Worksheet 5

Foundations of Bayesian Methodology

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

Spring Semester 2022

Contents

1 Exercise 3 - Normal example in JAGS

Confused finding/declaration: although the set.seed(44566) is implemented in every chunk that will generate random number, the results still change every time. That is the reason why the results and interpretation are NOT consistent with compiled version.

1.1 rjags interface to JAGS

The steps of `list.factories` and `set.factories` show and control over the status of factories in JAGS modules. The steps of `jags.modules` shows the names of the currently loaded modules and also loads or unloads JAGS modules.

```
remove(list=ls())
set.seed(44566)
# set the path to the 05normal_exmple_JAGS.txt file
path <- ""

suppressPackageStartupMessages(library(rjags))
list.factories(type = "rng")
```

```
##         factory status
## 1 base::BaseRNG   TRUE
```

```
list.factories(type = "monitor")
```

```
##          factory status
## 1 base::Variance   TRUE
## 2     base::Mean   TRUE
## 3    base::Trace   TRUE
```

```
list.factories(type = "sampler")
```

```
##             factory status
## 1  bugs::BinomSlice   TRUE
## 2         bugs::RW1   TRUE
## 3    bugs::Censored   TRUE
## 4         bugs::Sum   TRUE
## 5        bugs::DSum   TRUE
## 6   bugs::Conjugate   TRUE
## 7   bugs::Dirichlet   TRUE
## 8     bugs::MNormal   TRUE
## 9      base::Finite   TRUE
## 10      base::Slice   TRUE
```

```
set.factory(name = "base::Slice", type = "sampler", state = FALSE)
```

```
## NULL
```

```
list.factories(type = "sampler")
```

```
##             factory status
## 1  bugs::BinomSlice   TRUE
## 2         bugs::RW1   TRUE
## 3    bugs::Censored   TRUE
## 4         bugs::Sum   TRUE
## 5        bugs::DSum   TRUE
## 6   bugs::Conjugate   TRUE
## 7   bugs::Dirichlet   TRUE
## 8     bugs::MNormal   TRUE
## 9      base::Finite   TRUE
## 10      base::Slice  FALSE
```

```
set.factory(name = "base::Slice", type = "sampler", state = TRUE)
```

```
## NULL
```

```
list.factories(type = "sampler")
```

```
##              factory status
## 1  bugs::BinomSlice   TRUE
## 2         bugs::RW1   TRUE
## 3    bugs::Censored   TRUE
## 4         bugs::Sum   TRUE
## 5        bugs::DSum   TRUE
## 6   bugs::Conjugate   TRUE
## 7   bugs::Dirichlet   TRUE
## 8     bugs::MNormal   TRUE
## 9      base::Finite   TRUE
## 10      base::Slice   TRUE
```

```
list.modules()
```

```
## [1] "basemod" "bugs"
```

```
load.module("glm")
```

```
## module glm loaded
```

```
list.modules()
```

```
## [1] "basemod" "bugs"    "glm"
```

```
unload.module("glm")
```

```
## Module glm unloaded
```

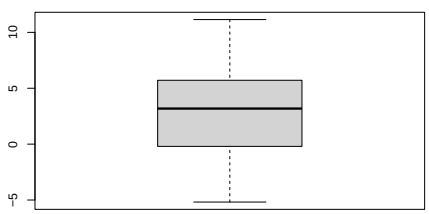
```
list.modules()
```

```
## [1] "basemod" "bugs"
```

1.1.1 Introduction

The basic setup of normal example is as following codes

```
y <- c(3.048,2.980,2.029,7.249,-0.259,3.061,4.059,6.370,7.902,1.926,
       9.094,10.489,-0.384,-3.096,2.315,5.830,-1.542,-1.544,5.714,
       -5.182,3.828,-4.038,2.169,5.087,-0.201,4.880,3.302,3.859,
       11.144,5.564)
par(mfrow = c(1, 1))
boxplot(y)
```



```
summary(y)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -5.1820  0.3307  3.1815  3.1884  5.6765 11.1440
```

```
sd(y)
```

```
## [1] 4.046692
```

```
# Define the parameters of the prior distributions
mu0 <- -3
sigma2_0 <- 4
a0 <- 1.6
b0 <- 0.4
```

Boxplot shows that the median of observed data is around 3, the 25% quantile and 75% quantile are around 0 and 5, respectively. In addition, the minimum and maximum are around -5 and 11, respectively.

1.1.2 INLA exact result (motivation)

```
suppressPackageStartupMessages(library(INLA))
formula <- y ~ 1
inla.output <- inla(formula,data=data.frame(y=y),
                    control.family = list(hyper =
                                            list(prec = list(prior="loggamma",
                                                             param=c(a0,b0)))),
                    control.fixed = list(mean.intercept=mu0,
                                         prec.intercept=1/sigma2_0))
```

1.2 Step 2: JAGS model file as a string in rjags with coda

This step gives observed values, sets up initial values and specifies model.

```
set.seed(44566)
suppressPackageStartupMessages(library(rjags))
suppressPackageStartupMessages(library(coda))
wb_data <- list( N=30,
                 y=c(3.048,2.980,2.029,7.249,-0.259,3.061,4.059,6.370,7.902,1.926,
                     9.094,10.489,-0.384,-3.096,2.315,5.830,-1.542,-1.544,5.714,
                     -5.182,3.828,-4.038,2.169,5.087,-0.201,4.880,3.302,3.859,
                     11.144,5.564)
)
wb_inits <- list( mu=-0.2381084, inv_sigma2=0.3993192 )
modelString = " # open quote for modelString
model{
# likelihood
for (i in 1:N){
y[i] ~ dnorm( mu, inv_sigma2 )
}
# Priors
mu ~ dnorm( -3, 0.25 ) # prior for mu N(mu0, prec=1/sigma2_0)
inv_sigma2 ~ dgamma( 1.6, 0.4 ) # prior for precision G(a0, b0)
```

```
# transformations
# deterministic definition of variance
sigma2 <- 1/inv_sigma2

# deterministic definition of standard deviation
sigma <- sqrt(sigma2)
}
" # close quote for modelString
writeLines(modelString, con="TempModelexe3.txt") # write to a file
```

1.3 JAGS only one chain

```
suppressPackageStartupMessages(library(MASS))
suppressPackageStartupMessages(library(FNN))
# model initiation
set.seed(44566)
model.jags <- jags.model(
  file = "TempModelexe3.txt",
  data = wb_data,
  inits = wb_inits,
  n.chains = 1,
  n.adapt = 4000
)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 30
##    Unobserved stochastic nodes: 2
##    Total graph size: 41
##
## Initializing model
```

```
str(model.jags)
```

```
## List of 8
##  $ ptr      :function ()
##  $ data     :function ()
##  $ model    :function ()
##  $ state    :function (internal = FALSE)
##  $ nchain   :function ()
##  $ iter     :function ()
##  $ sync     :function ()
##  $ recompile:function ()
##  - attr(*, "class")= chr "jags"
```

```
class(model.jags)
```

```
## [1] "jags"
```

```
attributes(model.jags)
```

```
## $names
## [1] "ptr"       "data"      "model"     "state"     "nchain"    "iter"
## [7] "sync"      "recompile"
##
## $class
## [1] "jags"
```

```
list.samplers(model.jags)
```

```
## $`bugs::ConjugateNormal`
## [1] "mu"
##
## $`bugs::ConjugateGamma`
## [1] "inv_sigma2"
```

```
update(model.jags, n.iter = 4000) # burn-in
# sampling
fit.jags.coda <- coda.samples(
  model = model.jags,
  variable.names = c("mu", "sigma2", "inv_sigma2"),
  n.iter = 10000,
  thin = 1
)
str(fit.jags.coda)
```

```
## List of 1
##  $ : 'mcmc' num [1:10000, 1:3] 0.0582 0.0698 0.0997 0.0455 0.0795 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:3] "inv_sigma2" "mu" "sigma2"
##   ..- attr(*, "mcpar")= num [1:3] 4001 14000 1
##  - attr(*, "class")= chr "mcmc.list"
```

```
class(fit.jags.coda)
```

```
## [1] "mcmc.list"
```

```
attributes(fit.jags.coda)
```

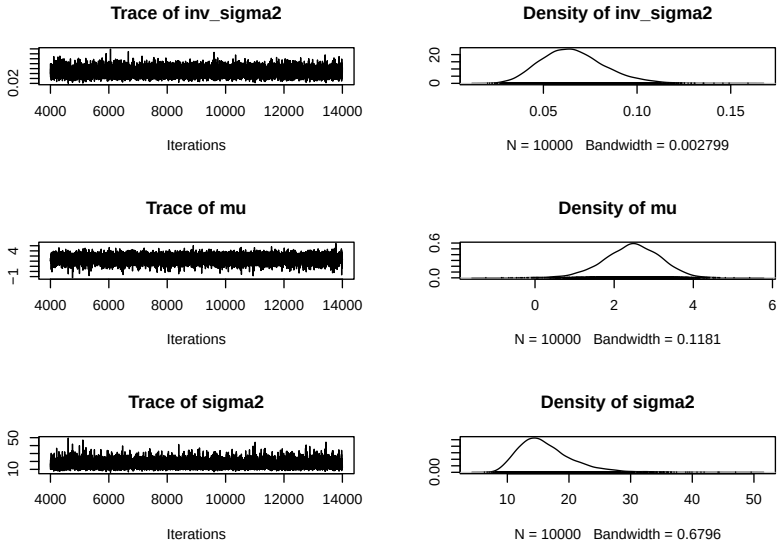
```
## $class
## [1] "mcmc.list"
```

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

```
##
## Iterations = 4001:14000
## Thinning interval = 1
## Number of chains = 1
## 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
```

```
## inv_sigma2  0.06584 0.01691 0.0001691      0.0001806
## mu           2.45066 0.71390 0.0071390      0.0076697
## sigma2      16.27022 4.51878 0.0451878      0.0495859
##
## 2. Quantiles for each variable:
##
##                2.5%      25%      50%     75%   97.5%
## inv_sigma2 0.03681  0.05398  0.06463  0.0763  0.1027
## mu         0.98370  1.99674  2.47552  2.9388  3.7624
## sigma2     9.73811 13.10550 15.47347 18.5261 27.1659
```

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



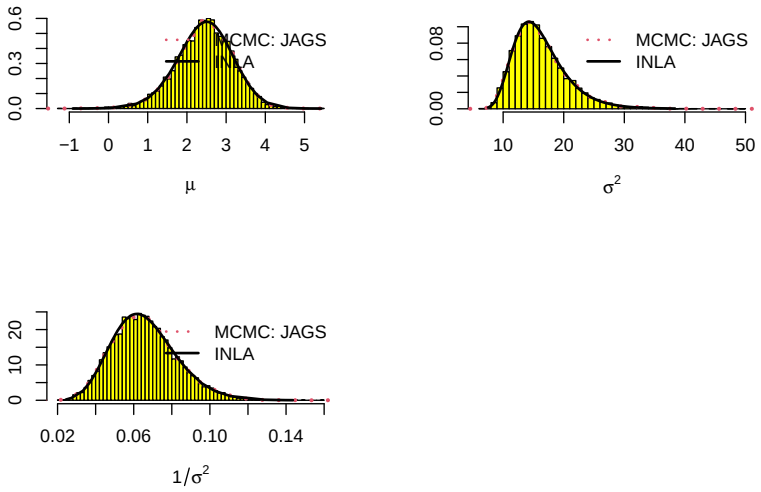

```
# store samples for each parameter from the chain into separate objects
m.fit.jags.coda <- as.matrix(fit.jags.coda)
mu.sim <- m.fit.jags.coda[,"mu"]
sigma2.sim <- m.fit.jags.coda[,"sigma2"]
inv_sigma2.sim <- m.fit.jags.coda[,"inv_sigma2"]

par(mfrow=c(2,2))
# plot for mean
rg <- range(inla.output$marginals.fixed$"(Intercept)"[,2])
truehist(mu.sim, prob=TRUE, col="yellow", xlab=expression(mu),ylim=rg)
lines(density(mu.sim),lty=3,lwd=3, col=2)
lines(inla.output$marginals.fixed$"(Intercept)",lwd=2)
legend("topright",c("MCMC: JAGS","INLA"),lty=c(3,1),
       lwd=c(2,2),col=c(2,1),cex=1.0,bty="n")
KL.divergence(inla.rmarginal(length(mu.sim), inla.output$marginals.fixed[[1]]),
              mu.sim, k = 1)
```

```
## [1] -0.007744871
```

```
# plot for variance
m_var <-inla.tmarginal(function(x) 1/x, inla.output$marginals.hyperpar[[1]])
rg <- range(m_var[,2])
truehist(sigma2.sim, prob=TRUE, col="yellow", xlab=expression(sigma^2),ylim=rg)
lines(density(sigma2.sim),lty=3,lwd=3, col=2)
```

```
lines(m_var,lwd=2)
legend("topright",c("MCMC: JAGS","INLA"),lty=c(3,1),lwd=c(2,2),col=c(2,1),
       cex=1.0,bty="n")
# plot for precision
truehist(inv_sigma2.sim, prob=TRUE, col="yellow", xlab=expression(1/sigma^2))
lines(density(inv_sigma2.sim),lty=3,lwd=3, col=2)
lines(inla.output$marginals.hyperpar[[1]],lwd=2)
legend("topright",c("MCMC: JAGS","INLA"),lty=c(3,1),lwd=c(2,2),col=c(2,1),
       cex=1.0,bty="n")
KL.divergence(inla.rmarginal(length(mu.sim),
                             inla.output$marginals.hyperpar[[1]]),
              inv_sigma2.sim, k = 1)
```

```
## [1] -0.00714838
```



The empirical mean and standard deviation for each variable and corresponding quantiles are shown in the output. The traceplot as well as the plot of density indicate good mixing for three different parameters: precision, mean and variance. When compared to the result of INLA, whether plots or the results of Kullback-Leibler divergence proves the similarity between the output of INLA and JAGS.

1.4 JAGS several chains

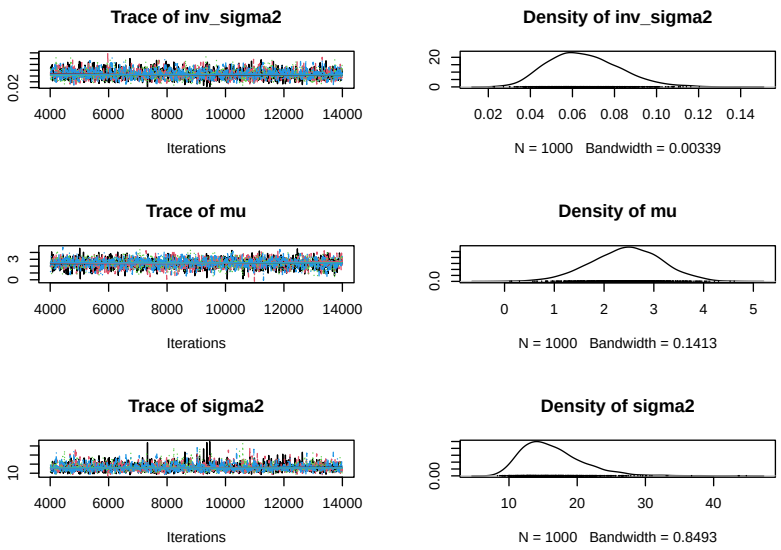
```
set.seed(44566)
wb_inits <- function() {
  list(mu = rnorm(1),
       inv_sigma2 = runif(1)
  )
}
# model initialisation
model.jags <- jags.model(
  file = "TempModelexe3.txt",
  data = wb_data,
  inits = wb_inits,
  n.chains = 4,
  n.adapt = 4000
)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 30
##    Unobserved stochastic nodes: 2
##    Total graph size: 41
##
## Initializing model
```

```
update(model.jags, n.iter = 4000) # burn-in
# sampling/monitoring
fit.jags.coda <- coda.samples(
  model = model.jags,
  variable.names = c("mu", "sigma2", "inv_sigma2"),
  n.iter = 10000,
  thin = 10
)
summary(fit.jags.coda)
```

```
##
## Iterations = 4010:14000
## 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
## inv_sigma2  0.06551 0.0168 0.0002656      0.0002679
## mu          2.44835 0.7004 0.0110737      0.0111218
## sigma2     16.35025 4.5271 0.0715801      0.0707483
##
## 2. Quantiles for each variable:
##
##               2.5%     25%      50%      75%   97.5%
## inv_sigma2 0.03737  0.0533  0.06419  0.07622  0.1016
## mu         1.04499  1.9790  2.46831  2.93248  3.7827
## sigma2     9.83847 13.1201 15.57934 18.76022 26.7627
```

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

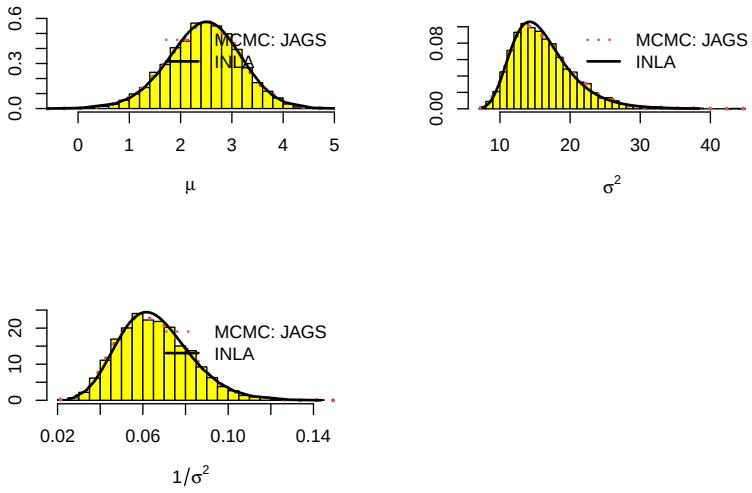



```
# store samples for each parameter from the chains into separate vectors
m.fit.jags.coda <-as.matrix(fit.jags.coda)
mu.sim <- m.fit.jags.coda[,"mu"]
sigma2.sim <- m.fit.jags.coda[,"sigma2"]
inv_sigma2.sim <- m.fit.jags.coda[,"inv_sigma2"]
par(mfrow=c(2,2))
# plot for mean
rg <- range(inla.output$marginals.fixed$"(Intercept)"[,2])
truehist(mu.sim, prob=TRUE, col="yellow", xlab=expression(mu),ylim=rg)
lines(density(mu.sim),lty=3,lwd=3, col=2)
lines(inla.output$marginals.fixed$"(Intercept)",lwd=2)
legend("topright",c("MCMC: JAGS","INLA"),lty=c(3,1),lwd=c(2,2),col=c(2,1),
       cex=1.0,bty="n")
KL.divergence(inla.rmarginal(length(mu.sim), inla.output$marginals.fixed[[1]]),
              mu.sim, k = 1)
```

```
## [1] 0.05547802
```

```
# plot for variance
m_var <-inla.tmarginal(function(x) 1/x, inla.output$marginals.hyperpar[[1]])
rg <- range(m_var[,2])
truehist(sigma2.sim, prob=TRUE, col="yellow", xlab=expression(sigma^2),ylim=rg)
lines(density(sigma2.sim),lty=3,lwd=3, col=2)
lines(m_var,lwd=2)
legend("topright",c("MCMC: JAGS","INLA"),lty=c(3,1),lwd=c(2,2),col=c(2,1),
       cex=1.0,bty="n")
# plot for precision
truehist(inv_sigma2.sim, prob=TRUE, col="yellow", xlab=expression(1/sigma^2))
lines(density(inv_sigma2.sim),lty=3,lwd=3, col=2)
lines(inla.output$marginals.hyperpar[[1]],lwd=2)
legend("topright",c("MCMC: JAGS","INLA"),lty=c(3,1),lwd=c(2,2),col=c(2,1),
       cex=1.0,bty="n")
KL.divergence(inla.rmarginal(length(mu.sim),
                             inla.output$marginals.hyperpar[[1]]),
              inv_sigma2.sim, k = 1)
```

```
## [1] 0.06334182
```



The empirical mean and standard deviation for each variable and corresponding quantiles are shown in the output. The result of four chains is similar as the result of one chain: From traceplot and the plot of density, we can find good mixing for three parameters; INLA and JAGS have similar results.

1.4.1 CODA

```
effectiveSize(fit.jags.coda)
```

```
## inv_sigma2         mu     sigma2
##   3936.568   3991.371   4100.741
```

```
lapply(fit.jags.coda, effectiveSize)
```

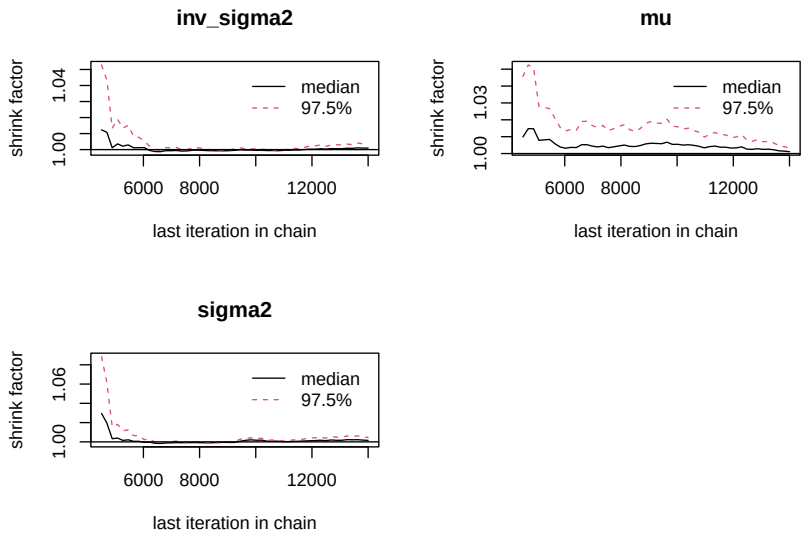
```
## [[1]]
## inv_sigma2         mu     sigma2
##   1000.000   1093.052   1000.000
##
## [[2]]
## inv_sigma2         mu     sigma2
##       1000       1000       1000
##
## [[3]]
## inv_sigma2         mu     sigma2
##   936.5681  1000.0000  1100.7410
##
## [[4]]
## inv_sigma2         mu     sigma2
##  1000.0000   898.3188  1000.0000
```

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

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

```
## inv_sigma2          1          1
## mu                  1          1
## sigma2              1          1
##
## Multivariate psrf
##
## 1
```

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

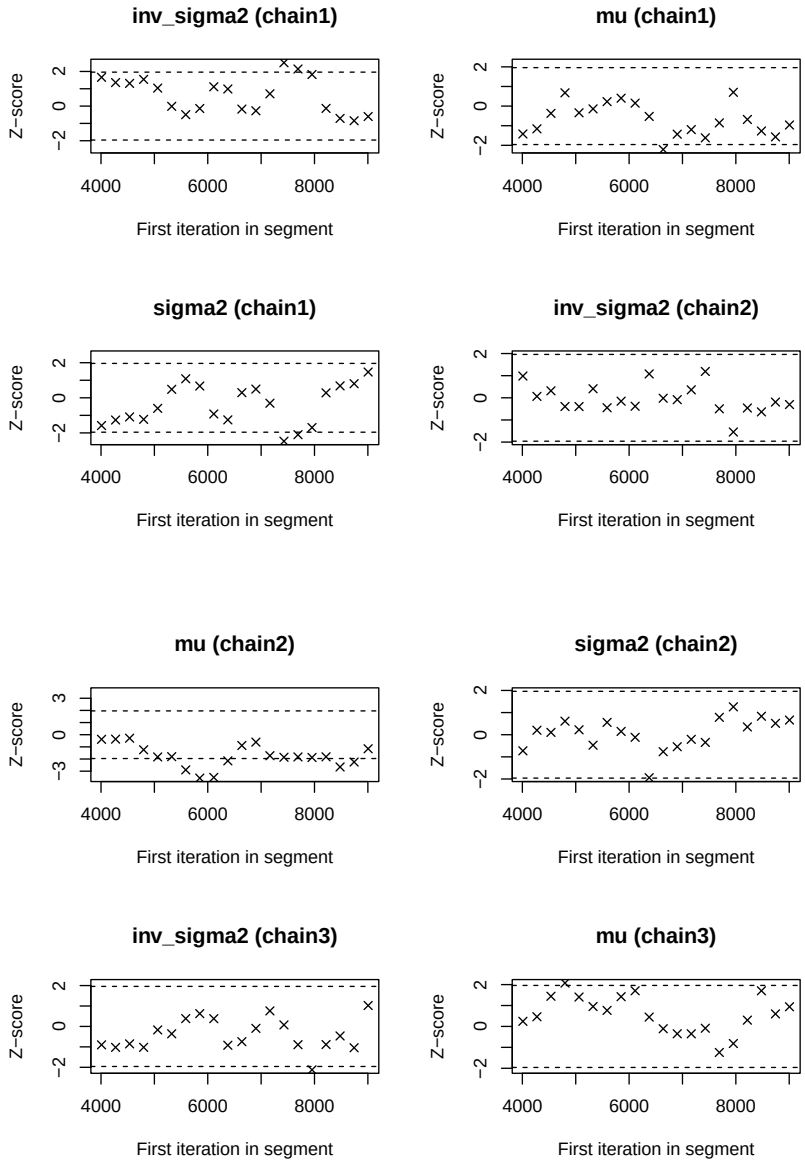


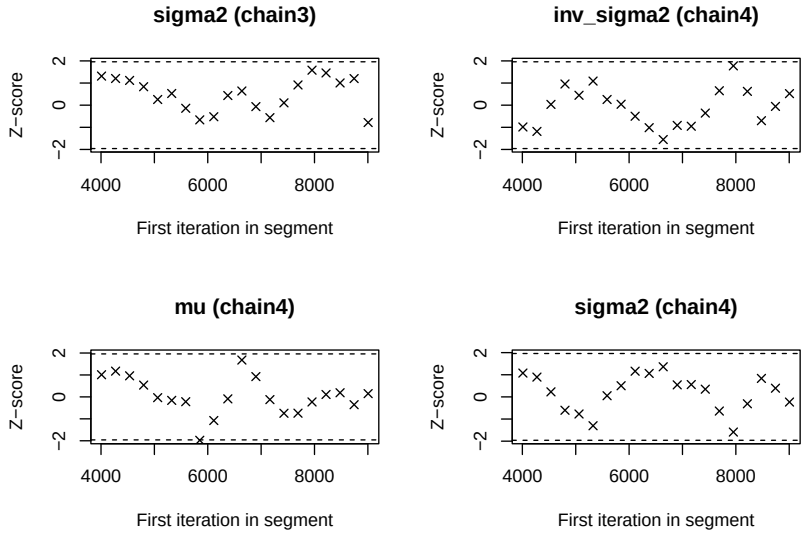

```
geweke.diag(fit.jags.coda)
```

```
## [[1]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
## inv_sigma2         mu     sigma2
##      1.661     -1.421     -1.580
##
##
## [[2]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
## inv_sigma2         mu     sigma2
##     0.9837    -0.3785    -0.7277
##
##
## [[3]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
## inv_sigma2         mu     sigma2
##    -0.9014     0.2443     1.3131
```

```
##
##
## [[4]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
## inv_sigma2         mu     sigma2
##    -0.9848     1.0110     1.0768
```

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






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

```
## [[1]]
##
##            Stationarity start     p-value
##            test         iteration
## inv_sigma2 passed       101       0.130
## mu         passed         1       0.297
## sigma2     passed         1       0.170
##
##            Halfwidth Mean    Halfwidth
##            test
## inv_sigma2 passed     0.0647 0.00109
## mu         passed     2.4056 0.03981
## sigma2     passed    16.5144 0.28623
##
## [[2]]
##
##            Stationarity start     p-value
##            test         iteration
## inv_sigma2 passed        1        0.84536
## mu         failed       NA        0.00445
## sigma2     passed        1        0.78726
##
##            Halfwidth Mean    Halfwidth
##            test
## inv_sigma2 passed     0.0657 0.00107
## mu         <NA>          NA      NA
## sigma2     passed    16.3508 0.28405
##
## [[3]]
##
##            Stationarity start     p-value
##            test         iteration
## inv_sigma2 passed       1         0.734
## mu         passed       1         0.437
## sigma2     passed       1         0.580
##
```

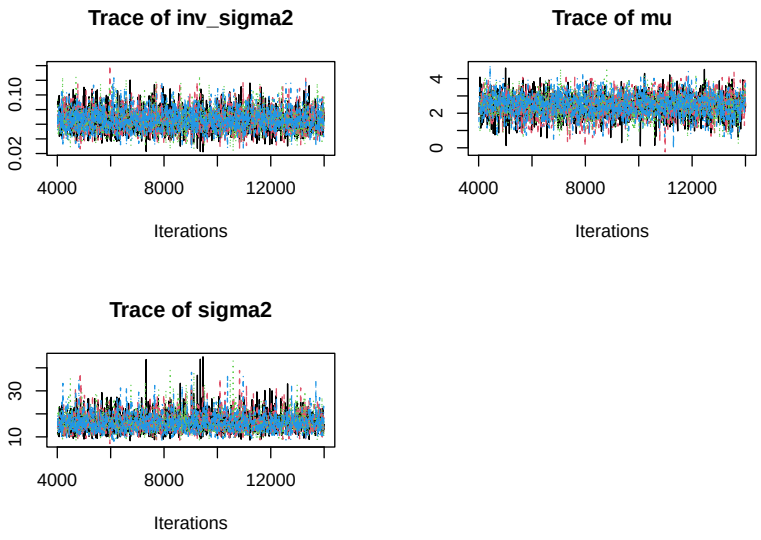
```
##            Halfwidth Mean    Halfwidth
##            test
## inv_sigma2 passed     0.0656 0.00108
## mu         passed     2.4872 0.04320
## sigma2     passed    16.3385 0.26954
##
## [[4]]
##
##            Stationarity start     p-value
##            test         iteration
## inv_sigma2 passed       1         0.855
## mu         passed       1         0.744
## sigma2     passed       1         0.608
##
##            Halfwidth Mean    Halfwidth
##            test
## inv_sigma2 passed     0.0658 0.00101
## mu         passed     2.4599 0.04662
## sigma2     passed    16.1973 0.26906
```

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

```
## [[1]]
##
## Quantile (q) = 0.025
## Accuracy (r) = +/- 0.005
## Probability (s) = 0.95
##
## You need a sample size of at least 3746 with these values of q, r and s
##
## [[2]]
##
## Quantile (q) = 0.025
## Accuracy (r) = +/- 0.005
## Probability (s) = 0.95
##
## You need a sample size of at least 3746 with these values of q, r and s
##
## [[3]]
##
## Quantile (q) = 0.025
## Accuracy (r) = +/- 0.005
## Probability (s) = 0.95
##
## You need a sample size of at least 3746 with these values of q, r and s
##
## [[4]]
##
## Quantile (q) = 0.025
## Accuracy (r) = +/- 0.005
## Probability (s) = 0.95
##
## You need a sample size of at least 3746 with these values of q, r and s
```

```
par(mfrow=c(2,2))
coda:::traceplot(fit.jags.coda)
```

```
# "DIC" penalised expected deviance computation
dic1<-dic.samples(model=model.jags, n.iter=1000, type="popt")
```



ESS: The Effective Sample Size of three parameters are shown in the table. The ESS must be large enough to get stable inferences for quantities of interest. In the case of out of 10000 iterations, the global ESS around 4000 are large enough.

	Global	1st Chain	2nd Chain	3rd Chain	4th Chain
Precision	3863.425	981.847	1000.000	881.578	1000
Mean	4099.911	1199.804	900.106	1000.000	1000
Variance	3931.376	931.376	1000.000	1000.000	1000

Table 1: Effective Sample Size

Gelman-Rubin-Brooks: The estimated shrink factors and corresponding upper CI of three parameters of Gelman and Rubin's convergence diagnostic are within the acceptable range. In addition, Gelman-Rubin-Brooks plot shows the evolution of shrink factor as the number of iterations increases. The trends of the median of shrink factors are mostly stable among three parameters in the range of 6000 to 14000 iterations.

Geweke: The results of Geweke's convergence diagnostic for 4 chaines are all in the range of -2 to 2. However, the Geweke-Brooks plot indicates that the 4th chain has more samples who holds a Z-score that out of the range from -2 to 2, which proves that the 4th chain has not reached equilibrium.

Heidelberger & Welch: Heidelberger and Welch's convergence diagnostic shows a similar result as Geweke-Brooks plot. The parameter of variance in the 4th chain fails stationary test indicating non convergence therefore a longer MCMC run is needed. Except of this parameter, all parameters also pass the halfwidth test that means the length of the sample is long enough to estimate the mean with sufficient accuracy

Raftery & Lewis Raftery and Lewis's diagnostic gives the he minimum sample size based on zero autocorrelation. All the four chains need a minimum sample size of 3746.

Trace plot The traceplots show agian that all the four chains of three parameters hold good mixing.

1.5 Additional sampling in several chains, preparation for BGR/Gelman with runjags

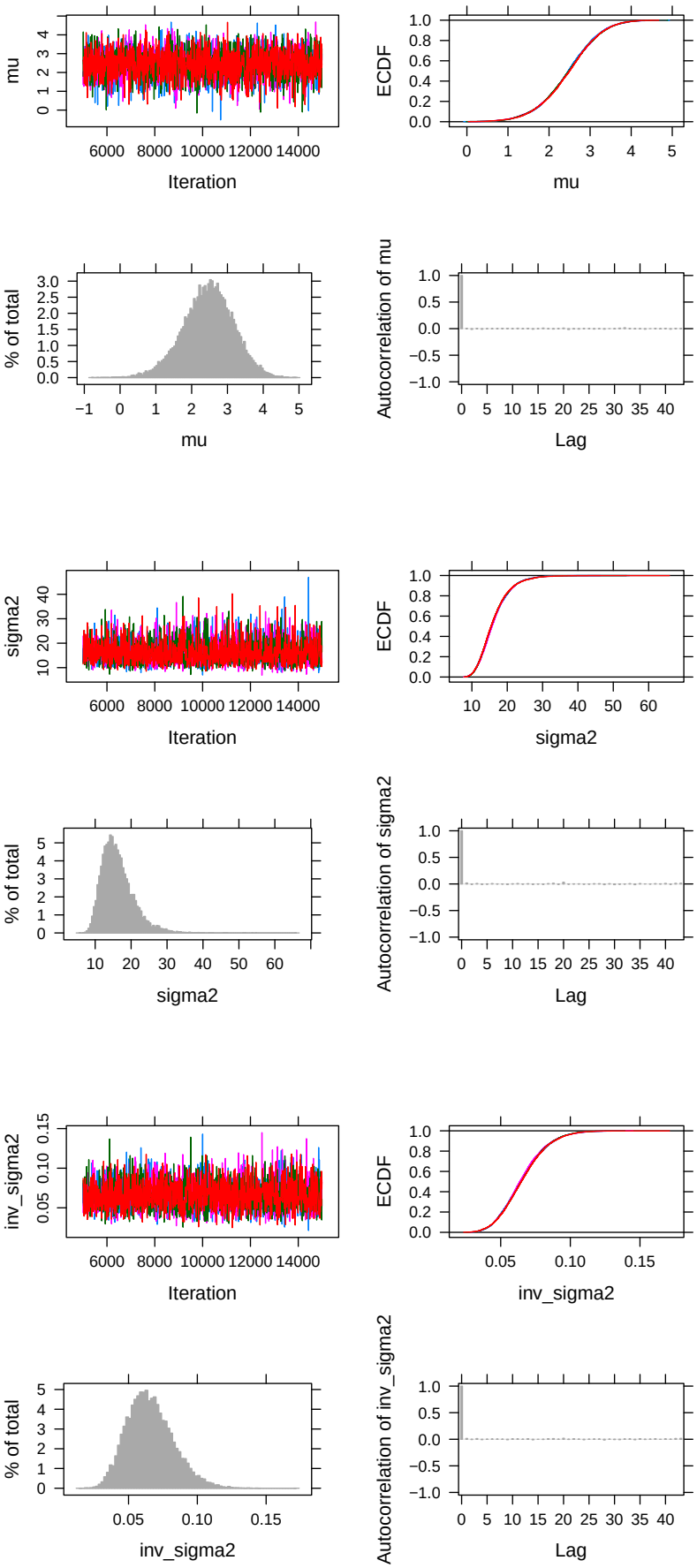
1.5.1 runjags interface with a link to a file

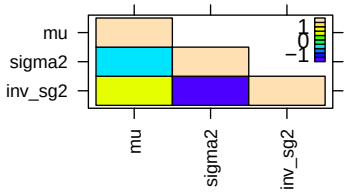
```
suppressPackageStartupMessages(library(runjags))
set.seed(44566)
wb_data <- list( N=30,
                 y=c(3.048,2.980,2.029,7.249,-0.259,3.061,4.059,6.370,7.902,1.926,
                     9.094,10.489,-0.384,-3.096,2.315,5.830,-1.542,-1.544,5.714,
                     -5.182,3.828,-4.038,2.169,5.087,-0.201,4.880,3.302,3.859,
                     11.144,5.564)
)
wb_inits <- function() {
  list(mu = rnorm(1),
       inv_sigma2 = runif(1)
  )
}
fit.runjags<-run.jags(model=paste(path,"05normal_exmple_JAGS.txt",sep=""),
                      monitor=c("mu", "sigma2", "inv_sigma2"),
                      data=wb_data,
                      inits=wb_inits,
                      n.chains=4,
                      burnin=4000,
                      sample=5000,
                      adapt=1000,
                      thin=2)
```

```
## Compiling rjags model...
## Calling the simulation using the rjags method...
## Note: the model did not require adaptation
## Burning in the model for 4000 iterations...
## Running the model for 10000 iterations...
## Simulation complete
## Calculating summary statistics...
## Calculating the Gelman-Rubin statistic for 3 variables....
## Finished running the simulation
```

```
plot(fit.runjags)
```

```
## Generating plots...
```


mu
Iteration
ECDF
mu
% of total
mu
Autocorrelation of mu
Lag
sigma2
Iteration
ECDF
sigma2
% of total
sigma2
Autocorrelation of sigma2
Lag
inv_sigma2
Iteration
ECDF
inv_sigma2
% of total
inv_sigma2
Autocorrelation of inv_sigma2
Lag




```
print(fit.runjags)
```

```
##
## JAGS model summary statistics from 20000 samples (thin = 2; chains = 4; adapt+burnin = 5000):
##
##             Lower95   Median  Upper95     Mean       SD Mode      MCerr MC%ofSD
## mu            1.022   2.4861   3.8188   2.4608  0.70895   --  0.0049784     0.7
## sigma2       8.9822   15.548   25.313    16.29   4.5207   --   0.032795     0.7
## inv_sigma2 0.033409 0.064318 0.098241 0.065734 0.016864   -- 0.00012082     0.7
##
##            SSeff      AC.20    psrf
## mu         20279 -0.0013516 0.99998
## sigma2     19002  0.0030124  1.0001
## inv_sigma2 19481  0.0064424  1.0001
##
## Total time taken: 0.8 seconds
```

```
# CODA
fit.runjags.coda<-as.mcmc.list(fit.runjags)
summary(fit.runjags.coda)
```

```
##
## Iterations = 5001:14999
## Thinning interval = 2
## Number of chains = 4
## Sample size per chain = 5000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                Mean      SD  Naive SE Time-series SE
## mu          2.46082 0.70895 0.0050130      0.0049803
## sigma2     16.29031 4.52069 0.0319661      0.0328513
## inv_sigma2  0.06573 0.01686 0.0001192      0.0001209
##
## 2. Quantiles for each variable:
##
```

```
##                2.5%      25%      50%      75%   97.5%
## mu          1.00086  2.00693  2.48614  2.93601  3.8056
## sigma2      9.74068 13.10828 15.54778 18.60860 27.3611
## inv_sigma2 0.03655  0.05374  0.06432  0.07629  0.1027
```

```
# conduct CODA
```

Using `plot` function except ordinary plots that appeared before, autocorrelation plots and a cross-correlation plot are produced.

Autocorrelation plots give an overview of autocorrelation in different thinning interval. For all three parameters, when the lag is larger than 1, the autocorrelation is close to 0.

Cross-correlation plot shows the correlation between parameters. The correlation between variance and mean is close to 0, while variance and precision hold an inverse correlation.

1.5.2 R2jags wrapper to rjags interface to JAGS several chains

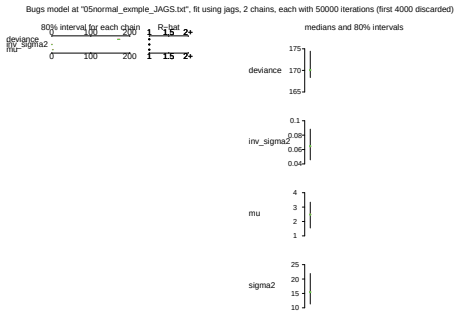
```
suppressPackageStartupMessages(library(R2jags))
set.seed(44566)
wb_data <- list( N=30,
                 y=c(3.048,2.980,2.029,7.249,-0.259,3.061,4.059,6.370,7.902,1.926,
                     9.094,10.489,-0.384,-3.096,2.315,5.830,-1.542,-1.544,5.714,
                     -5.182,3.828,-4.038,2.169,5.087,-0.201,4.880,3.302,3.859,
                     11.144,5.564)
)
#define parameters
params<-c("mu", "sigma2", "inv_sigma2")
# define inits
inits1 <- list(mu=rnorm(1), inv_sigma2=runif(1),
               .RNG.name="base::Super-Duper", .RNG.seed=1)
inits2 <- list(mu=rnorm(1), inv_sigma2=runif(1),
               .RNG.name="base::Wichmann-Hill", .RNG.seed=2)
wb_inits <- list(inits1,inits2)
fit.R2jags<-jags(data=wb_data,
                 inits=wb_inits,
                 parameters.to.save=params,
                 model.file="05normal_exmple_JAGS.txt",
                 n.chains=2,
                 n.iter=50000,
                 n.burnin=4000,
                 n.thin=5,
                 DIC = TRUE,
                 jags.seed = 321,
                 refresh =100,
                 digits = 4,
                 jags.module = c("glm","dic"))
```

```
## module glm loaded
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 30
##    Unobserved stochastic nodes: 2
```

```
##    Total graph size: 41
##
## Initializing model
```

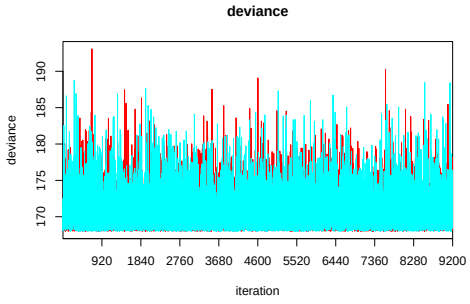
```
# Standard plots of the monitored variables
plot(fit.R2jags)
```

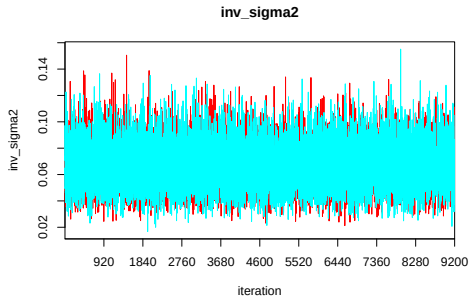


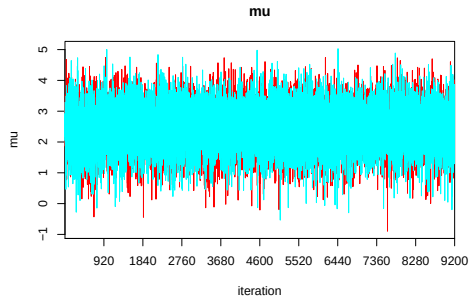

```
# Display summary statistics
print(fit.R2jags)
```

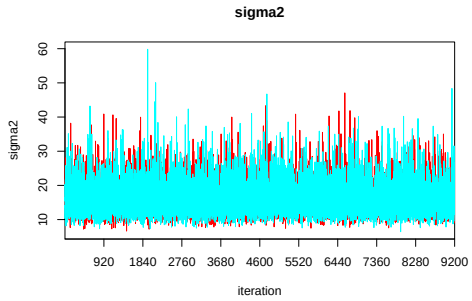
```
## Inference for Bugs model at "05normal_exmple_JAGS.txt", fit using jags,
##  2 chains, each with 50000 iterations (first 4000 discarded), n.thin = 5
##  n.sims = 18400 iterations saved
##             mu.vect sd.vect    2.5%     25%     50%     75%   97.5%  Rhat n.eff
## inv_sigma2    0.066   0.017   0.037   0.054   0.065   0.077   0.102 1.001 12000
## mu            2.465   0.707   1.009   2.019   2.484   2.943   3.795 1.001 18000
## sigma2       16.215   4.492   9.761  13.057  15.487  18.497  27.158 1.001 12000
## deviance    170.863   2.681 168.080 168.906 170.077 171.993 177.932 1.001 18000
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 3.6 and DIC = 174.5
## DIC is an estimate of expected predictive error (lower deviance is better).
```

```
# traceplot
traceplot(fit.R2jags)
```










```
# CODA
fit.R2jags.coda<-as.mcmc(fit.R2jags)
summary(fit.R2jags.coda)
```

```
##
## Iterations = 4001:49996
## Thinning interval = 5
## Number of chains = 2
## Sample size per chain = 9200
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                Mean      SD Naive SE Time-series SE
## deviance   170.86273 2.68094 0.019764       0.019952
## inv_sigma2   0.06601 0.01682 0.000124       0.000124
## mu           2.46507 0.70680 0.005211       0.005211
## sigma2      16.21452 4.49229 0.033118       0.033118
##
## 2. Quantiles for each variable:
##
##                2.5%       25%       50%       75%     97.5%
```

```
## deviance   168.08028 168.90636 170.07685 171.99286 177.9320
## inv_sigma2   0.03682   0.05406   0.06457   0.07659   0.1024
## mu           1.00920   2.01852   2.48364   2.94292   3.7945
## sigma2       9.76115  13.05698  15.48714  18.49724  27.1576
```

```
# conduct CODA
```

The function `R2jags::jags` compute deviance when `DIC` is `true`, where DIC is an estimate of expected predictive error and lower deviance is better. Besides, using `plot` function, the quantiles for each variable are visualized.

2 Exercise 4 - Logistic regression in JAGS

2.1 Bayesian inference

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

Table 2: Mice data from Collett (2003)

x	y	n	centered x	p	*alive*
0.0028	26	28	-0.0121833	0.9285714	2
0.0028	9	12	-0.0121833	0.7500000	3
0.0056	21	40	-0.0093833	0.5250000	19
0.0112	9	40	-0.0037833	0.2250000	31
0.0225	6	40	0.0075167	0.1500000	34
0.0450	1	40	0.0300167	0.0250000	39

Logistic model:

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

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

```
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)
library(ggplot2)
```

```
## 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
)
```

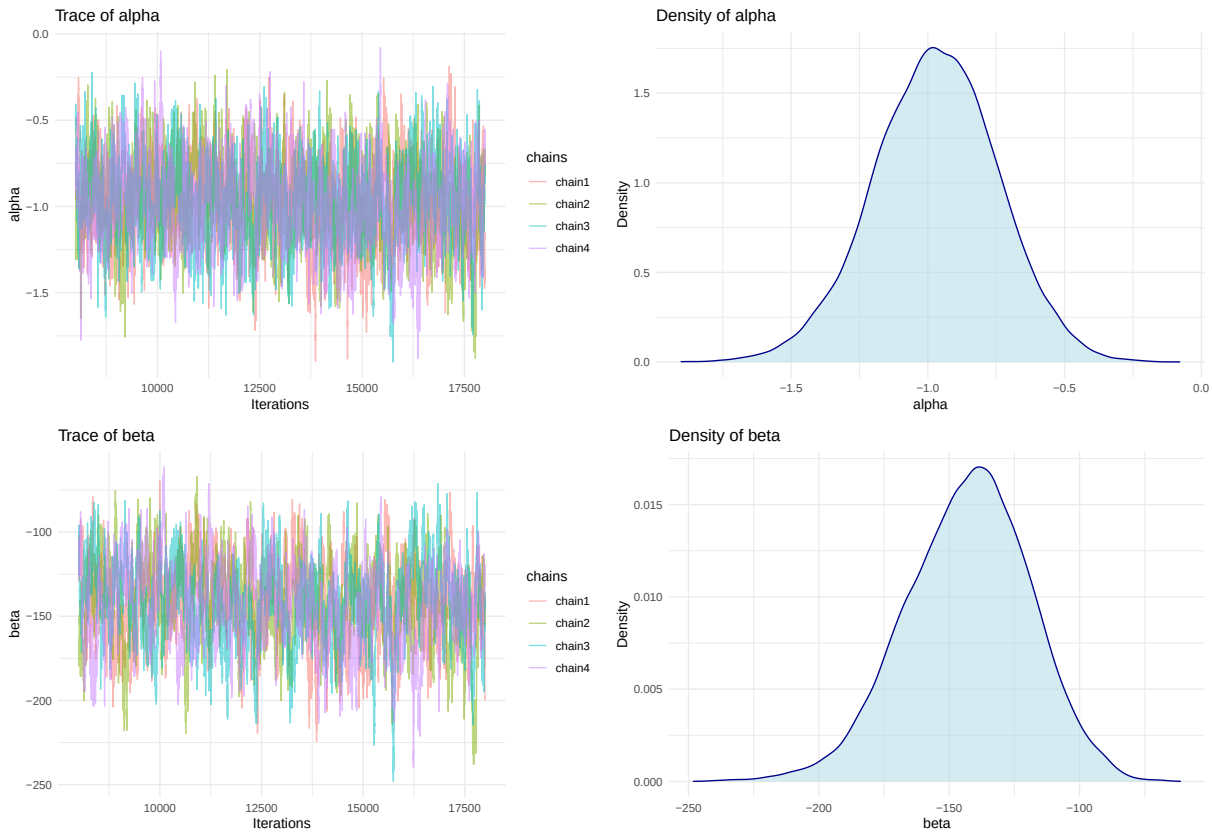
```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 6
##    Unobserved stochastic nodes: 2
##    Total graph size: 37
##
## Initializing model
```

```
## 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
)
```

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

```
##
## Iterations = 4001:14000
## 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   -0.9724  0.2282 0.001141       0.008263
## beta  -142.9774 23.9476 0.119738       1.122918
##
## 2. Quantiles for each variable:
##
##           2.5%      25%      50%       75%    97.5%
## alpha   -1.433   -1.124   -0.969   -0.8183  -0.5335
## beta  -191.984 -158.689 -141.771 -126.3879 -98.6019
```

```r
m.fit1.jags.coda <-as.matrix(fit1.jags.coda)
d.chains <- data.frame(
  iterations = rep(8001:18000, times=4),
  alpha = m.fit1.jags.coda[, "alpha"],
  beta = m.fit1.jags.coda[, "beta"],
  chains = rep(c("chain1", "chain2", "chain3", "chain4"), each=10000),
  alphaRanks = rank(m.fit1.jags.coda[, "alpha"]),
  betaRanks = rank(m.fit1.jags.coda[, "beta"])
)

ggplot(d.chains, aes(x=iterations, y=alpha, color=chains)) + geom_line(alpha=0.5) +
  labs(title="Trace of alpha", x="Iterations") + theme_minimal()

ggplot(d.chains, aes(x=alpha, y=..density..)) +
  geom_density(color="darkblue", fill="lightblue", alpha=0.5) +
  labs(title="Density of alpha", y="Density") + theme_minimal()

ggplot(d.chains, aes(x=iterations, y=beta, color=chains)) + geom_line(alpha=0.5) +
  labs(title="Trace of beta", x="Iterations") + theme_minimal()

ggplot(d.chains, aes(x=beta, y=..density..)) +
  geom_density(color="darkblue", fill="lightblue", alpha=0.5) +
  labs(title="Density of beta", y="Density") + theme_minimal()
```




```
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 Bayesian approach")
```

Table 3: Summary statistics Bayesian approach

	Mean	SD	2.5%	Median	97.5%
alpha	-0.9724034	0.228228	-1.433141	-0.9690259	-0.533536
beta	-142.9773832	23.947566	-191.983614	-141.7706975	-98.601885

2.2 Classic Logistic regression

```
fit_glm_classic <- glm(cbind(y, (n-y)) ~ x_centered, data = d.mice, family = binomial)
summary(fit_glm_classic)
```

```
##
## Call:
## glm(formula = cbind(y, (n - y)) ~ x_centered, family = binomial,
##     data = d.mice)
##
## 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_centered  -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
```

3 Exercise 5 - CODA for logistic regression in JAGS

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

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

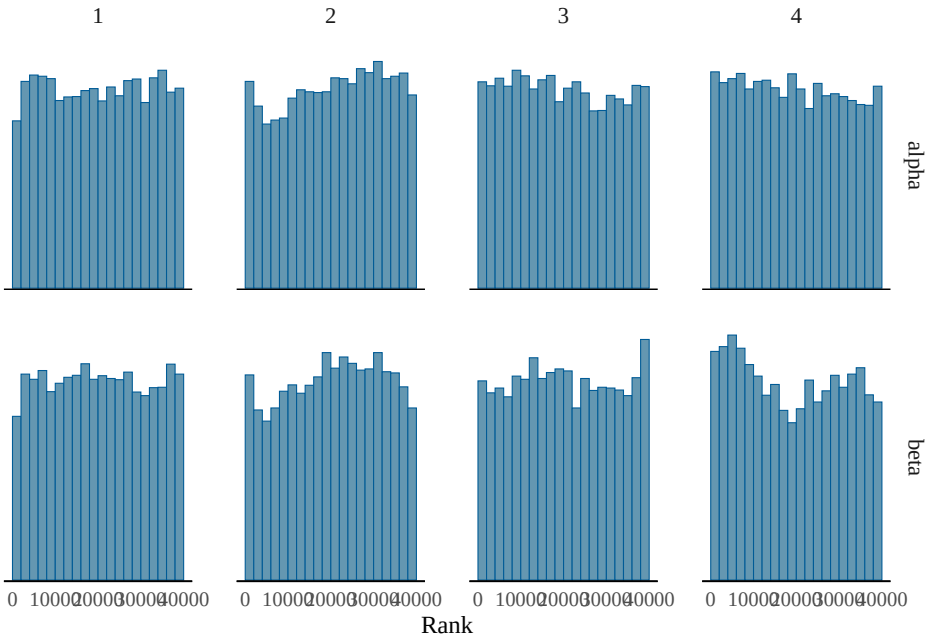
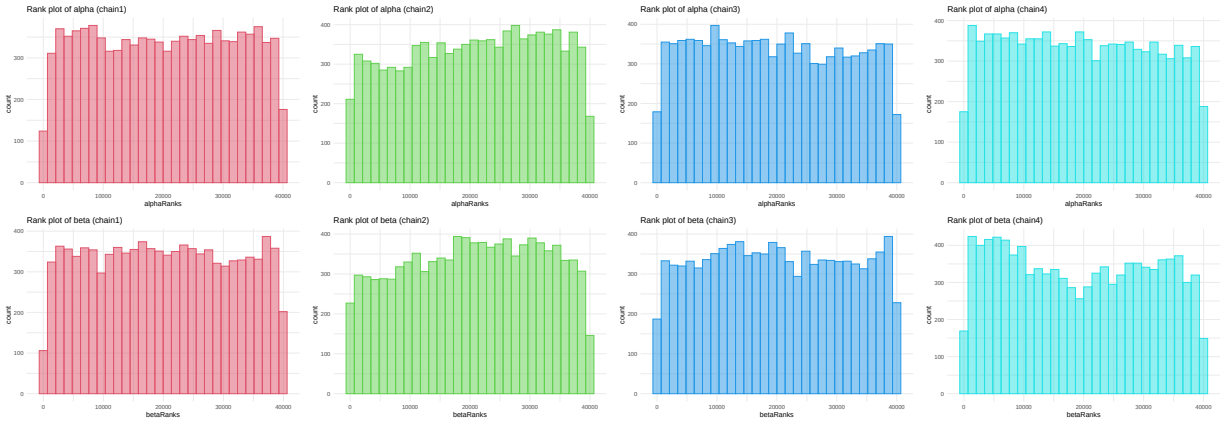


Figure 1: Rank Plot.

```
## Rank plot of alpha
ggplot(d.chains[1:10000, ], aes(x=alphaRanks)) +
  geom_histogram(bins=30, color=2, fill=2, alpha=0.5) +
  labs(title="Rank plot of alpha (chain1)") + theme_minimal()

ggplot(d.chains[10001:20000, ], aes(x=alphaRanks)) +
```

```
  geom_histogram(bins=30, color=3, fill=3, alpha=0.5) +
  labs(title="Rank plot of alpha (chain2)") + theme_minimal()

ggplot(d.chains[20001:30000, ], aes(x=alphaRanks)) +
  geom_histogram(bins=30, color=4, fill=4, alpha=0.5) +
  labs(title="Rank plot of alpha (chain3)") + theme_minimal()

ggplot(d.chains[30001:40000, ], aes(x=alphaRanks)) +
  geom_histogram(bins=30, color=5, fill=5, alpha=0.5) +
  labs(title="Rank plot of alpha (chain4)") + theme_minimal()

## Rank plot of beta
ggplot(d.chains[1:10000, ], aes(x=betaRanks)) +
  geom_histogram(bins=30, color=2, fill=2, alpha=0.5) +
  labs(title="Rank plot of beta (chain1)") + theme_minimal()

ggplot(d.chains[10001:20000, ], aes(x=betaRanks)) +
  geom_histogram(bins=30, color=3, fill=3, alpha=0.5) +
  labs(title="Rank plot of beta (chain2)") + theme_minimal()

ggplot(d.chains[20001:30000, ], aes(x=betaRanks)) +
  geom_histogram(bins=30, color=4, fill=4, alpha=0.5) +
  labs(title="Rank plot of beta (chain3)") + theme_minimal()

ggplot(d.chains[30001:40000, ], aes(x=betaRanks)) +
  geom_histogram(bins=30, color=5, fill=5, alpha=0.5) +
  labs(title="Rank plot of beta (chain4)") + theme_minimal()
```



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 rankings mixing or overlapping in a uniform distribution. See Vehtari et al. (2019) for details.

It can be seen in the figure that the rank histograms nearly follow a normal distribution.

3.1 Convergence diagnostics

3.1.1 Heidelberger & Welch (Convergence to stationarity)

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

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

```
##
##       Stationarity start     p-value
##       test         iteration
## alpha passed       1         0.356
## beta  passed       1         0.481
##
##       Halfwidth Mean     Halfwidth
##       test
## alpha passed      -0.969 0.0288
## beta  passed    -142.454 3.9710
##
## [[2]]
##
##       Stationarity start     p-value
##       test         iteration
## alpha passed       4001      0.1389
## beta  passed          1      0.0813
##
##       Halfwidth Mean     Halfwidth
##       test
## alpha passed      -0.992 0.0413
## beta  passed    -142.410 4.4718
##
## [[3]]
##
##       Stationarity start     p-value
##       test         iteration
## alpha passed       1         0.470
## beta  passed       1         0.438
##
##       Halfwidth Mean     Halfwidth
##       test
## alpha passed      -0.979 0.0347
## beta  passed    -142.655 4.5456
##
## [[4]]
##
##       Stationarity start     p-value
##       test         iteration
## alpha failed         NA      0.0209
## beta  passed       2001      0.0650
##
##       Halfwidth Mean Halfwidth
##       test
## alpha <NA>         NA   NA
## beta  passed    -146 5.13
```

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 is 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.

3.1.2 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 40       43560 3746         11.6
##  beta  65       63385 3746         16.9
##
##
## [[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 63       67263  3746         18.0
##  beta  132      138918 3746         37.1
##
##
## [[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 55       57435  3746         15.3
##  beta  88       103400 3746         27.6
##
##
## [[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 24       24480 3746          6.53
##  beta  52       57504 3746         15.40
```

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.

3.1.3 Geweke (Convergence to stationarity)

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

```
## [[1]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##  alpha   beta
## 0.6215 0.2479
##
##
## [[2]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
## alpha  beta
## 1.544 1.471
##
##
## [[3]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
## alpha  beta
## 1.317 1.865
##
##
## [[4]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##  alpha   beta
## 0.9854 0.3025
```

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)
```

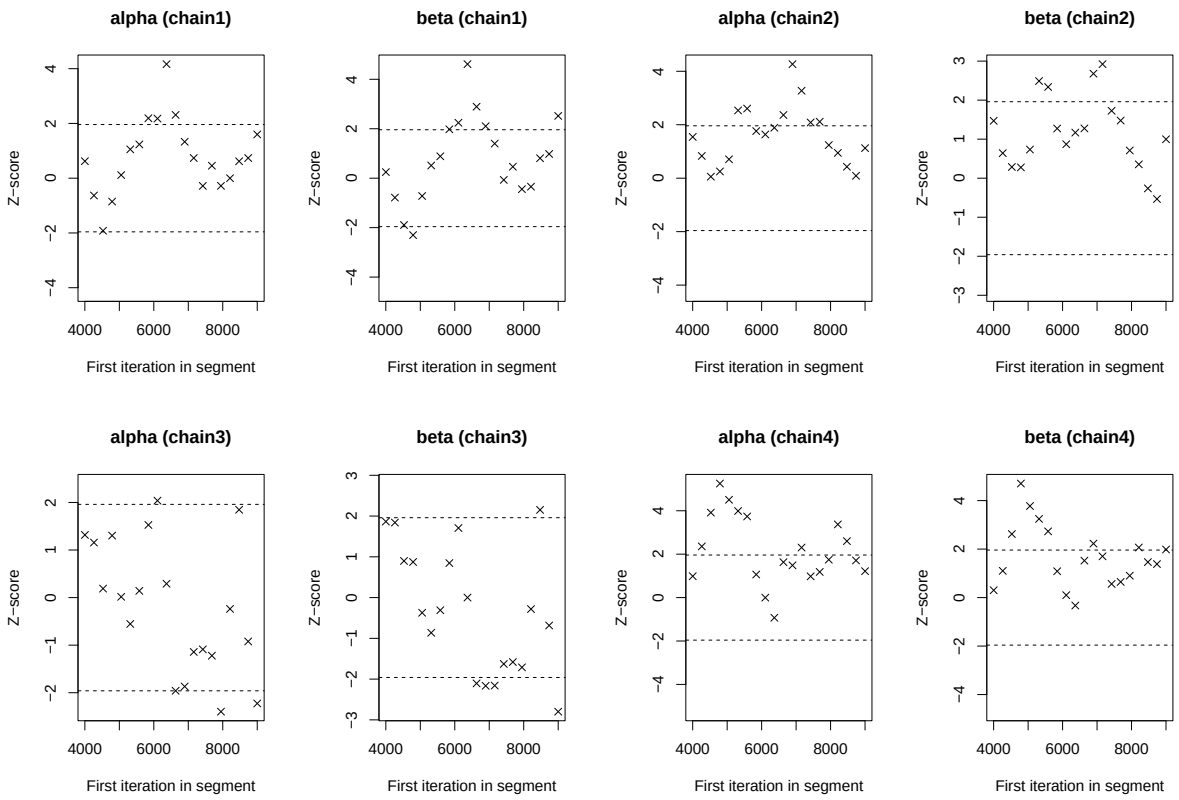


Figure 2: Geweke Plot.

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.

3.1.4 Gelman and Rubin's convergence diagnostic

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

```
## Potential scale reduction factors:
##
##       Point est. Upper C.I.
## alpha          1       1.00
## beta           1       1.01
##
## 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)
```

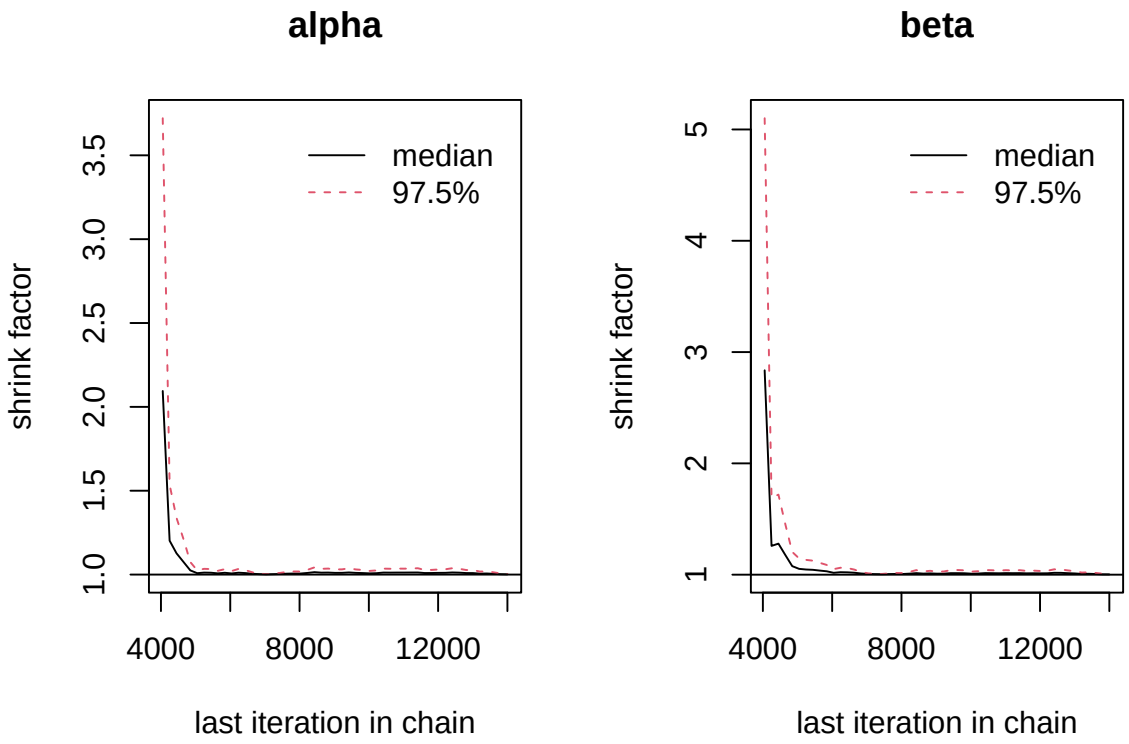


Figure 3: Gelman plot.

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.

3.1.5 Gelman-Rubin diagnostic using stable variance estimators

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

```
## $psrf
## [1] 1.000858 1.001369
##
## $mpsrf
## [1] 1.001806
##
```

```
## $means
##       alpha        beta
##   -0.973203 -143.069023
##
## $n.eff
## [1] 1076.652
##
## $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.

3.1.6 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.

3.2 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
```

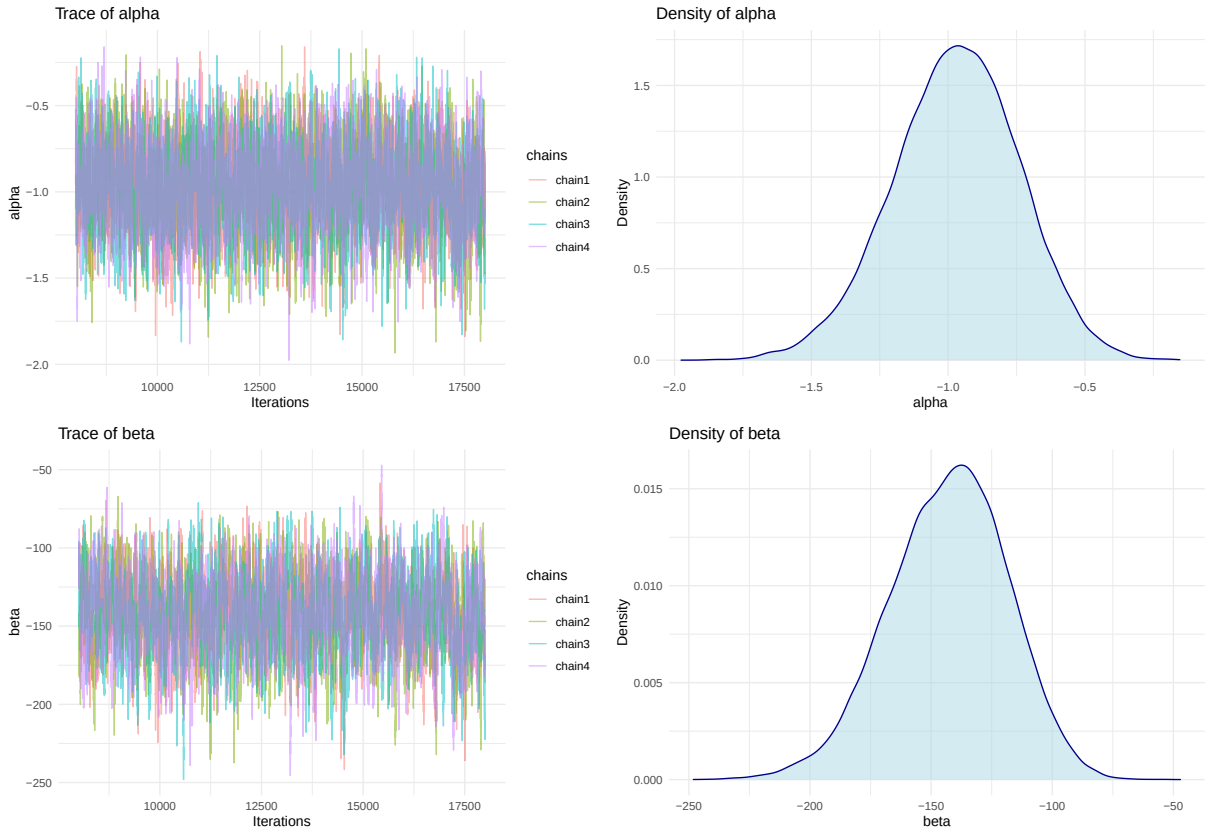
```
## Burn-in (increase burn-in iterations)
update(model2.jags, n.iter = 4000)

## 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 = 4003:34000
## 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    -0.9716  0.2317 0.001159       0.005384
## beta  -142.9854 24.4416 0.122208       0.716055
##
## 2. Quantiles for each variable:
##
##           2.5%      25%       50%       75%    97.5%
## alpha   -1.447   -1.124   -0.9654   -0.8131  -0.5361
## beta  -193.368 -159.092 -141.7906 -125.8210 -98.1524
```

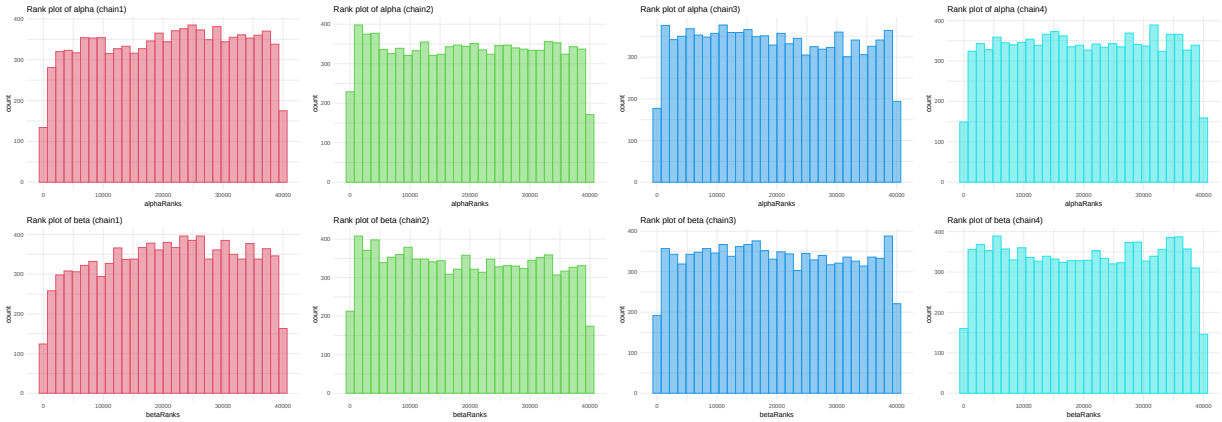
```
m.fit2.jags.coda <-as.matrix(fit2.jags.coda)
d.chains <- data.frame(
  iterations = rep(8001:18000, times=4),
  alpha = m.fit2.jags.coda[, "alpha"],
  beta = m.fit2.jags.coda[, "beta"],
  chains = rep(c("chain1", "chain2", "chain3", "chain4"), each=10000),
  alphaRanks = rank(m.fit2.jags.coda[, "alpha"]),
  betaRanks = rank(m.fit2.jags.coda[, "beta"])
)

ggplot(d.chains, aes(x=iterations, y=alpha, color=chains)) + geom_line(alpha=0.5) +
  labs(title="Trace of alpha", x="Iterations") + theme_minimal()

ggplot(d.chains, aes(x=alpha, y=..density..)) +
  geom_density(color="darkblue", fill="lightblue", alpha=0.5) +
```

```
  labs(title="Density of alpha", y="Density") + theme_minimal()

ggplot(d.chains, aes(x=iterations, y=beta, color=chains)) + geom_line(alpha=0.5) +
  labs(title="Trace of beta", x="Iterations") + theme_minimal()

ggplot(d.chains, aes(x=beta, y=..density..)) +
  geom_density(color="darkblue", fill="lightblue", alpha=0.5) +
  labs(title="Density of beta", y="Density") + theme_minimal()
```




```
## Rank plot of alpha
ggplot(d.chains[1:10000, ], aes(x=alphaRanks)) +
  geom_histogram(bins=30, color=2, fill=2, alpha=0.5) +
  labs(title="Rank plot of alpha (chain1)") + theme_minimal()

ggplot(d.chains[10001:20000, ], aes(x=alphaRanks)) +
  geom_histogram(bins=30, color=3, fill=3, alpha=0.5) +
  labs(title="Rank plot of alpha (chain2)") + theme_minimal()

ggplot(d.chains[20001:30000, ], aes(x=alphaRanks)) +
  geom_histogram(bins=30, color=4, fill=4, alpha=0.5) +
  labs(title="Rank plot of alpha (chain3)") + theme_minimal()

ggplot(d.chains[30001:40000, ], aes(x=alphaRanks)) +
  geom_histogram(bins=30, color=5, fill=5, alpha=0.5) +
  labs(title="Rank plot of alpha (chain4)") + theme_minimal()

## Rank plot of beta
ggplot(d.chains[1:10000, ], aes(x=betaRanks)) +
  geom_histogram(bins=30, color=2, fill=2, alpha=0.5) +
  labs(title="Rank plot of beta (chain1)") + theme_minimal()
```

```
ggplot(d.chains[10001:20000, ], aes(x=betaRanks)) +
  geom_histogram(bins=30, color=3, fill=3, alpha=0.5) +
  labs(title="Rank plot of beta (chain2)") + theme_minimal()

ggplot(d.chains[20001:30000, ], aes(x=betaRanks)) +
  geom_histogram(bins=30, color=4, fill=4, alpha=0.5) +
  labs(title="Rank plot of beta (chain3)") + theme_minimal()

ggplot(d.chains[30001:40000, ], aes(x=betaRanks)) +
  geom_histogram(bins=30, color=5, fill=5, alpha=0.5) +
  labs(title="Rank plot of beta (chain4)") + theme_minimal()
```




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

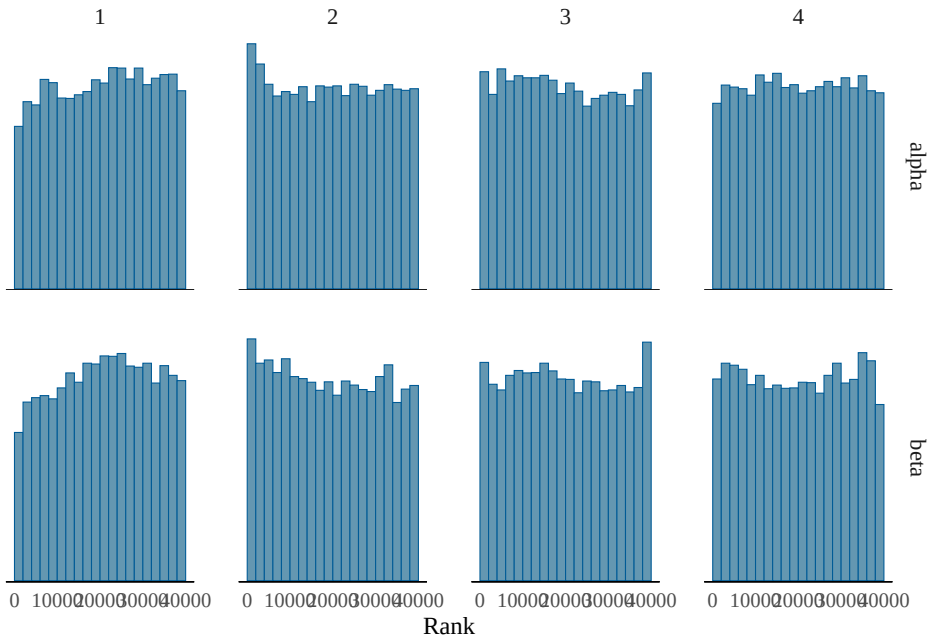


Figure 4: Rank Plot.

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         0.343
## beta  passed       1         0.313
##
##       Halfwidth Mean    Halfwidth
##       test
## alpha passed      -0.96 0.0191
## beta  passed    -141.42 2.6028
##
## [[2]]
##
##       Stationarity start     p-value
##       test         iteration
## alpha passed       1         0.241
## beta  passed       1         0.358
##
##       Halfwidth Mean     Halfwidth
##       test
## alpha passed      -0.979 0.0224
## beta  passed    -144.288 2.9090
##
## [[3]]
##
##       Stationarity start     p-value
##       test         iteration
## alpha passed       1         0.908
## beta  passed       1         0.931
##
##       Halfwidth Mean     Halfwidth
##       test
## alpha passed      -0.977 0.0211
## beta  passed    -143.131 2.9174
##
## [[4]]
##
##       Stationarity start     p-value
##       test         iteration
## alpha passed       1         0.740
## beta  passed       1         0.771
##
##       Halfwidth Mean    Halfwidth
##       test
## alpha passed      -0.97 0.0216
## beta  passed    -143.10 2.7870
```

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 54       61668  3746         16.5
##  beta  108      110403 3746         29.5
##
##
## [[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 72       104202 3746         27.8
##  beta  99       105228 3746         28.1
##
##
## [[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 63       61911  3746         16.5
##  beta  120      149892 3746         40.0
##
##
## [[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 63       65313 3746         17.4
##  beta  72       79542 3746         21.2
```

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
```

```
## 0.8379 0.6160
##
##
## [[2]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
## alpha  beta
## 1.538 1.355
##
##
## [[3]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##  alpha   beta
## 0.8589 1.4979
##
##
## [[4]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##  alpha   beta
## 0.6240 0.6357
```

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       1.01
## beta           1       1.01
##
## Multivariate psrf
##
## 1
```

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

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

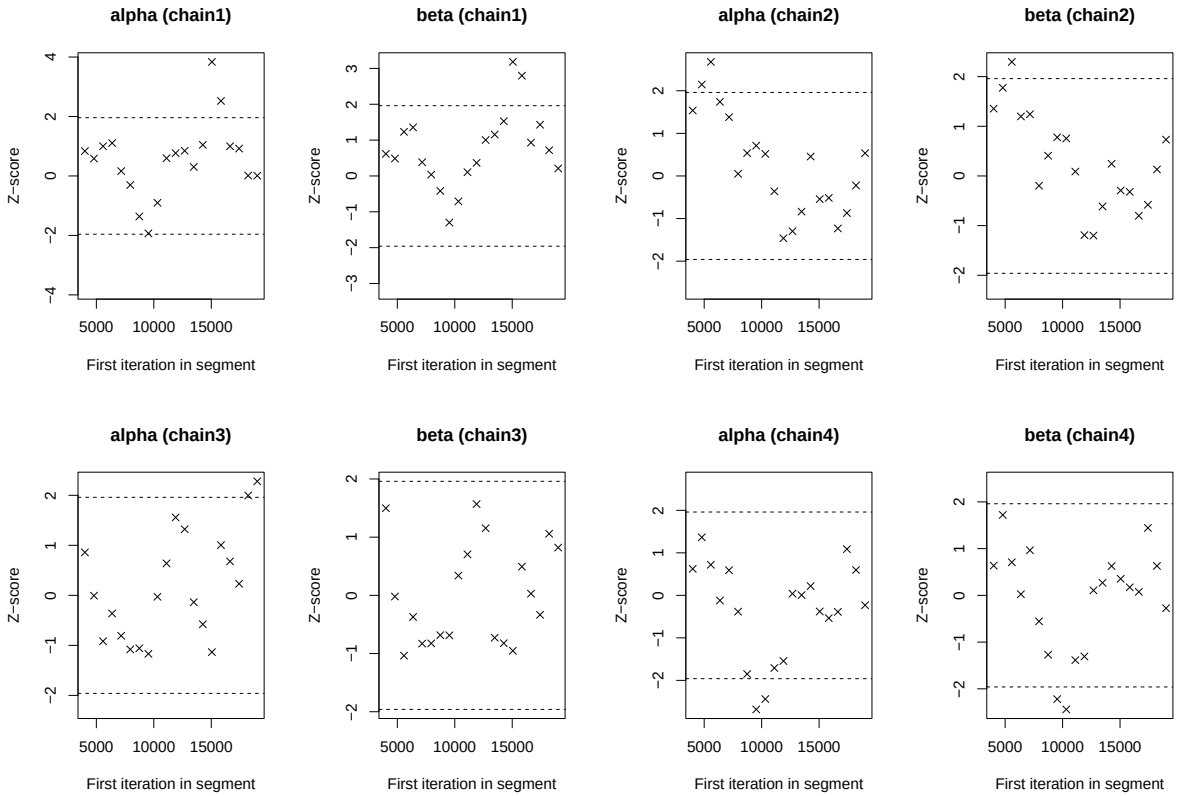


Figure 5: Geweke Plot.

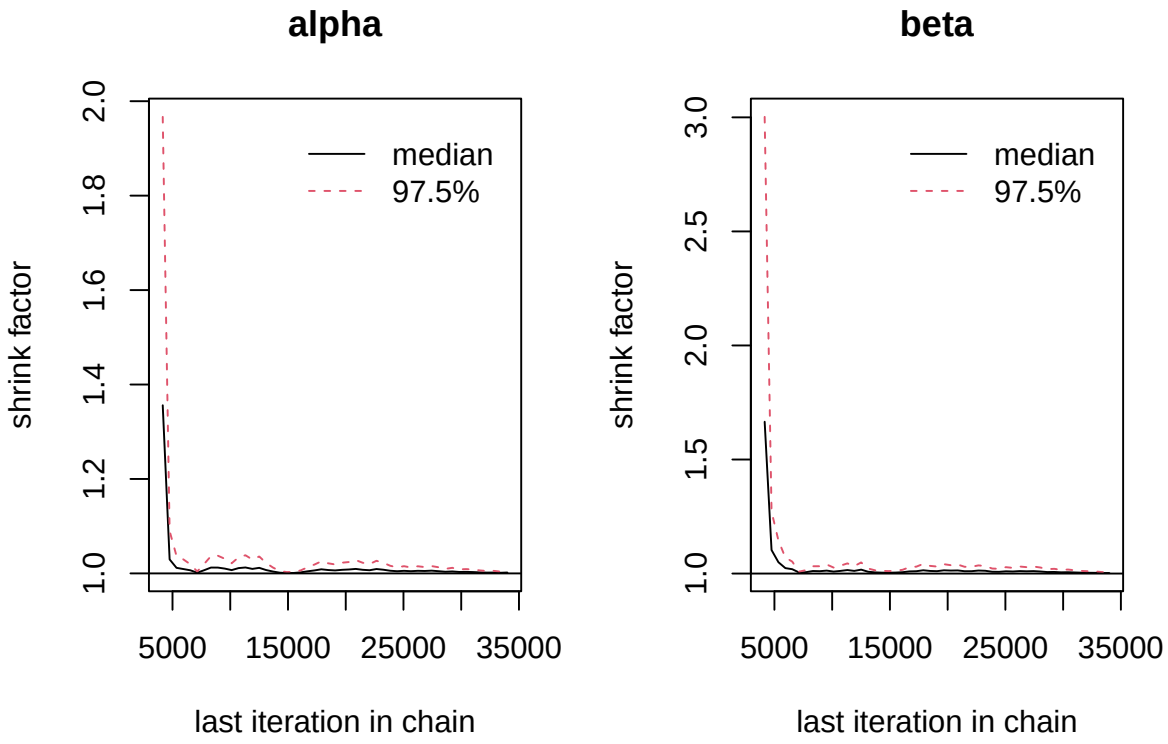


Figure 6: Gelman plot.

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

```
## $psrf
## [1] 1.000280 1.000464
##
## $mpsrf
## [1] 1.000601
##
## $means
##        alpha         beta
##   -0.9717448 -143.0073286
##
## $n.eff
## [1] 3071.043
##
## $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.

4 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)
}
```

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

where $\text{ACF}(k) = \text{corr}(\theta_t^*, \theta_{t+k}^*)$

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)) )
```

```r
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(
              "method" = c("ESS-Script", "ESS-Script", "stableGR", "stableGR"),
              "thinning" = c(1,10,1,10),
              "n_eff" = c(ess_script1,ess_script2,ess_function1$n.eff,ess_function2$n.eff)
              )
```

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

method	thinning	n_eff
ESS-Script	1	11.96032
ESS-Script	10	115.43639
stableGR	1	10.92997
stableGR	10	92.17642

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