Exercise 4

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.

Compare the output provided by the classic logistic regression and the Bayesian inference. What are the differences?

```
remove(list=ls())
rm(.Random.seed, envir=globalenv())
set.seed(44566)

# Load data
y <- c(26,9,21,9,6,1)
n <- c(28,12,40,40,40,40)
x <- c(0.0028, 0.0028, 0.0056, 0.0112, 0.0225, 0.0450)

# Priors
mu0 <- 0
prec_0 <- 1.0E-04
```

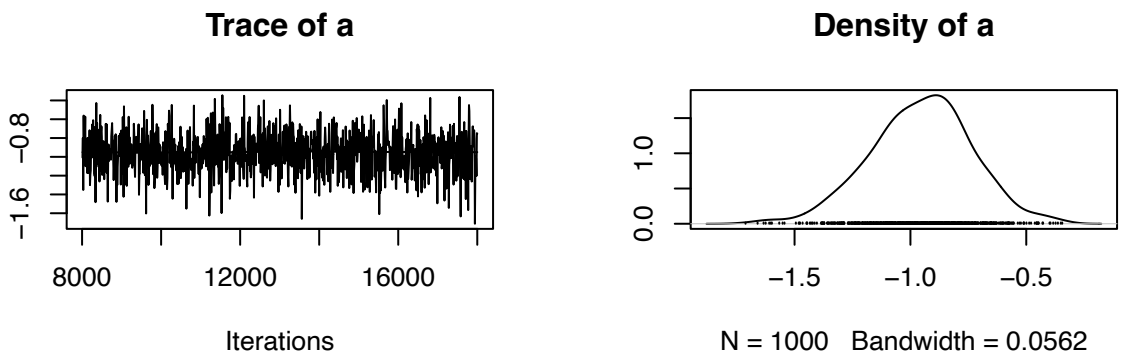
INLA

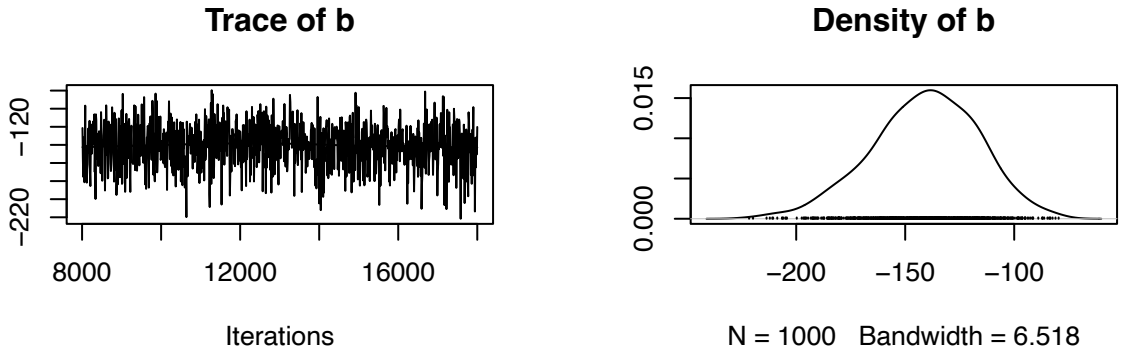
```
formula <- y ~ 1 + x

inla.output <- inla(formula,
                    data = data.frame(y = y, x = x-mean(x)),
                    family="binomial",
                    Ntrials = n,
                    control.fixed = list(mean = list(intercept = 0, x = 0),
                                         prec = list(intercept = prec_0, x = prec_0)),
                    control.predictor = list(compute = T, link= 1))
```

JAGS: one chain

```
mice_data <- list(Y = y, x = x - mean(x), n = n)

inits_1chain <- list(a = 0, b = 0, .RNG.name = "base::Wichmann-Hill", .RNG.seed = 123456)

modelString = " # open quote for modelString
model {
# likelihood
for(i in 1:length(Y)) {
  Y[i] ~ dbin(p[i], n[i])
  logit(p[i]) <- a+b*x[i]
}
# priors
a ~ dnorm(0, 1.0E-4)
b ~ dnorm(0, 1.0E-4)
}
" # close quote for modelString
```

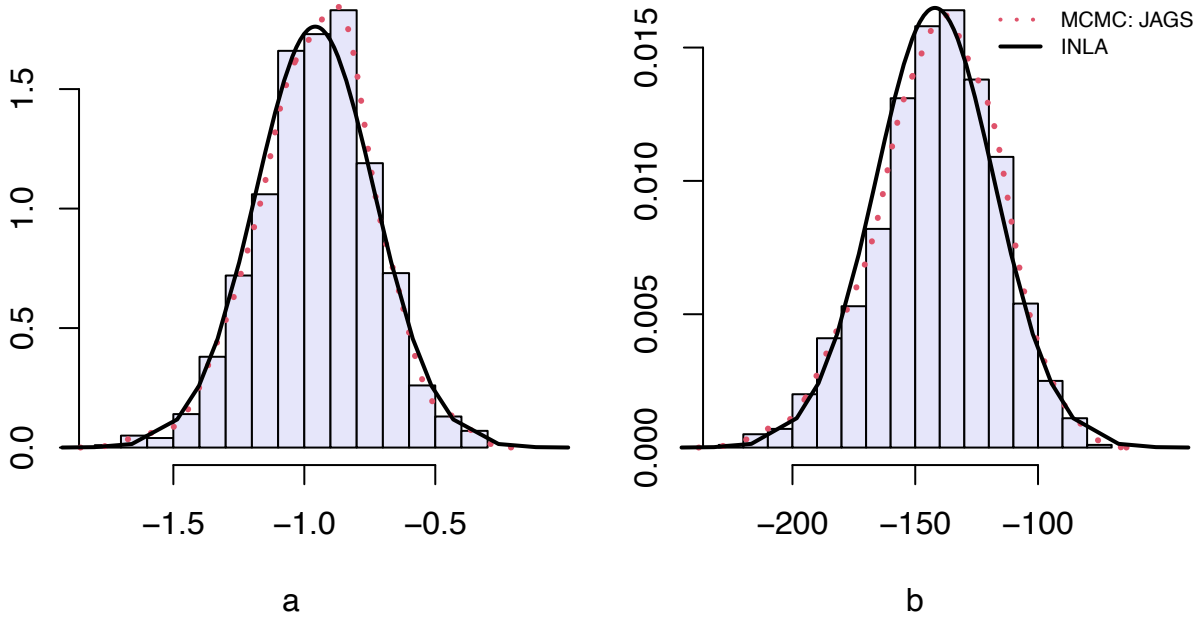
```r
writeLines(modelString, con="MiceModel.txt") # write to a file

# model initiation
model.jags <- jags.model(
  file = "MiceModel.txt",
  data = mice_data,
  inits = inits_1chain,
  n.chains = 1,
  n.adapt = 4000
)

# burn-in
update(model.jags, n.iter = 4000)

# sampling
fit.jags.coda <- coda.samples(
  model = model.jags,
  variable.names = c("a", "b"),
  n.iter = 10000,
  thin = 10
)

#summary(fit.jags.coda)
plot(fit.jags.coda)
```






```r
# store samples for each parameter from the chain into separate objects
m.fit.jags.coda <- as.matrix(fit.jags.coda)
a.sim <- m.fit.jags.coda[,"a"]
b.sim <- m.fit.jags.coda[,"b"]
```

```r
par(mfrow=c(1,2), mar=c(6.1, 3.1, 4.1, 2.1), xpd=TRUE)
# plot for intercept
truehist(a.sim, prob=TRUE, col="lavender", xlab=expression(a))
lines(density(a.sim),lty=3,lwd=3, col=2)
lines(inla.output$marginals.fixed$"(Intercept)",lwd=2)

# plot for slope
truehist(b.sim, prob=TRUE, col="lavender", xlab=expression(b))
lines(density(b.sim),lty=3,lwd=3, col=2)
lines(inla.output$marginals.fixed$"x",lwd=2)
legend("topright",c("MCMC: JAGS","INLA"), inset=c(-0.18,0),
       lty=c(3,1),lwd=c(2,2),col=c(2,1),cex=0.7,bty="n")
```

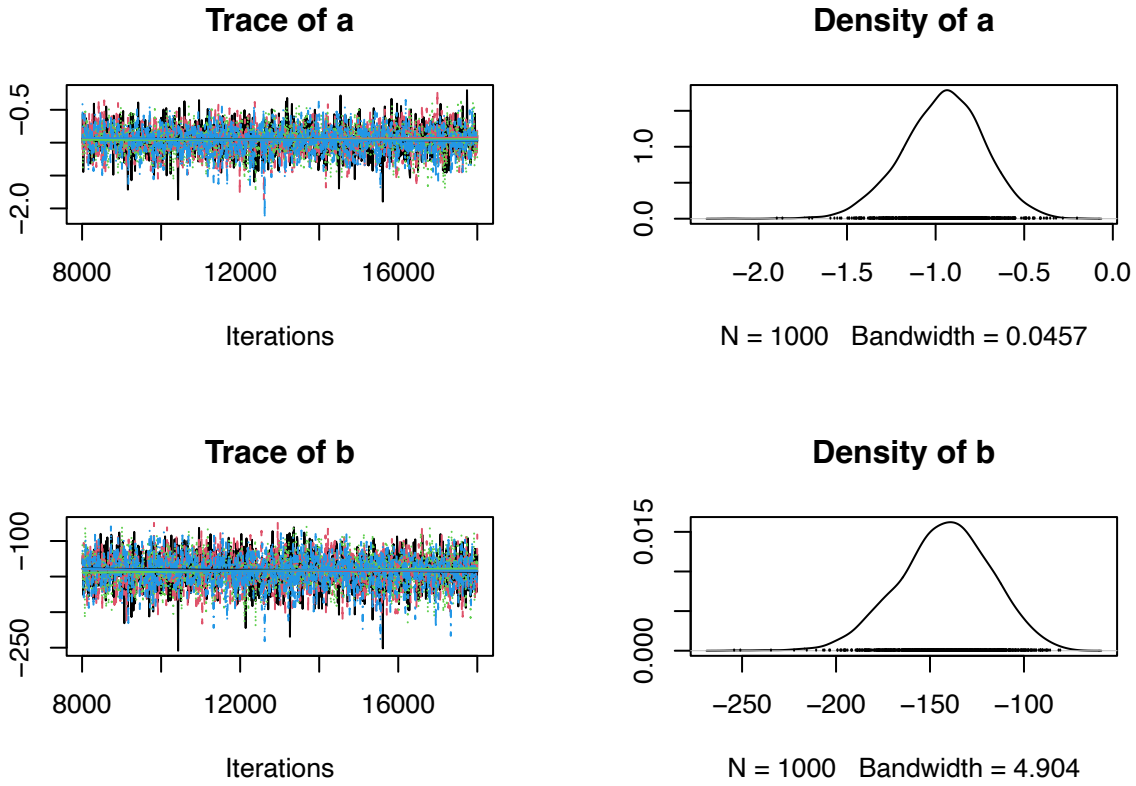



```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 6
##    Unobserved stochastic nodes: 2
##    Total graph size: 37
##
## Initializing model
```

JAGS: several chains

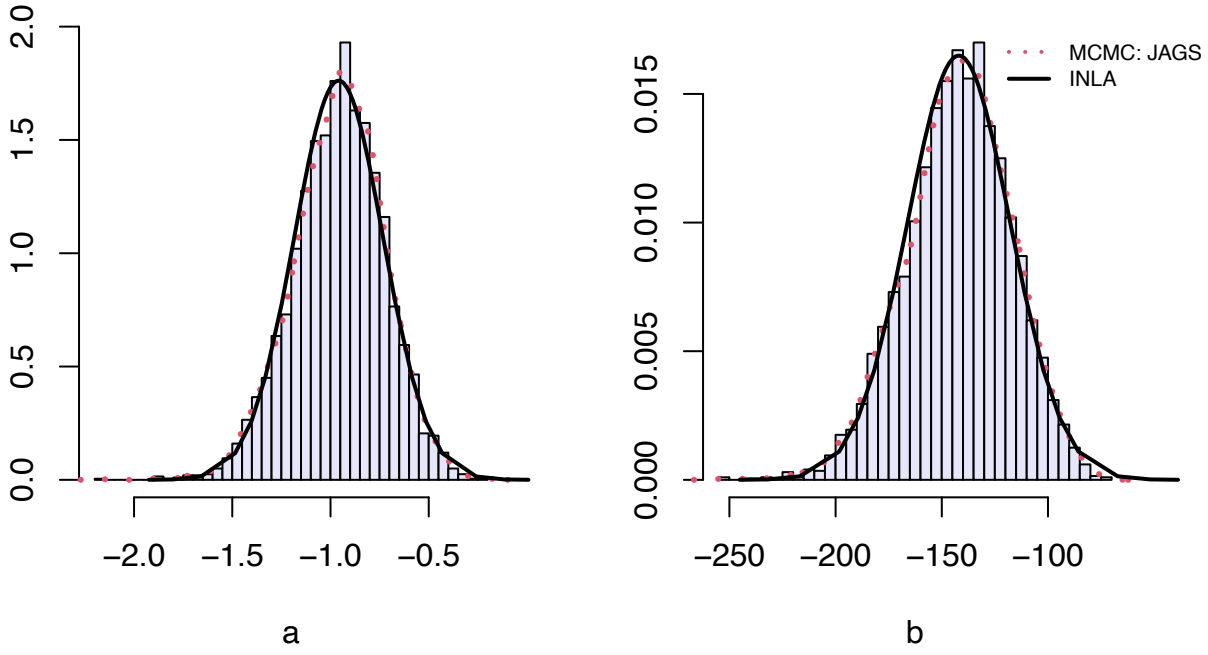
```r
inits_4chain <- list(
  list(a=0, b = 0, .RNG.name = "base::Super-Duper", .RNG.seed = 12345),
  list(a=0, b = 0, .RNG.name = "base::Super-Duper", .RNG.seed = 123456),
  list(a=0, b = 0, .RNG.name = "base::Super-Duper", .RNG.seed = 1234567),
  list(a=0, b = 0, .RNG.name = "base::Super-Duper", .RNG.seed = 12345678))

# model initialisation
model.jags <- jags.model(
```

```
  file = "MiceModel.txt",
  data = mice_data,
  inits = inits_4chain,
  n.chains = 4,
  n.adapt = 4000
)

# burn-in
update(model.jags, n.iter = 4000)

# sampling/monitoring
fit.jags.coda <- coda.samples(
  model = model.jags,
  variable.names = c("a", "b"),
  n.iter = 10000,
  thin = 10
)

summary(fit.jags.coda)
plot(fit.jags.coda)
```




```
# store samples for each parameter from the chain into separate objects
m.fit.jags.coda <- as.matrix(fit.jags.coda)
a.sim <- m.fit.jags.coda[,"a"]
b.sim <- m.fit.jags.coda[,"b"]

par(mfrow=c(1,2), mar=c(6.1, 3.1, 4.1, 2.1), xpd=TRUE)
# plot for intercept
truehist(a.sim, prob=TRUE, col="lavender", xlab=expression(a))
```

```r
lines(density(a.sim),lty=3,lwd=3, col=2)
lines(inla.output$marginals.fixed$"(Intercept)",lwd=2)

# plot for slope
truehist(b.sim, prob=TRUE, col="lavender", xlab=expression(b))
lines(density(b.sim),lty=3,lwd=3, col=2)
lines(inla.output$marginals.fixed$"x",lwd=2)
legend("topright",c("MCMC: JAGS","INLA"), inset=c(-0.2,0),
       lty=c(3,1),lwd=c(2,2),col=c(2,1),cex=0.7,bty="n")
```




```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 6
##    Unobserved stochastic nodes: 2
##    Total graph size: 37
##
## Initializing model
##
##
## Iterations = 8010:18000
## Thinning interval = 10
## Number of chains = 4
## 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    -0.9576  0.2298 0.003633       0.003591
## b -142.1170 24.6851 0.390306       0.368080
##
```

```
## 2. Quantiles for each variable:
##
##      2.5%      25%       50%       75%    97.5%
## a   -1.43   -1.105   -0.9472   -0.8019  -0.5237
## b -191.98 -157.657 -141.1805 -125.0899 -96.6721
```

Classical logistic regression model

```
df <- data.frame("dosage" = x, "numtot" = n, "prop" = y/n)

# Centering dosages
df$dosage_centered <- df$dosage - mean(df$dosage)

glm.model <- glm(prop ~ dosage_centered, df, family= "binomial", weight= n)

summary(glm.model)$coef
```

```
##                     Estimate Std. Error   z value     Pr(>|z|)
## (Intercept)       -0.9800475  0.2399331 -4.084669 4.413963e-05
## dosage_centered -146.6927209 26.3629619 -5.564349 2.631328e-08
```

Conclusion/Comparison:

The estimates for the intercept a and slope b from the JAGS model are actually relatively similar compared to the classical logistic regression with the function glm(). Since we estimated the distribution of the slope and intercept, we can even obtain quantiles of a and b, which is not possible with the classical approach. Furthermore, we have a quantification of the computational MCMC error with the naive se and time-series se, which do not exist in the classical framework. Finally, we can see the fundamental differences between the classical and Bayesian statistics, reflected in the standard error of the estimates in the classical approach, and the empirical standard deviation in the Bayesian model.

Exercise 6

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

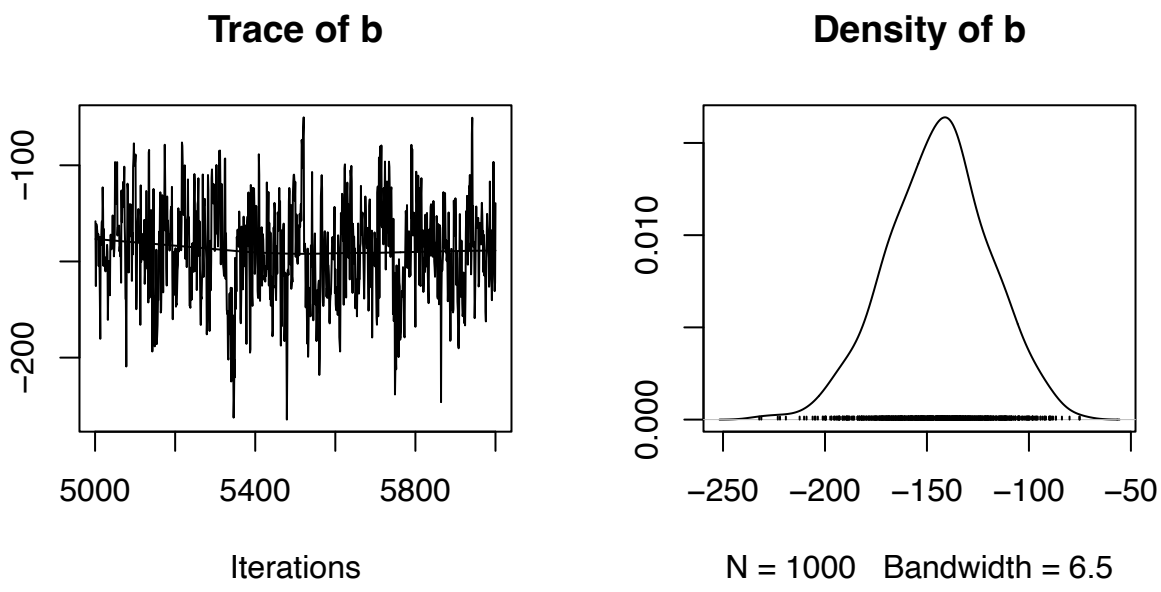
1

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.

```
set.seed(44566)

inits_1chain <- list(a = 0, b = 0, .RNG.name = "base::Wichmann-Hill", .RNG.seed = 123456)

# model initiation
model.jags <- jags.model(
  file = "MiceModel.txt",
  data = mice_data,
  inits = inits_1chain,
  n.chains = 1,
  n.adapt = 1000)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 6
##    Unobserved stochastic nodes: 2
##    Total graph size: 37
##
## Initializing model
```

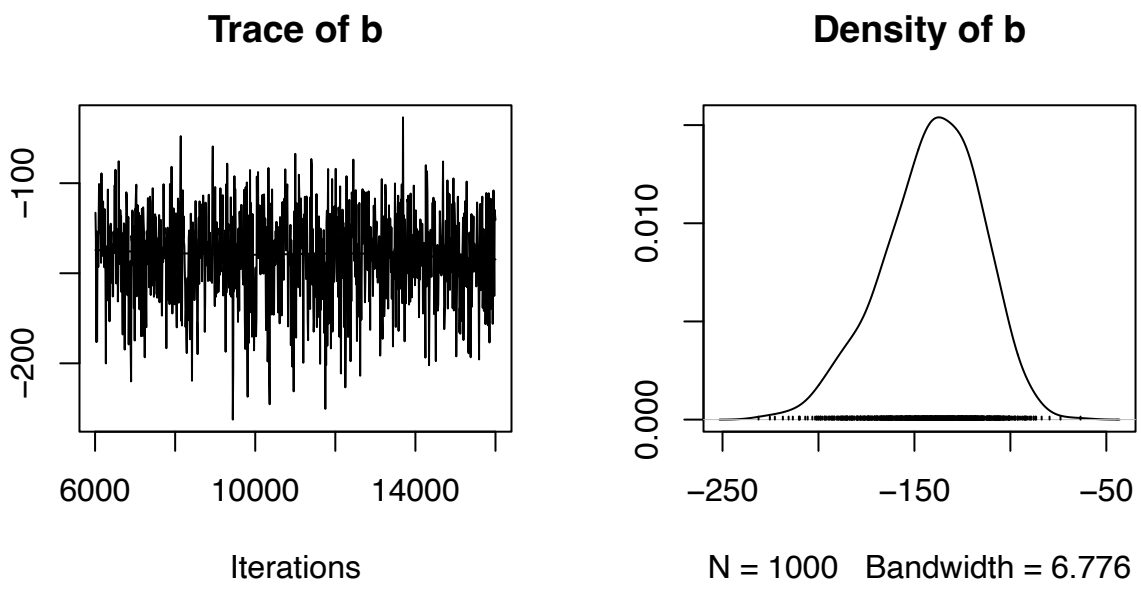
```
# burn-in
update(model.jags, n.iter = 4000)

# sampling
fit.jags.coda_1 <- coda.samples(
  model = model.jags,
  variable.names = "b",
  n.iter = 1000,
  thin = 1)

#summary(fit.jags.coda)
par(mfrow=c(1,2), mar=c(10,2,4,3))
plot(fit.jags.coda_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.

```
set.seed(44566)

# sampling
fit.jags.coda_2 <- coda.samples(
  model = model.jags,
  variable.names = "b",
  n.iter = 10000,
  thin = 10)

#summary(fit.jags.coda)
par(mfrow=c(1,2), mar=c(10,2,4,3))
plot(fit.jags.coda_2)
```



(a)

For which of the above conditions the ESS estimates will be larger and why?

Without thinning, the samples are dependent on preceding samples since the Markov Chain is kept intact. With thinning, only every x'th sample is chosen, so this dependence is disrupted, and autocorrelation is reduced. Which lower autocorrelation, the resulting ESS is higher.

(b)

To check your answer: Apply both the 05ess.R code and the function effectiveSize from the coda package. Compare the ESS estimates with those obtained with the n.eff function from package stableGR (Vats and Knudson, 2021). Please report your findings.

```
ess_t1 <-   ess(fit.jags.coda_1[[1]], 1000)
ess_t10 <- ess(fit.jags.coda_2[[1]], 1000)

coda_ess_t1 <- as.numeric(effectiveSize(fit.jags.coda_1))
coda_ess_t10 <- as.numeric(effectiveSize(fit.jags.coda_2))

stableGR_ess_t1 <- as.numeric(n.eff(fit.jags.coda_1)$n.eff)
stableGR_ess_t10 <- as.numeric(n.eff(fit.jags.coda_2)$n.eff)
```

Setting	custom function	coda::effectiveSize	stableGR::n.eff
n.iter = 1000, thin = 1	144.97	147.34	141.46
n.iter = 10000, thin = 10	954.434	857.59	954.434

Indeed, the effective sample sizes are higher with thinning. The different packages don't all return exactly the same values, but they're in comparable ranges.