Group tasks

Exercise 3 (Normal example in JAGS - 3 points)

Run the code provided in the file `05normal_example_JAGS.R` and explore the interfaces in R to JAGS. Comment your findings.

Exercise 4 (Logistic regression in JAGS - 10 points)

Extend the code available in the file `05normal_example_JAGS.R` to deal with the logistic regression example for mice data from Collett (2003, p.71) provided in Table 1, ws05.

Solution:

Classical logistic regression

```
remove(list=ls())
######################
## Classical analysis
######################
# the covariate values (dose)
x_original <- c(0.0028, 0.0028, 0.0056, 0.0112, 0.0225, 0.0450)
# center covariate values
x <- x_original - mean(x_original)
# number of mice deaths
y <- c(26, 9, 21, 9, 6, 1)
# total number of mice
n <- c(28, 12, 40, 40, 40, 40)

## Fit a classical generalized linear model (logistic regression for binomial data)
m <- glm(formula = cbind(y, n - y) ~ x, family = binomial)
summary(m)

##
## Call:
## glm(formula = cbind(y, n - y) ~ x, family = binomial)
##
## Deviance Residuals:
##       1        2        3        4        5        6
##  3.0784   0.4474  -0.9319  -2.2893   0.7546   1.3344
```

```
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -0.9800     0.2399  -4.085 4.41e-05 ***
## x           -146.6927    26.3630  -5.564 2.63e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 92.287  on 5  degrees of freedom
## Residual deviance: 18.136  on 4  degrees of freedom
## AIC: 40.805
##
## Number of Fisher Scoring iterations: 5
```

```
vcov(m)
```

```
##             (Intercept)          x
## (Intercept)  0.05756791   4.263589
## x            4.26358925 695.005759
```

As a result of classical analysis the estimated intercept and its standard error are -0.98 (0.24) and the estimated slope and its standard error are -146.693 (26.363).

Bayesian logistic regression in `rjags`

```
######################
## JAGS analysis, rjags interface
######################
set.seed(44566)
library(rjags)

#create Data
data <- matrix(NA, nrow=6, ncol=3)
colnames(data) <- list("x", "y", "n")
data[,1] <- x
data[,2] <- y
data[,3] <- n

mice.data <- list(y=data[,2], x=data[,1], n=data[,3], N = nrow(data))

#define parameters
mice.params <- c("a", "b")

# define initial values for 2 chains
```

```
inits1 <- list(a=rnorm(1), b=rnorm(1),
               .RNG.name="base::Super-Duper", .RNG.seed=1)
inits2 <- list(a=rnorm(1), b=rnorm(1),
               .RNG.name="base::Wichmann-Hill", .RNG.seed=2)
mice.inits <- list(inits1, inits2)

modelString = "
# open quote for modelString
model{
  for(i in 1:N){
    y[i] ~ dbin(p[i],n[i])
    logit(p[i]) <- a+b*x[i]
  }
  a ~ dnorm(0,0.0001)
  b ~ dnorm(0,0.0001)
}" # close quote for modelString

writeLines(modelString, con="05_mice_logistic_regression_JAGS.txt") # write to a file

# model initiation
mice.rjags <- jags.model(
  file = "05_mice_logistic_regression_JAGS.txt",
  data = mice.data,
  inits = mice.inits,
  n.chains = 2,
  n.adapt = 4000)

## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 6
##    Unobserved stochastic nodes: 2
##    Total graph size: 38
##
## Initializing model

N.iter <- 10000
N.thin <- 1
N.burnin <- 4000
# burn-in
update(mice.rjags, n.iter = N.burnin)

# sampling/monitoring
fit.rjags.coda.mice <- coda.samples(
```

```
  model = mice.rjags,
  variable.names = mice.params,
  n.iter = N.iter,
  thin = N.thin)
# store samples for each parameter from the two chains into separate vectors
a.sample <- fit.rjags.coda.mice[,"a"]
b.sample <- fit.rjags.coda.mice[,"b"]

summary(fit.rjags.coda.mice)

##
## Iterations = 8001:18000
## Thinning interval = 1
## Number of chains = 2
## Sample size per chain = 10000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##        Mean      SD Naive SE Time-series SE
## a   -0.9612  0.2314 0.001636       0.003209
## b -142.3448 24.7462 0.174982       0.349327
##
## 2. Quantiles for each variable:
##
##       2.5%      25%       50%       75%    97.5%
## a   -1.432   -1.116   -0.9532   -0.8015  -0.5289
## b -193.673 -158.294 -141.3912 -125.0619 -96.7406
```

```
par(mfrow=c(1,2))
acf(as.matrix(a.sample), main = "a", ci = "")
acf(as.matrix(b.sample), main = "b", ci = "")
```

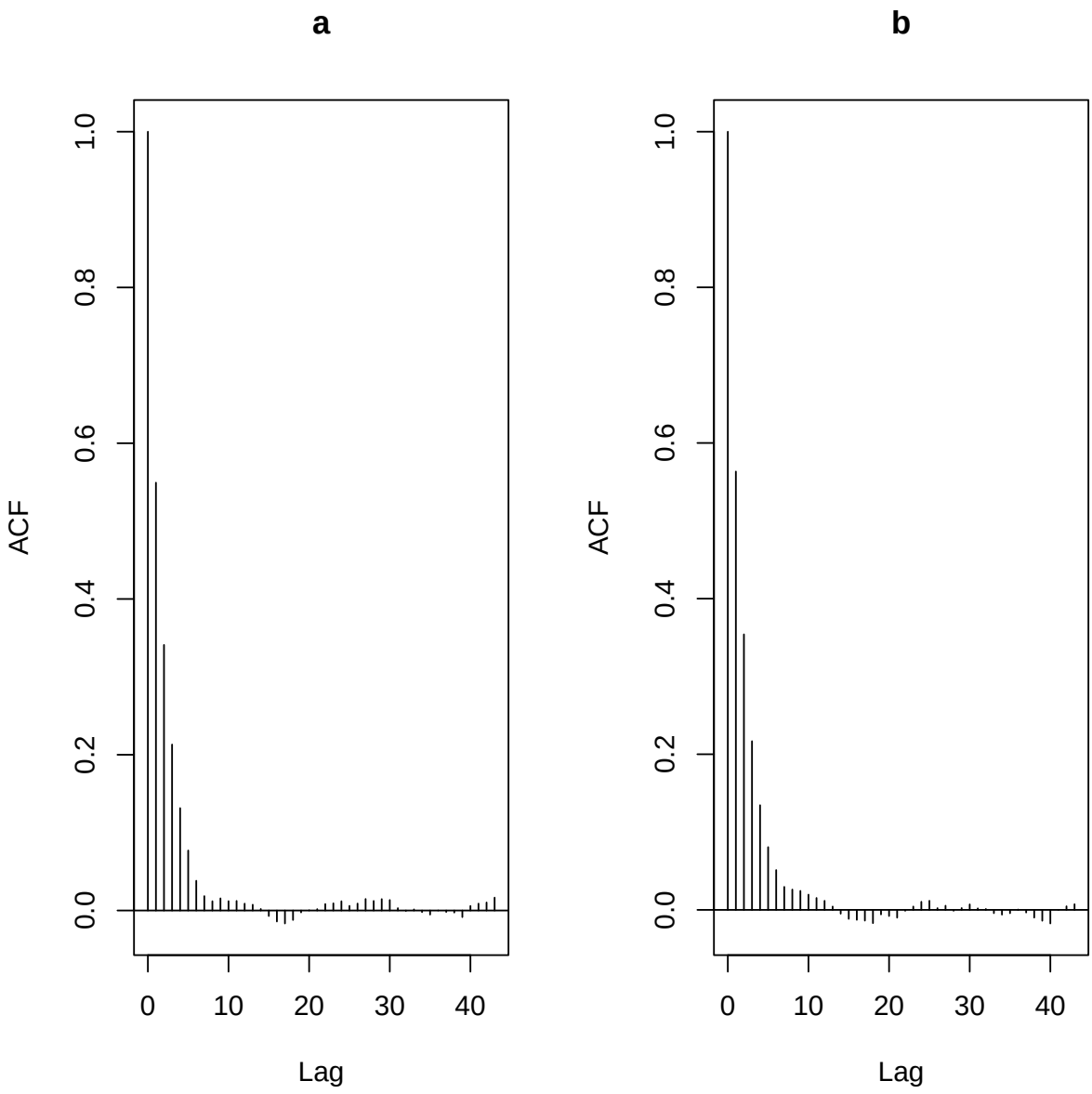


Figure 1: ACF of samples for a and b.

Bayesian analysis provides mean posterior estimates of intercept -0.961 and the slope -142.345. Note that different prior parameter assumptions would lead to different estimates. ACF plots in Figure 1 show high dependence.

```
plot(fit.rjags.coda.mice)
```

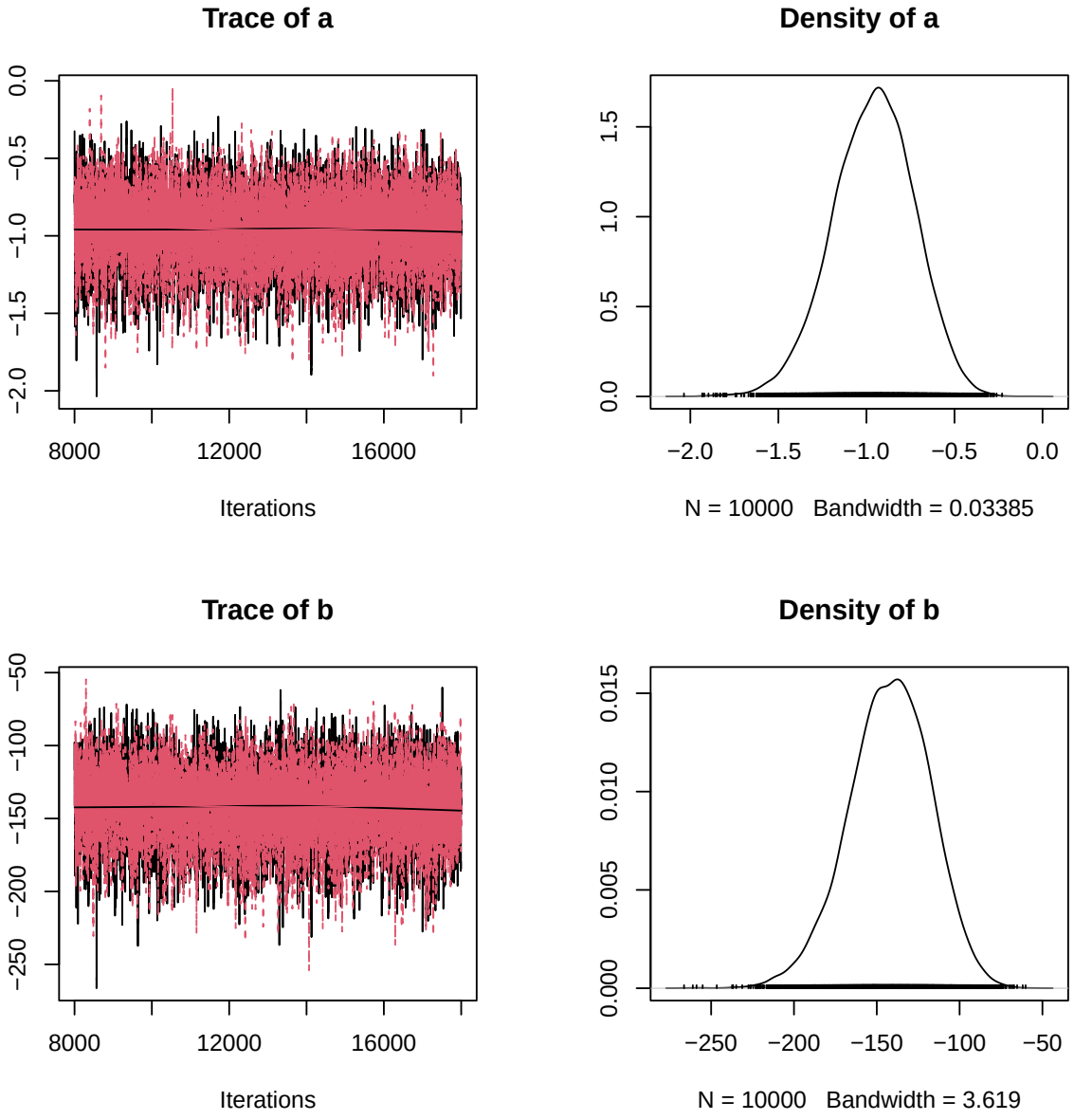


Figure 2: Traceplots and the density plots for the parameters a and b from the Bayesian logistic regression model fitted in JAGS for the mice data.

Exercise 5 (CODA for the logistic regression in JAGS - 10 points)

Run the logistic regression in JAGS for mice data from Collett (2003, p.71) provided in Table 1, which you developed in Exercise 4 above.

1. Rank plots are histograms of ranked posterior draws (ranked over all chains) plotted separately for each chain, which were recommended by Vehtari, Gelman, Simpson, Carpenter, and Bürkner (2021) instead of traceplots. Implement, apply rank plots, and interpret them.

   Solution:

```
rank_plot <- function(jags.object, param.name){
  #plot the rank plots for each chain separately ranked over all chains
  ##jags.object:
  ##param.name: the name of the parameter for which
    #the rank plots have to be plotted
  library(MASS) #to get the truehist()

  nchains <- length(jags.object) #number of chains
  jags.object_m <- as.matrix(jags.object)
  tl <- dim(jags.object_m)[1] #total length
  cl <- tl/nchains #length of one chain

  #rank the samples
  ranked_param <- rank(jags.object_m[,param.name])

  #plotting of rank plots
  if(nchains == 1) {
    par(mfrow = c(1,1), pty = "s")
  }
  if(nchains == 2) {
    par(mfrow = c(1,2), pty = "s")
  }
  if(nchains > 2) {
    par(mfrow = c(2,2), pty = "s")
  }

  for(i in 1:nchains) {
    truehist(ranked_param[(i-1)*cl+c(1:cl)],
             main = paste(paste("chain", i, "," , sep = ""), param.name),
             col = "skyblue3", yaxt = 'n', xlab = "", ylab = "")
  }
}
```

Figures 3 and 4 show rank plots which indicate uniformity for all the chains. This shows that the chains have converged and are mixing well.

```
rank_plot(fit.rjags.coda.mice, param.name = "a")
```

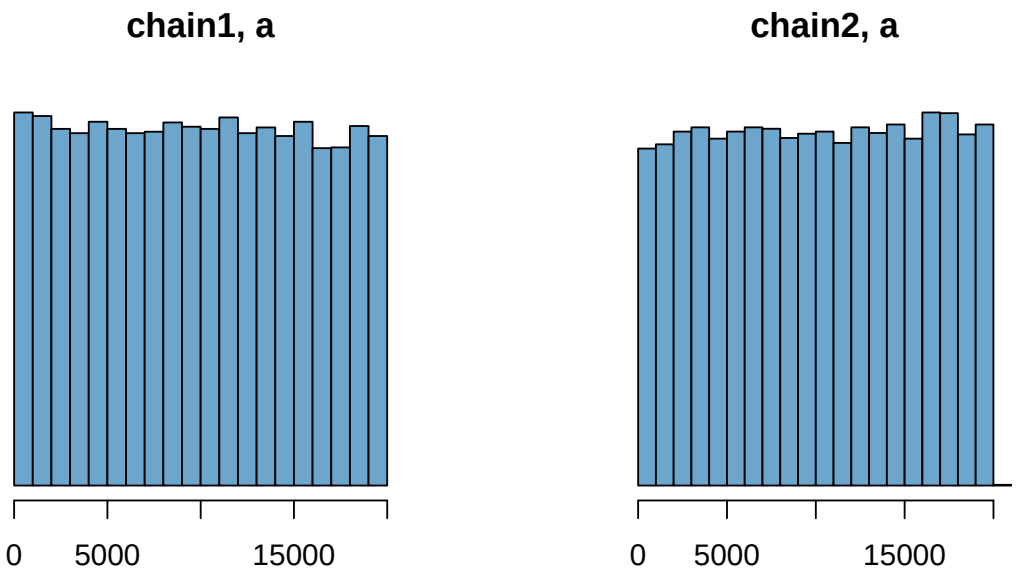


Figure 3: Rank plots for the JAGS samples of a in mice data example.

```
rank_plot(fit.rjags.coda.mice, param.name = "b")
```

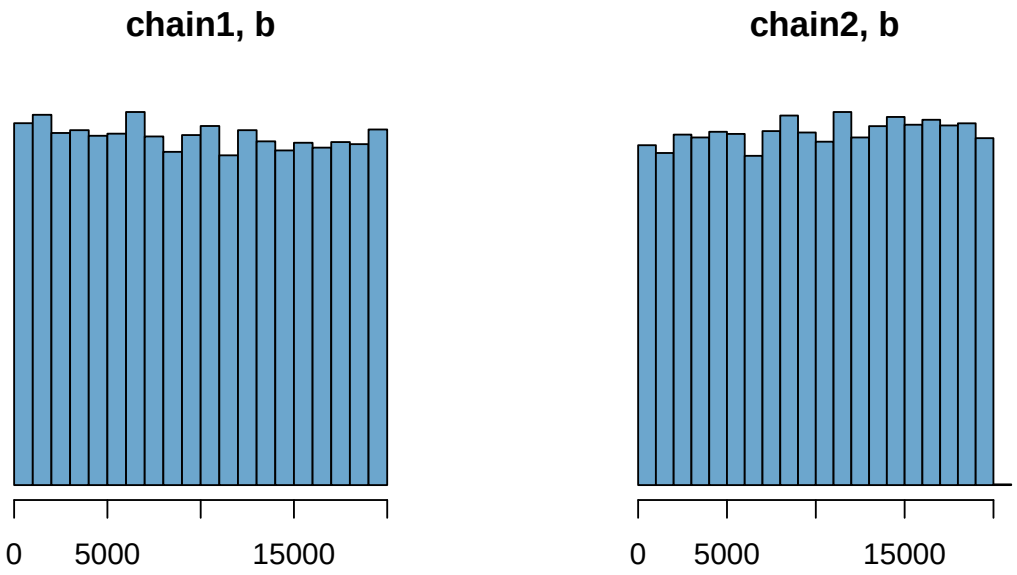


Figure 4: Rank plots for the JAGS samples of b in mice data example.

2. Run the following formal convergence diagnostics:

   Solution:

   - `heidel.diag`

```
heidel.diag(fit.rjags.coda.mice)
```

```
## [[1]]
##
##   Stationarity start     p-value
##   test         iteration
## a passed       1         0.688
## b passed       1         0.442
##
##   Halfwidth Mean     Halfwidth
##   test
## a passed      -0.964 0.009
## b passed    -142.693 0.979
##
## [[2]]
##
##   Stationarity start     p-value
##   test         iteration
## a passed       1         0.161
## b passed       1         0.160
##
##   Halfwidth Mean     Halfwidth
##   test
## a passed      -0.958 0.00879
## b passed    -141.996 0.95760
```

`heidel.diag` checks the convergence to stationarity and shows that the chains passed the stationarity and halfwidth tests.

- `raftery.diag`

```
raftery.diag(fit.rjags.coda.mice)
```

```
## [[1]]
##
## Quantile (q) = 0.025
## Accuracy (r) = +/- 0.005
## Probability (s) = 0.95
##
##    Burn-in  Total Lower bound  Dependence
##    (M)      (N)   (Nmin)       factor (I)
##  a 10       10782 3746         2.88
##  b 10       10946 3746         2.92
##
##
## [[2]]
```

```
##
## Quantile (q) = 0.025
## Accuracy (r) = +/- 0.005
## Probability (s) = 0.95
##
##    Burn-in  Total Lower bound  Dependence
##    (M)      (N)   (Nmin)       factor (I)
##  a 10       12690 3746         3.39
##  b 12       13200 3746         3.52
```

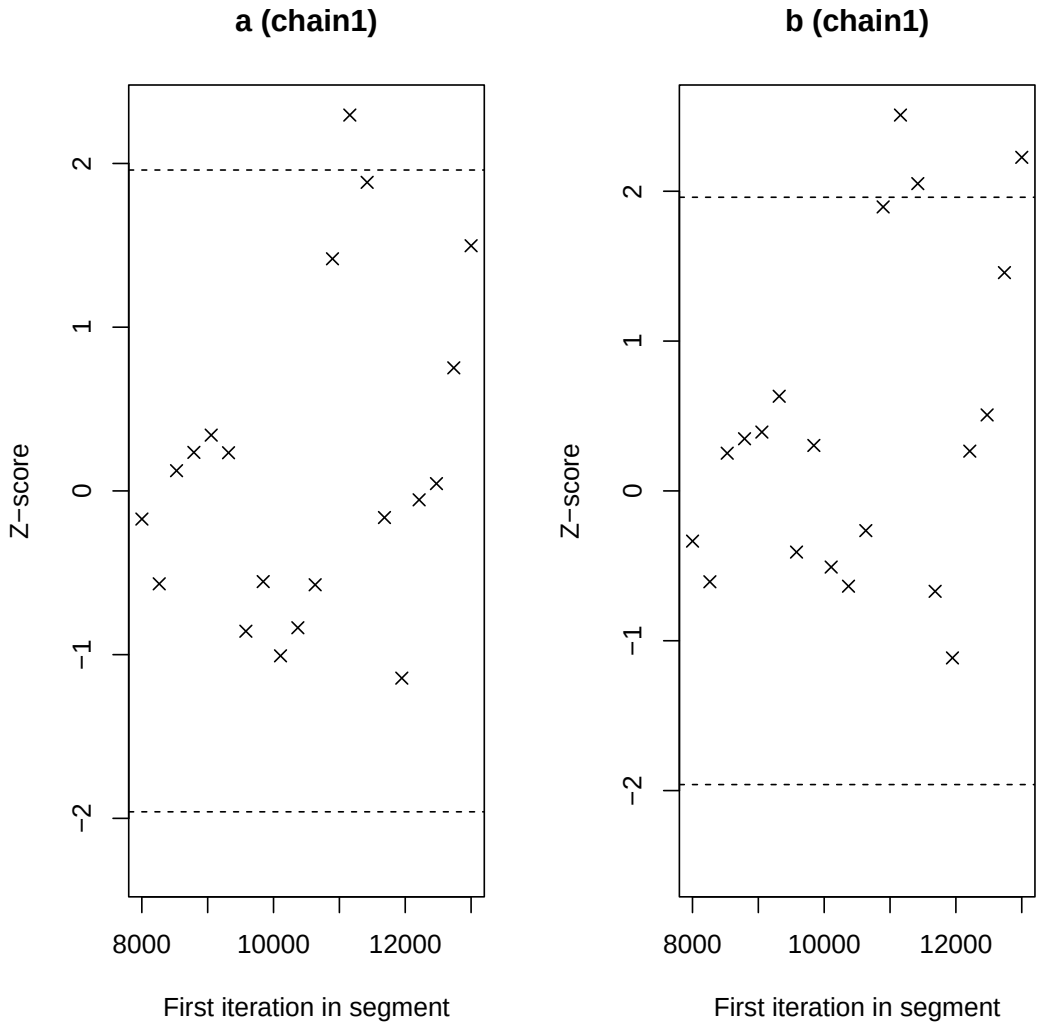
`raftery.diag` checks the convergence to ergodic average and suggests a thinning equal to 4 (maximum of all dependence factors I).

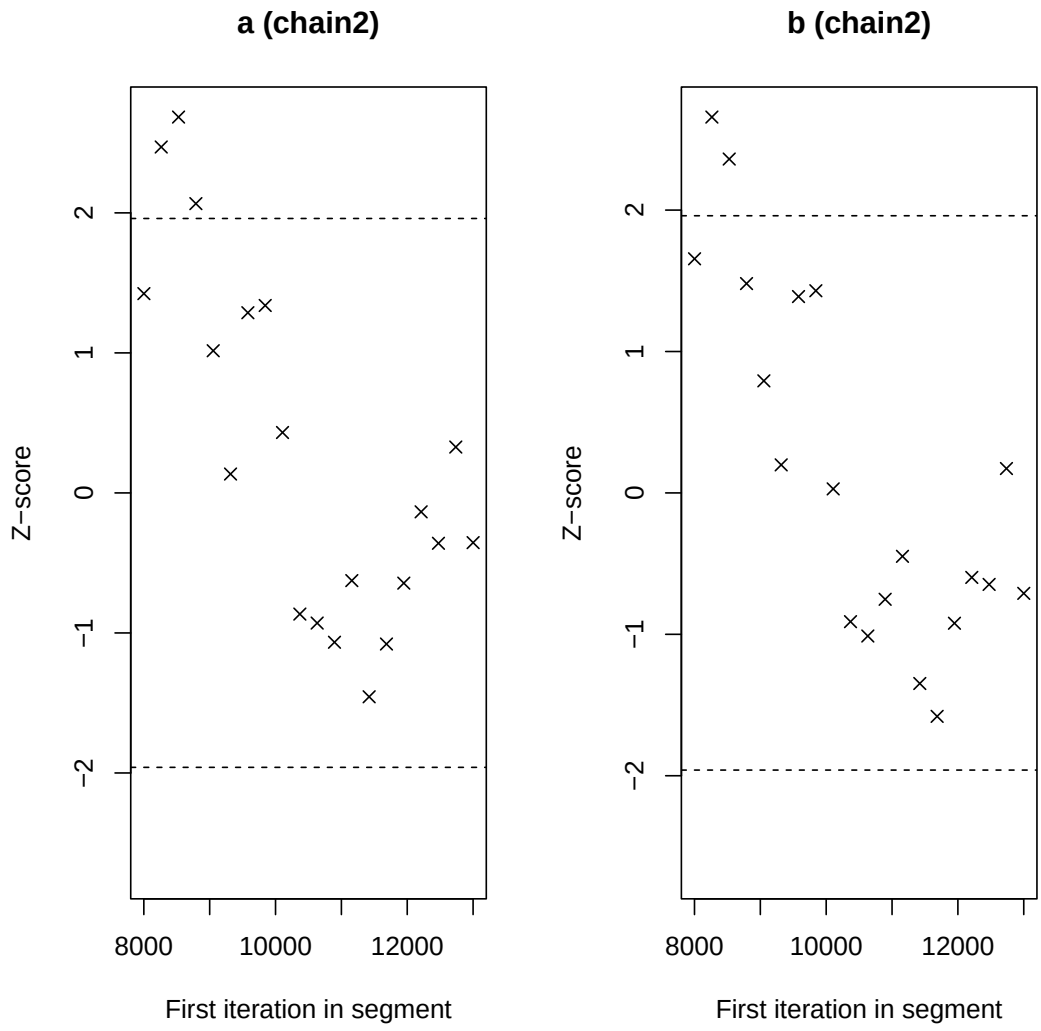
- `geweke.diag` and `geweke.plot` check the convergence to stationarity

```
geweke.diag(fit.rjags.coda.mice)
```

```
## [[1]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##       a       b
## -0.1719 -0.3351
##
##
## [[2]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##     a     b
## 1.423 1.656
```

```
geweke.plot(fit.rjags.coda.mice)
```


a (chain1)
b (chain1)
Z−score
2
1
0
−1
−2
8000
10000
12000
First iteration in segment



`geweke.diag` shows that the z-scores are not larger than 1.96 and `geweke.plot` shows that only a few values fall out of the bound (-1.96, 1.96).

- `gelman.diag` and `gelman.plot` deal with the convergence to ergodic average

```
gelman.diag(fit.rjags.coda.mice)

## Potential scale reduction factors:
##
##   Point est. Upper C.I.
## a          1          1
## b          1          1
##
## Multivariate psrf
##
## 1
```

Figure 5 demonstrates that the shrink factor ($\widehat{R}$) decreases to 1 with increasing

number of iterations.

- `stable.GR` from package `stableGR` (Vats and Knudson (2020)).

```
library(stableGR)
stable.GR(fit.rjags.coda.mice)

## $psrf
## [1] 1.000062 1.000062
##
## $mpsrf
## [1] 1.000058
##
## $means
##             a             b
##   -0.9612116 -142.3464775
##
## $n.eff
## [1] 9251.218
##
## $blather
## [1] FALSE
```

`psrf` is the "potential scale reduction factor" which is the $\widehat{R}$ and these are smaller than 1.01.

3. Modify the original `n.burnin`, `n.iter` and `n.thin` parameters as suggested by the convergence diagnostics and re-run the MCMC simulation. Do the new results differ much from those in the first run?

```
N.iter <- 15000
N.thin <- 4
N.burnin <- 4000
# burn-in
update(mice.rjags, n.iter = N.burnin)

# sampling/monitoring
fit.rjags.coda.mice <- coda.samples(
  model = mice.rjags,
  variable.names = mice.params,
  n.iter = N.iter,
  thin = N.thin)
summary(fit.rjags.coda.mice)

##
## Iterations = 22004:37000
```

```
gelman.plot(fit.rjags.coda.mice)
```

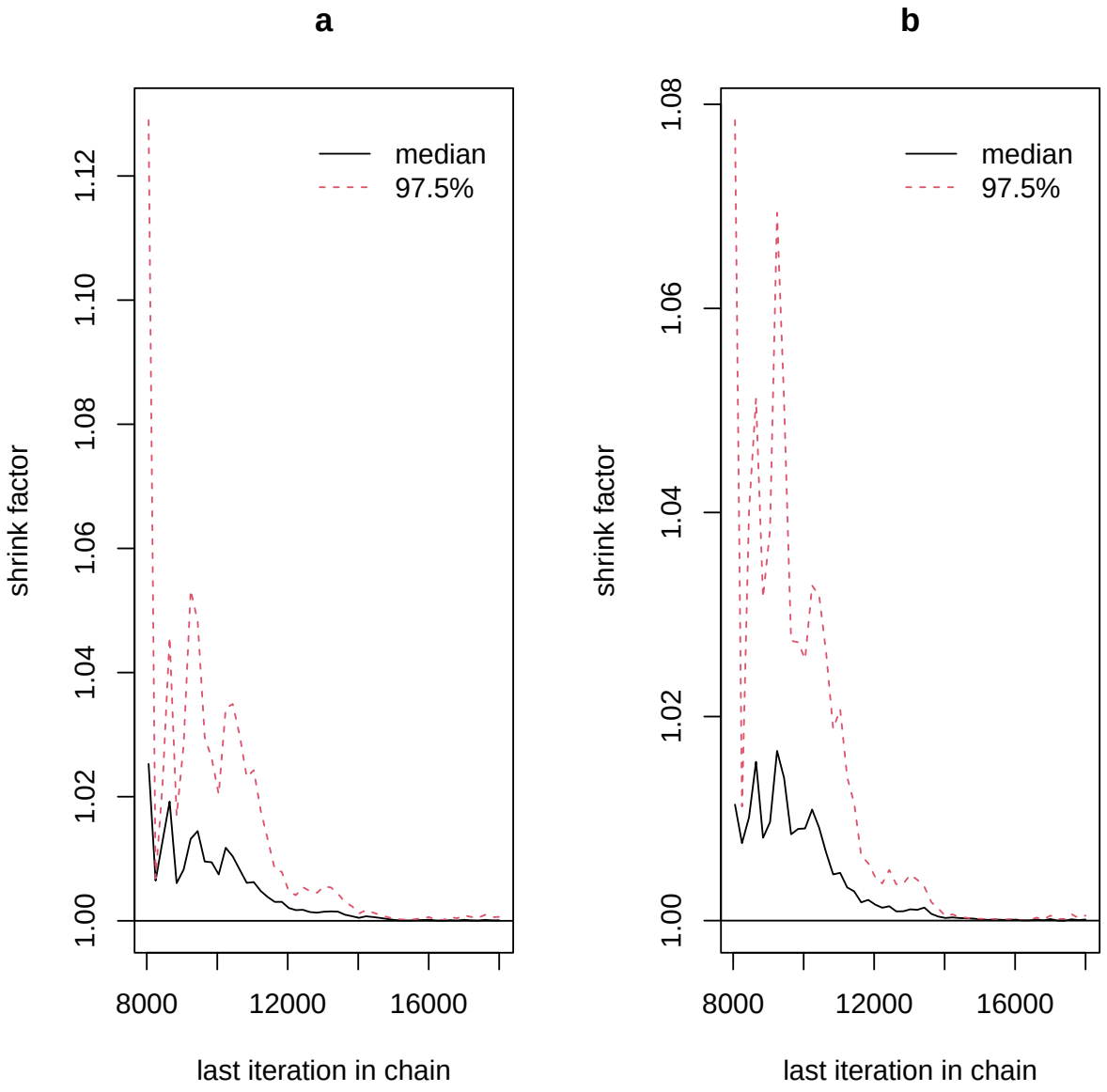


Figure 5: Gelman-Rubin-Brooks plot.

```
## Thinning interval = 4
## Number of chains = 2
## Sample size per chain = 3750
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##        Mean      SD Naive SE Time-series SE
## a   -0.9551  0.2276 0.002628       0.003165
## b -141.3846 24.1005 0.278288       0.318112
##
## 2. Quantiles for each variable:
##
##       2.5%      25%       50%       75%    97.5%
## a   -1.425   -1.105   -0.9464   -0.7978  -0.5373
## b -191.628 -156.790 -140.3102 -124.7471 -96.6151
```

```
par(mfrow = c(1,2))
acf(as.matrix(fit.rjags.coda.mice[,"a"]), main = "a", ci = "")
acf(as.matrix(fit.rjags.coda.mice[,"b"]), main = "a", ci = "")
```

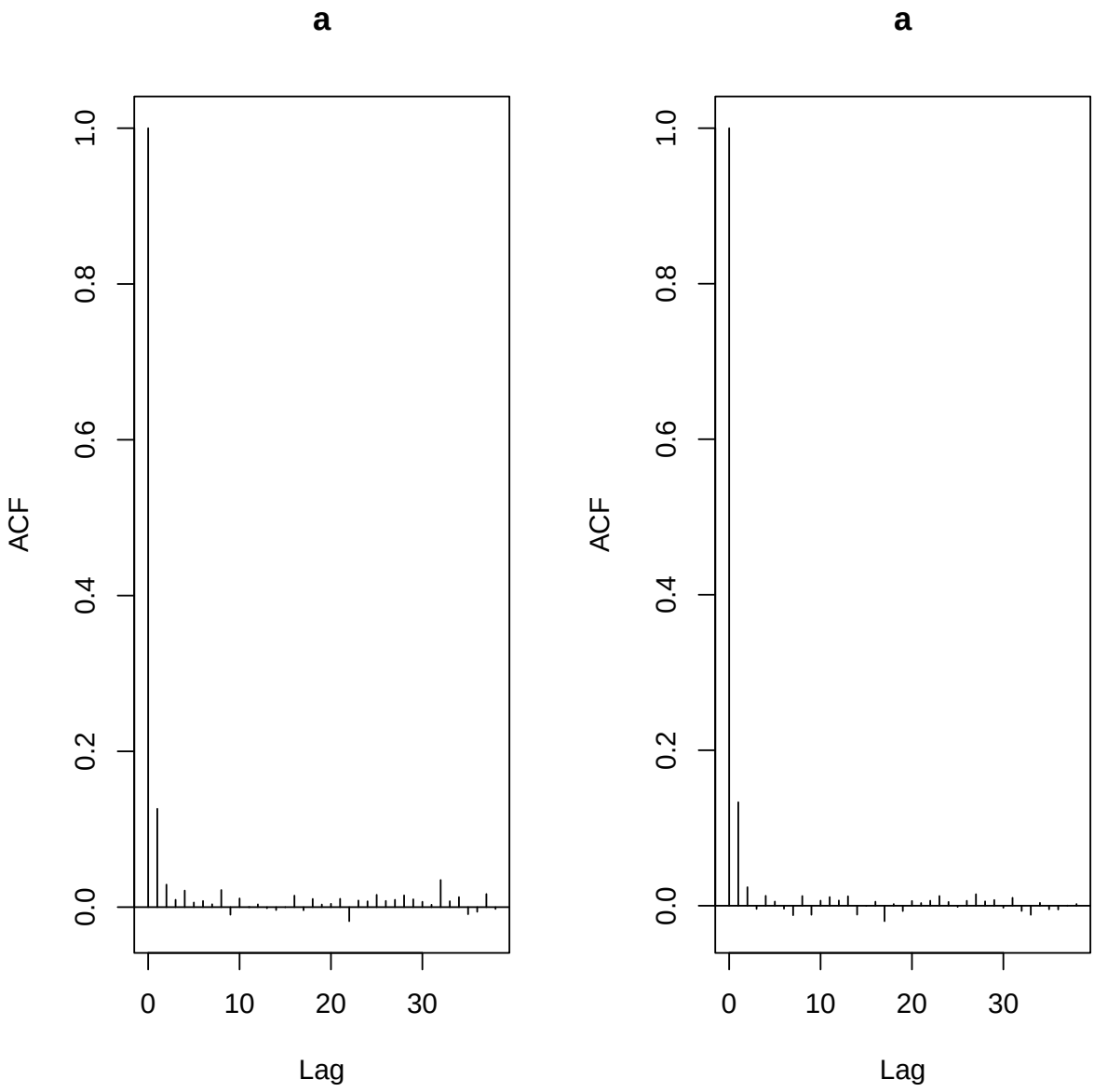


Figure 6: ACF of samples for a and b with new tuning parameters.

ACF plots in Figure 6 improve quite a lot with larger number of thinning and the number of iterations as compared to Figure 1.

Exercise 6 (ESS - 7 points)

Run the code from the previous exercise with mice data with only one chain monitoring beta under the following two conditions:

Solution:

1. After an adaptation phase of 1000 and a burn-in of 4000 draw a sample of 1000 observations in one chain with thinning set to 1.

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 6
##    Unobserved stochastic nodes: 2
##    Total graph size: 38
##
## Initializing model
##
## Iterations = 5001:6000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 1000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##         Mean     SD Naive SE Time-series SE
## a   -0.9585  0.238 0.007525        0.01696
## b -142.3792 24.395 0.771422        1.57742
##
## 2. Quantiles for each variable:
##
##       2.5%     25%       50%       75%    97.5%
## a   -1.447   -1.11   -0.9531   -0.7873  -0.5162
## b -190.664 -158.64 -141.4738 -125.5740 -97.4291
```

```
par(mfrow = c(1,2))
acf(as.matrix(fit.rjags.coda.mice.1[,"a"]), main = "a", ci = "")
acf(as.matrix(fit.rjags.coda.mice.1[,"b"]), main = "a", ci = "")
```

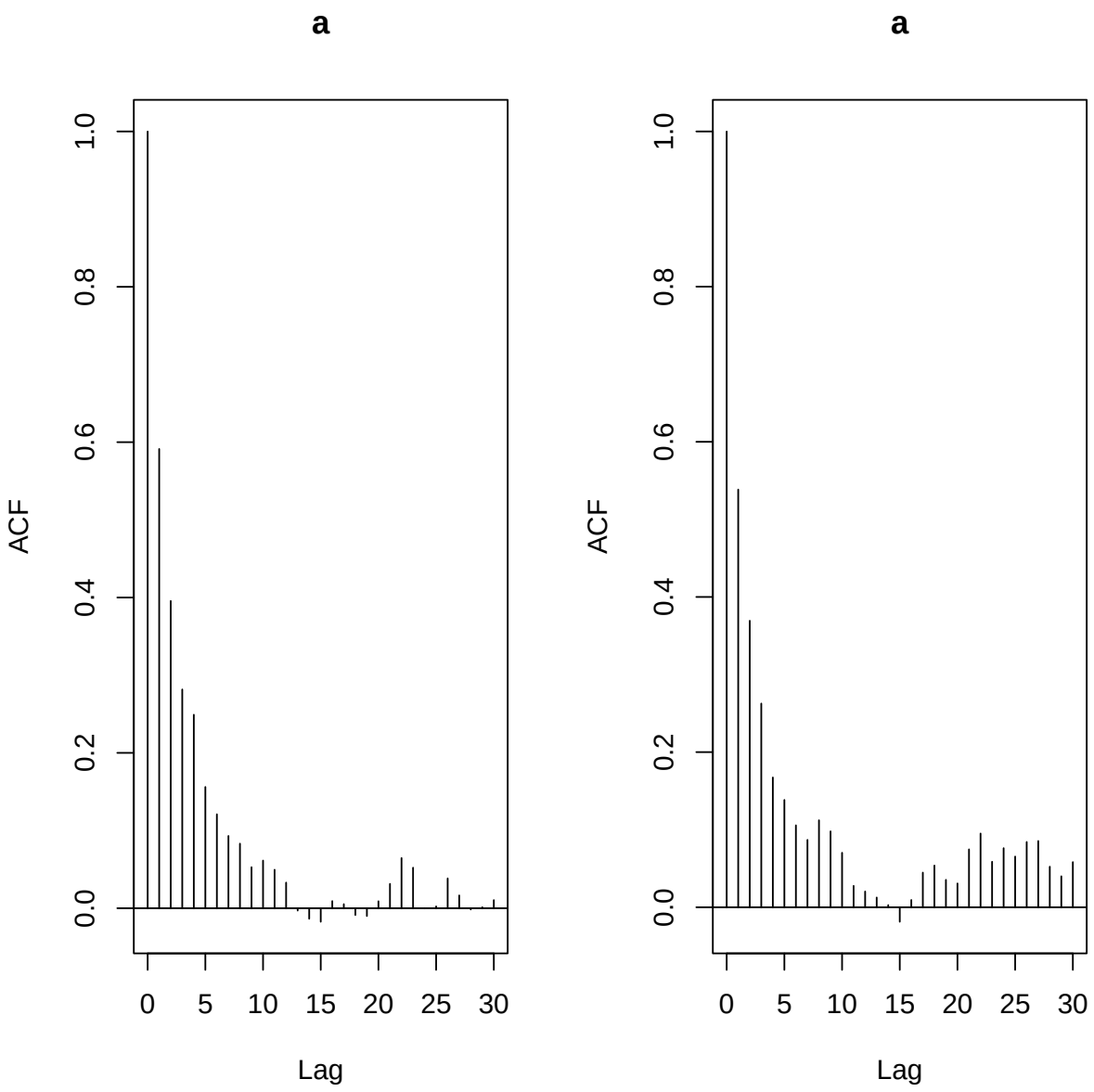


Figure 7: ACF of samples for a and b with thinning equal to 1.

2. After an adaptation phase of 1000 and a burn-in of 4000 draw a sample of 10000 observations in one chain with thinning set to 10.

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 6
##    Unobserved stochastic nodes: 2
##    Total graph size: 38
##
## Initializing model
##
## Iterations = 5010:15000
## Thinning interval = 10
## Number of chains = 1
## Sample size per chain = 1000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##        Mean      SD Naive SE Time-series SE
## a   -0.9605  0.2309 0.007302       0.007302
## b -144.2164 24.4952 0.774605       0.774605
##
## 2. Quantiles for each variable:
##
##       2.5%      25%       50%       75%    97.5%
## a   -1.412   -1.114   -0.9447   -0.8111  -0.5253
## b -197.332 -160.305 -143.0586 -126.9475 -98.6173
```

```
par(mfrow = c(1,2))
acf(as.matrix(fit.rjags.coda.mice.2[,"a"]), main = "a", ci = "")
acf(as.matrix(fit.rjags.coda.mice.2[,"b"]), main = "a", ci = "")
```

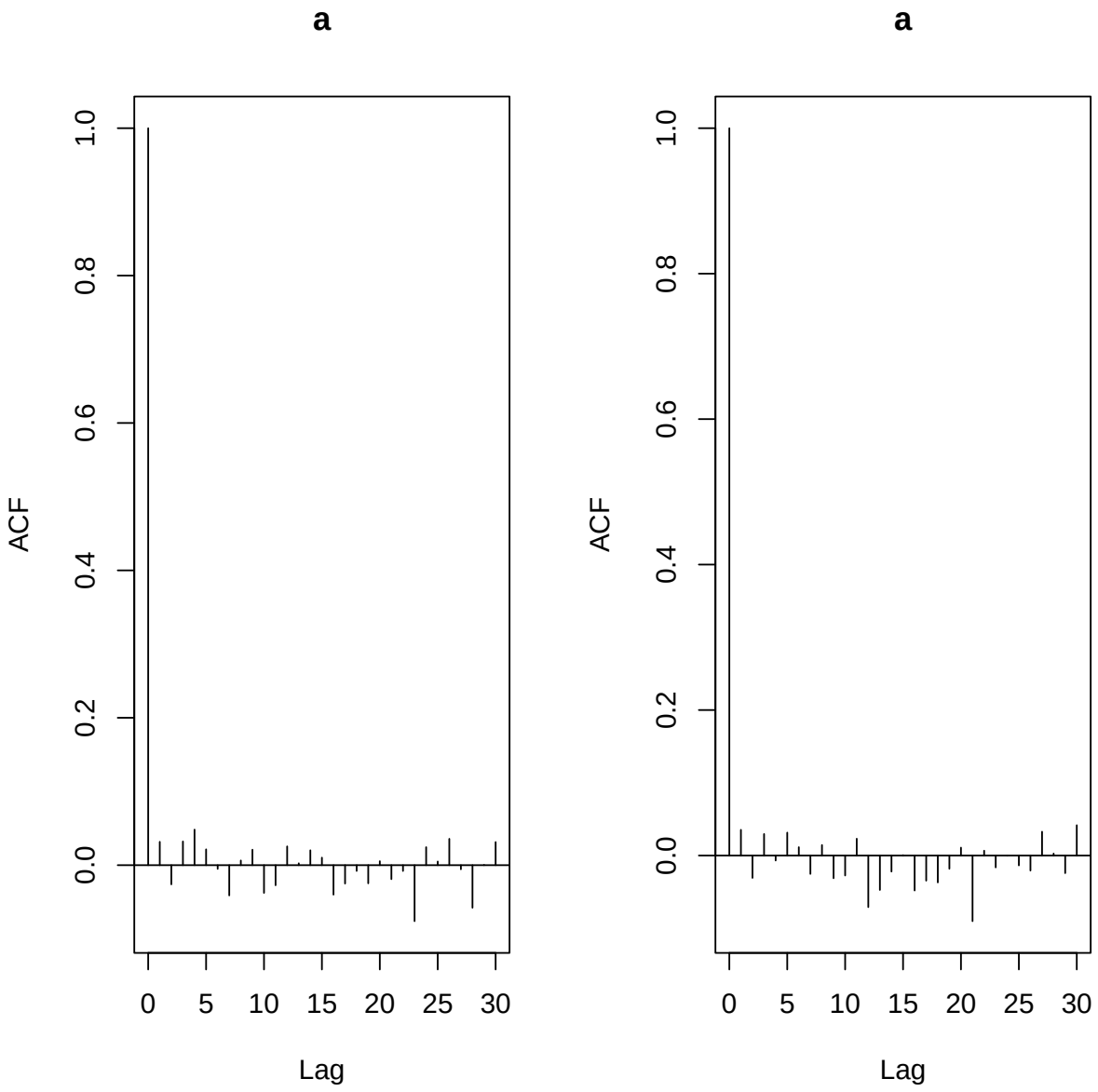


Figure 8: ACF of samples for a and b with thinning equal to 10.

Figure 7 shows larger autocorrelation than Figure 8.

- ESS will be larger for the second case because there is less autocorrelation between MCMC samples due to thinning.

- ```
  (ess1 <- ess(as.mcmc(fit.rjags.coda.mice.1[, "b"]), N.iter1))

  ## [1] 230.5437

  (ess2 <- ess(as.mcmc(fit.rjags.coda.mice.2[, "b"]), N.iter2))

  ## [1] 9361.881

  (effectiveSize(fit.rjags.coda.mice.1[, "b"]))

  ##     var1
  ## 239.1615

  (effectiveSize(fit.rjags.coda.mice.2[, "b"]))

  ## var1
  ## 1000

  (n.eff(as.matrix(fit.rjags.coda.mice.1[, "b"])))

  ## $n.eff
  ##       se
  ## 196.3447
  ##
  ## $converged
  ## [1] FALSE
  ##
  ## $n.target
  ##    se
  ## 31303

  (n.eff(as.matrix(fit.rjags.coda.mice.2[, "b"])))

  ## $n.eff
  ## var1
  ## 1000
  ##
  ## $converged
  ## [1] FALSE
  ##
  ## $n.target
  ## var1
  ## 6147
  ```

  All of the three methods confirm that the effective sample size is larger in the second case. In addition, `n.eff` from the R package `stableGR` informs about the convergence status.
  ```