Worksheet 5

Foundations of Bayesian Methodology

Wenje Tu Lea Bührer Jerome Sepin Zhixuan Li
Elia-Leonid Mastropietro Jonas Raphael Füglistaler

Spring Semester 2022

Exercise 3 - Normal example in JAGS

Exercise 4 - Logistic regression in JAGS

Exercise 5 - CODA for logistic regression in JAGS

```
x <- c(0.0028, 0.0028, 0.0056, 0.0112, 0.0225, 0.0450)
# the centered covariate values (centered dose) from the Mice data from Collett
x_centered <- x - mean(x)
# number of mice deaths
# y <- c(35, 21, 9, 6, 1)
y <- c(26, 9, 21, 9, 6, 1)
# total number of mice
# n <- c(40, 40, 40, 40, 40)
n <- c(28, 12, 40, 40, 40, 40)
```

```
d.mice <- data.frame(
  x, y, n, x_centered, y/n
)
colnames(d.mice) <- c("$x$", "$y$", "$n$", "centered $x$", "$p$")
knitr::kable(d.mice, align="c", caption="Mice data from Collett (2003)")
```

Table 1: Mice data from Collett (2003)

x	y	n	centered x	p
0.0028	26	28	-0.0121833	0.9285714
0.0028	9	12	-0.0121833	0.7500000
0.0056	21	40	-0.0093833	0.5250000
0.0112	9	40	-0.0037833	0.2250000
0.0225	6	40	0.0075167	0.1500000
0.0450	1	40	0.0300167	0.0250000

Logistic model:

$$\text{logit}(p_i) = \ln\left(\frac{p_i}{1-p_i}\right) = \alpha + \beta x_i$$

$$p_i = \frac{\exp(\alpha + \beta x_i)}{1 + \exp(\alpha + \beta x_i)}$$

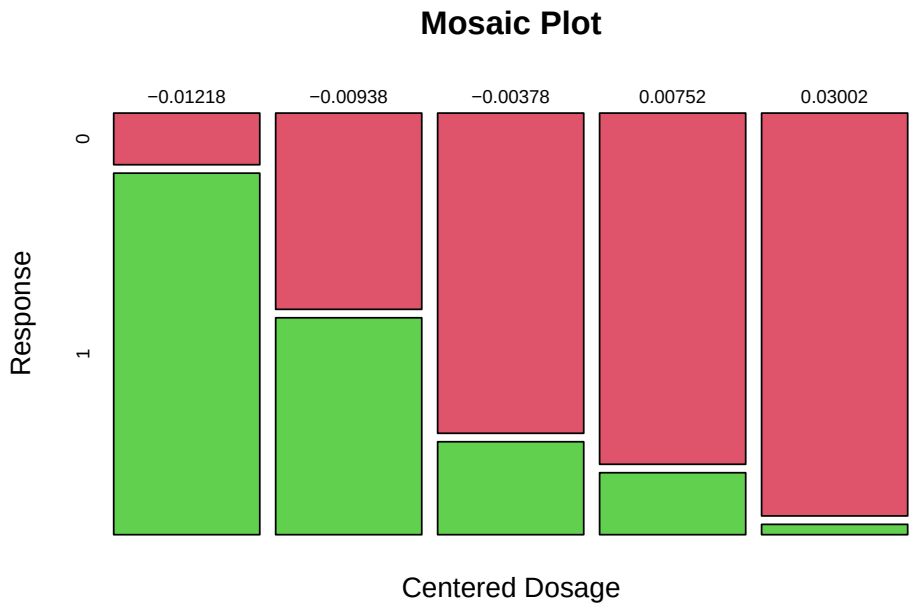
```
## Disaggregate the data
d.mice1 <- data.frame(Response=rep(c(1, 0), c(sum(y), sum(n)-sum(y))),
                      CenteredDose=c(rep(round(x_centered, 5), y),
```

```
                                    rep(round(x_centered, 5), n-y)))

knitr::kable(table(d.mice1))
```

	-0.01218	-0.00938	-0.00378	0.00752	0.03002
0	5	19	31	34	39
1	35	21	9	6	1

```
mosaicplot(CenteredDose ~ Response, data=d.mice1, color=2:3,
           main="Mosaic Plot", xlab="Centered Dosage")
```




```
modelString <- "model{
  for (i in 1:length(y)) {
    y[i] ~ dbin(p[i],n[i])
    p[i] <- ilogit(alpha + beta * x[i])
  }

  alpha ~ dnorm(0, 1.0E-04)
  beta ~ dnorm(0, 1.0E-04)
}"

writeLines(modelString, con="LogitModel.txt")
```

```
library(rjags)
library(coda)
```

```
## Generate data list for JAGS
dat.jags <- list("y"=y, "x"=x_centered, "n"=n)

## Initialize starting points (let JAGS initialize) and set seed
inits.jags <- list(list(.RNG.name="base::Wichmann-Hill", .RNG.seed=314159),
                   list(.RNG.name="base::Marsaglia-Multicarry", .RNG.seed=159314),
                   list(.RNG.name="base::Super-Duper", .RNG.seed=413159),
                   list(.RNG.name="base::Mersenne-Twister", .RNG.seed=143915))

## Compile JAGS model
model1.jags <- jags.model(
  file = "LogitModel.txt",
```

```
  data = dat.jags,
  inits = inits.jags,
  n.chains = 4,
  n.adapt = 4000
)
```

```
## 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(model1.jags, n.iter = 4000)

## Sampling
fit1.jags.coda <- coda.samples(
  model = model1.jags,
  variable.names = c("alpha", "beta"),
  n.iter = 10000,
  thin = 1
)
```

```
summary(fit1.jags.coda)
```

```
##
## Iterations = 8001:18000
## Thinning interval = 1
## Number of chains = 4
## 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
## alpha   -0.961  0.23  0.00115       0.002288
## beta  -142.233 24.71  0.12357       0.243683
##
## 2. Quantiles for each variable:
##
##          2.5%      25%       50%       75%    97.5%
## alpha   -1.43   -1.111   -0.9529   -0.8029  -0.5309
## beta  -193.85 -158.058 -141.1766 -125.0540 -96.9159
```

```
plot(fit1.jags.coda)
```

```
m.fit1.jags.coda <-as.matrix(fit1.jags.coda)
d.summary <- t(rbind(
  colMeans(m.fit1.jags.coda),
  apply(m.fit1.jags.coda, 2, function(x) sd(x)),
  apply(m.fit1.jags.coda, 2, function(x) quantile(x, probs=c(0.025, 0.5, 0.975)))
))

colnames(d.summary) <- c("Mean", "SD", "2.5%", "Median", "97.5%")
knitr::kable(d.summary, align="c", caption="Summary statistics")
```

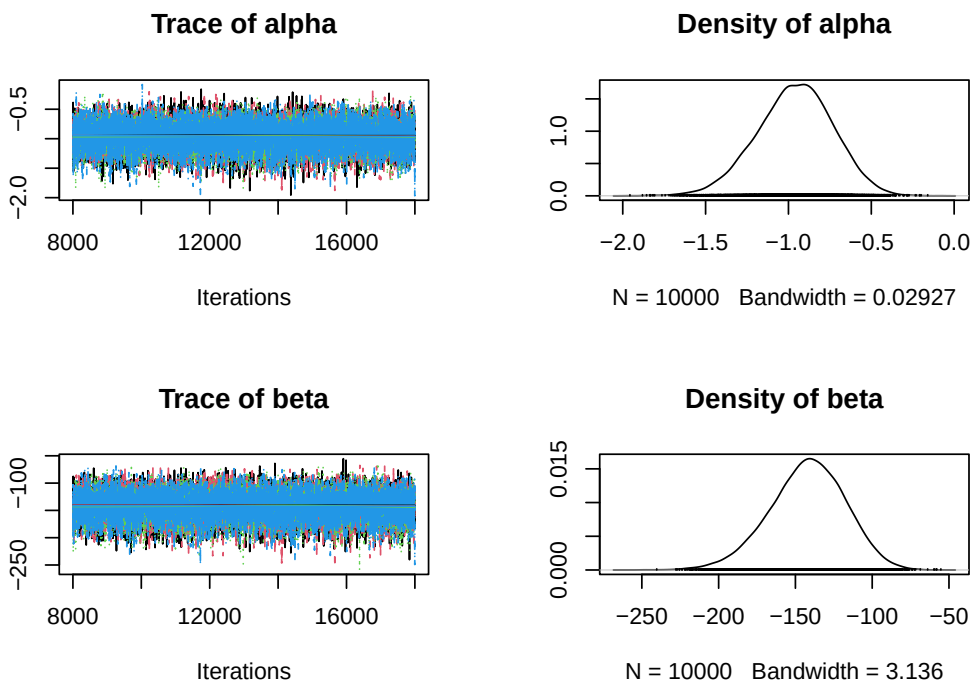


Figure 1: Trace and density plot.

Table 3: Summary statistics

	Mean	SD	2.5%	Median	97.5%
alpha	-0.9609658	0.2299865	-1.430185	-0.9529379	-0.5308684
beta	-142.2333579	24.7146657	-193.853777	-141.1766066	-96.9158787

5.1 Rank plots

```
library(bayesplot)
library(stableGR)
```

```
mcmc_rank_hist(fit1.jags.coda)
```

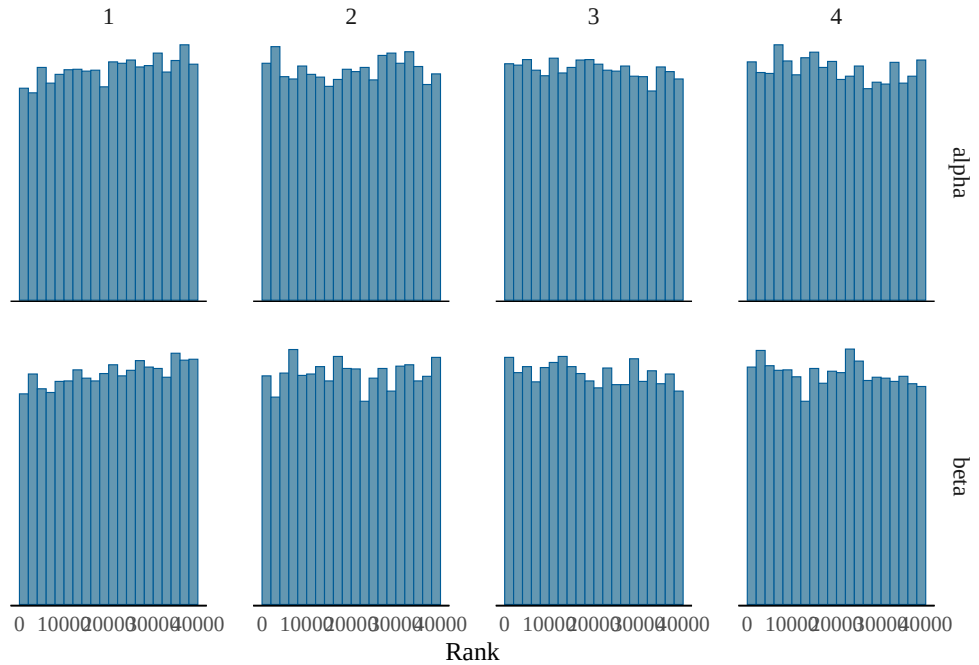


Figure 2: Rank Plot.

The figure illustrates the histograms of the ranked posterior drawn, ranked over all chains for α and β. Whereas traditional trace plots visualize how the chains mix over the course of sampling, rank histograms visualize how the values from the chains mix together in terms of ranking. An ideal plot would show the

```
##       test
## alpha passed      -0.953 0.00896
## beta  passed    -141.317 0.95695
##
## [[2]]
##
##       Stationarity start     p-value
##       test         iteration
## alpha passed       1         0.243
## beta  passed       1         0.162
##
##       Halfwidth Mean     Halfwidth
##       test
## alpha passed      -0.961 0.00904
## beta  passed    -142.063 0.97236
##
## [[3]]
##
##       Stationarity start     p-value
##       test         iteration
## alpha passed       1         0.836
## beta  passed       1         0.364
##
##       Halfwidth Mean     Halfwidth
##       test
## alpha passed      -0.965 0.00894
## beta  passed    -142.776 0.96457
##
## [[4]]
##
##       Stationarity start     p-value
##       test         iteration
## alpha passed       1         0.371
## beta  passed       1         0.481
##
##       Halfwidth Mean     Halfwidth
##       test
## alpha passed      -0.965 0.00893
## beta  passed    -142.777 0.92644
```

The output from `heidel.diag` shows the summary results for the four generated chains. The daignostic on one hand checks if the length of the sample it long enough and on the other hand checks if the means are estimated from a converged chain.

The stationarity test uses the Cramer-von-Mises statistic to test the null hypothesis that the sampled values come from a stationary distribution. The half-width test calculates a 95% confidence interval for the mean, using the portion of the chain which passed the stationarity test.

We see that both tests are passed for the four chains and all p-values are larger than $0.05 = \alpha$. We conclude that the sampled values come from a stationary distribution and the length of the sample is considered long enough to estimate the mean with sufficient accuracy.

```
raftery.diag(fit1.jags.coda)
```

(b) Raftery & Lewis (Convergence to ergodic average)

```
## [[1]]
##
## Quantile (q) = 0.025
```

```
## Accuracy (r) = +/- 0.005
## Probability (s) = 0.95
##
##        Burn-in  Total Lower bound  Dependence
##        (M)      (N)   (Nmin)       factor (I)
##  alpha 12       12590 3746         3.36
##  beta  7        8197  3746         2.19
##
##
## [[2]]
##
## Quantile (q) = 0.025
## Accuracy (r) = +/- 0.005
## Probability (s) = 0.95
##
##        Burn-in  Total Lower bound  Dependence
##        (M)      (N)   (Nmin)       factor (I)
##  alpha 10       10966 3746         2.93
##  beta  8        8276  3746         2.21
##
##
## [[3]]
##
## Quantile (q) = 0.025
## Accuracy (r) = +/- 0.005
## Probability (s) = 0.95
##
##        Burn-in  Total Lower bound  Dependence
##        (M)      (N)   (Nmin)       factor (I)
##  alpha 7        7893  3746         2.11
##  beta  8        10866 3746         2.90
##
##
## [[4]]
##
## Quantile (q) = 0.025
## Accuracy (r) = +/- 0.005
## Probability (s) = 0.95
##
##        Burn-in  Total Lower bound  Dependence
##        (M)      (N)   (Nmin)       factor (I)
##  alpha 8        11042 3746         2.95
##  beta  8        10204 3746         2.72
```

The output from `raftery.diag` shows the summary results for the four generated chains. `raftery.diag` is a run length control diagnostic based on a criterion of accuracy of estimation of the quantile q. The dependence factor I ($I = \frac{M+N}{N_{\min}}$), indicates to which extent autocorrelation inflates the required sample size. $I > 5$ indicates strong autocorrelation which may be due to a poor choice of starting value, high posterior correlations or "stickiness" of the MCMC algorithm. We see that the dependence factors for the four chains are all smaller than 5 and hence no strong autocorrelation exists.

```
geweke.diag(fit1.jags.coda)
```

(c) Geweke (Convergence to stationarity)

```
## [[1]]
##
## Fraction in 1st window = 0.1
```

```
## Fraction in 2nd window = 0.5
##
##  alpha   beta
## 0.2719 0.6407
##
##
## [[2]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
## alpha  beta
## 1.358 1.629
##
##
## [[3]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##    alpha     beta
## -0.09937 -0.20974
##
##
## [[4]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##  alpha   beta
## 0.9955 0.6094
```

The output from `geweke.diag` shows the summary results for the four generated chains. `geweke.diag` is a convergence diagnostic for Markov chains based on a test for equality of the means of the first and last part of a Markov chain (by default the first 10% and the last 50%). If the samples are drawn from the stationary distribution of the chain, the two means are equal and Geweke's statistic has an asymptotically standard normal distribution. The test statistic is a standard Z-score.

The idea behind the Geweke's diagnostic is that in a long enough chain whose trace plots suggest convergence to the target distribution, we assume the second half of the chain has converged to the target distribution and we test if the first 10% can be treated as burn-in. So we mimic the simple two-sample test of means: if the mean of the first 10% is not significantly different from the last 50%, then we conclude the target distribution converged somewhere in the first 10% of the chain. So we'll use this 10% as burn-in.

We see that for the four chains the absolute values of Z-scores for variables alpha and beta are smaller than 2, which indicates that the equilibrium may have been reached.

```
geweke.plot(fit1.jags.coda)
```

If `geweke.diag` indicates that the first and last part of a sample from a Markov chain are not drawn from the same distribution, it may be useful to discard the first few iterations to see if the rest of the chain has "converged". `geweke.plot` shows what happens to Geweke's Z-score when successively larger numbers of iterations are discarded from the beginning of the chain. To preserve the asymptotic conditions required for Geweke's diagnostic, the plot never discards more than half the chain. For beta (chain1), alpha (chain2), and alpha (chain3), there exist several segments out of the 95% confidence bands ($|Z| > 2$). The plot shows that we mainly obtain z-score values below an absolute value of 2. So we can assume that the chain has converged.

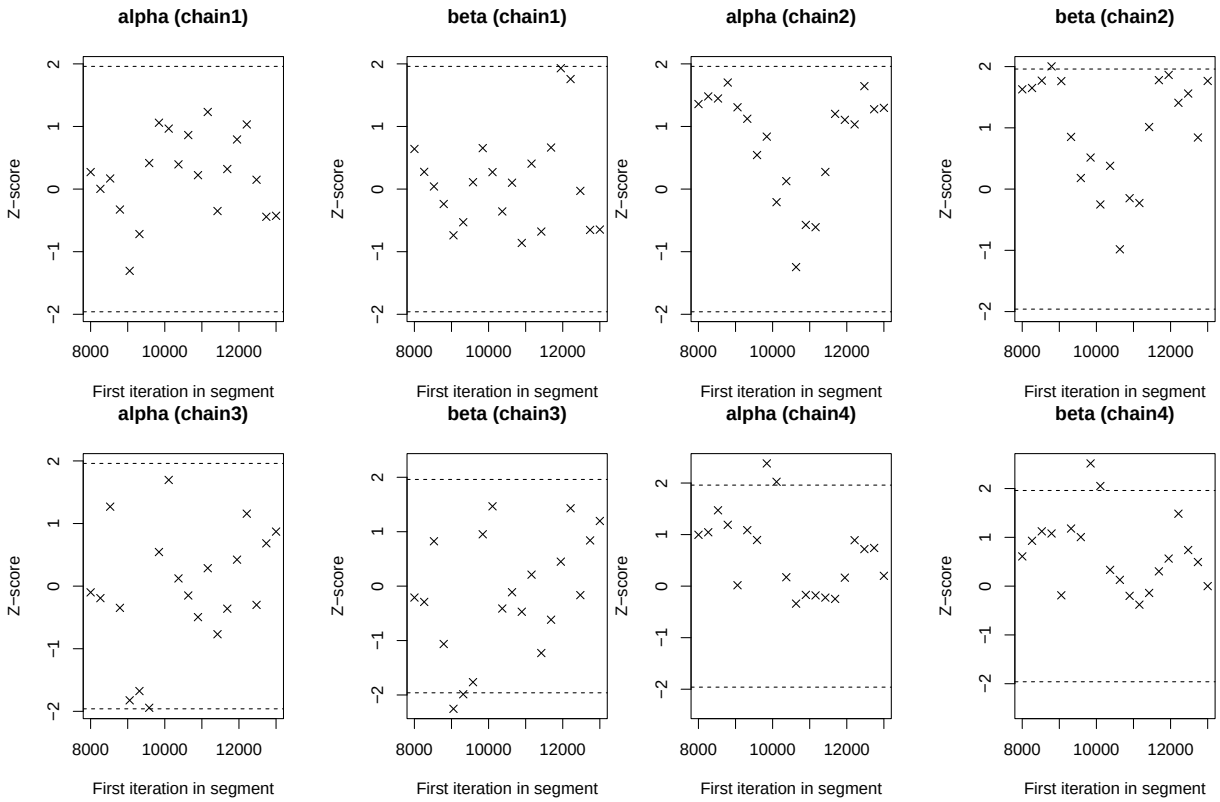


Figure 3: Geweke Plot.

```
gelman.diag(fit1.jags.coda, autoburnin=TRUE)
```

(d) Gelman and Rubin's convergence diagnostic

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

The idea of Gelman and Rubin's convergence diagnostic is to run multiple chains from widely dispersed starting values and perform an Analysis of Variance to see if the between-chain variability (B) is large in relation to the average variability within ($W + B$) the pooled chain.

$$\sqrt{\hat{R}} \approx \sqrt{\frac{W+B}{W}}$$

$\hat{R}$ is so-called "potential scale reduction factor" and it is calculated for each variable. We see that the "Potential scale reduction factors" for both alpha and beta are close to 1, which indicates that the between-chain variability (B) is close to 0. In other words, the separate four chains have mixed quite well.

```
gelman.plot(fit1.jags.coda, autoburnin=TRUE)
```

The `gelman.plot` shows that the evolution of Gelman and Rubin's shrink factor as the number of iterations increases. We see that the shrink factors for both variables quickly decrease to 1 after 10000

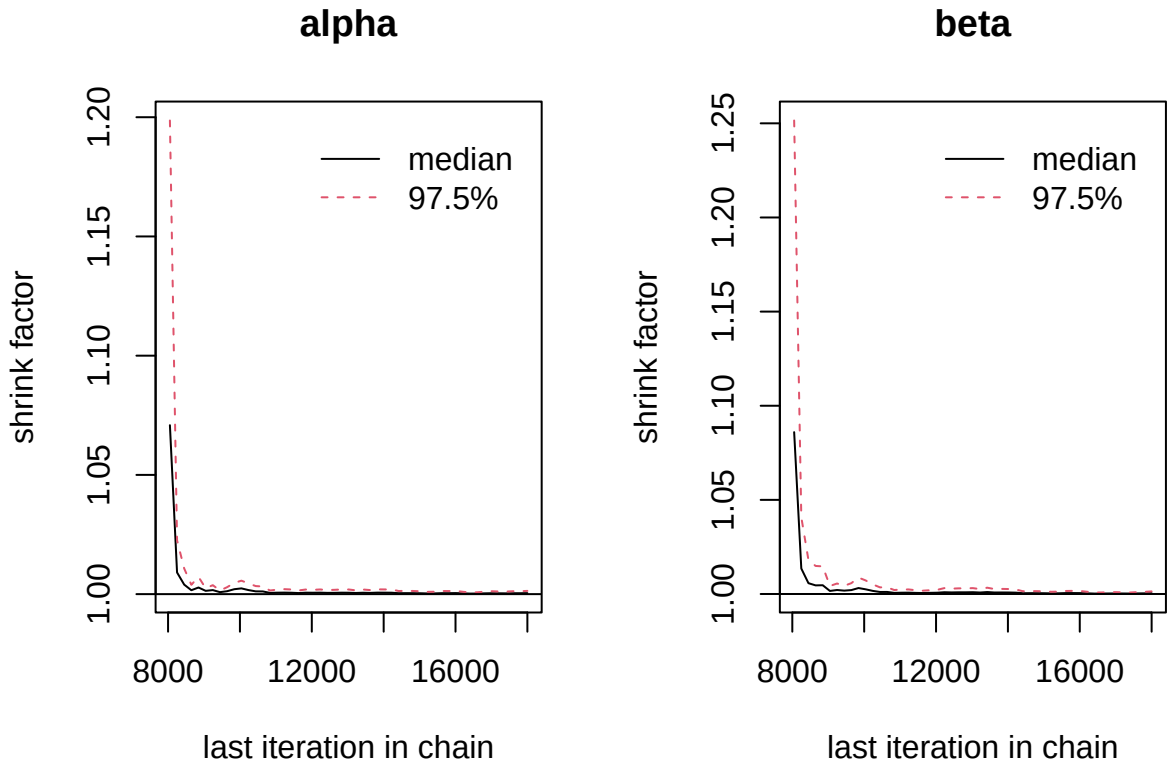


Figure 4: Gelman plot.

iterations and the variability of shrink factors becomes more stable as the number of iterations keeps increasing.

```
stable.GR(fit1.jags.coda)
```

(e) Gelman-Rubin diagnostic using stable variance estimators

```
## $psrf
## [1] 1.000002 1.000004
##
## $mpsrf
## [1] 1.000062
##
## $means
##        alpha         beta
##   -0.9608901 -142.2202900
##
## $n.eff
## [1] 17806.81
##
## $blather
## [1] FALSE
```

`stable.GR` extends Gelman-Rubin diagnostic using stable variance estimators. We see that the univariate potential scale reduction factors for both alpha and beta are calculated and they are close to 1. Multivariate `psrf` is also calculated by taking into account the interdependence of the chain's components. The PSRFs decrease to 1 as the chain length increases. When the PSRF becomes sufficiently close to 1, the sample collected by the Markov chain has converged to the target distribution. Means of variables (alpha and beta) and the effective sample size are also reported in the output.

Conclusions:

- Heidel: We conclude that the sampled values come from a stationary distribution and the length of the sample is considered long enough to estimate the mean with sufficient accuracy.
- Raftery & Lewis: The dependence factors for the four chains are all smaller than 5 and hence no strong autocorrelation exists.
- Geweke: We assume that the equilibrium has been reached. We achieve convergence towards the target distribution.
- Gelman & Rubin: The between-chain variability is close to 0. The separate four chains have mixed quite well.
- Gelman & Rubin for stable variance: The chains have converged to the target distribution.

All in all, it can be concluded that no issues arose during the diagnostics.

5.3 Re-run the MCMC simulation

In the following steps the MCMC simulation is adapted to the findings from subtask 5.2. We keep the same model as before but we change n.iter from 10'000 to 30'000 and n.thin from 1 to 3. The number of burn-in iterations (4000) is kept. **WHY do we change to this values???? the diagnostics do not really get better... (excluding the rank_hist)**

```
## Compile JAGS model
model2.jags <- jags.model(
  file = "LogitModel.txt",
  data = dat.jags,
  inits = inits.jags,
  n.chains = 4,
  n.adapt = 4000
)
```

```
## 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 (increase burn-in iterations)
update(model2.jags, n.iter = 4000)

## Initialize starting points (let JAGS initialize) and set seed
inits.jags <- list(list(.RNG.name="base::Wichmann-Hill", .RNG.seed=314159),
                   list(.RNG.name="base::Marsaglia-Multicarry", .RNG.seed=159314),
                   list(.RNG.name="base::Super-Duper", .RNG.seed=413159),
                   list(.RNG.name="base::Mersenne-Twister", .RNG.seed=143915))

## Sampling
fit2.jags.coda <- coda.samples(
  model = model2.jags,
  variable.names = c("alpha", "beta"),
  n.iter = 30000,
  thin = 3
)
```

```
summary(fit2.jags.coda)
```

```
##
## Iterations = 8003:38000
```

```
## Thinning interval = 3
## Number of chains = 4
## 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
## alpha   -0.9584  0.2301 0.001151       0.001439
## beta  -142.0398 24.4139 0.122070       0.150805
##
## 2. Quantiles for each variable:
##
##           2.5%     25%       50%       75%    97.5%
## alpha   -1.431   -1.11   -0.9521   -0.7997  -0.5281
## beta  -192.709 -157.99 -141.0777 -124.8791 -97.0048
```

```
plot(fit2.jags.coda)
```

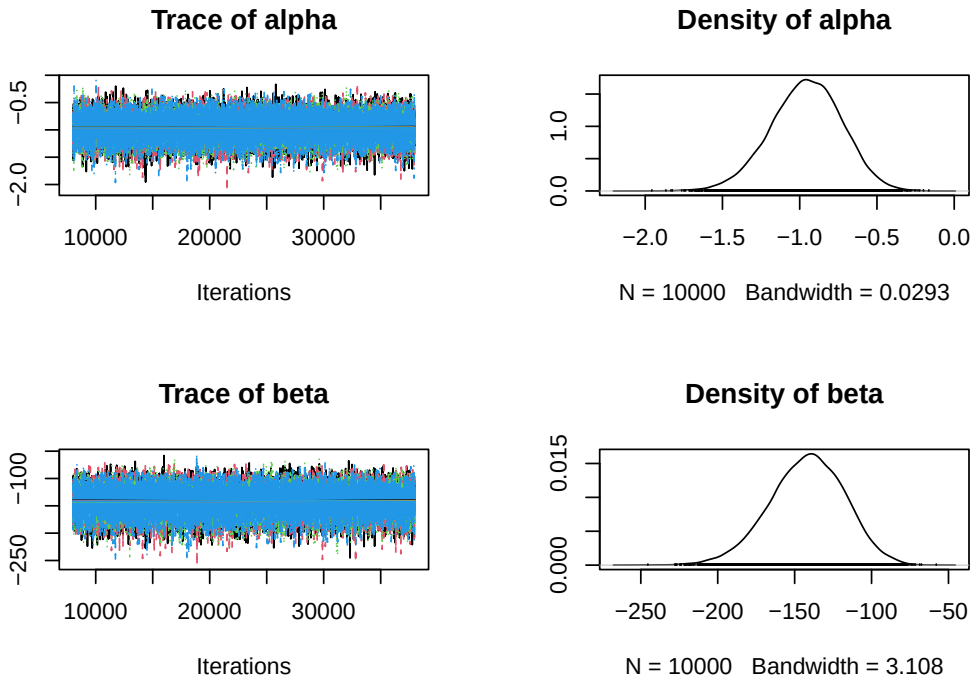


Figure 5: Trace and density plot.

```
mcmc_rank_hist(fit2.jags.coda)
```

Compared to the rank plots obtained with the original model, the rank plots for β in the new model do better follow a normal distribution. The rank plots of α are comparable.

```
## Heidelberger & Welch (Convergence to stationarity)
heidel.diag(fit2.jags.coda)
```

```
## [[1]]
##
##       Stationarity start     p-value
##       test         iteration
## alpha passed       1         0.942
## beta  passed       1         0.397
##
##       Halfwidth Mean     Halfwidth
```

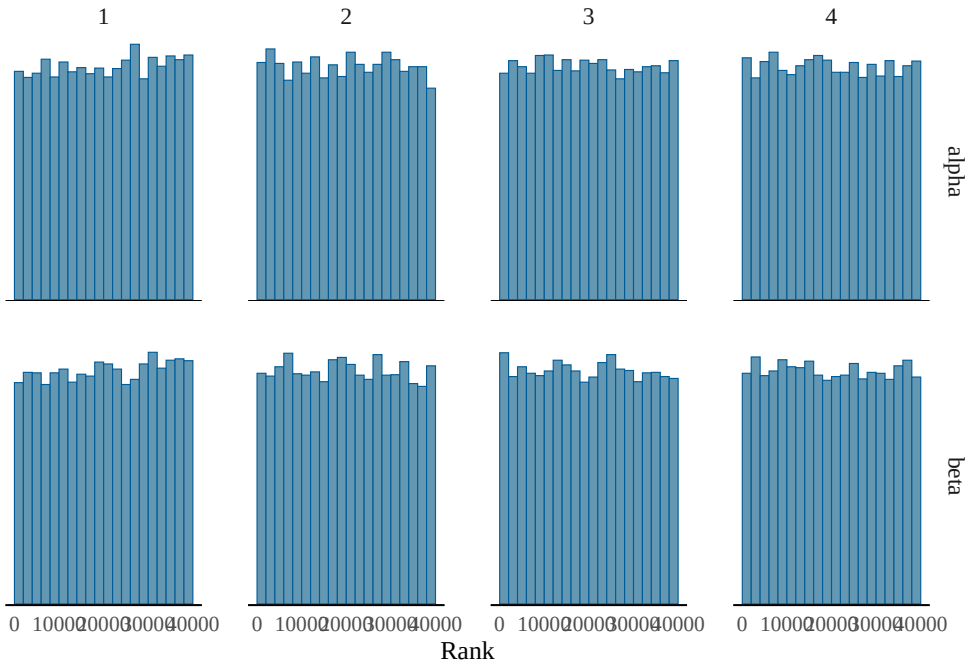


Figure 6: Rank Plot.

```
##       test
## alpha passed      -0.954 0.00567
## beta  passed    -141.443 0.58374
##
## [[2]]
##
##       Stationarity start     p-value
##       test         iteration
## alpha passed       1         0.510
## beta  passed       1         0.489
##
##       Halfwidth Mean     Halfwidth
##       test
## alpha passed      -0.961 0.00563
## beta  passed    -142.173 0.60000
##
## [[3]]
##
##       Stationarity start     p-value
##       test         iteration
## alpha passed       1         0.409
## beta  passed       1         0.151
##
##       Halfwidth Mean     Halfwidth
##       test
## alpha passed      -0.959 0.0056
## beta  passed    -142.340 0.5986
##
## [[4]]
##
##       Stationarity start     p-value
##       test         iteration
## alpha passed       1         0.455
## beta  passed       1         0.433
```

```
##
##       Halfwidth Mean    Halfwidth
##       test
## alpha passed      -0.96 0.00567
## beta  passed    -142.20 0.58211
```

The p-values of the Heidelberg diagnostics have slightly changed. One test is now even failed (alpha 3).

```
## Raftery & Lewis (Convergence to ergodic average)
raftery.diag(fit2.jags.coda)
```

```
## [[1]]
##
## Quantile (q) = 0.025
## Accuracy (r) = +/- 0.005
## Probability (s) = 0.95
##
##        Burn-in  Total Lower bound  Dependence
##        (M)      (N)   (Nmin)       factor (I)
##  alpha 12       14376 3746         3.84
##  beta  9        13338 3746         3.56
##
##
## [[2]]
##
## Quantile (q) = 0.025
## Accuracy (r) = +/- 0.005
## Probability (s) = 0.95
##
##        Burn-in  Total Lower bound  Dependence
##        (M)      (N)   (Nmin)       factor (I)
##  alpha 12       14616 3746         3.90
##  beta  12       13905 3746         3.71
##
##
## [[3]]
##
## Quantile (q) = 0.025
## Accuracy (r) = +/- 0.005
## Probability (s) = 0.95
##
##        Burn-in  Total Lower bound  Dependence
##        (M)      (N)   (Nmin)       factor (I)
##  alpha 12       14022 3746         3.74
##  beta  12       14256 3746         3.81
##
##
## [[4]]
##
## Quantile (q) = 0.025
## Accuracy (r) = +/- 0.005
## Probability (s) = 0.95
##
##        Burn-in  Total Lower bound  Dependence
##        (M)      (N)   (Nmin)       factor (I)
##  alpha 9        13449 3746         3.59
##  beta  9        13122 3746         3.50
```

The values for the dependence factor I have decreased. They are all still below 5. So we do not obtain any strong autocorrelation.

```
## Geweke (Convergence to stationarity)
geweke.diag(fit2.jags.coda)
```

```
## [[1]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##  alpha   beta
## 0.5573 0.3418
##
##
## [[2]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##  alpha   beta
## 0.5888 1.0447
##
##
## [[3]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##  alpha   beta
## -1.196 -1.400
##
##
## [[4]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##  alpha   beta
## 0.4101 0.5777
```

The obtained absolute values for α and β are all below 2, which indicates convergence to the target distribution. The plot shows that for the third and fourth chain some z-scores lie above and one below a z-score of 2.0.

```
geweke.plot(fit2.jags.coda)
```

```
## Gelman and Rubin's convergence diagnostic
gelman.diag(fit2.jags.coda, autoburnin=TRUE)
```

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

The Gelman and Rubin diagnostic output is equivalent for this second adapted model to the one for the original model.

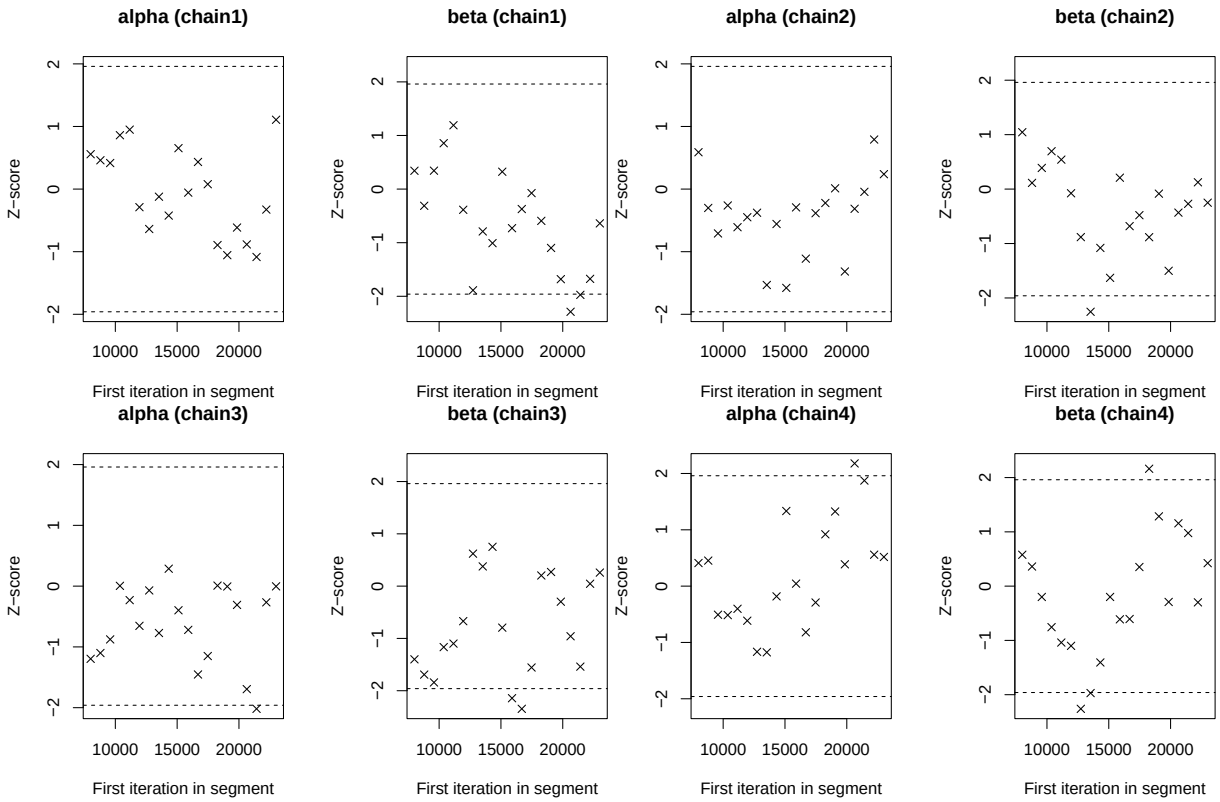


Figure 7: Geweke Plot.

```
gelman.plot(fit2.jags.coda, autoburnin=TRUE)
```

```
stable.GR(fit2.jags.coda)
```

```
## $psrf
## [1] 0.9999701 0.9999708
##
## $mpsrf
## [1] 1.000016
##
## $means
##        alpha         beta
##   -0.9584487 -142.0399259
##
## $n.eff
## [1] 30374.44
##
## $blather
## [1] FALSE
```

The univariate potential scale reduction factors for both alpha and beta are both close to 1. Compared to the original model the effective sample size has been increased from 16766.74 to 29704.35.

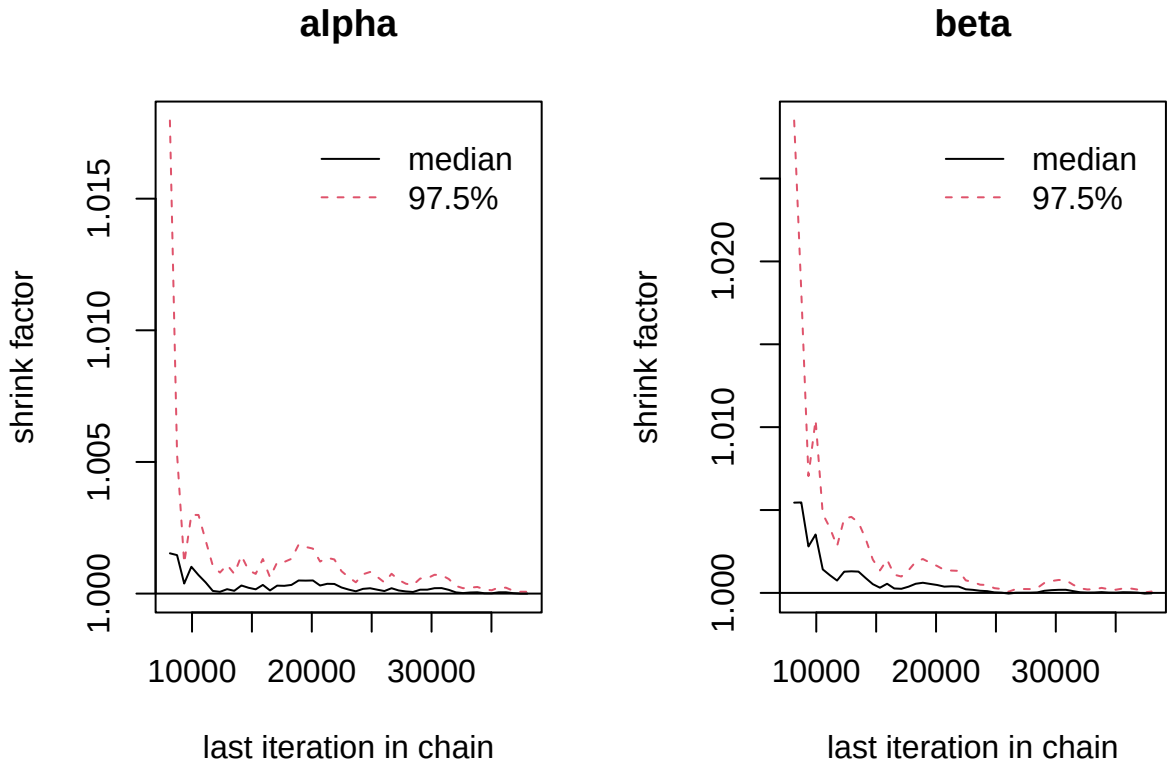


Figure 8: Gelman plot.

Exercise 6 (ESS)

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.**

```
## Initialize starting points (let JAGS initialize) and set seed
inits3.jags <- list(list(.RNG.name="base::Wichmann-Hill", .RNG.seed=314159))

## Compile JAGS model
model3.jags <- jags.model(
  file = "LogitModel.txt",
  data = dat.jags,
  n.chains = 1,
  inits = inits3.jags,
  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 (increase burn-in iterations)
update(model3.jags, n.iter = 4000)
```

```
## Posterior Sampling
fit3.jags.coda <- coda.samples(
  model = model3.jags,
  variable.names = c("beta"), #monitoring only beta
  n.iter = 1000,
  thin = 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.**

```
## Initialize starting points (let JAGS initialize) and set seed
inits4.jags <- list(list(.RNG.name="base::Wichmann-Hill", .RNG.seed=314159))

## Compile JAGS model
model4.jags <- jags.model(
  file = "LogitModel.txt",
  data = dat.jags,
  n.chains = 1,
  inits = inits4.jags,
  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 (increase burn-in iterations)
update(model4.jags, n.iter = 4000)

## Posterior Sampling
fit4.jags.coda <- coda.samples(
  model = model4.jags,
  variable.names = c("beta"), #monitoring only beta
  n.iter = 10000,
  thin = 10
)
```

(a) **For which of the above conditions the ESS estimates will be larger and why?**

```
# Function to test if a vector is monoton decreasing,
# a boolean value is returned
monotone <- function(vec){
  a <- TRUE
  if(length(vec) == 1){
    return(a)
  }
  for(i in 2:length(vec)){
    if(vec[i] > vec[i-1]){
      a <- FALSE
      break;
    }
  }
  return(a)
```

```
}

# Function to find the lag to stop the ESS
# calculation compare:
#
#      Geyer (1992),
#     "Practical Markov Chain Monte Carlo".
#      Statistical Science, 7: 473- 511
#
# Gamma_i = gamma(2*i) + gamma(2*i + 1)
#
# m is the greatest integer at which Gamma_i > 0
# and Gamma_i is monotone for i = 1, ..., m.
# Thereby gamma(i) is the sample autocorrelation
# at lag i.
#
# Parameter:
# vec - sample vector (mcmc object)
#
# Output
# m <- greatest integer where both criteria are
#     fulfilled
geyer <- function(vec){
  g <- c()
  res <- 1
   for(i in 1:(length(vec)/2 - 1)){
    g <- c(g,
           autocorr(vec, lags = i) + autocorr(vec, lags = i + 1)
           )
    if(monotone(g) == FALSE || g[i] < 0){
      break
    }
  }
  if(i==1){
    res <- 1
  }
  else{
    res <- i-1
  }
  return(res)
}
# Function to calculate the effective sample
# size for one MCMC chain.
#
# Parameter:
# mcmc - mcmc object
# M - number of sampled values
#
# Output:
# effective sample size
ess <- function(mcmc, M){
  m <- geyer(mcmc)
  y <- M / (1 + 2 * sum(autocorr(mcmc, lag = 1:(2*m +1)) ) )
  return(y)
}
```

$$ESS = N_{eff} = \frac{M}{1 + 2\sum_{k\geq 1}^{\infty} ACF(k)}$$

$$\text{where } ACF(k) = \frac{Cov(\boldsymbol{x_t}, \boldsymbol{x_{t+k}})}{Var(\boldsymbol{x_t})} = \frac{\frac{1}{N}\sum_{t=1}^{N-k}(x_t - \bar{x})(x_{t+k} - \bar{x})}{\frac{1}{N}\sum_{t=1}^{N}(x_t - \bar{x})^2}$$

For practical computation, the infinite sum in the definition of ESS may be stopped earlier. Here the stopping is defined by the criteria of Geyer 1992.

We have in the second model a thinning parameter of 10 and thus expect the autocorrelation in the MCMC-sample to have less correlation as we pick only the 10th entry each time and put it into our MCMC-sample. Thus the sum will be smaller for the less correlated sample (`fit4.jags.coda`). M is in both models the same with 1000. Therefore we expect the ESS to be larger in the second model!

(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.**

```
library(stableGR)
#ESS from the script

ess_script1 <- ess(mcmc = as.mcmc(fit3.jags.coda), M= length(as.mcmc(fit3.jags.coda)) )
ess_script2 <- ess(mcmc = as.mcmc(fit4.jags.coda), M= length(as.mcmc(fit4.jags.coda)) )

#ESS from the stableGR package
ess_function1 <- stableGR::n.eff(as.mcmc(fit3.jags.coda))
ess_function2 <- stableGR::n.eff(as.mcmc(fit4.jags.coda))

results <- data.frame(
              "n_eff" = c(ess_script1,ess_script2,ess_function1$n.eff,ess_function2$n.eff),
              "thining" = c(1,10,1,10),
              "method"  = c("ESS-Script", "ESS-Script", "stableGR", "stableGR")
              )
```

Table 4: Effective sample size with different thinning and functions.

n_eff	thining	method
343.9597	1	ESS-Script
930.7882	10	ESS-Script
312.2273	1	stableGR
1000.0000	10	stableGR

As expected, the effective sample size is with both methods higher in the model with thinning = 10 in comparison with thinning = 1.