Worksheet 5

Foundations of Bayesian Methodology

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

Spring Semester 2022

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:

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

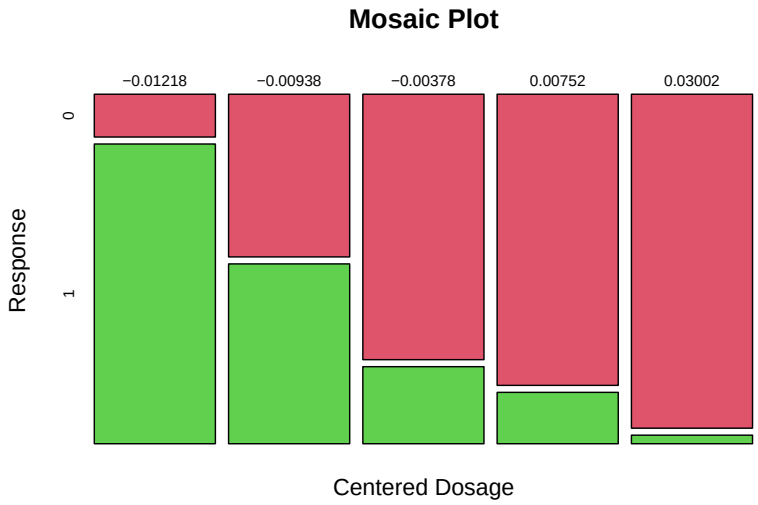
$$p_i = \frac{\exp(\alpha + \beta x_i)}{1 + \exp(\alpha + \beta x_i)}$$

```
## 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] ~ dbern(p[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)
```

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

```
## Linked to JAGS 4.3.0
```

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

```r
library(coda)
```

```r
## Generate data list for JAGS
dat.jags <- with(d.mice1, list(y=Response, x=CenteredDose))

## 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: 200
##    Unobserved stochastic nodes: 2
##    Total graph size: 419
##
## Initializing model
```

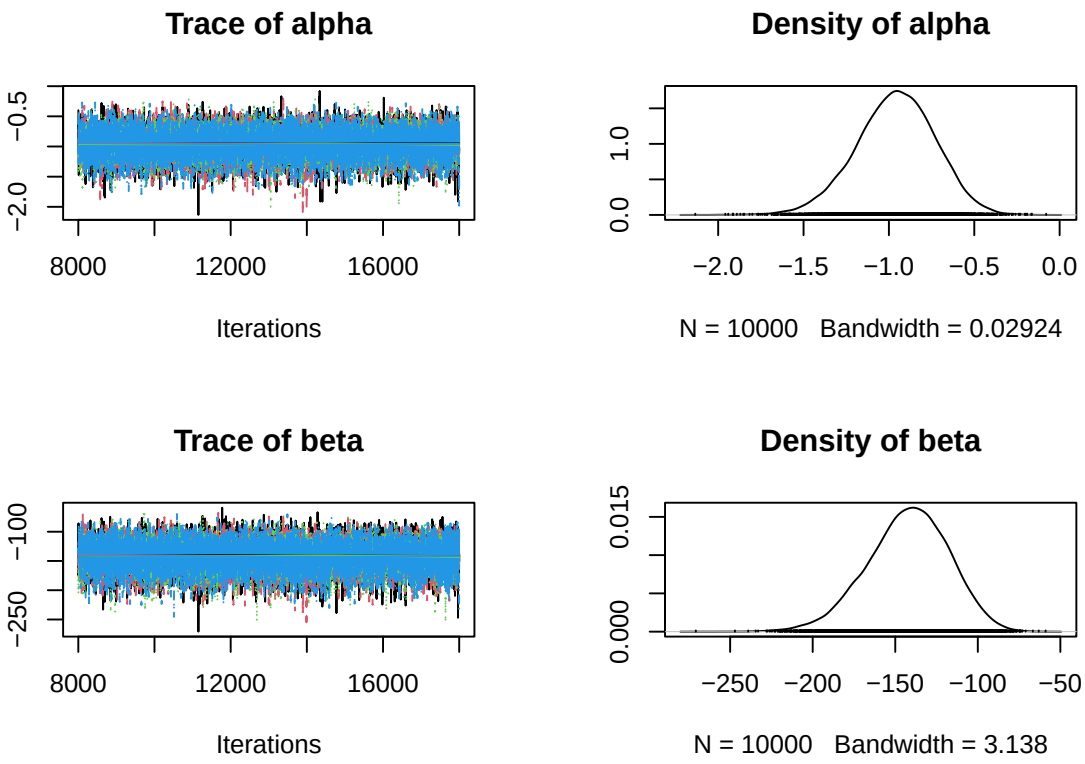
```r
## Burn-in
update(model1.jags, n.iter = 4000)

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

```r
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  0.2312 0.001156        0.00232
## beta  -142.347 24.6489 0.123244        0.24914
##
```

```
## 2. Quantiles for each variable:
##
##           2.5%     25%      50%       75%    97.5%
## alpha   -1.436   -1.11   -0.954   -0.8026  -0.5304
## beta  -193.929 -158.13 -141.108 -125.0617 -97.5972
```

```
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")
```

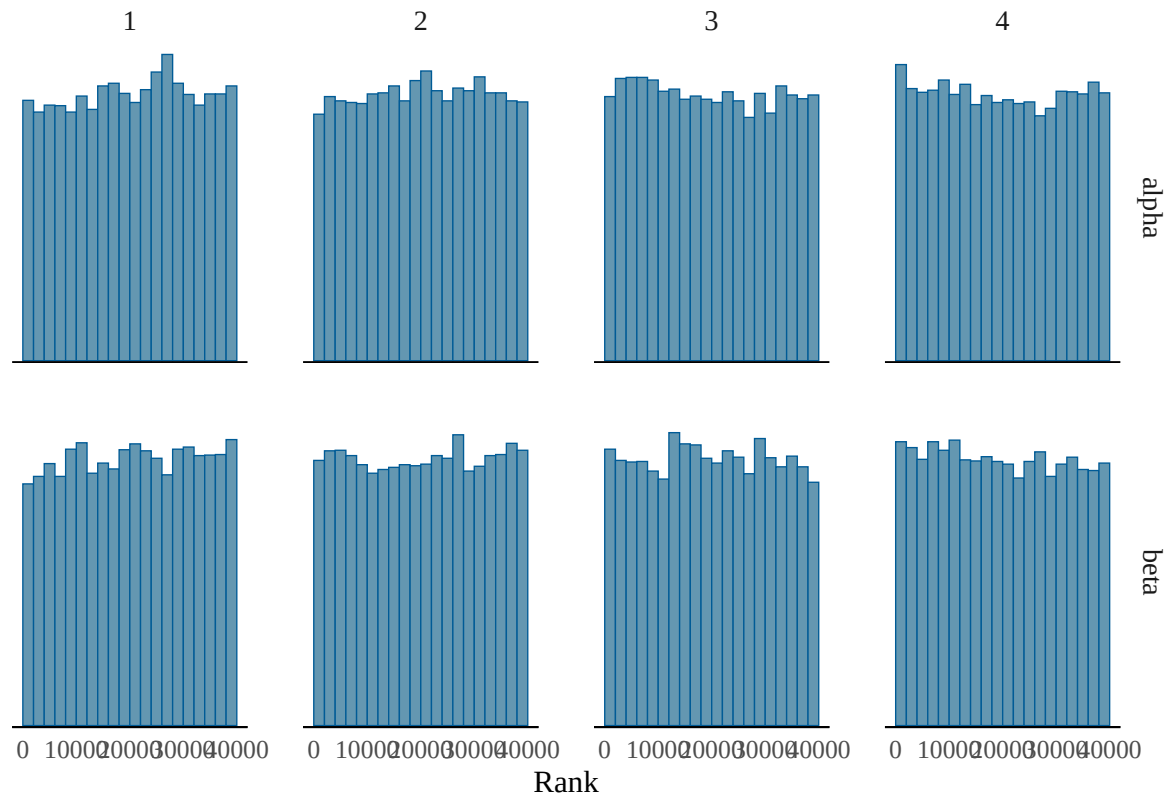
Table 3: Summary statistics

	Mean	SD	2.5%	Median	97.5%
alpha	-0.9609822	0.2312012	-1.436231	-0.9539647	-0.5304126
beta	-142.3473139	24.6488847	-193.928630	-141.1080375	-97.5972303

5.1 Rank plots

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

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



5.2 Convergence diagnostics

(a) Heidelberger & Welch (Convergence to stationarity)

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

```
## [[1]]
##
##       Stationarity start     p-value
##       test         iteration
## alpha passed       1         0.909
## beta  passed       1         0.880
##
##       Halfwidth Mean     Halfwidth
##       test
## alpha passed      -0.956 0.00886
## beta  passed    -141.614 0.96605
##
## [[2]]
##
##       Stationarity start     p-value
##       test         iteration
## alpha passed       1         0.167
## beta  passed       1         0.291
##
##       Halfwidth Mean     Halfwidth
##       test
## alpha passed      -0.959 0.00934
```

```
## beta  passed    -142.186 0.99778
##
## [[3]]
##
##       Stationarity start     p-value
##       test         iteration
## alpha passed       1         0.596
## beta  passed       1         0.635
##
##       Halfwidth Mean     Halfwidth
##       test
## alpha passed      -0.965 0.00902
## beta  passed    -142.646 0.99899
##
## [[4]]
##
##       Stationarity start     p-value
##       test         iteration
## alpha passed       1         0.538
## beta  passed       1         0.325
##
##       Halfwidth Mean     Halfwidth
##       test
## alpha passed      -0.964 0.00916
## beta  passed    -142.943 0.94250
```

The output from `heidel.diag` shows the summary results for the four generated chains. 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. 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.

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

```
raftery.diag(fit1.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 10       12334 3746         3.29
##  beta  10       12294 3746         3.28
##
##
## [[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       12488 3746         3.33
##  beta  10       11984 3746         3.20
##
##
## [[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 10       11462 3746         3.06
##  beta  10       11664 3746         3.11
##
##
## [[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 6        7002  3746         1.87
##  beta  8        11086 3746         2.96
```

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. We see that the dependence factors for the four chains are all smaller than 5 and hence no strong autocorrelation exists.

(c) Geweke (Convergence to stationarity)

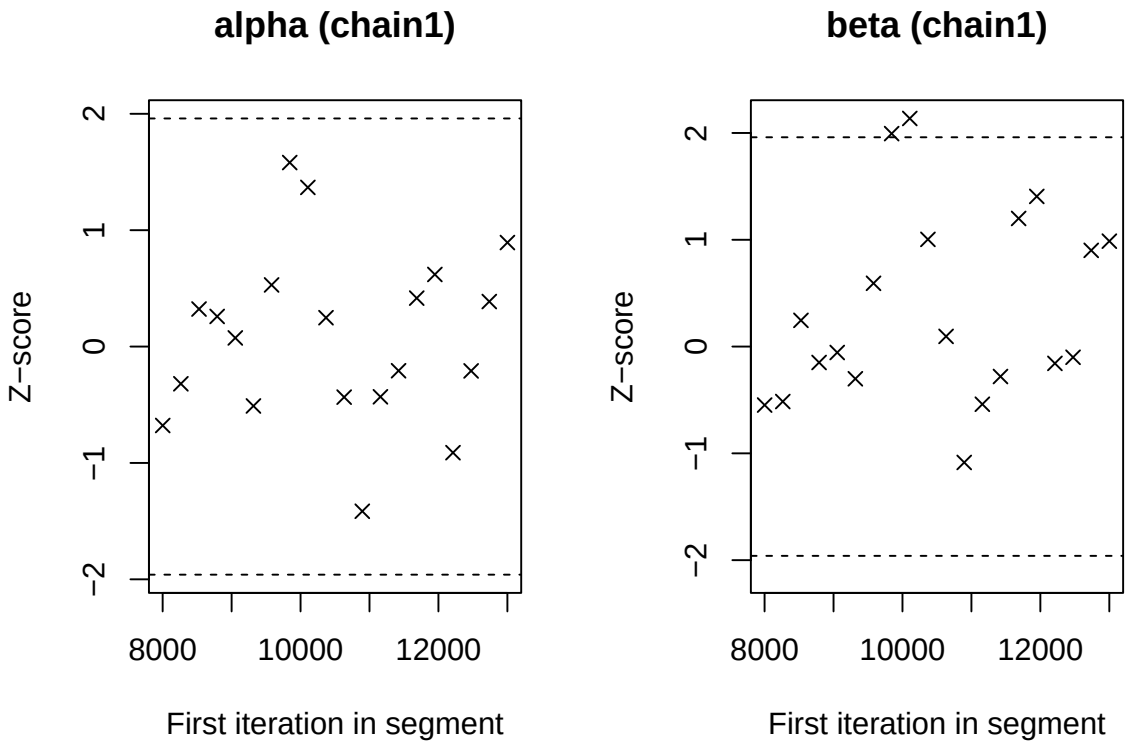
```
geweke.diag(fit1.jags.coda)
```

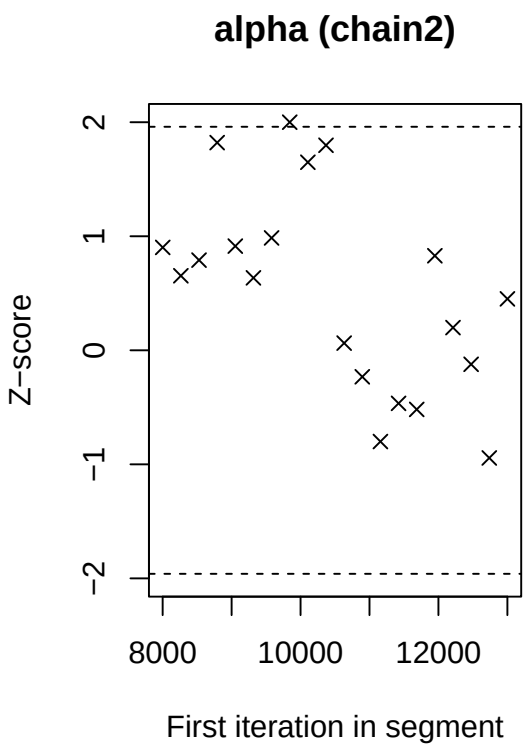
```
## [[1]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##   alpha    beta
## -0.6783 -0.5478
##
##
## [[2]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##  alpha   beta
## 0.9021 1.5789
##
##
## [[3]]
```

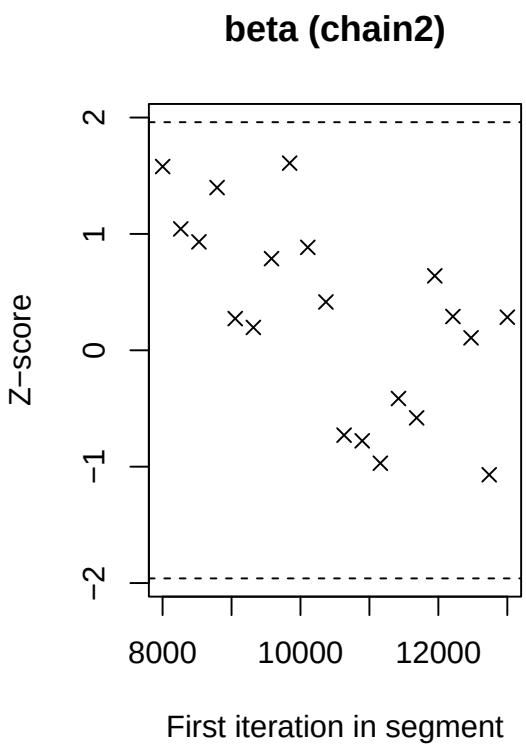
```
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##    alpha     beta
## -0.73994 -0.05956
##
##
## [[4]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
## alpha  beta
## 1.592 1.697
```

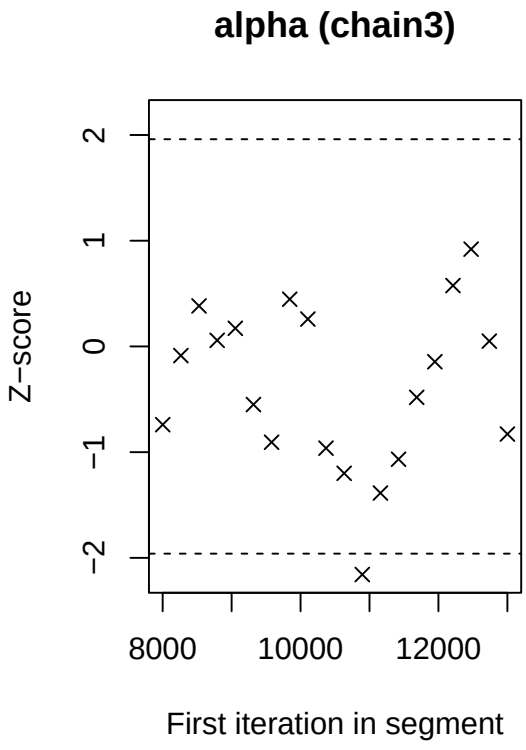
The output from `geweke.diag` shows the summary results for the four generated chains. `geweke.diag` is a convergence diagnostic for Markov chains based on a test for equality of the means of the first and last part of a Markov chain (by default the first 10% and the last 50%). If the samples are drawn 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. 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.

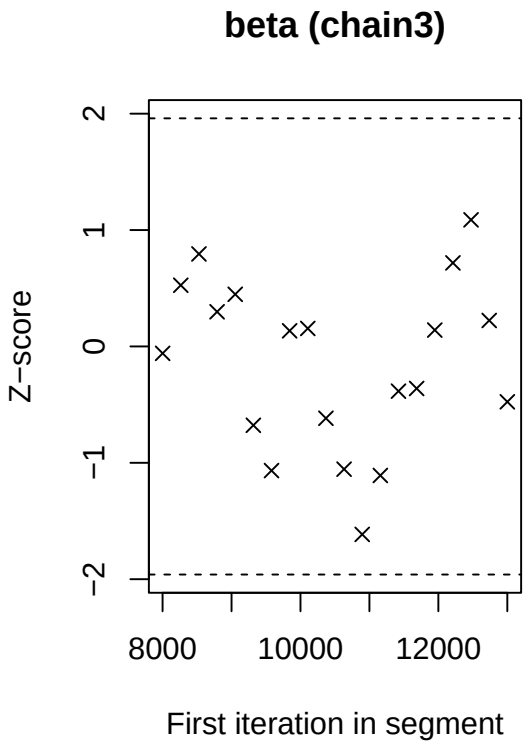
```
geweke.plot(fit1.jags.coda)
```

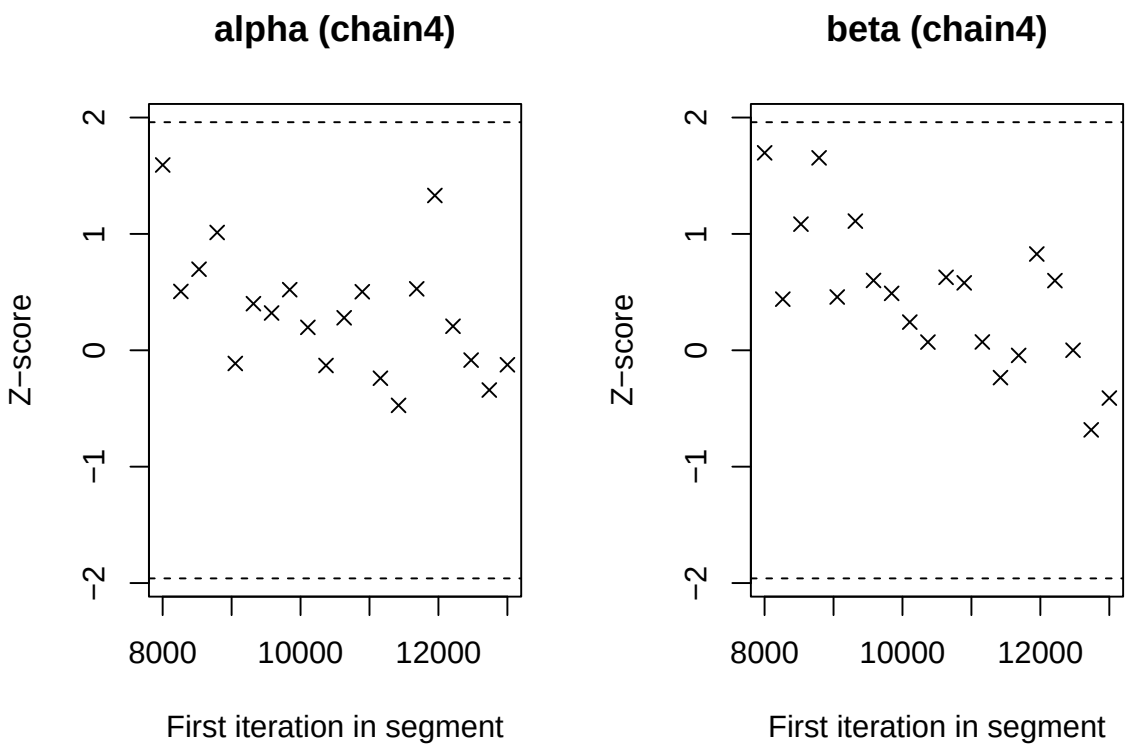



alpha (chain2)
Z−score
2
1
0
−1
−2
8000
10000
12000
First iteration in segment


beta (chain2)
Z−score
2
1
0
−1
−2
8000
10000
12000
First iteration in segment


alpha (chain3)
Z−score
2
1
0
−1
−2
8000
10000
12000
First iteration in segment


beta (chain3)
Z−score
2
1
0
−1
−2
8000
10000
12000
First iteration in segment



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 ($|Z| > 2$).

(d) Gelman and Rubin's convergence diagnostic

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

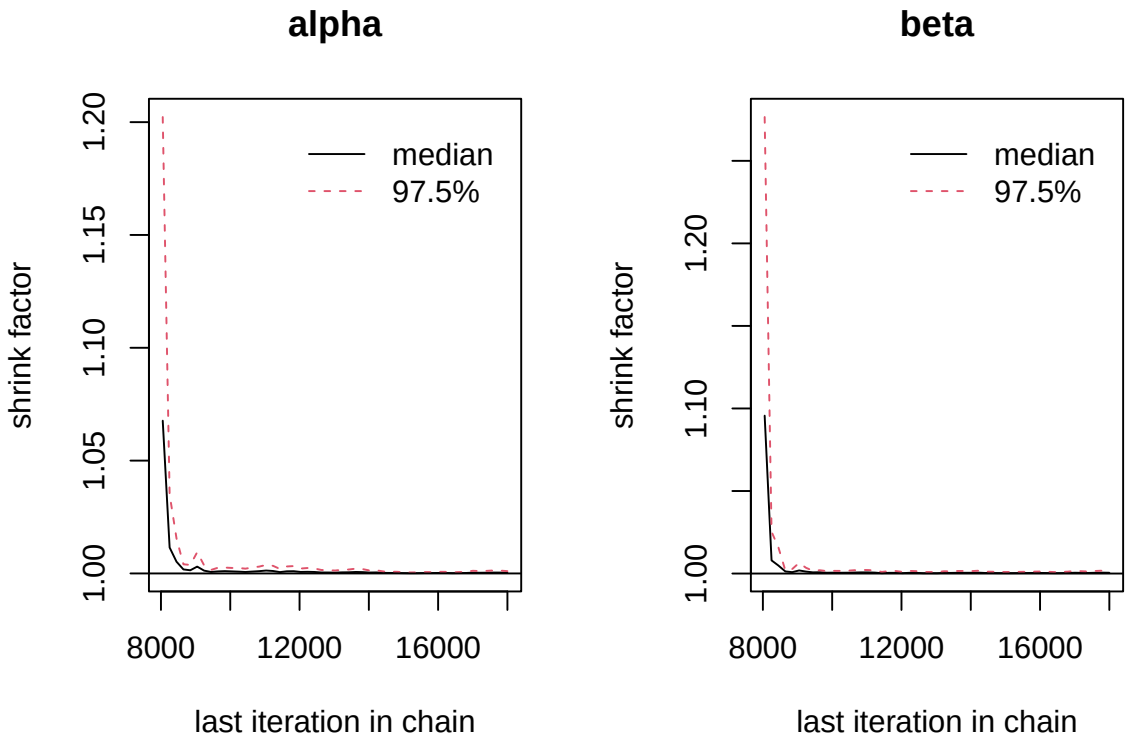
```
## Potential scale reduction factors:
##
##       Point est. Upper C.I.
## alpha          1          1
## beta           1          1
##
## Multivariate psrf
##
## 1
```

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

(e) Gelman-Rubin diagnostic using stable variance estimators

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

```
## $psrf
## [1] 1.000008 1.000009
##
## $mpsrf
## [1] 1.000069
##
## $means
##        alpha         beta
##   -0.9609695 -142.3475523
##
## $n.eff
## [1] 16766.74
##
## $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.

Mutlivariate psrf is also calculated by taking into account the interdependence of the chain's components. Means of variables (alpha and beta) and the effective sample size are also reported in the output.

5.3 Re-run the MCMC simulation

```
## 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: 200
##    Unobserved stochastic nodes: 2
##    Total graph size: 419
##
## Initializing model
```

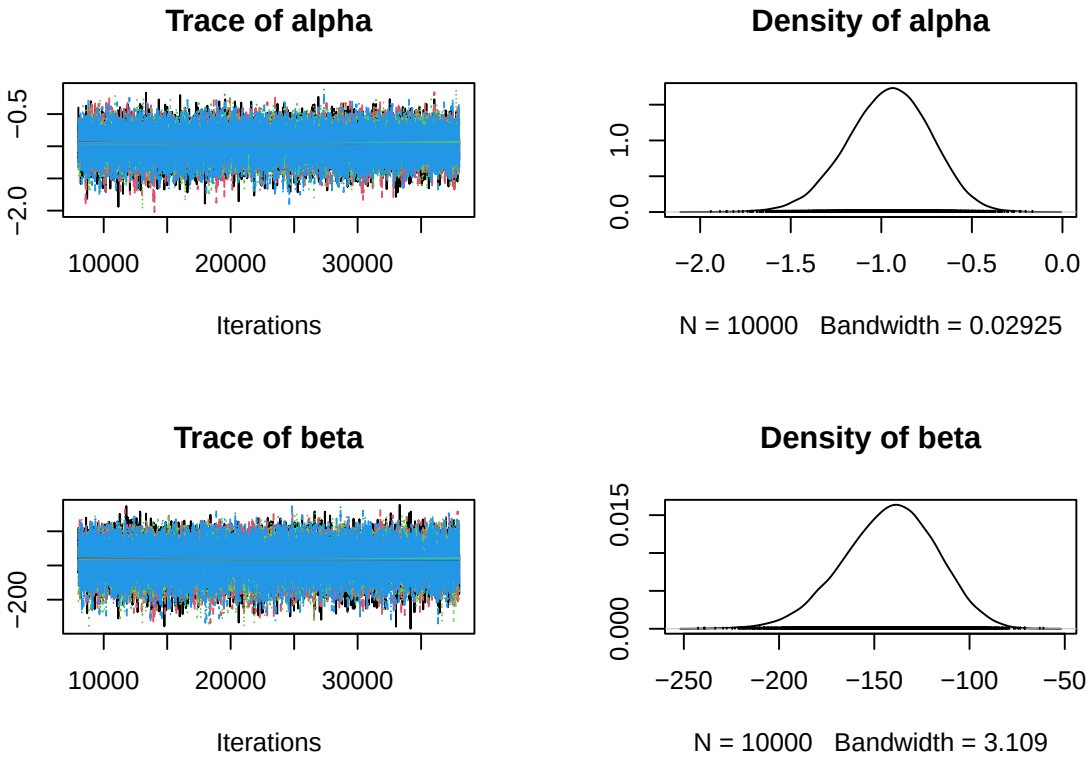
```
## Burn-in (increase burn-in iterations)
update(model2.jags, n.iter = 4000)

## 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)
```

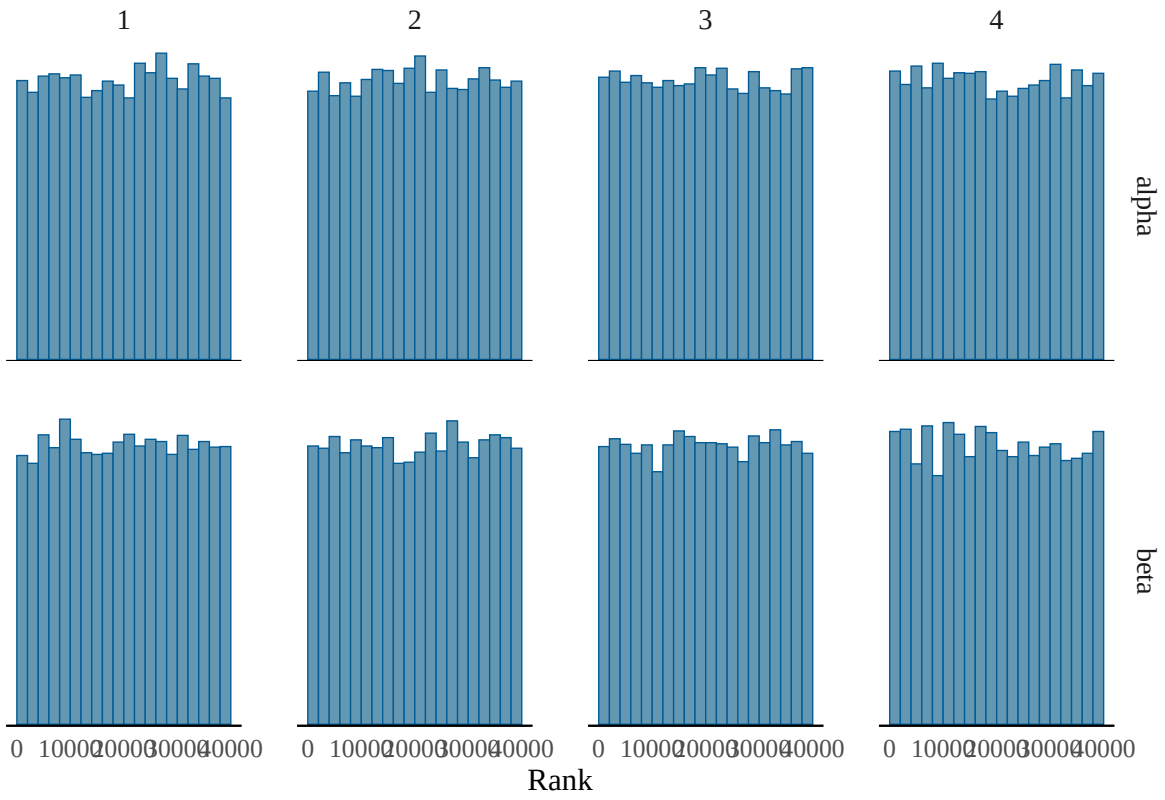
```
##
## Iterations = 8003:38000
## Thinning interval = 3
## Number of chains = 4
## Sample size per chain = 10000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##             Mean      SD Naive SE Time-series SE
## alpha    -0.9585  0.2297 0.001149        0.00144
## beta  -141.9249 24.4204 0.122102        0.15330
##
## 2. Quantiles for each variable:
```

```
##
##           2.5%      25%       50%       75%    97.5%
## alpha   -1.427   -1.109   -0.9509   -0.7995  -0.5306
## beta  -192.369 -157.877 -140.7864 -124.8710 -97.1010
```

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




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




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

```
## [[1]]
##
##       Stationarity start     p-value
##       test         iteration
## alpha passed       1         0.717
## beta  passed       1         0.531
##
##       Halfwidth Mean     Halfwidth
##       test
## alpha passed      -0.958 0.00552
## beta  passed    -141.789 0.58577
##
## [[2]]
##
##       Stationarity start     p-value
##       test         iteration
## alpha passed       1         0.847
## beta  passed       1         0.654
##
##       Halfwidth Mean     Halfwidth
##       test
## alpha passed      -0.957 0.00578
## beta  passed    -141.847 0.61738
##
## [[3]]
##
##       Stationarity start     p-value
##       test         iteration
```

```
## alpha failed         NA      0.00534
## beta  passed       2001      0.07354
##
##       Halfwidth Mean Halfwidth
##       test
## alpha <NA>        NA    NA
## beta  passed    -142 0.642
##
## [[4]]
##
##       Stationarity start     p-value
##       test         iteration
## alpha passed       1         0.748
## beta  passed       1         0.361
##
##       Halfwidth Mean    Halfwidth
##       test
## alpha passed      -0.96 0.00575
## beta  passed    -142.19 0.60495
```

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

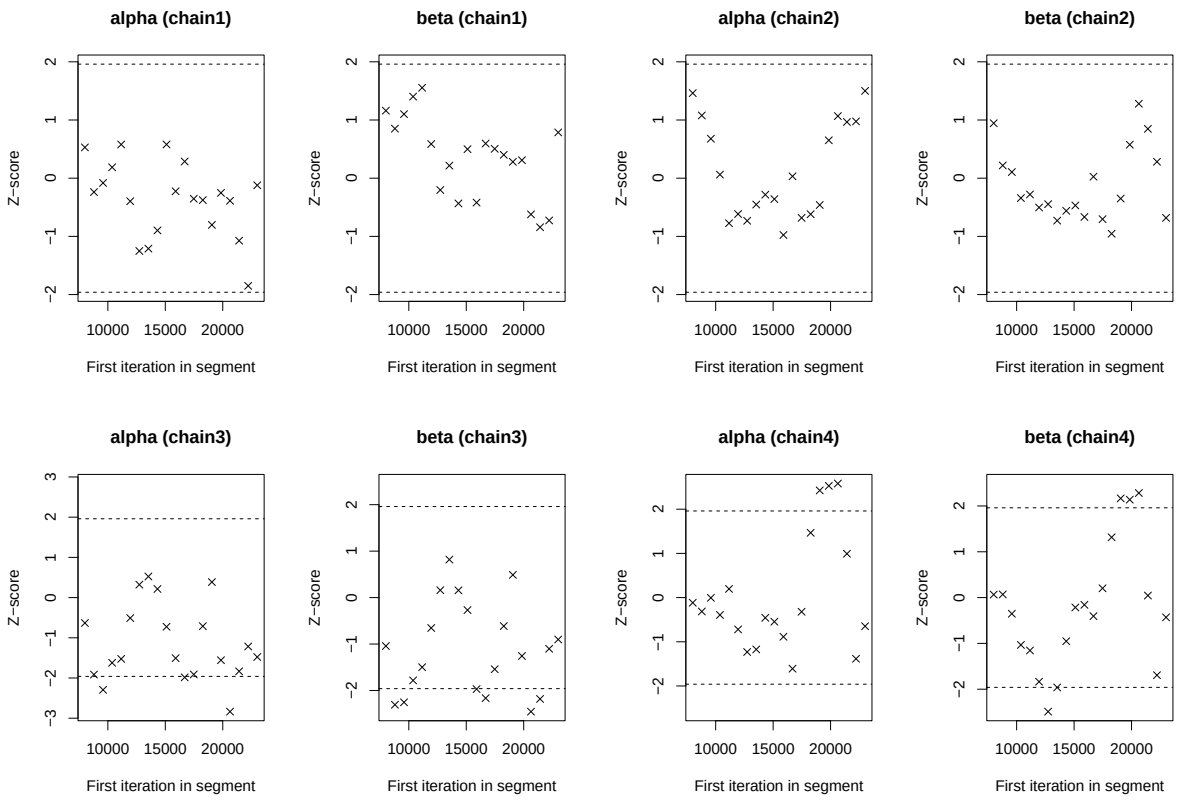
```
## [[1]]
##
## Quantile (q) = 0.025
## Accuracy (r) = +/- 0.005
## Probability (s) = 0.95
##
##        Burn-in  Total Lower bound  Dependence
##        (M)      (N)   (Nmin)       factor (I)
##  alpha 12       14139 3746         3.77
##  beta  9        13338 3746         3.56
##
##
## [[2]]
##
## 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       14139 3746         3.77
##  beta  12       14139 3746         3.77
##
##
## [[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       14280 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        12591 3746         3.36
##  beta  9        13674 3746         3.65
```

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

```
## [[1]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##  alpha   beta
## 0.5294 1.1604
##
##
## [[2]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##  alpha   beta
## 1.4619 0.9448
##
##
## [[3]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##   alpha    beta
## -0.6318 -1.0428
##
##
## [[4]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##    alpha     beta
## -0.11640  0.06524
```

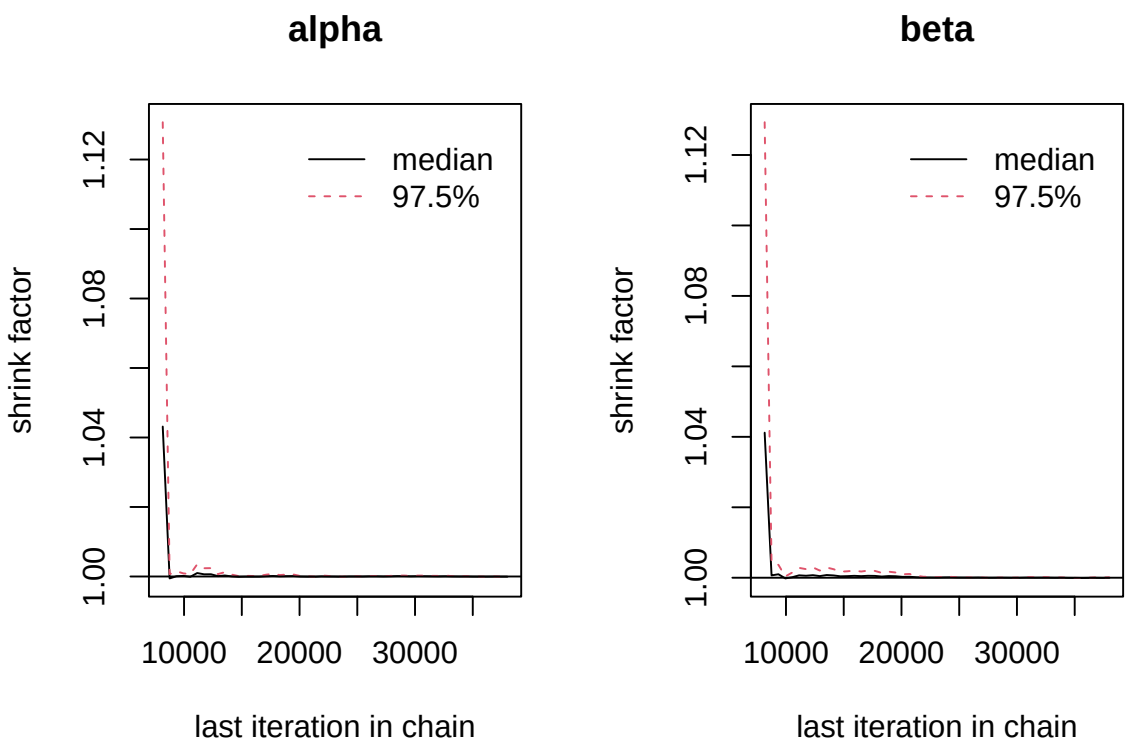
```
geweke.plot(fit2.jags.coda)
```




```
## 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
```

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




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

```
## $psrf
## [1] 0.9999711 0.9999712
##
## $mpsrf
## [1] 1.000017
##
## $means
##        alpha         beta
##   -0.9585019 -141.9221871
##
## $n.eff
## [1] 29704.35
##
## $blather
## [1] FALSE
```