wk6-bayes-hw-kenwan

AAPL returns

Using the methods of the workshop estimate parameters of normal model and robust normal model for the Apple returns.

Estimate realized arithmetic volatility of Apple returns using both normal and robust models.

Volatility is calculated as: σ=Sd*sqrt(365)

where Sd is standard deviation of daily arithmetic returns S(t)−S(t−1), S(t) is stock price.

```
getSymbols("AAPL", from="2015-1-1",to="2015-12-31")
```

```
## 'getSymbols' currently uses auto.assign=TRUE by default, but will
## use auto.assign=FALSE in 0.5-0. You will still be able to use
## 'loadSymbols' to automatically load data. getOption("getSymbols.env")
## and getOption("getSymbols.auto.assign") will still be checked for
## alternate defaults.
##
## This message is shown once per session and may be disabled by setting
## options("getSymbols.warning4.0"=FALSE). See ?getSymbols for details.
```

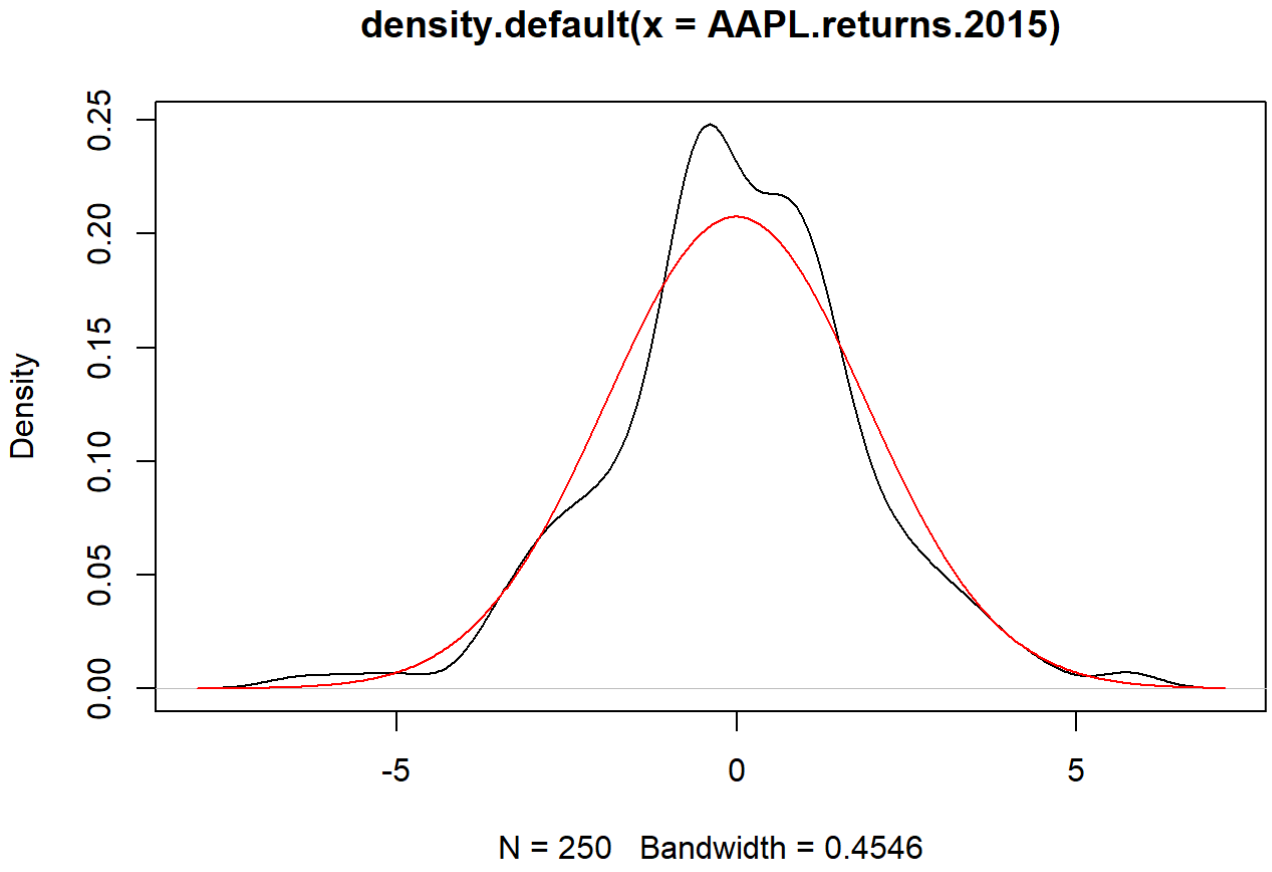
```
##
## WARNING: There have been significant changes to Yahoo Finance data.
## Please see the Warning section of '?getSymbols.yahoo' for details.
##
## This message is shown once per session and may be disabled by setting
## options("getSymbols.yahoo.warning"=FALSE).
```

```
## [1] "AAPL"
```

```
AAPL.1.day<-as.matrix(AAPL)
AAPL.returns.2015<-diff(AAPL.1.day[,6])

AAPL.ret<-read.csv(paste(dataPath,"/wk6/AAPL_2015.csv",sep="/"))$x
```

Visualize the distribution of the returns.

```
AAPL.dens<-density(AAPL.returns.2015)
plot(AAPL.dens)
lines(AAPL.dens$x,dnorm(AAPL.dens$x,mean(AAPL.ret),sd(AAPL.ret)),col="red")
```



fit normal model for 1 group with no predictors.

Prep of data.

```
# Observed sigma.
(sigma = sd(AAPL.returns.2015)*sqrt(365))
```

```
## [1] 35.81267
```

```
# preparing data
set.seed(9384756)
y <- AAPL.returns.2015
Ntotal = length(y)

dataList = list(
  y = y ,
  Ntotal = Ntotal ,
  mean_mu = mean(y) ,
  sd_mu = sd(y)
)
```

Let's run it in jAGS.

```
modelString = "
model {
  for ( i in 1:Ntotal ) {
    y[i] ~ dnorm( mu , 1/sigma^2 )
  }
  mu ~ dnorm(mean_mu , 1/(100*sd_mu)^2)
  sigma ~ dunif( sd_mu/1000 , sd_mu*1000 )
}
" # close quote for modelString
# Write out modelString to a text file
writeLines( modelString , con="TEMPmodel.txt" )
```

Init

```
initsList <-function() {
  upDown<-sample(c(1,-1),1)
  m <- mean(y)*(1+upDown*.05)
  s <- sd(y)*(1-upDown*.1)
  list( mu = m , sigma = s)
}
```

Run

```
parameters = c( "mu" , "sigma")     # The parameters to be monitored
adaptSteps = 500               # Number of steps to "tune" the samplers
burnInSteps = 1000
numSavedSteps=50000
nChains = 4
thinSteps = 1
nIter = ceiling( ( numSavedSteps * thinSteps ) / nChains )
  # Create, initialize, and adapt the model:

jagsModel = jags.model( "TEMPmodel.txt" , data=dataList , inits=initsList ,
                        n.chains=nChains , n.adapt=adaptSteps )
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 250
##    Unobserved stochastic nodes: 2
##    Total graph size: 266
##
## Initializing model
```

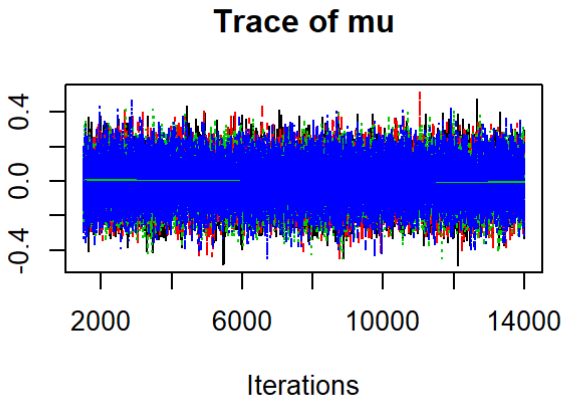
```
# Burn-in:
update( jagsModel , n.iter=burnInSteps )

# Run it
# The saved MCMC chain:
codaSamples = coda.samples( jagsModel , variable.names=parameters ,
                        n.iter=nIter , thin=thinSteps )
```

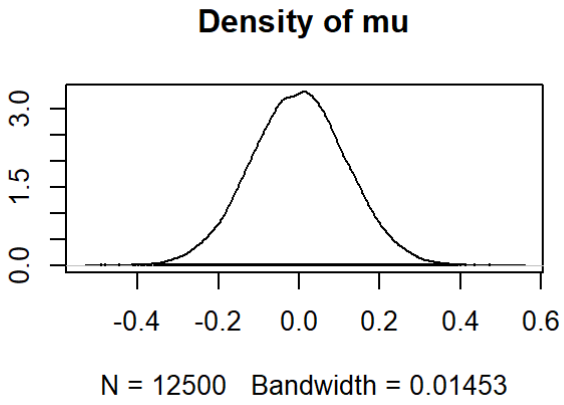
Check

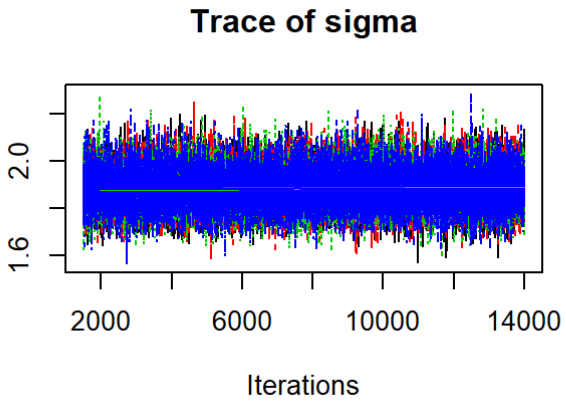
```
summary(codaSamples)
```

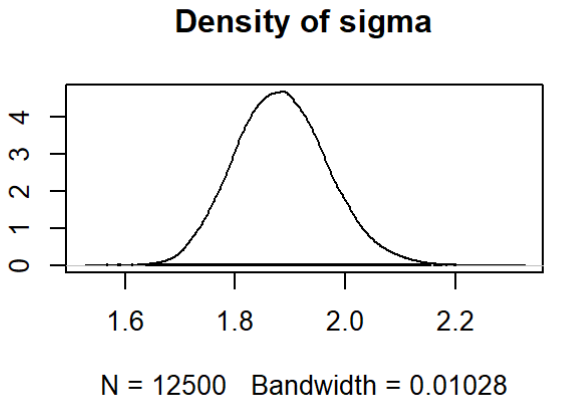
```
##
## Iterations = 1501:14000
## Thinning interval = 1
## Number of chains = 4
## Sample size per chain = 12500
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##             Mean      SD  Naive SE Time-series SE
## mu    -0.0006495 0.11935 0.0005337      0.0005337
## sigma  1.8845269 0.08445 0.0003777      0.0004958
##
## 2. Quantiles for each variable:
##
##          2.5%      25%        50%     75%  97.5%
## mu    -0.2363 -0.08059 -0.0002131 0.07929 0.2334
## sigma  1.7275  1.82605  1.8821045 1.93966 2.0591
```

```
plot(codaSamples)
```





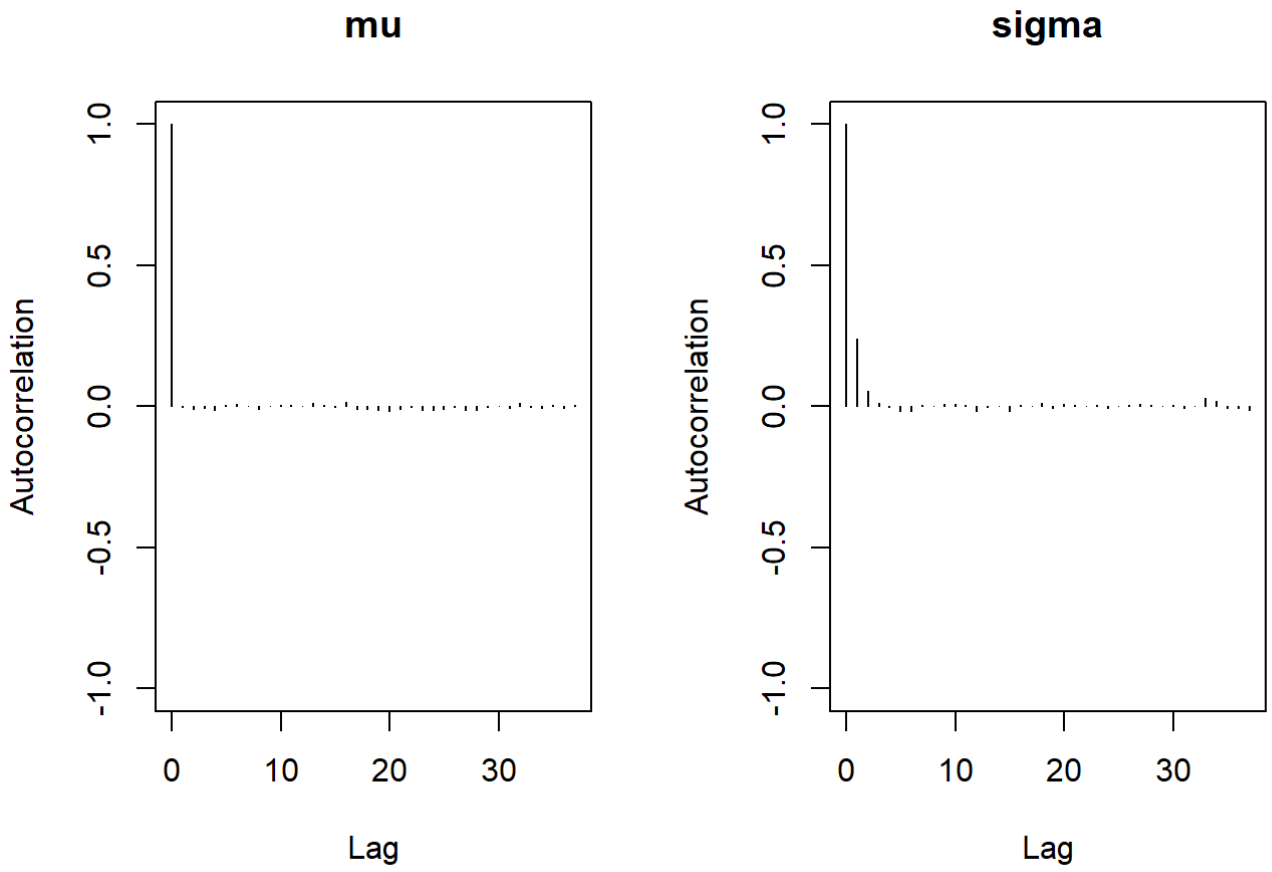


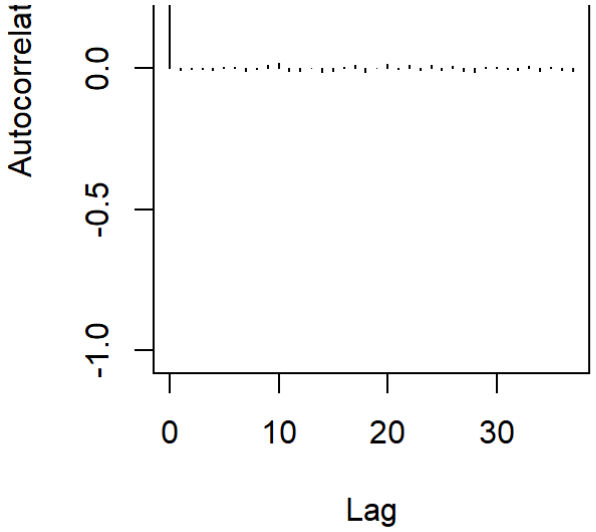


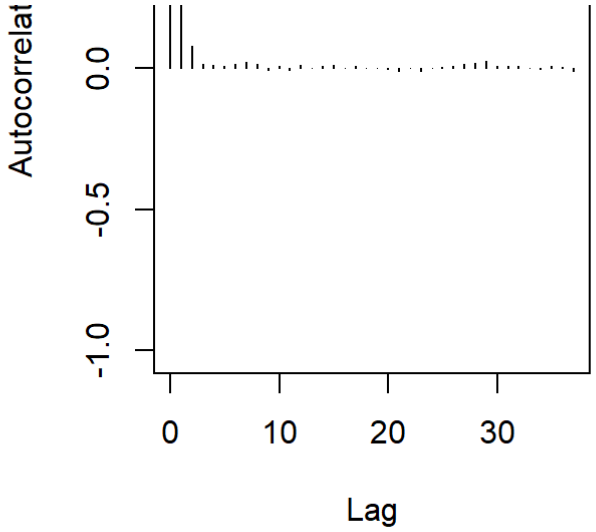

```
sd(y)
```

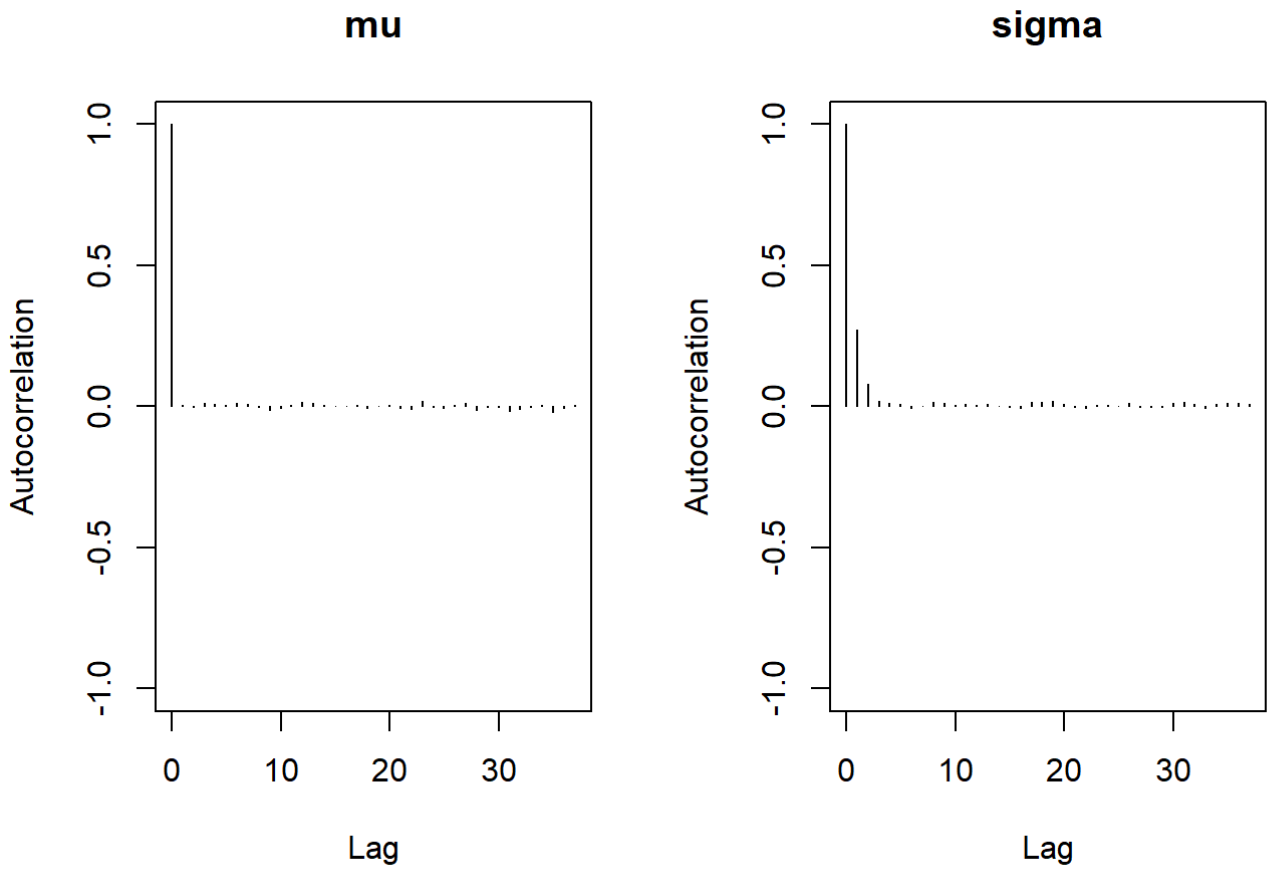
```
## [1] 1.874521
```

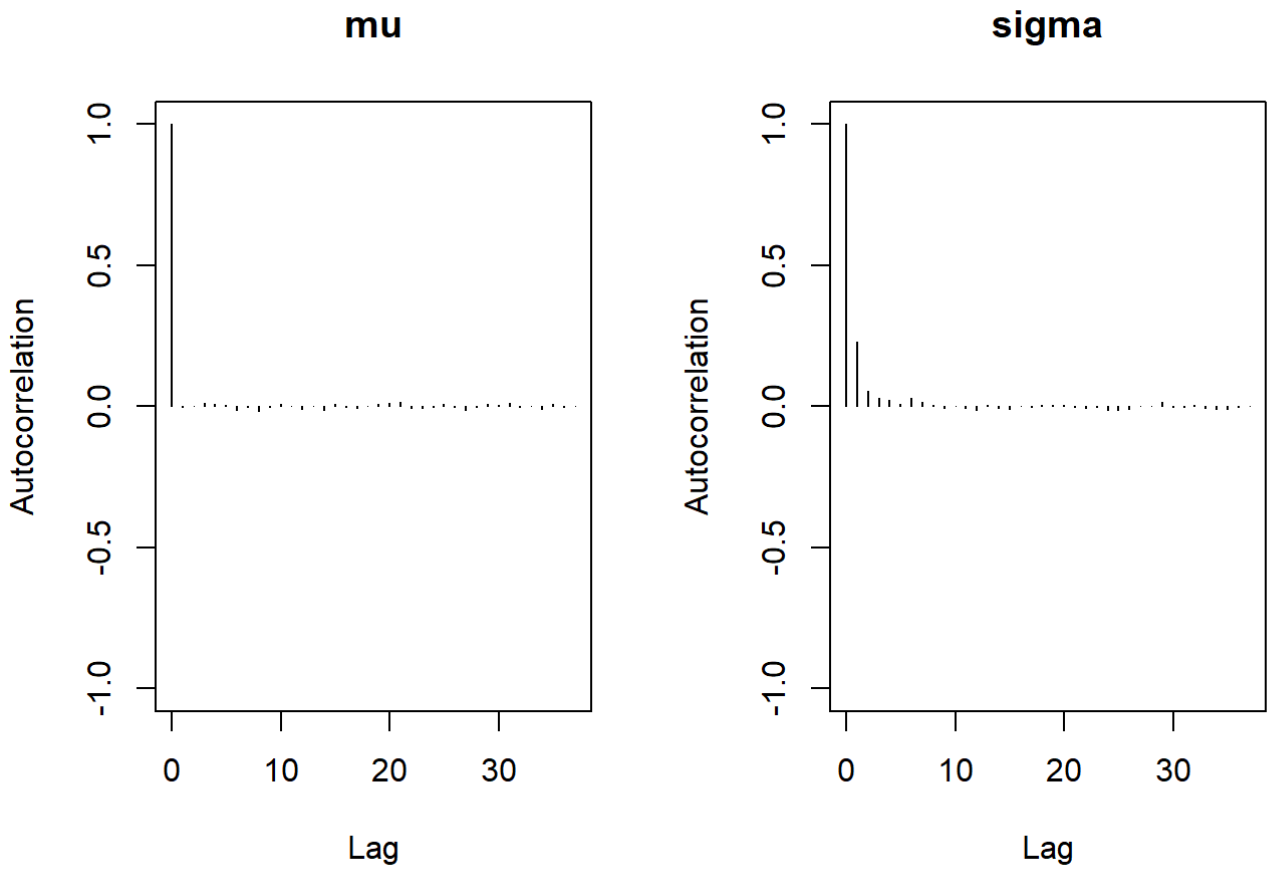
```
autocorr.plot(codaSamples,ask=F)
```











SD is closed to the observed and looks significant. Mu is not.

Overall, looks like it had no issues converging, bbut sigma plot does show some slight autocorrelation after lag 1.

More dianostics.

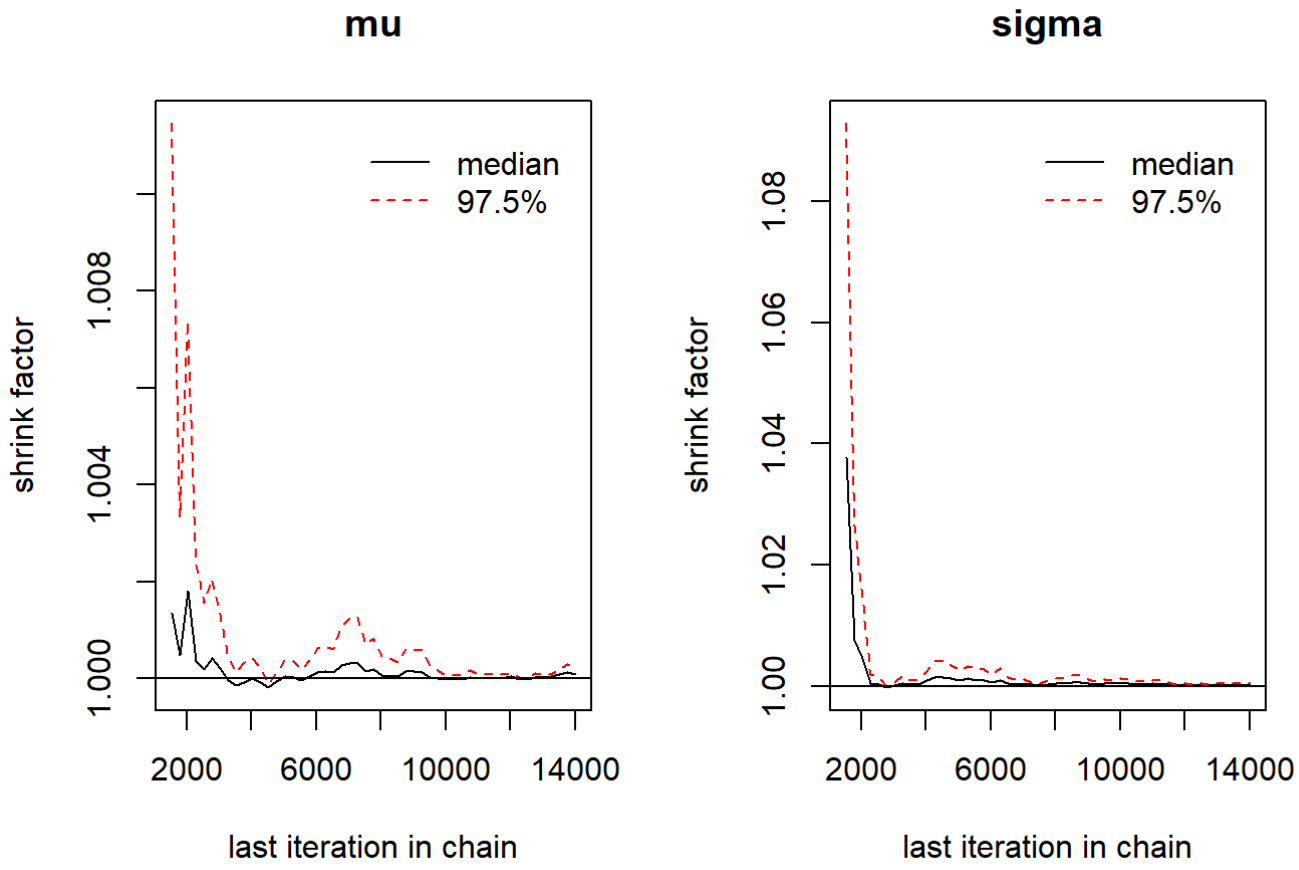
```
Ntotal
```

```
## [1] 250
```

```
effectiveSize(codaSamples)
```

```
##       mu    sigma
## 50000.00 29046.65
```

```
gelman.plot(codaSamples)
```



```
lapply(codaSamples,function(z) hdi(as.matrix(z)))
```

```
## [[1]]
##                mu    sigma
## lower -0.2339639 1.724318
## upper  0.2277296 2.051052
## attr(,"credMass")
## [1] 0.95
##
## [[2]]
##                mu    sigma
## lower -0.2345372 1.720200
## upper  0.2355874 2.049839
## attr(,"credMass")
## [1] 0.95
##
## [[3]]
##                mu    sigma
## lower -0.2305202 1.721172
## upper  0.2343708 2.048856
## attr(,"credMass")
## [1] 0.95
##
## [[4]]
##                mu    sigma
## lower -0.2368205 1.717573
## upper  0.2381566 2.046750
## attr(,"credMass")
## [1] 0.95
```

samples

```
head(codaSamples[1])
```

```
## [[1]]
## Markov Chain Monte Carlo (MCMC) output:
## Start = 1501
## End = 1507
## Thinning interval = 1
##               mu    sigma
## [1,] -0.09937405 1.855925
## [2,] -0.18990554 1.914891
## [3,] -0.02852219 1.834646
## [4,]  0.01963885 1.843079
## [5,]  0.12527833 1.803096
## [6,] -0.10954132 1.842440
## [7,] -0.06134141 2.004177
##
## attr(,"class")
## [1] "mcmc.list"
```

HDI

```
(HDIofChains<-lapply(codaSamples,function(z) hdi(as.matrix(z))))
```

```
## [[1]]
##               mu    sigma
## lower -0.2339639 1.724318
## upper  0.2277296 2.051052
## attr(,"credMass")
## [1] 0.95
##
## [[2]]
##               mu    sigma
## lower -0.2345372 1.720200
## upper  0.2355874 2.049839
## attr(,"credMass")
## [1] 0.95
##
## [[3]]
```

```
##               mu    sigma
## lower -0.2305202 1.721172
## upper  0.2343708 2.048856
## attr(,"credMass")
## [1] 0.95
##
## [[4]]
##               mu    sigma
## lower -0.2368205 1.717573
## upper  0.2381566 2.046750
## attr(,"credMass")
## [1] 0.95
```

Now let's repeat with a robust normal.

Let's run it in jAGS.

```
modelString = "
model {
for ( i in 1:Ntotal ) {
    y[i] ~ dt(mu,1/sigma^2,nu)
}
mu ~ dnorm( mean_mu , 1/(100*sd_mu)^2 )
sigma ~ dunif( sd_mu/1000 , sd_mu*1000 )
nu ~ dexp(1/30.0)
}
  " # close quote for modelString
  # Write out modelString to a text file
writeLines( modelString , con="TEMPmodel.txt" )
```

Init

```
initsList <-function() {
  upDown<-sample(c(1,-1),1)
  m <- mean(y)*(1+upDown*.05)
```

```
  s <- sd(y)*(1-upDown*.1)
  list( mu = m , sigma = s,nu=2)
}
```

Run

```
parameters = c( "mu" , "sigma" , "nu" )     # The parameters to be monitored
adaptSteps = 500               # Number of steps to "tune" the samplers
burnInSteps = 1000
nChains = 3
thinSteps = 1
numSavedSteps=50000
(nIter = ceiling( ( numSavedSteps * thinSteps ) / nChains ))
```

```
## [1] 16667
```

```
# Create, initialize, and adapt the model:
jagsModel = jags.model( "TEMPmodel.txt" , data=dataList , inits=initsList ,
                  n.chains=nChains , n.adapt=adaptSteps )
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 250
##    Unobserved stochastic nodes: 3
##    Total graph size: 269
##
## Initializing model
```

```
# Burn-in:
update( jagsModel , n.iter=burnInSteps )
# The saved MCMC chain:
```

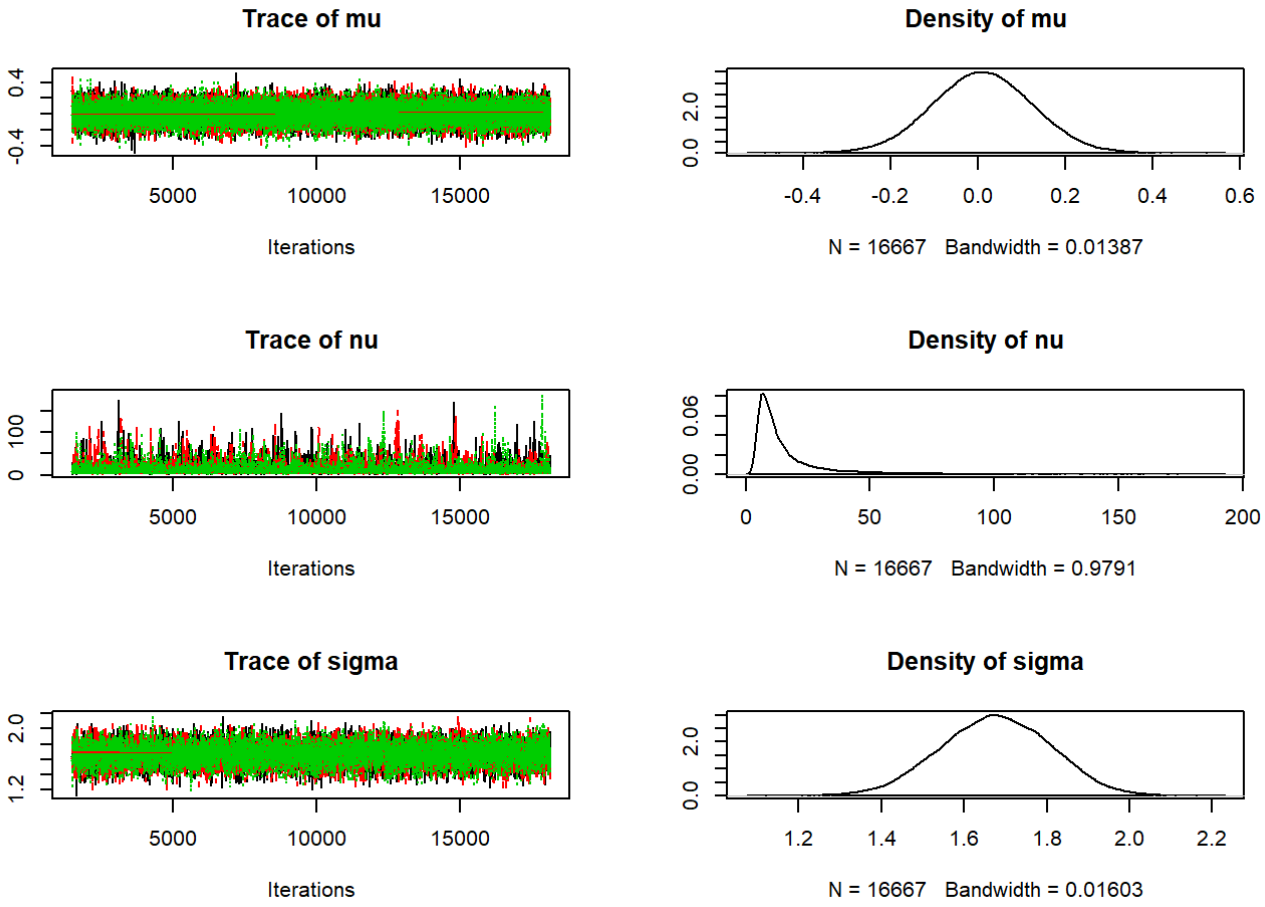
```
codaSamples = coda.samples( jagsModel , variable.names=parameters ,
                      n.iter=nIter , thin=thinSteps )
```

Check

```
summary(codaSamples)
```

```
##
## Iterations = 1501:18167
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 16667
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##            Mean      SD  Naive SE Time-series SE
## mu     0.009645  0.1139 0.0005095      0.0006405
## nu    15.999595 16.1068 0.0720310      0.3389063
## sigma  1.675480  0.1317 0.0005888      0.0016824
##
## 2. Quantiles for each variable:
##
##          2.5%      25%       50%      75%   97.5%
## mu    -0.2133 -0.06716  0.009307  0.08612  0.2324
## nu     4.1246  7.06968 10.509945 17.84480 63.5852
## sigma  1.4174  1.58548  1.676331  1.76773  1.9264
```

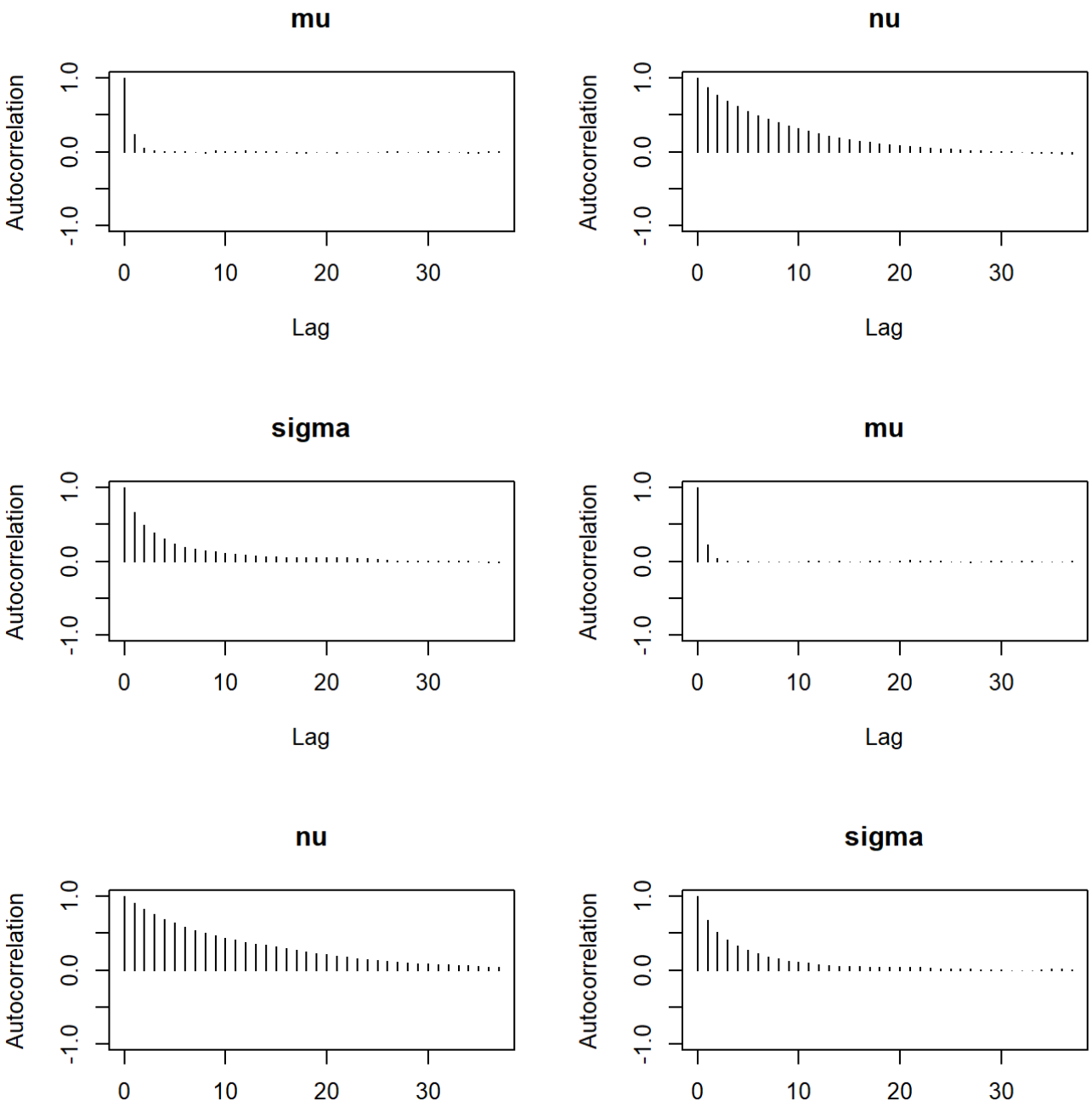
```
plot(codaSamples)
```

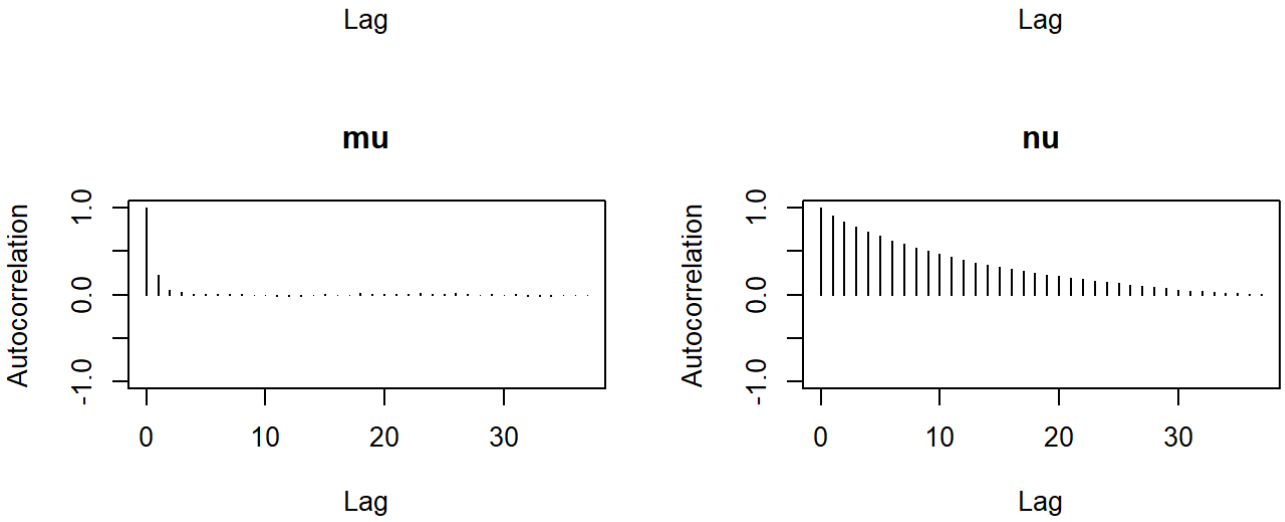


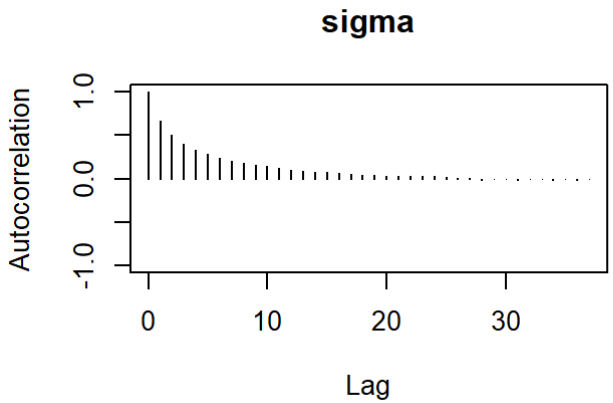

```
sd(y)
```

```
## [1] 1.874521
```

```
autocorr.plot(codaSamples,ask=F)
```







Mu and sigma trace plots look okay.

Autocorrelation does seem like an issue. The lag autocorrelation is quite long (15 or so).

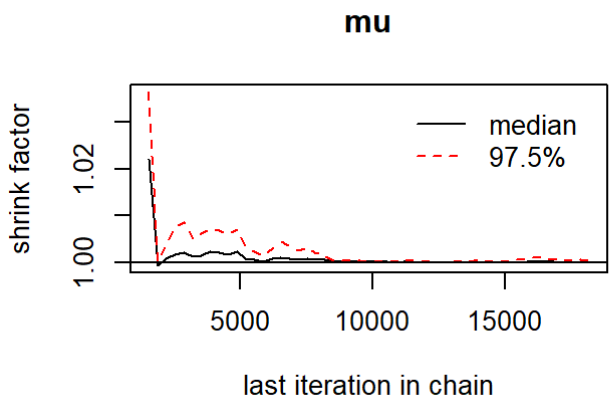
```
Ntotal
```

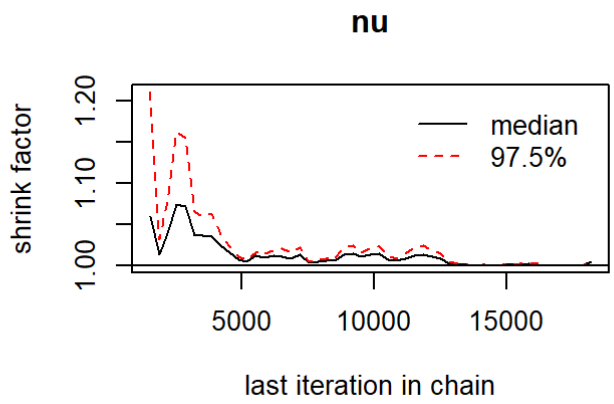
```
## [1] 250
```

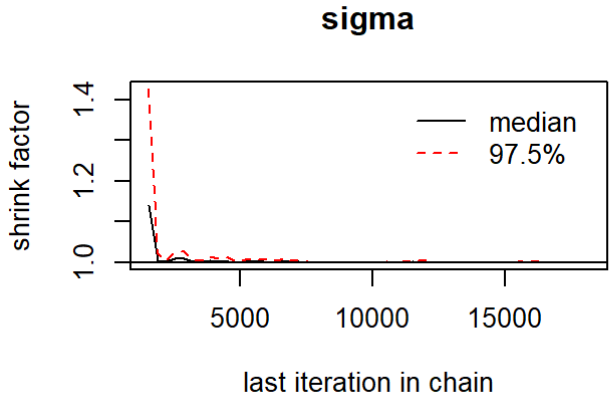
```
effectiveSize(codaSamples)
```

```
##        mu        nu     sigma
## 31655.996  2321.338  6138.830
```

```
gelman.plot(codaSamples)
```








```
lapply(codaSamples,function(z) hdi(as.matrix(z)))
```

```
## [[1]]
##                mu        nu    sigma
## lower -0.2054288  2.668874 1.418590
## upper  0.2384919 49.131664 1.923963
## attr(,"credMass")
## [1] 0.95
##
## [[2]]
##                mu        nu    sigma
## lower -0.2130176  2.437007 1.426751
## upper  0.2322023 47.776510 1.940113
## attr(,"credMass")
## [1] 0.95
##
## [[3]]
##                mu        nu    sigma
## lower -0.2131374  2.595064 1.417140
## upper  0.2336948 47.289261 1.921236
## attr(,"credMass")
## [1] 0.95
```

HDI.

Nu HDI is huge!

```
head(codaSamples[1])
```

```
## [[1]]
## Markov Chain Monte Carlo (MCMC) output:
## Start = 1501
## End = 1507
## Thinning interval = 1
##                mu        nu    sigma
## [1,]  0.29461742  6.785205 1.640994
## [2,] -0.06322534  7.162731 1.499485
## [3,] -0.08146829  7.498677 1.585753
```

```
## [4,]  0.10208899  6.213823 1.532581
## [5,] -0.11830757  5.499770 1.553625
## [6,]  0.10233567  9.395436 1.622268
## [7,] -0.09470823 12.502827 1.662198
##
## attr(,"class")
## [1] "mcmc.list"
```

```
(HDIofChains<-lapply(codaSamples,function(z) hdi(as.matrix(z))))
```

```
## [[1]]
##               mu        nu    sigma
## lower -0.2054288  2.668874 1.418590
## upper  0.2384919 49.131664 1.923963
## attr(,"credMass")
## [1] 0.95
##
## [[2]]
##               mu        nu    sigma
## lower -0.2130176  2.437007 1.426751
## upper  0.2322023 47.776510 1.940113
## attr(,"credMass")
## [1] 0.95
##
## [[3]]
##               mu        nu    sigma
## lower -0.2131374  2.595064 1.417140
## upper  0.2336948 47.289261 1.921236
## attr(,"credMass")
## [1] 0.95
```

Non robust estimate of sigma is inside the HDI for all the chains.