Worksheet 05 Group 2

Andrea Staub Emanuel Mauch Holly Vuarnoz Jan Hohenheim
Sophie Haldemann

```
library(glue)
library(rjags)
library(INLA)
library(MASS)
library(ggplot2)
library(patchwork)
library(coda)
library(stableGR)
source("05ess.r")
```

Exercise 3

Run the code provided in the file 05normal_example_JAGS.R and explore the interfaces in R to JAGS. Comment your findings. -> The comments can be found below the code chunks.

```
###################
# Normal Example (Gibbs MCMC)
###################

remove(list=ls())
set.seed(44566)

# set the path to the 05normal_example_JAGS.txt file
path <- ""
```

clearing environment, `set.seed` for reproducibility and set path to .txt file with model

Interface in R to JAGS: rjags

```
############################################################
# rjags interface to JAGS
############################################################

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")
```

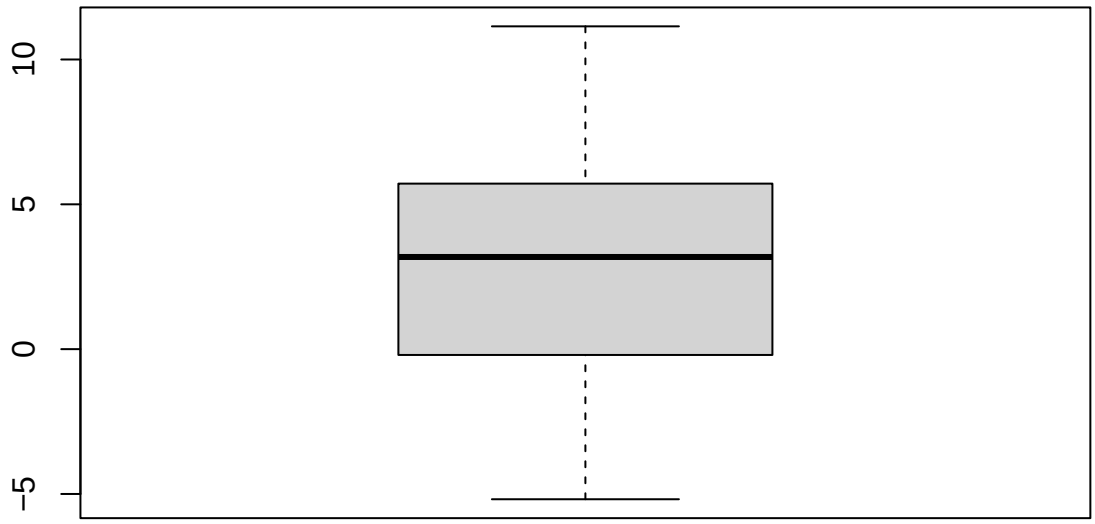
```
## Module glm unloaded
```

```
list.modules()
```

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

With `list.factories` a data frame with two columns is returned, the first column shows the names of the factory objects in the currently loaded modules, and the second column is a logical vector indicating whether the corresponding factory is active or not. With `list.modules()` the loaded modules can be listed. With `load.module("module_name")` and `unload.module("module_name")` modules can be loaded or unloaded.

```
####################
## Introduction
####################


# generating data
#mu <- 4
#sigma2 <- 16
#n <- 30
#y <- rnorm(n=n, mean=mu, sd=sqrt(sigma2))


# Load the data
# round(y,3)
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
```

30 random datapoints were generated from a normal distribution with mean = 4 and standard deviation = 4. The parameters of the prior distribution are defined. Mean value from summary is with 3.1884333 not that close to mean = 4.

```
################################
# INLA exact result (motivation)
################################

# https://www.r-inla.org/download-install
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))
```

As input for INLA the previously defined parameters and the random sampled datapoints are used. INLA is used to approximate distribution.

```
############################
# Step 2: JAGS model file as a string in rjags with coda
############################
set.seed(44566)

library(rjags)
library(coda)


#sessionInfo()

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="TempModel.txt") # write to a file
```

The input for JAGS is defined as a list. The model is printed to a new .txt file. N is set to 30 and N samples are drawn from a normal distribution with mean mu and the precision as standard deviation. Mu follows a normal prior and the precision a gamma prior. The variance and then the standard deviation are estimated from the precision.

```
############################################################
# JAGS only one chain
############################################################
```

```r
# model initiation
model.jags <- jags.model(
  file = "TempModel.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
```

```r
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"
```

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

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

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

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

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

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

A jags model with only one chain is initiated. The previously generated TempModel.txt file is used. The number of iterations for adaption is set to 4000.

```r
# burn-in

update(model.jags, n.iter = 4000)

# 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] 0.0607 0.1019 0.0835 0.0568 0.1072 ...
##   ..- 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"
```

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

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

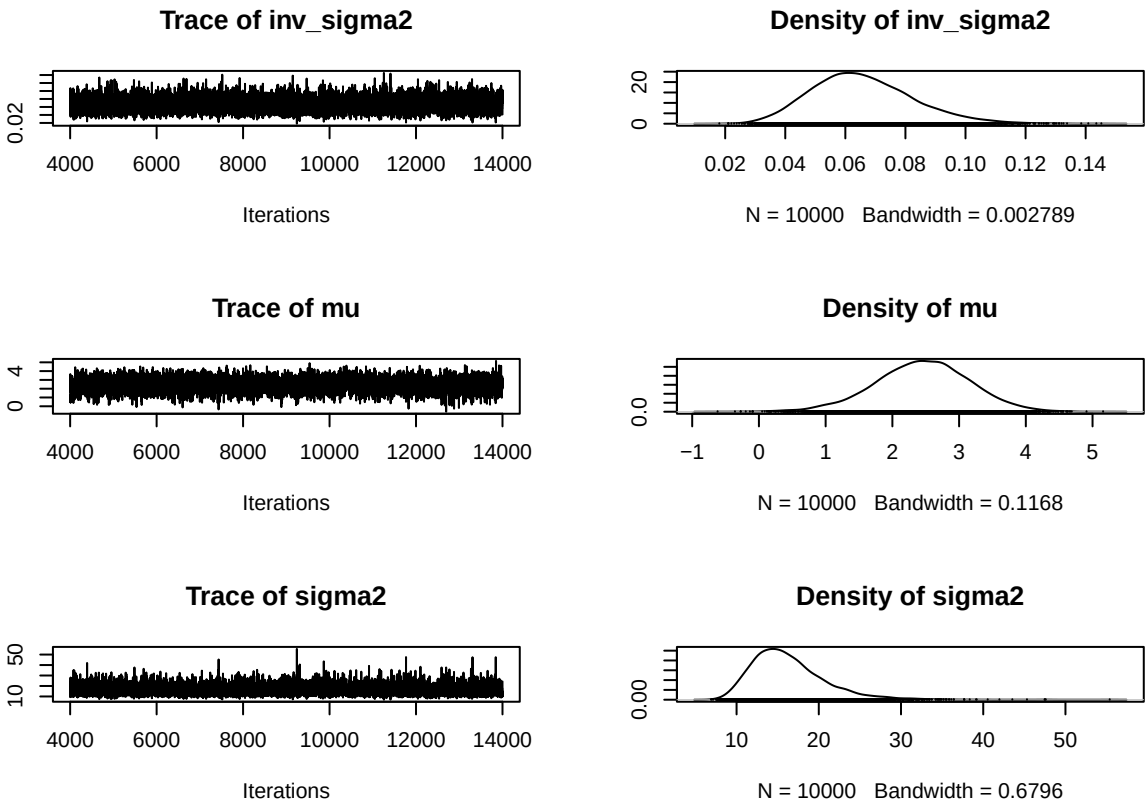
```r
attributes(fit.jags.coda)
```

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

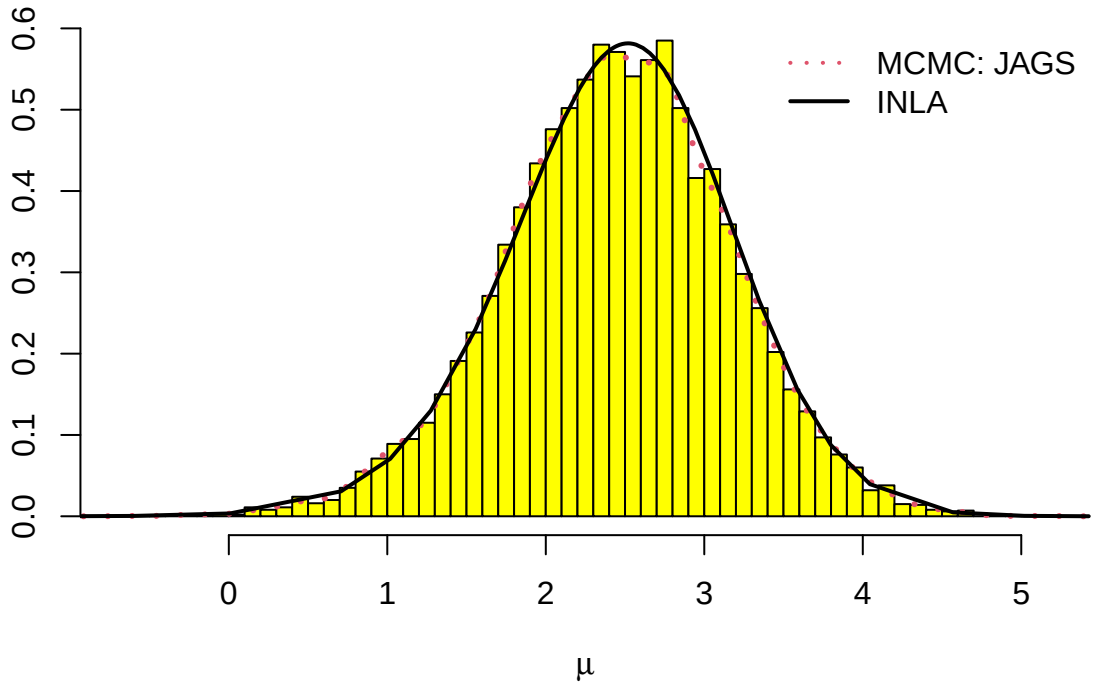
```r
summary(fit.jags.coda)
```

```
##
## 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  0.06572 0.01677 0.0001677      0.0001798
## mu          2.44852 0.70995 0.0070995      0.0076083
## sigma2     16.27816 4.46477 0.0446477      0.0483250
##
## 2. Quantiles for each variable:
##
##               2.5%     25%      50%      75%   97.5%
## inv_sigma2 0.03686  0.0539  0.06425  0.07614  0.1025
## mu         0.98945  1.9902  2.46585  2.92163  3.8129
## sigma2     9.75956 13.1336 15.56520 18.55397 27.1266
```

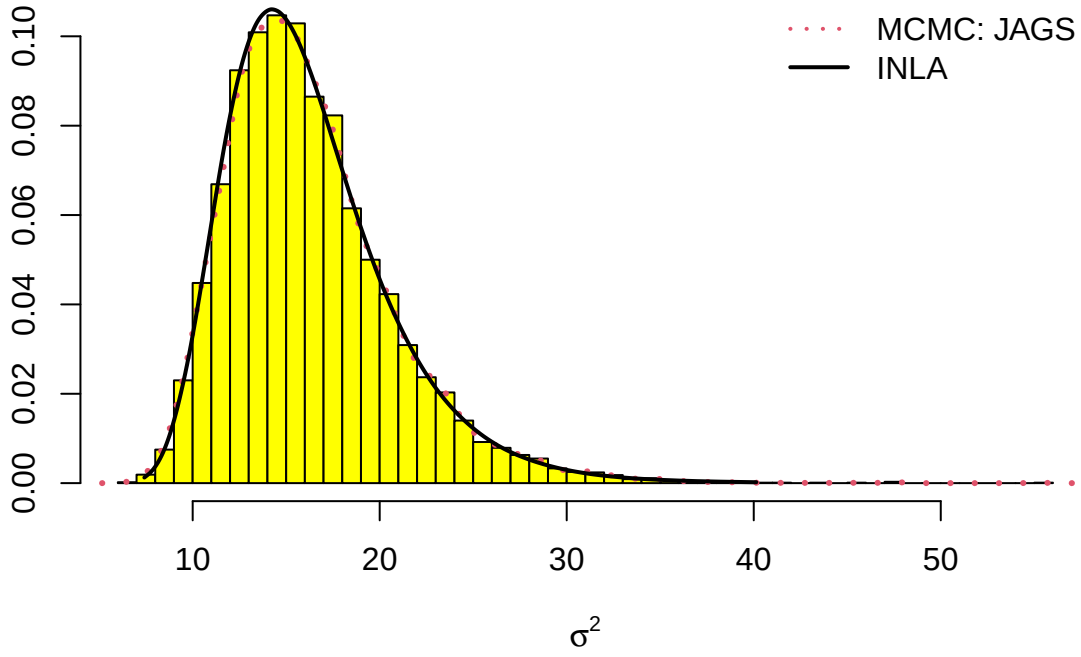
```r
#print(fit.jags.coda)
plot(fit.jags.coda)
```



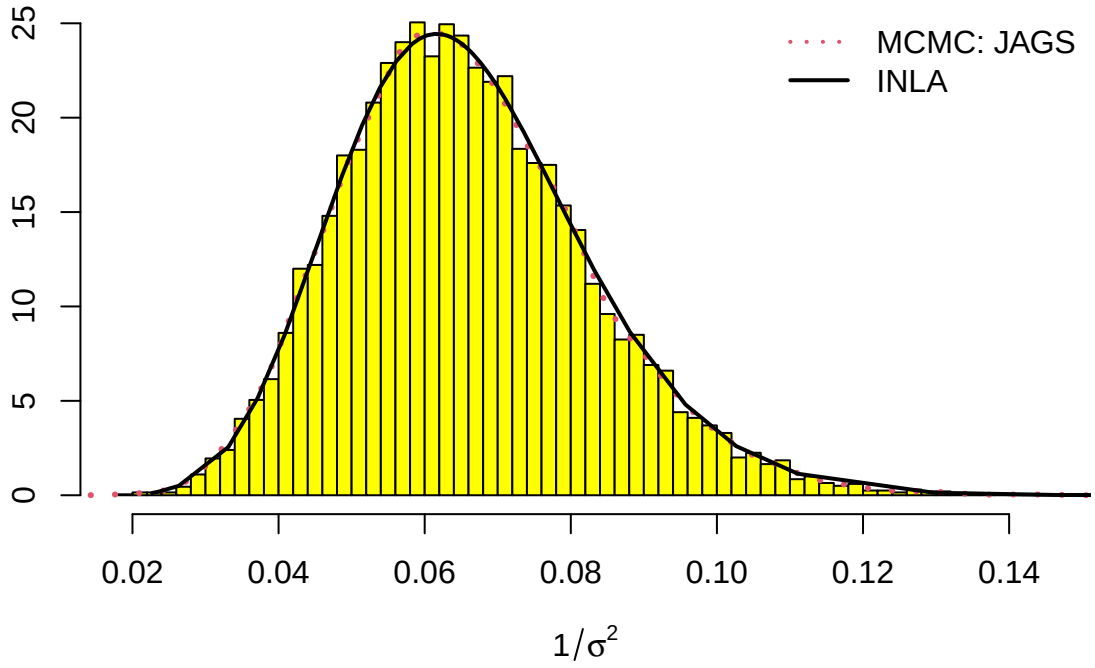

```r
# store samples for each parameter from the chain into separate objects
m.fit.jags.coda <- as.matrix(fit.jags.coda)
mu.sim <- m.fit.jags.coda[,"mu"]
sigma2.sim <- m.fit.jags.coda[,"sigma2"]
inv_sigma2.sim <- m.fit.jags.coda[,"inv_sigma2"]

library(MASS)

par(mfrow=c(1,1))
# 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")
```




```
# 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")
```



The burn in period is set to 4000. After the burn in 10000 iterations are done to fit the model. The traceplots look good and stable. When looking at the traceplots we can see that the approximations for mu and sigma^2 from JAGS and INLA look similar but differ just a little bit.

```
############################################################
# JAGS several chains
############################################################

wb_inits <- function() {
  list(mu = rnorm(1),
       inv_sigma2 = runif(1)
  )
}

# model initialisation
model.jags <- jags.model(
  file = "TempModel.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
```

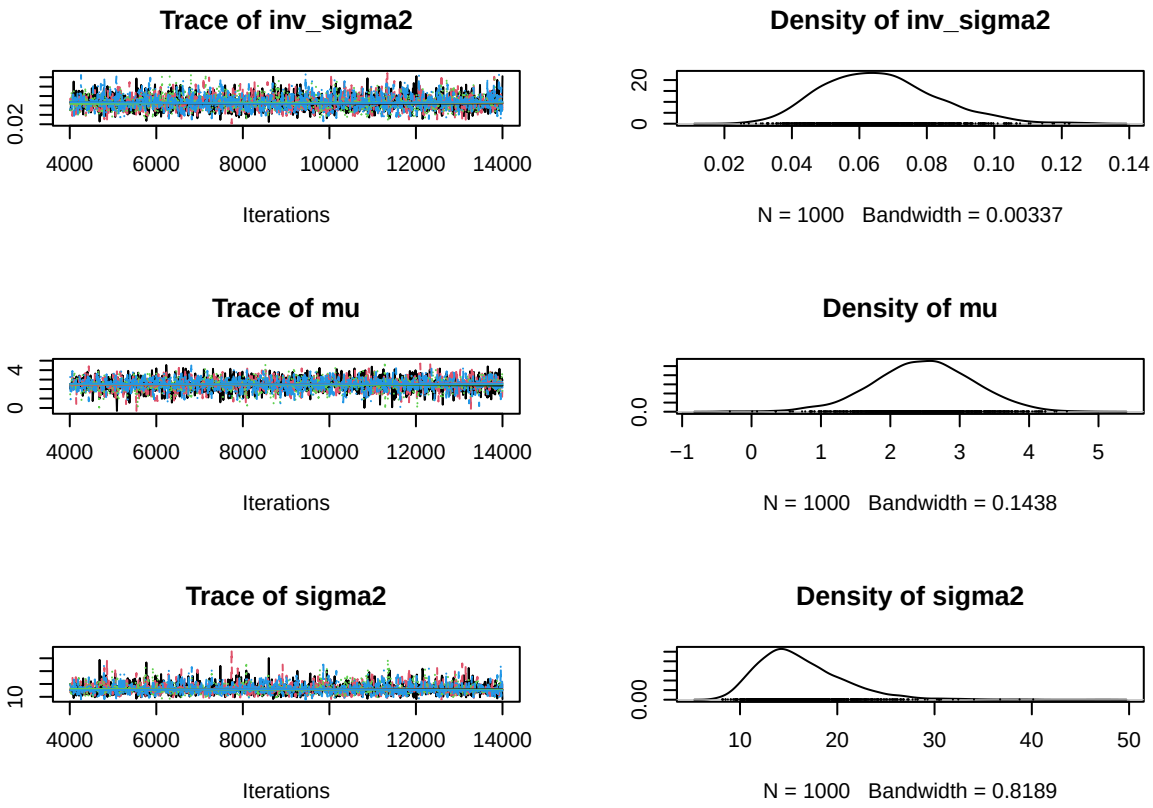
```
# burn-in

update(model.jags, n.iter = 4000)

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

#n.thin<-floor((n.iter-n.adapt)/500)
#floor((10000-4000)/500)=12
```

Now again JAGS is used with the same model but instead of only one chain, 4 chains are used instead. The burn in period is again 4000.

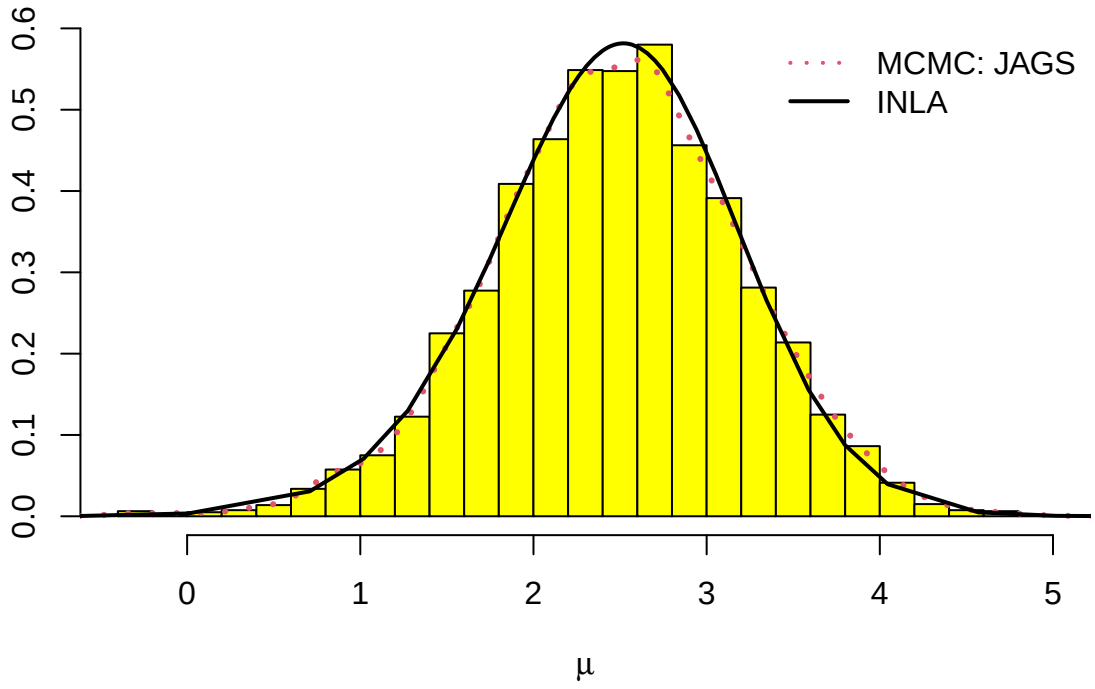
```
summary(fit.jags.coda)
```

```
##
## 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  0.06595 0.01683 0.000266      0.0002821
## mu          2.47872 0.71987 0.011382      0.0110102
## sigma2     16.21930 4.43271 0.070087      0.0751751
##
## 2. Quantiles for each variable:
##
##               2.5%      25%      50%      75%   97.5%
## inv_sigma2 0.03758  0.05393  0.06465  0.07631  0.1027
## mu         0.98807  2.00657  2.48768  2.96169  3.8591
## sigma2     9.73704 13.10448 15.46830 18.54279 26.6125
```

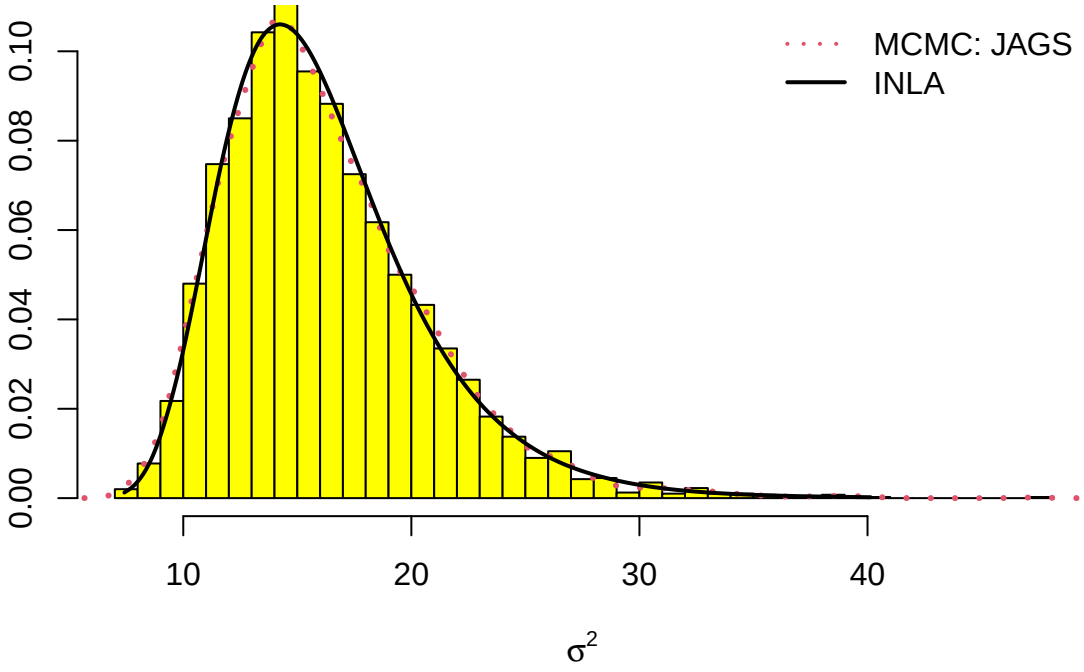
```
plot(fit.jags.coda)
```



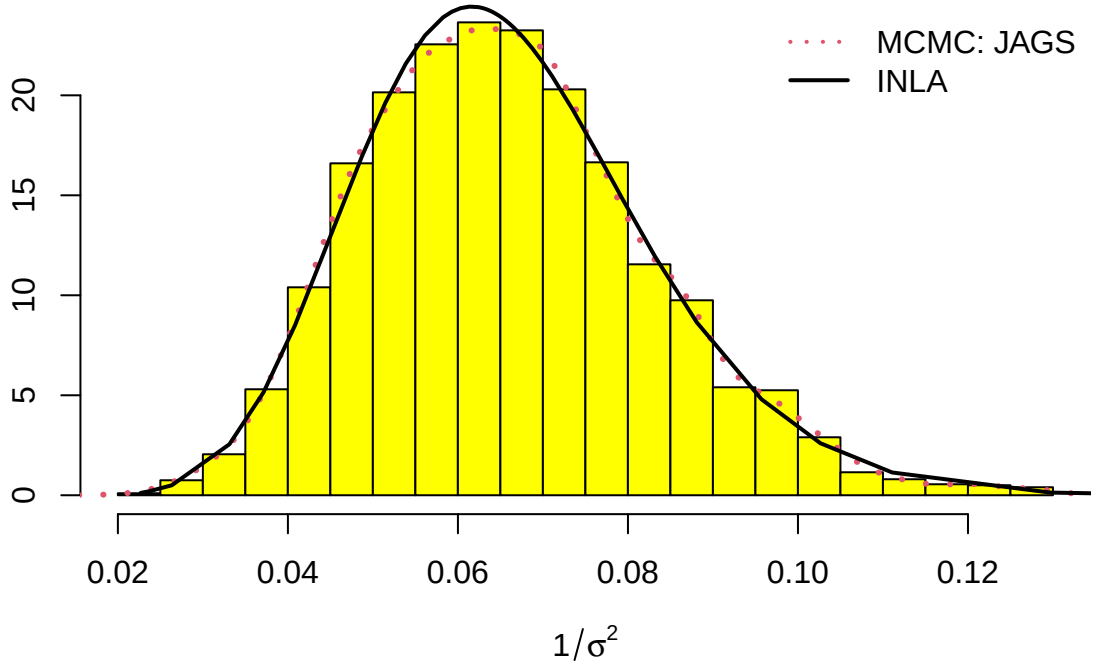

```r
# store samples for each parameter from the chains into separate vectors
m.fit.jags.coda <-as.matrix(fit.jags.coda)
mu.sim <- m.fit.jags.coda[,"mu"]
sigma2.sim <- m.fit.jags.coda[,"sigma2"]
inv_sigma2.sim <- m.fit.jags.coda[,"inv_sigma2"]


par(mfrow=c(1,1))
# 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")
```




```
# 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")
```




```
## CODA
#summary(fit.jags.coda)
#effectiveSize(fit.jags.coda)
#lapply(fit.jags.coda, effectiveSize)
#gelman.diag(fit.jags.coda,autoburnin=TRUE)
#gelman.plot(fit.jags.coda,autoburnin=TRUE)
#geweke.diag(fit.jags.coda)
#geweke.plot(fit.jags.coda)
#heidel.diag(fit.jags.coda)
#raftery.diag(fit.jags.coda)
#coda:::traceplot(fit.jags.coda)


# "DIC" penalised expected deviance computation
dic1<-dic.samples(model=model.jags, n.iter=1000, type="popt")
#dic2<-dic.samples(model=model.jags2, n.iter=1000, type="popt")

# "DIC" penalised expected deviance comparison
# There is no absolute scale for DIC comparison
# SE is very helpful

#diffdic(dic1,dic2)
```

The traceplots of the four chains are superimposed on each other. They look alright. Also this time the results from approximation with JAGS and INLA look quite similar.

Interface in R to JAGS: runjags

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

library(runjags)
```

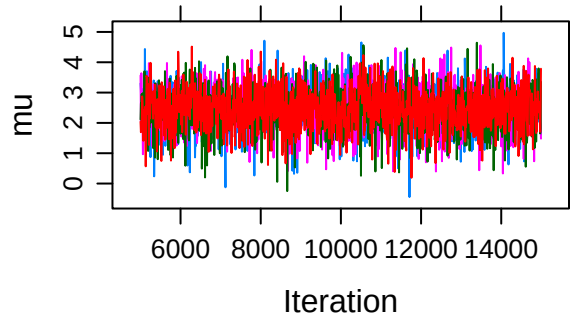
```
## Warning: package 'runjags' was built under R version 4.3.3
```

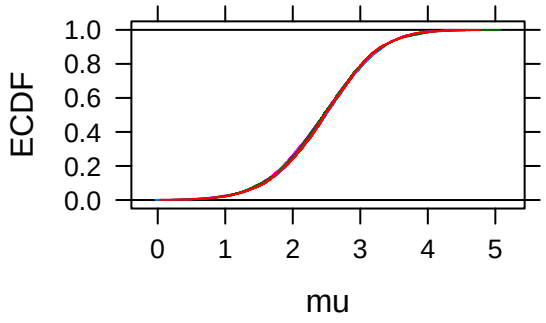
```
################
# runjags interface with a link to a file
################


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)
```

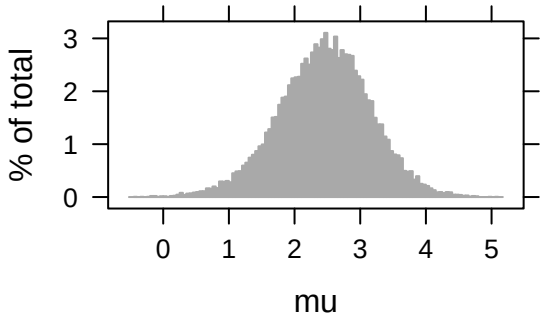
```
## Compiling rjags model...
## Calling the simulation using the rjags method...
## Note: the model did not require adaptation
## Burning in the model for 4000 iterations...
## Running the model for 10000 iterations...
## Simulation complete
## Calculating summary statistics...
## Calculating the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

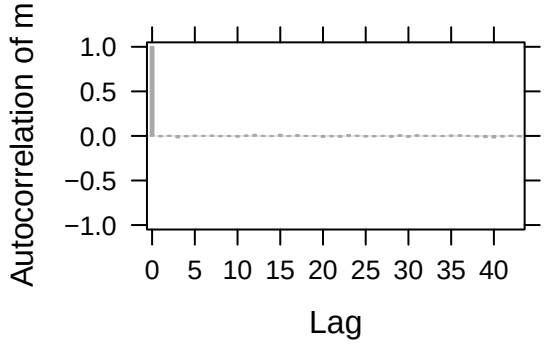
```
plot(fit.runjags)
```

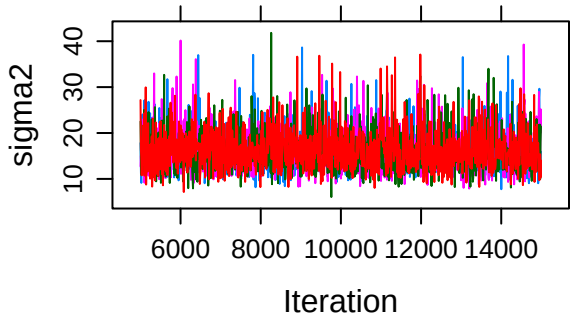
```
## Generating plots...
```

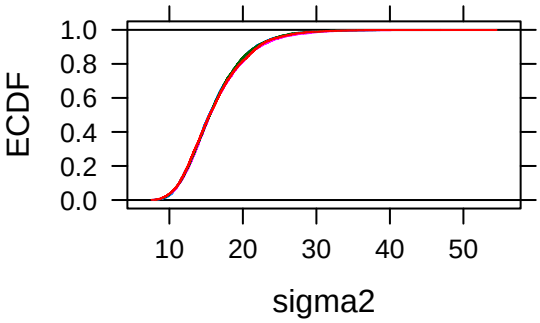

mu
0 1 2 3 4 5
6000 8000 10000 12000 14000
Iteration

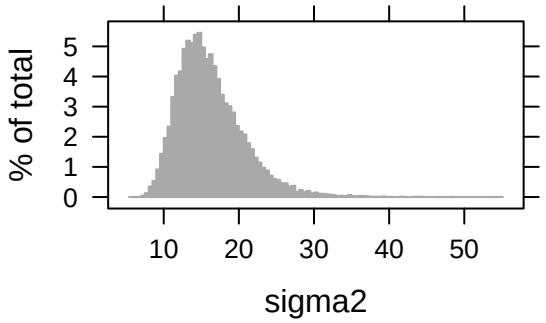

ECDF
1.0
0.8
0.6
0.4
0.2
0.0
0 1 2 3 4 5
mu

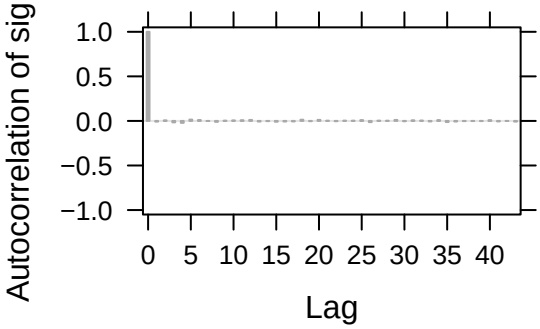

% of total
3
2
1
0
0 1 2 3 4 5
mu

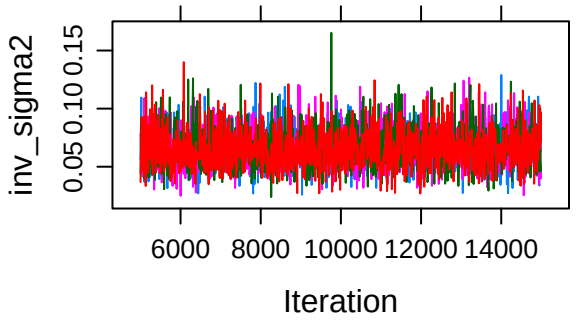

Autocorrelation of mu
1.0
0.5
0.0
−0.5
−1.0
0 5 10 15 20 25 30 35 40
Lag

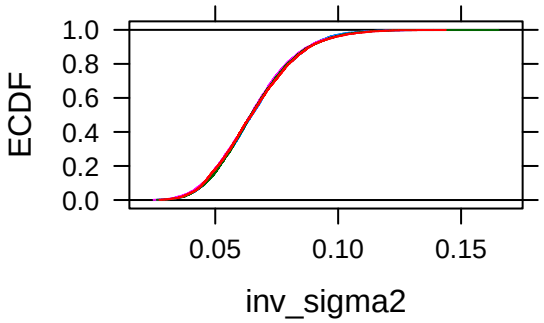

sigma2
10
20
30
40
6000
8000
10000
12000
14000
Iteration

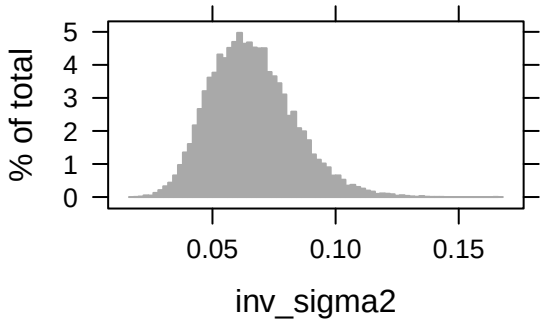

ECDF
0.0
0.2
0.4
0.6
0.8
1.0
10
20
30
40
50
sigma2

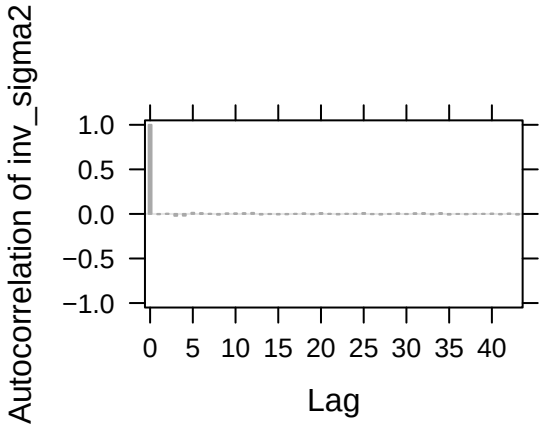

% of total
0
1
2
3
4
5
10
20
30
40
50
sigma2

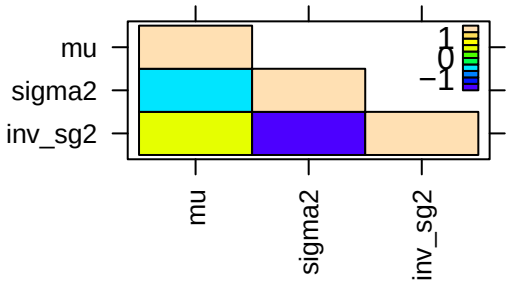

Autocorrelation of sigma2
−1.0
−0.5
0.0
0.5
1.0
0
5
10
15
20
25
30
35
40
Lag


inv_sigma2
0.05 0.10 0.15
6000 8000 10000 12000 14000
Iteration


ECDF
1.0
0.8
0.6
0.4
0.2
0.0
0.05 0.10 0.15
inv_sigma2


% of total
5
4
3
2
1
0
0.05 0.10 0.15
inv_sigma2


Autocorrelation of inv_sigma2
1.0
0.5
0.0
−0.5
−1.0
0 5 10 15 20 25 30 35 40
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            1.041   2.4689  3.8062   2.4526  0.69999   --   0.004857     0.7
## sigma2       8.7965   15.468  24.965   16.217   4.4475   --   0.031513     0.7
## inv_sigma2 0.03487 0.064648 0.09957 0.065974 0.016868   -- 0.00011849     0.7
##
##             SSeff      AC.20   psrf
## mu          20770 -0.0089682 1.0001
## sigma2      19919  0.0049786 1.0003
## inv_sigma2  20265  0.0061126 1.0003
##
## Total time taken: 1.3 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          2.45263 0.69999 0.0049497      0.0048595
## sigma2     16.21750 4.44754 0.0314488      0.0315398
## inv_sigma2  0.06597 0.01687 0.0001193      0.0001187
##
## 2. Quantiles for each variable:
##
##                2.5%      25%      50%      75%   97.5%
## mu          1.01469  1.99579  2.46895  2.91999  3.7884
## sigma2     9.70196 13.08395 15.46842 18.60047 26.7184
## inv_sigma2 0.03743  0.05376  0.06465  0.07643  0.1031
```

```
# conduct CODA
```

Another interface is runjags. Now the .txt file 05normal_exmple_JAGS.txt is used where the model is specified. Four chains are used the burnin period is again set to 4000 afterwards 5000 additional samples are taken. In the burnin the adaptive iterations to use for the simulation are not included. Thinning is set to 2. Here burnin is directly specified in the model instead of using `update()`. The traceplots look ok. We also get an empirical cumulative distribution function plot and an autocorrelation plot for the parameters and an additional plot showing the correlations between the parameters. The autocorrelation plots look good for all parameters.

Interface in R to JAGS: R2jags

```
############################################################
# R2jags wrapper to rjags interface to JAGS several chains
############################################################



library(R2jags)
```

```
##
## Attaching package: 'R2jags'
```

```
## The following object is masked from 'package:coda':
##
##     traceplot
```

```
#rm(list=ls())
```

A third interface is R2jags.

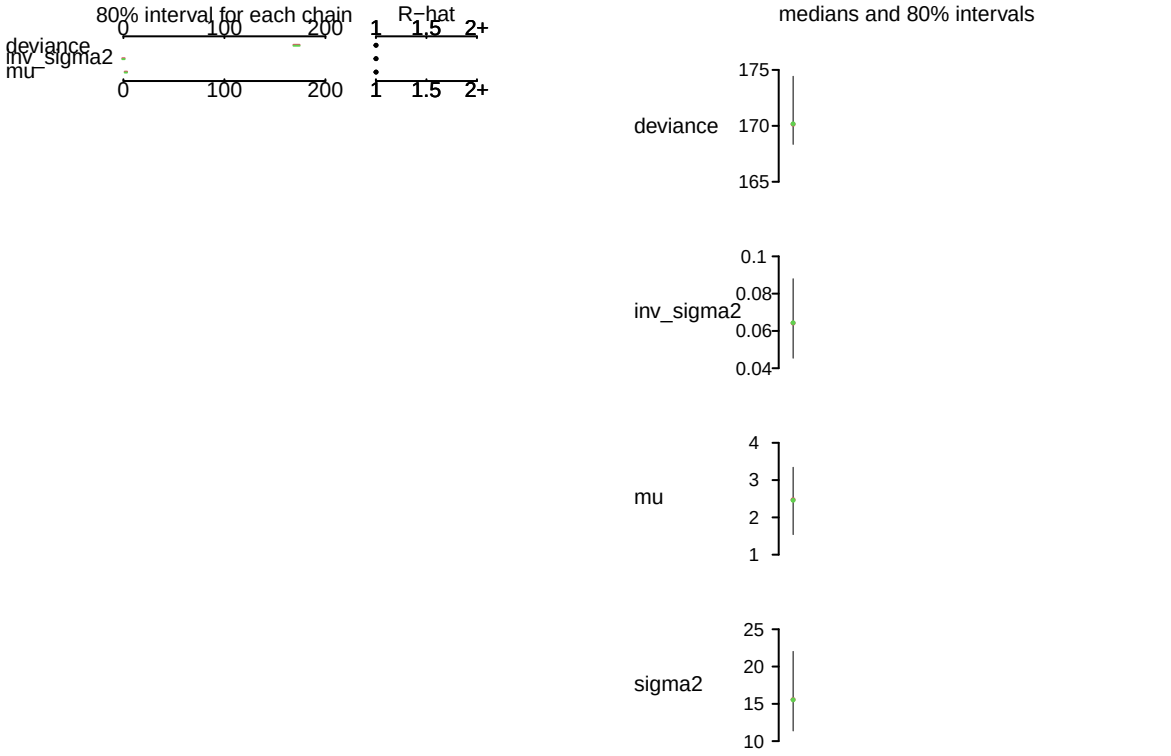
```
###############
# R2jags wrapper with a link to a file
###############


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)
)
```

```r
#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=paste(path,"05normal_exmple_JAGS.txt",sep=""),
                 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
```

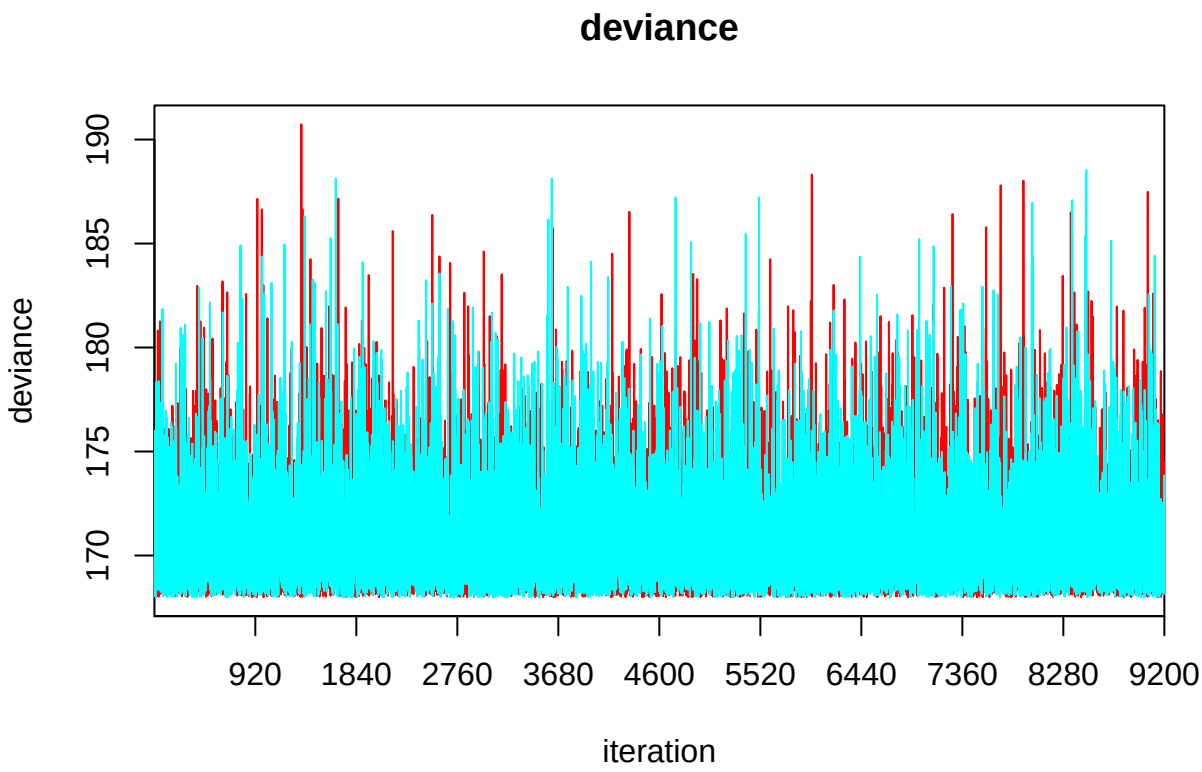
```r
# Standard plots of the monitored variables
plot(fit.R2jags)
```

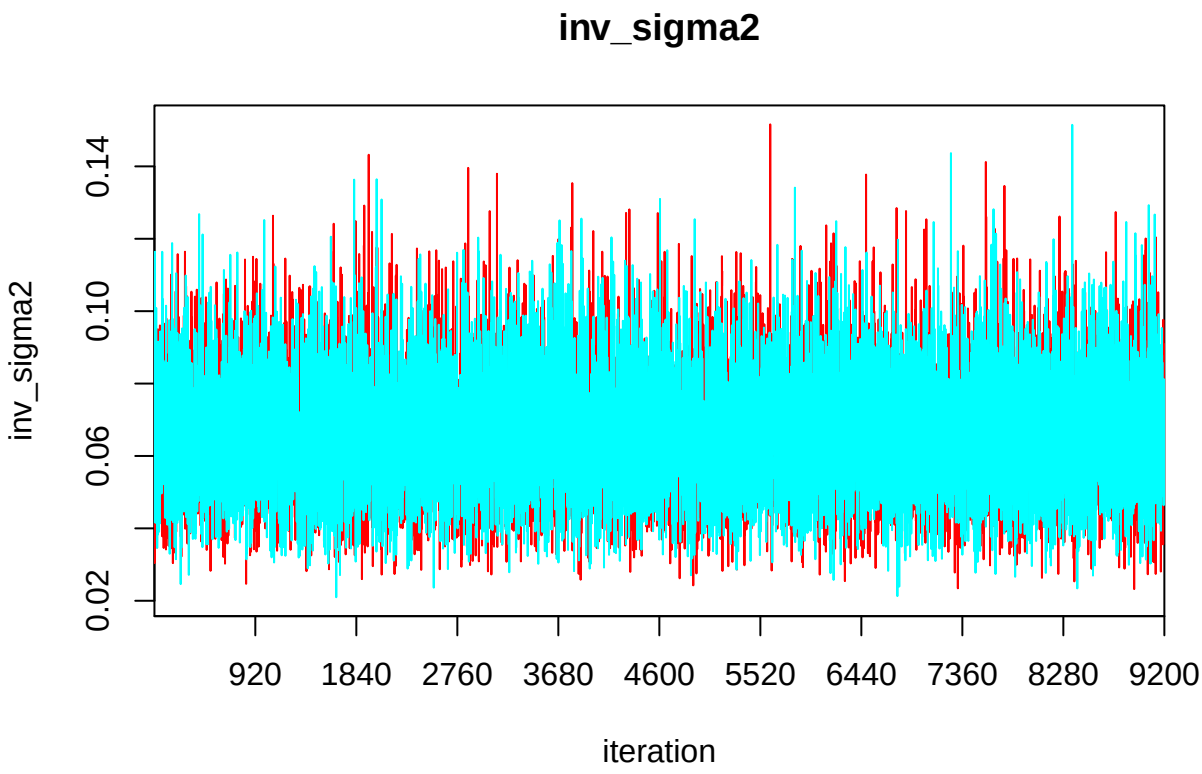


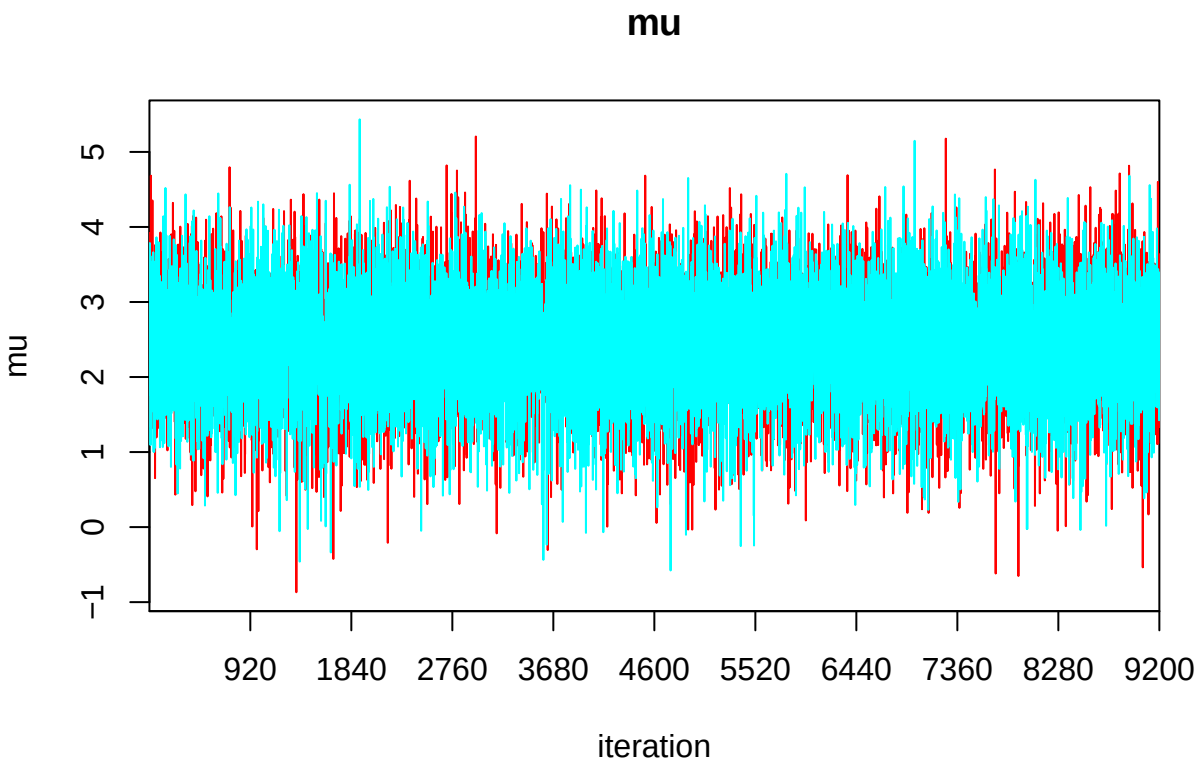

```
# Display summary statistics
print(fit.R2jags)
```

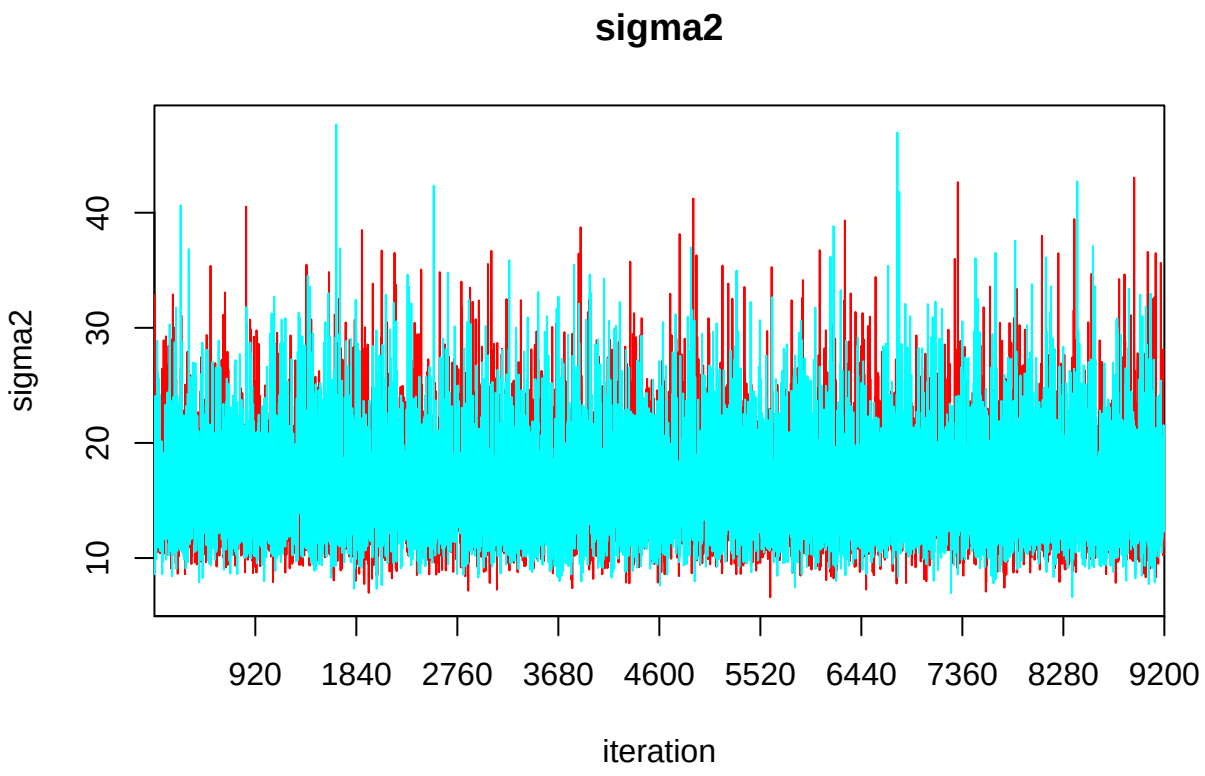
```
## Inference for Bugs model at "05normal_exmple_JAGS.txt", fit using jags,
##  2 chains, each with 50000 iterations (first 4000 discarded), n.thin = 5
##  n.sims = 18400 iterations saved
##             mu.vect sd.vect    2.5%     25%     50%     75%   97.5%  Rhat n.eff
## inv_sigma2   0.066   0.017   0.037   0.054   0.064   0.076   0.102 1.001 18000
## mu           2.451   0.715   0.993   1.991   2.466   2.933   3.823 1.001  6200
## sigma2      16.290   4.445   9.778  13.131  15.558  18.664  27.093 1.001 18000
## deviance   170.904   2.687 168.076 168.928 170.161 172.092 178.005 1.001  9300
##
## 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)
```


deviance
deviance
190
185
180
175
170
920
1840
2760
3680
4600
5520
6440
7360
8280
9200
iteration


inv_sigma2
inv_sigma2
0.02
0.06
0.10
0.14
920
1840
2760
3680
4600
5520
6440
7360
8280
9200
iteration


mu
mu
5
4
3
2
1
0
−1
920
1840
2760
3680
4600
5520
6440
7360
8280
9200
iteration




```
# 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   170.90404 2.68651 0.0198052      0.0196358
## inv_sigma2   0.06567 0.01677 0.0001236      0.0001266
## mu           2.45117 0.71547 0.0052745      0.0052335
## sigma2      16.28987 4.44454 0.0327656      0.0327655
##
## 2. Quantiles for each variable:
##
##                2.5%       25%       50%       75%    97.5%
## deviance   168.07627 168.92819 170.16068 172.09243 178.0055
## inv_sigma2   0.03691   0.05358   0.06428   0.07616   0.1023
## mu           0.99271   1.99140   2.46569   2.93277   3.8229
## sigma2       9.77825  13.13110  15.55786  18.66396  27.0926
```

```
# conduct CODA
```

Now 2 Markov chains are used and a burnin of 4000. This time `n.iter` = 50000 includes the burnin period. The thinning is set to 5 to save memory and computation time. Also the model is called from the 05normal_exmple_JAGS.txt file. Again burnin is set directly instead of updating the model. Also this time the traceplots look alright.

Exercise 4

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

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

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

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

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

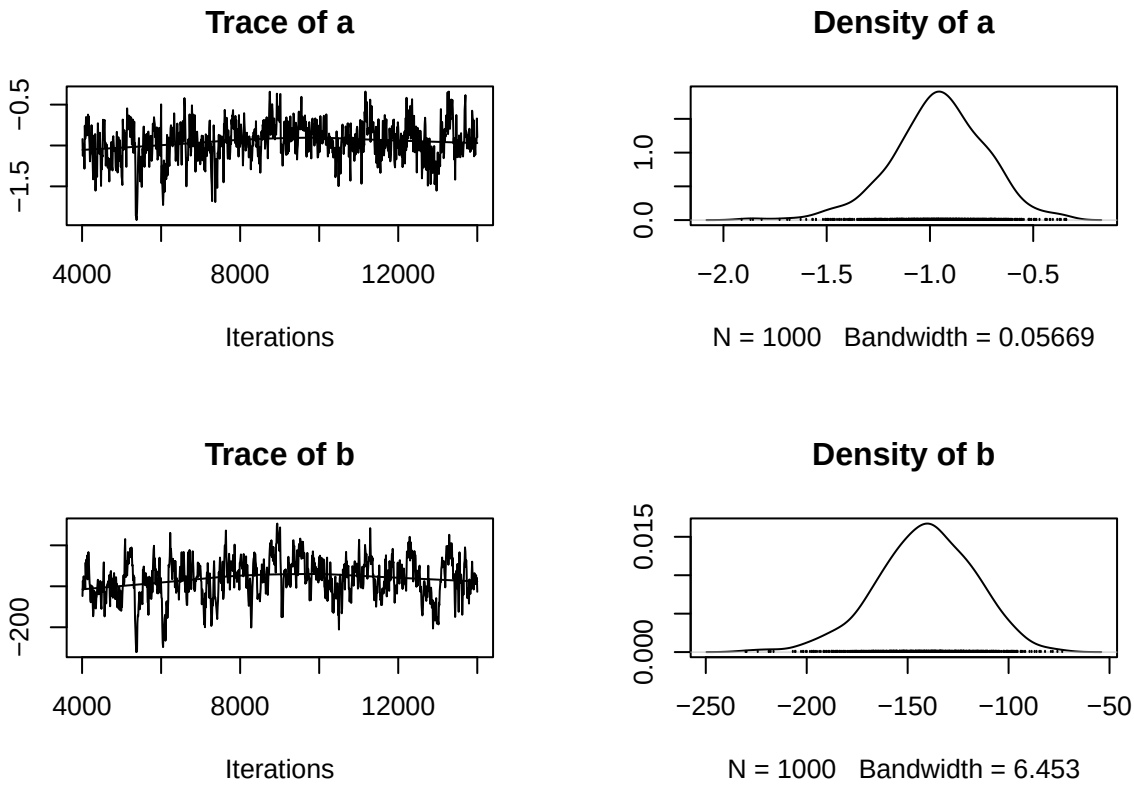
INLA

```
formula <- y ~ 1 + x

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

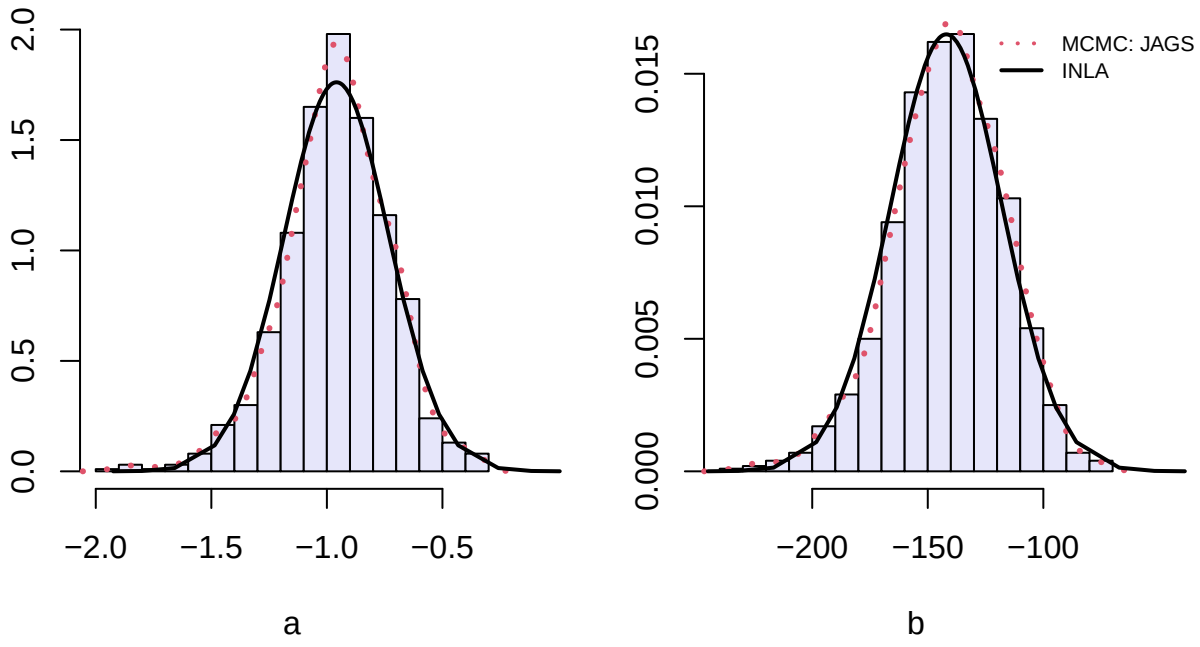
JAGS: one chain

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

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

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

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

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

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

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

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



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

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

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

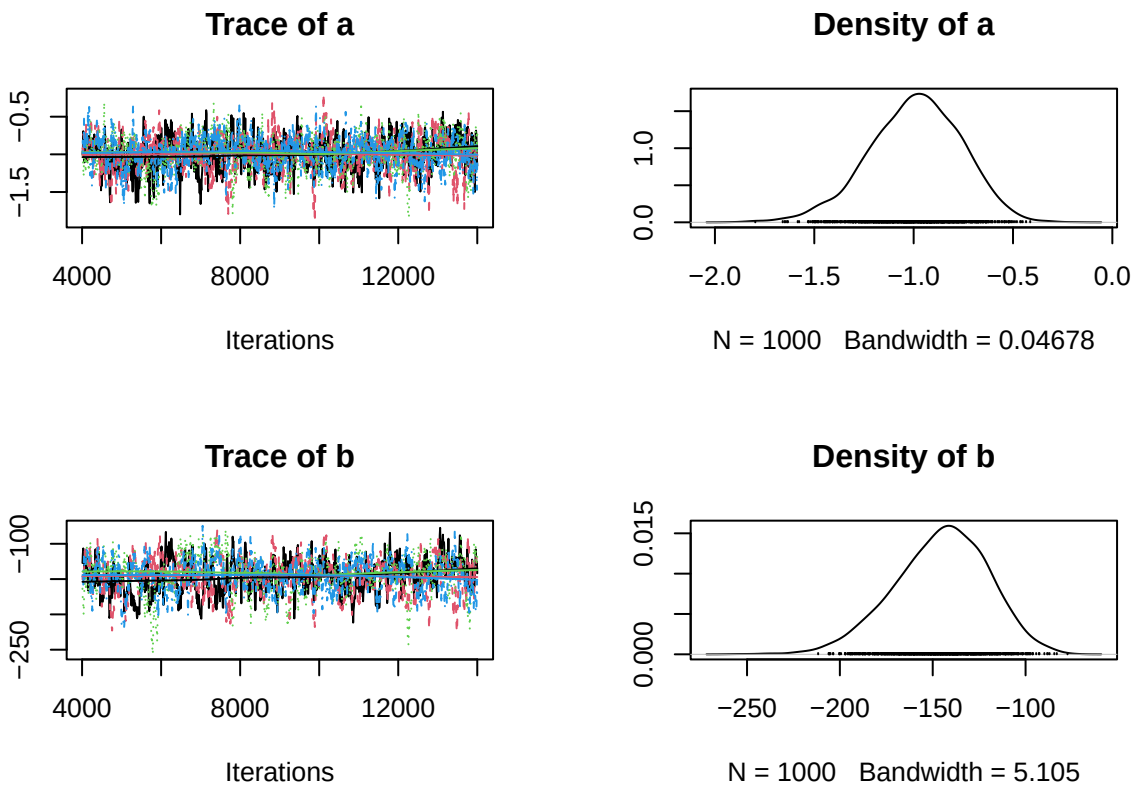



```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 6
##    Unobserved stochastic nodes: 2
##    Total graph size: 37
##
```

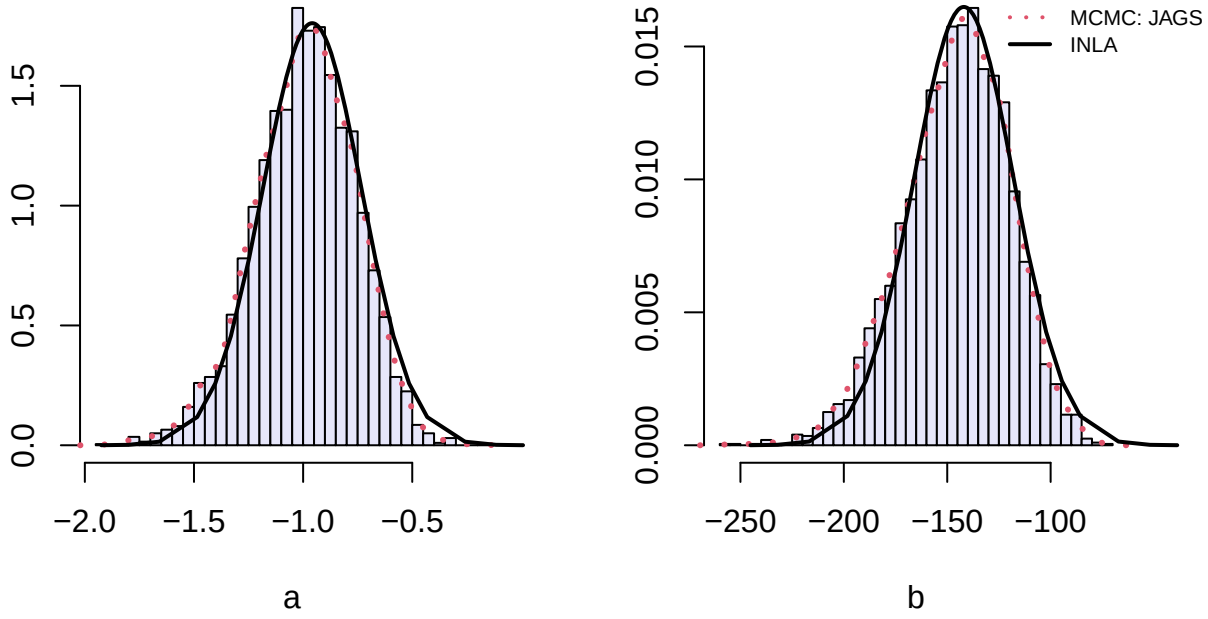
```
## Initializing model
```

JAGS: several chains

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

# model initialisation
model.jags <- jags.model(
  file = "MiceModel.txt",
  data = mice_data,
  inits = inits_4chain,
  n.chains = 4,
  n.adapt = 4000
)

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

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




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

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

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




```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 6
##    Unobserved stochastic nodes: 2
##    Total graph size: 37
##
## Initializing model
##
## [1] 14000
##
## 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
## a   -0.992  0.2318 0.003666        0.01011
## b -145.596 25.2995 0.400020        1.32299
##
## 2. Quantiles for each variable:
##
```

```
##       2.5%      25%       50%       75%   97.5%
## a   -1.483   -1.143   -0.9819   -0.8301   -0.57
## b -199.094 -161.784 -143.9363 -127.6517 -100.05
```

Classical logistic regression model

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

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

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

summary(glm.model)$coef
```

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

Conclusion/Comparison:

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

Exercise 5

Exercise 6

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

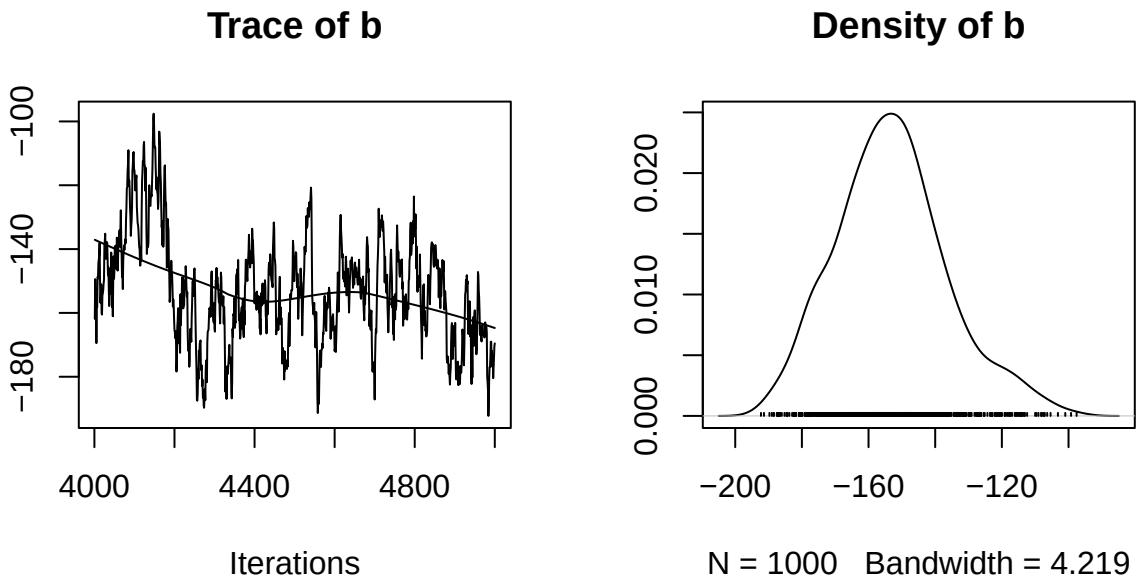
1

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

```
set.seed(44566)

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

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

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

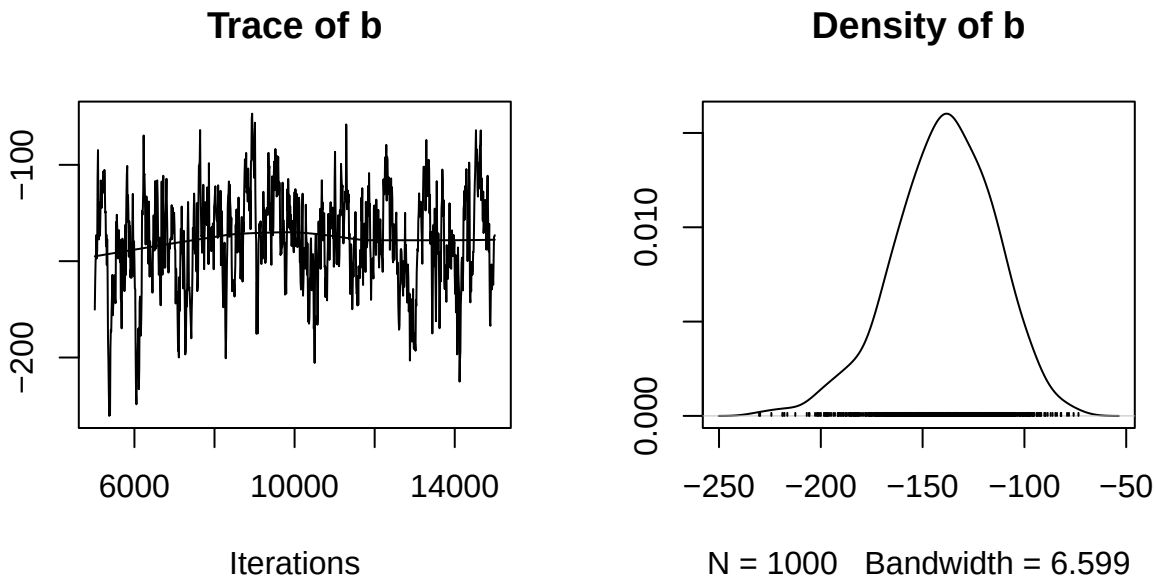
```
# burn-in
update(model.jags, n.iter = 4000)

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

#summary(fit.jags.coda)
par(mfrow=c(1,2), mar=c(10,2,4,3))
plot(fit.jags.coda_1)
```



2

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

```
set.seed(44566)

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

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



(a)

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

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

(b)

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

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

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

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

Setting	custom function	coda::effectiveSize	stableGR::n.eff
n.iter = 1000, thin = 1	21.01	26.67	17.54
n.iter = 10000, thin = 10	65.411	83.97	65.049

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