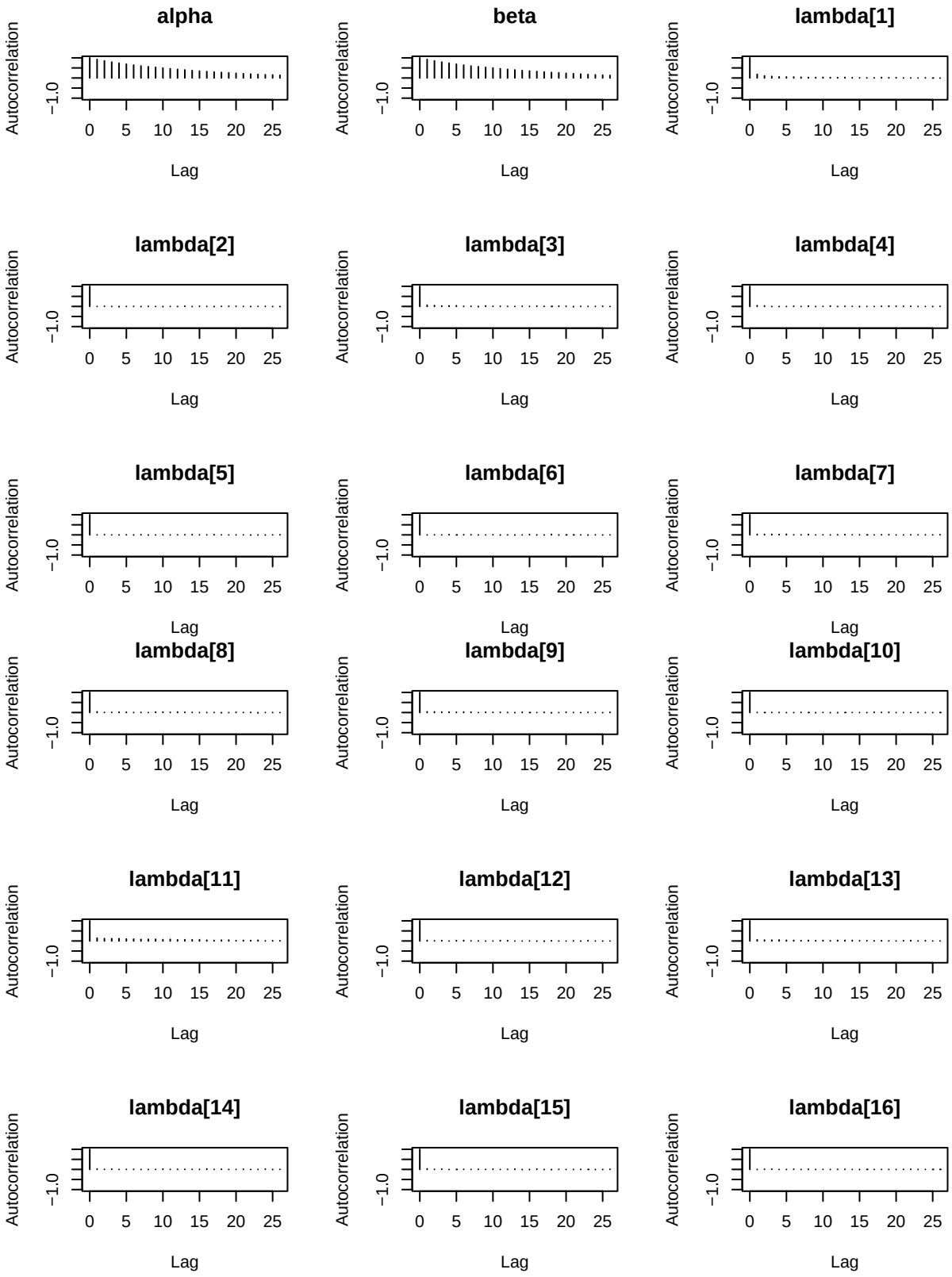

Trace of lambda[13]
1
2
3
4
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
2000
4000
6000
8000
10000
Iterations

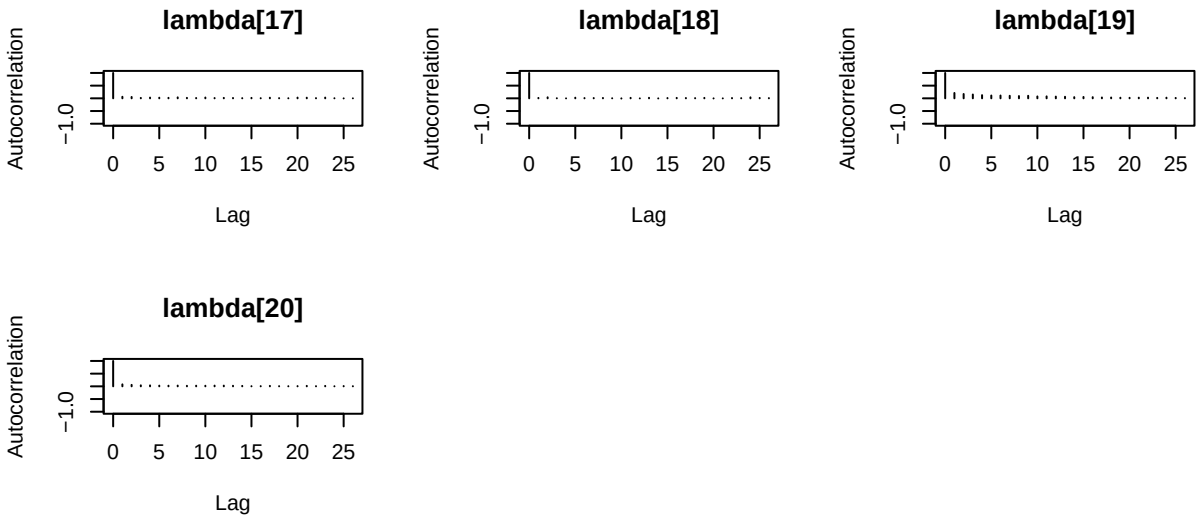

Trace of lambda[15]
0.5
1.0
1.5
2.0
2.5
3.0
0.4
0.6
0.8
1.0
1.2
0
2000
4000
6000
8000
10000
Iterations






```
autocorr.plot(x[1]) # plot autocorrelations of 1st chain
```

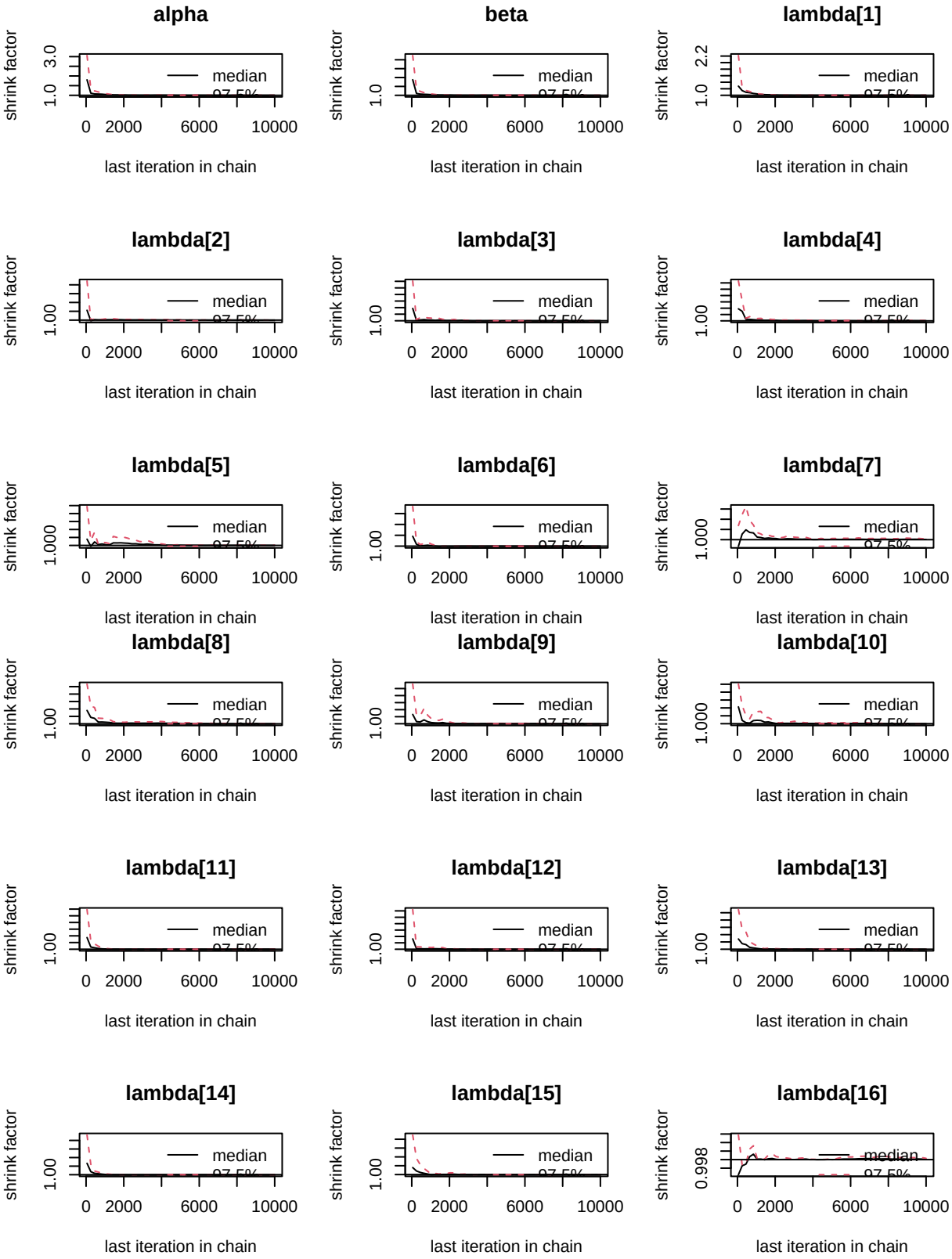

alpha
beta
lambda[1]
lambda[2]
lambda[3]
lambda[4]
lambda[5]
lambda[6]
lambda[7]
lambda[8]
lambda[9]
lambda[10]
lambda[11]
lambda[12]
lambda[13]
lambda[14]
lambda[15]
lambda[16]
Autocorrelation
−1.0
0
5
10
15
20
25
Lag

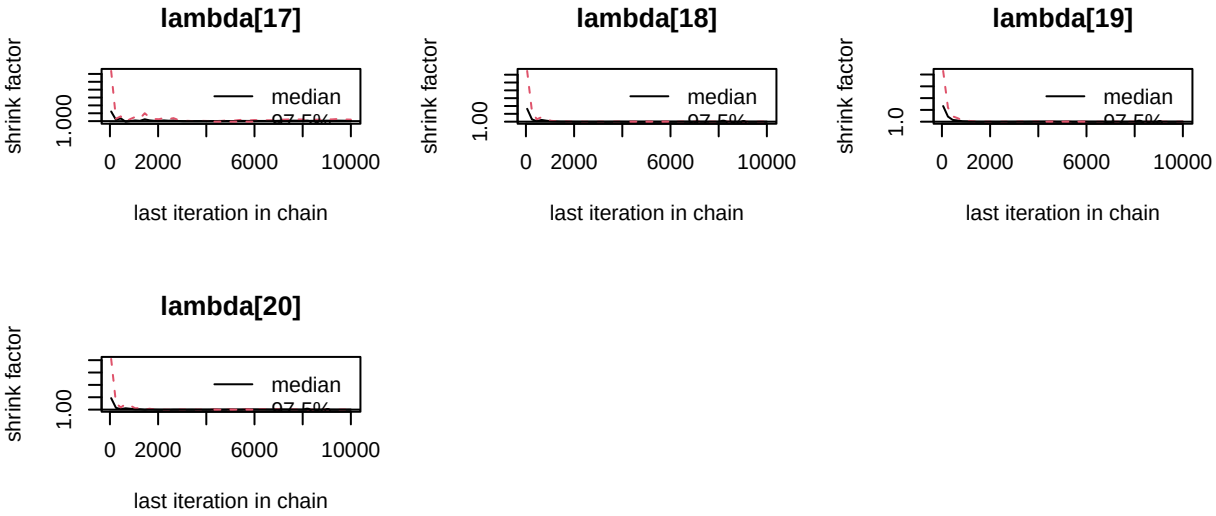



```
gelman.diag(x, autoburnin=FALSE) # Gelman-Rubin statistic
```

```
## Potential scale reduction factors:
##
##            Point est. Upper C.I.
## alpha               1          1
## beta                1          1
## lambda[1]           1          1
## lambda[2]           1          1
## lambda[3]           1          1
## lambda[4]           1          1
## lambda[5]           1          1
## lambda[6]           1          1
## lambda[7]           1          1
## lambda[8]           1          1
## lambda[9]           1          1
## lambda[10]          1          1
## lambda[11]          1          1
## lambda[12]          1          1
## lambda[13]          1          1
## lambda[14]          1          1
## lambda[15]          1          1
## lambda[16]          1          1
## lambda[17]          1          1
## lambda[18]          1          1
## lambda[19]          1          1
## lambda[20]          1          1
##
## Multivariate psrf
##
## 1
```

```
gelman.plot(x, autoburnin=FALSE) # Gelman-Rubin statistic plot
```


alpha
beta
lambda[1]
lambda[2]
lambda[3]
lambda[4]
lambda[5]
lambda[6]
lambda[7]
lambda[8]
lambda[9]
lambda[10]
lambda[11]
lambda[12]
lambda[13]
lambda[14]
lambda[15]
lambda[16]
shrink factor
median
97.5%
0
2000
6000
10000
last iteration in chain



Since the Gelman-Rubin statistic R < 1.05, we an declare convergence.

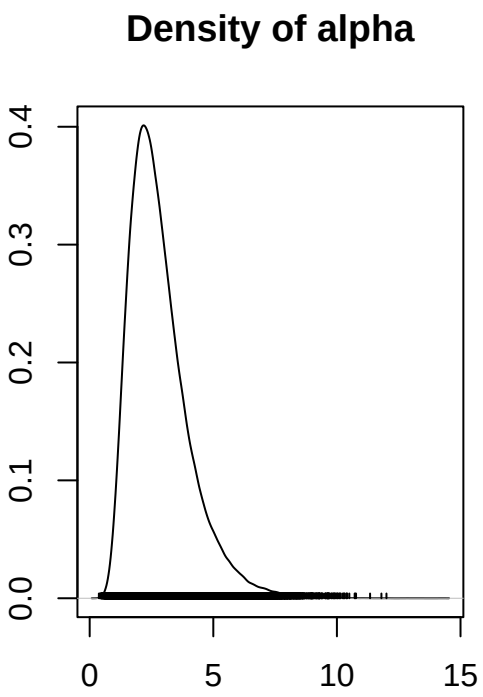
Part 1(c)

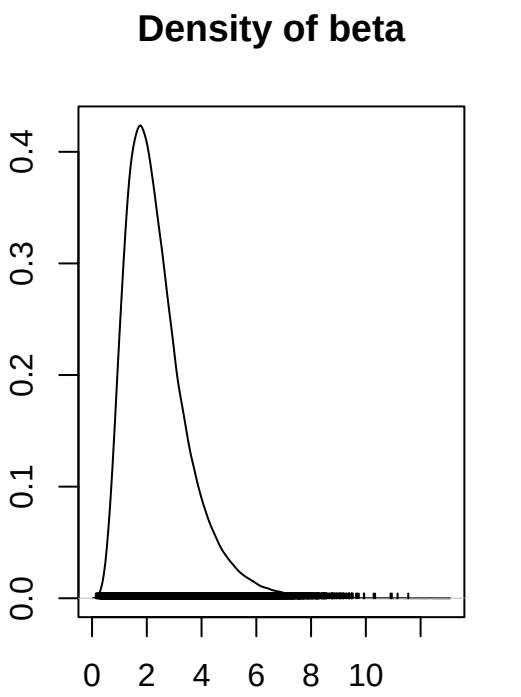
```
## 100000 iterations per chain (after burn-in)
x2 = coda.samples(model, c("alpha", "beta"), n.iter=100000 , n.burnin = 1000)
```

```
# summary output
summary(x2)
```

```
##
## Iterations = 10001:110000
## Thinning interval = 1
## Number of chains = 3
## 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
## alpha 2.833 1.230 0.002246        0.01203
## beta  2.380 1.181 0.002157        0.01159
##
## 2. Quantiles for each variable:
##
##         2.5%   25%   50%   75% 97.5%
## alpha 1.1421 1.963 2.597 3.441 5.896
## beta  0.7864 1.543 2.145 2.949 5.343
```

```
# graphical approximations of posterior densities
plot(x2[,c("alpha" , "beta")] , trace=FALSE)
```





Problem 2

PART 2(b)

```
# Generating probabilities
mu = 0
s2 = 1
n = 9
sigma2 = (n-1)/n
y = rnorm(length(n), mean = 0, sd =sigma2^0.5)

# Standard non-informative prior
t1 = qt(p=0.025, df= n-1, lower.tail = F)
upper_bound1 = t1 * (1/sqrt(n))
lower_bound1 = -t1 * (1/sqrt(n))
print(c(lower_bound1, upper_bound1))
```

```
## [1] -0.768668  0.768668
```

```
# Jeffreys prior
t2 = qt(p=0.025, df= n, lower.tail = F)
upper_bound2 = t2 * (sigma2/sqrt(n))
lower_bound2 = -t2 * (sigma2/sqrt(n))
print(c(lower_bound2, upper_bound2))
```

```
## [1] -0.6702688  0.6702688
```

The credible interval for standard non-informative prior is wider than the credible interval for Jeffreys' prior

Problem 3

PART 3(a)

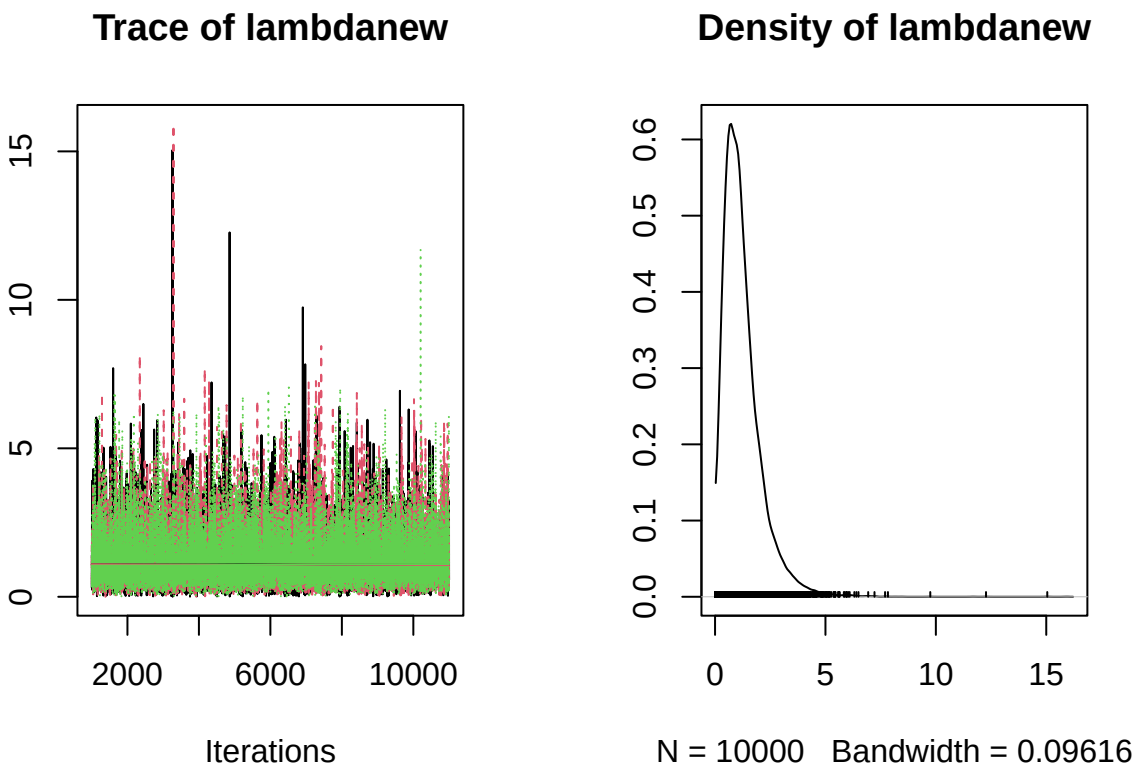
```
model_new = jags.model("new3model.bug", d, inits, n.chains=3)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 20
##    Unobserved stochastic nodes: 23
##    Total graph size: 84
##
## Initializing model
```

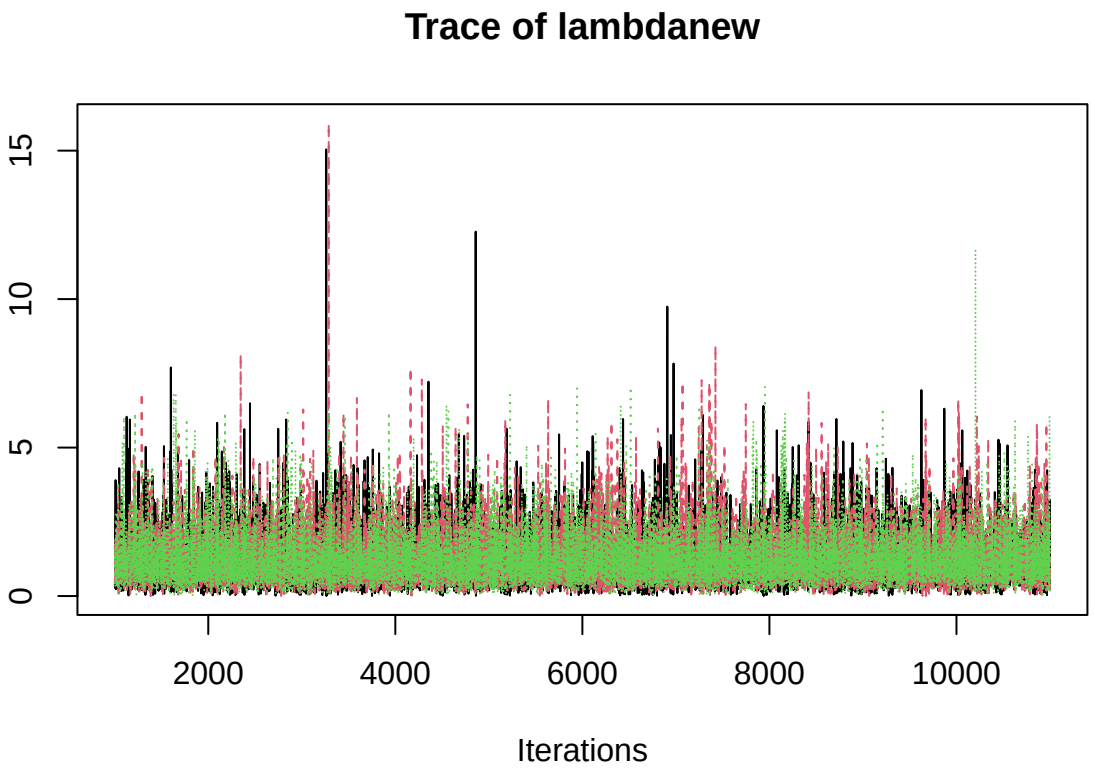
```
### Make a preliminary run of 1000 iterations
x_new = coda.samples(model_new, c("lambdanew"), n.iter=10000, n.burnin = 1000)
```

Assess convergence

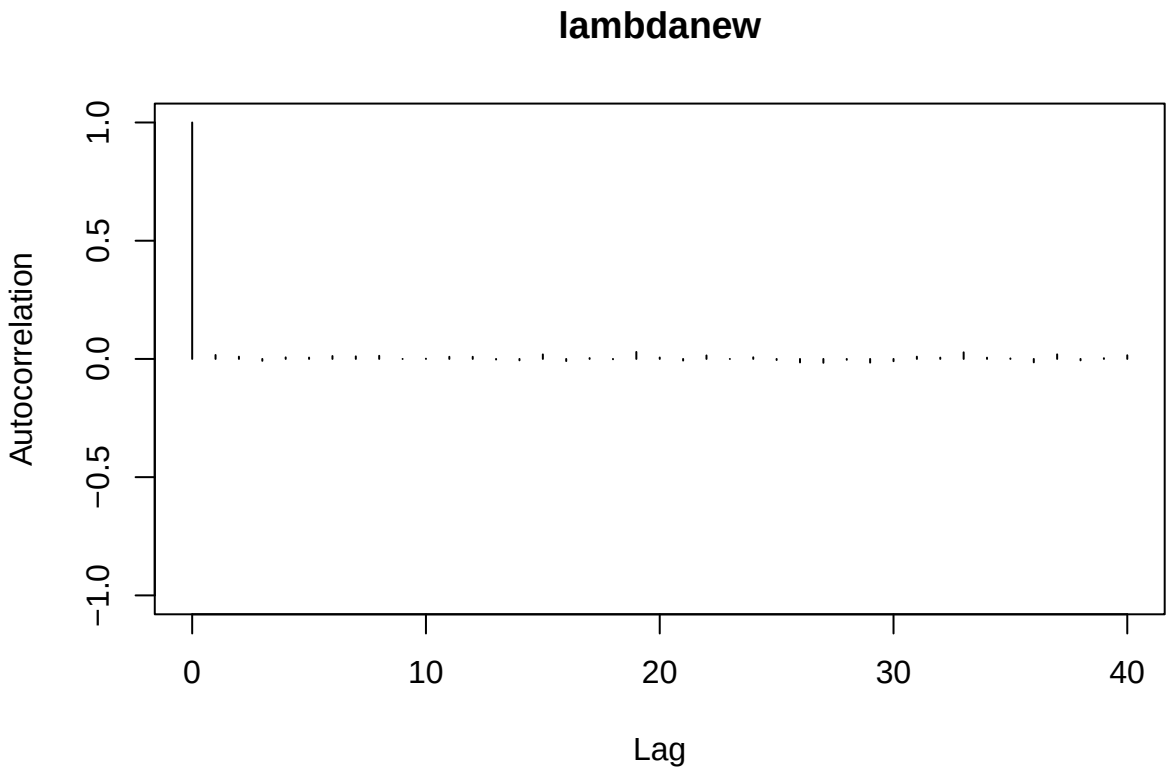
```
plot(x_new)
```



```
traceplot(x_new) # plot the chains
```



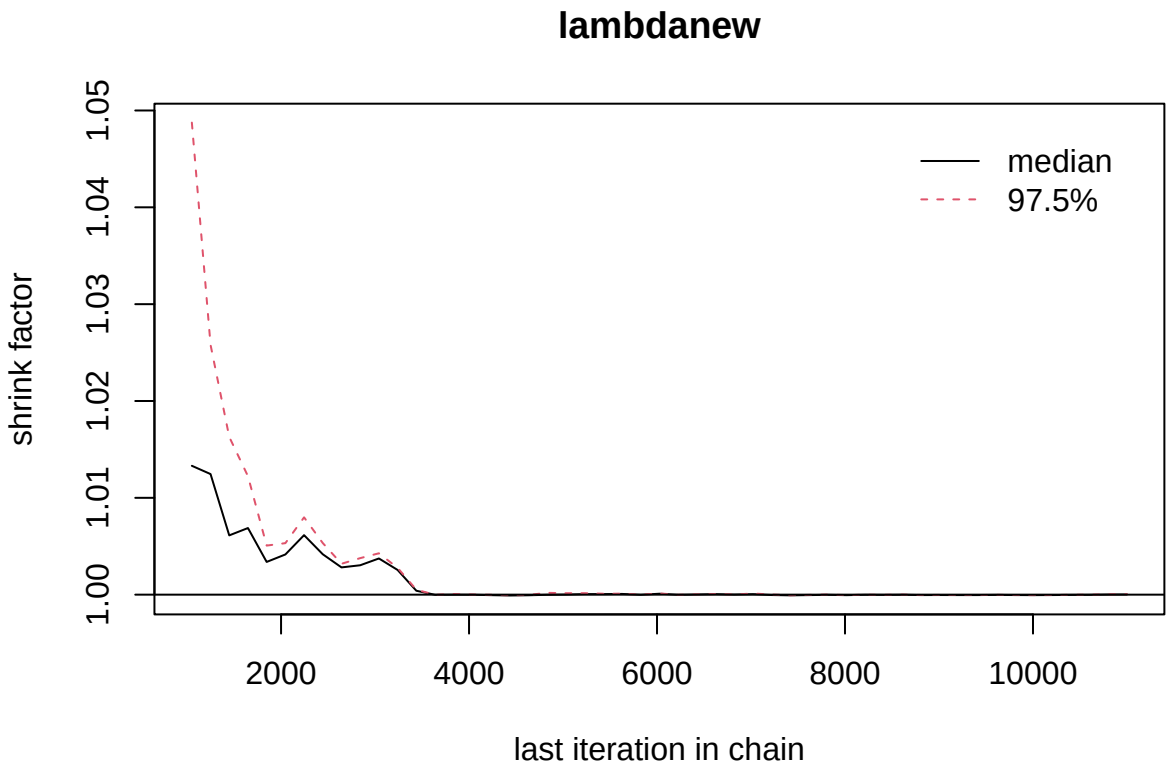

```
autocorr.plot(x_new[1]) # plot autocorrelations of 1st chain
```



```
gelman.diag(x_new, autoburnin=FALSE) # Gelman-Rubin statistic
```

```
## Potential scale reduction factors:
##
##           Point est. Upper C.I.
## lambdanew          1          1
```

```
gelman.plot(x_new, autoburnin=FALSE) # Gelman-Rubin statistic plot
```



Since Gelman-rubin statistic < 1.05, we can declare convergence.

```
## 100000 iterations per chain (after burn-in)
x2_new = coda.samples(model_new, c("lambdanew"), n.iter=100000 , n.burnin = 1000)
```

```
# summary output
summary(x2_new)
```

```
##
## Iterations = 11001:111000
## Thinning interval = 1
## Number of chains = 3
## 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
```

```
##       1.246790       0.872239       0.001592       0.001774
##
## 2. Quantiles for each variable:
##
##   2.5%    25%    50%    75%  97.5%
## 0.1700 0.6505 1.0589 1.6179 3.4425
```

PART 3(b)

```
### lambdanew
samps = rbind(x2_new[[1]], x2_new[[2]])

# Posterior mean and std. dev. of lambdanew
mean(samps)
```

```
## [1] 1.248083
```

```
sd(samps)
```

```
## [1] 0.8727906
```

```
# 95% credible interval for lambdanew (equal-tailed)
quantile(samps, c(0.025, 0.975))
```

```
##      2.5%     97.5%
## 0.1706138 3.4435471
```
