Bayesian Computation with JAGS

JAGS is

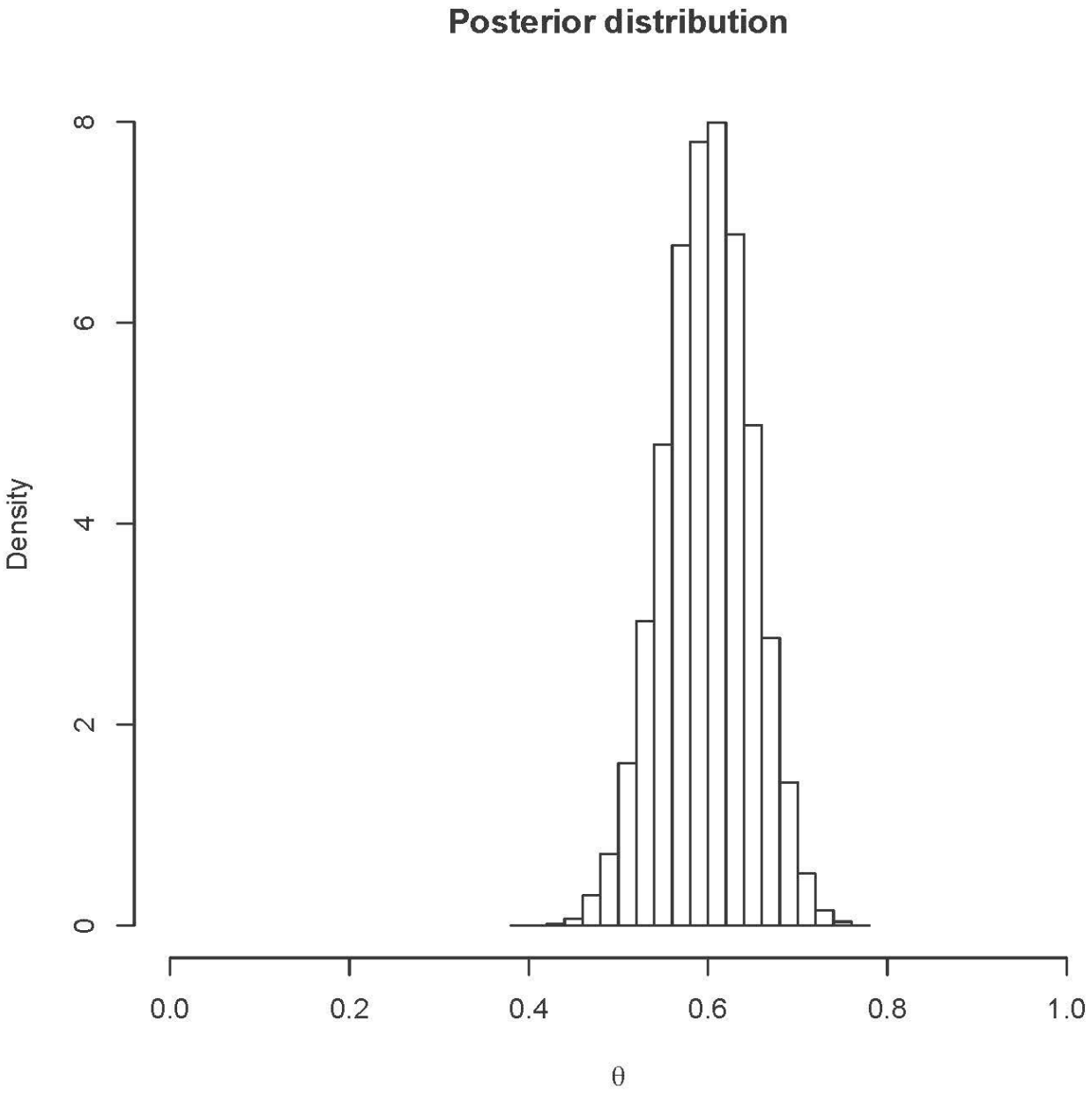
- Just Another Gibbs Sampler
- Cross-platform
- Accessible from within R

What I did

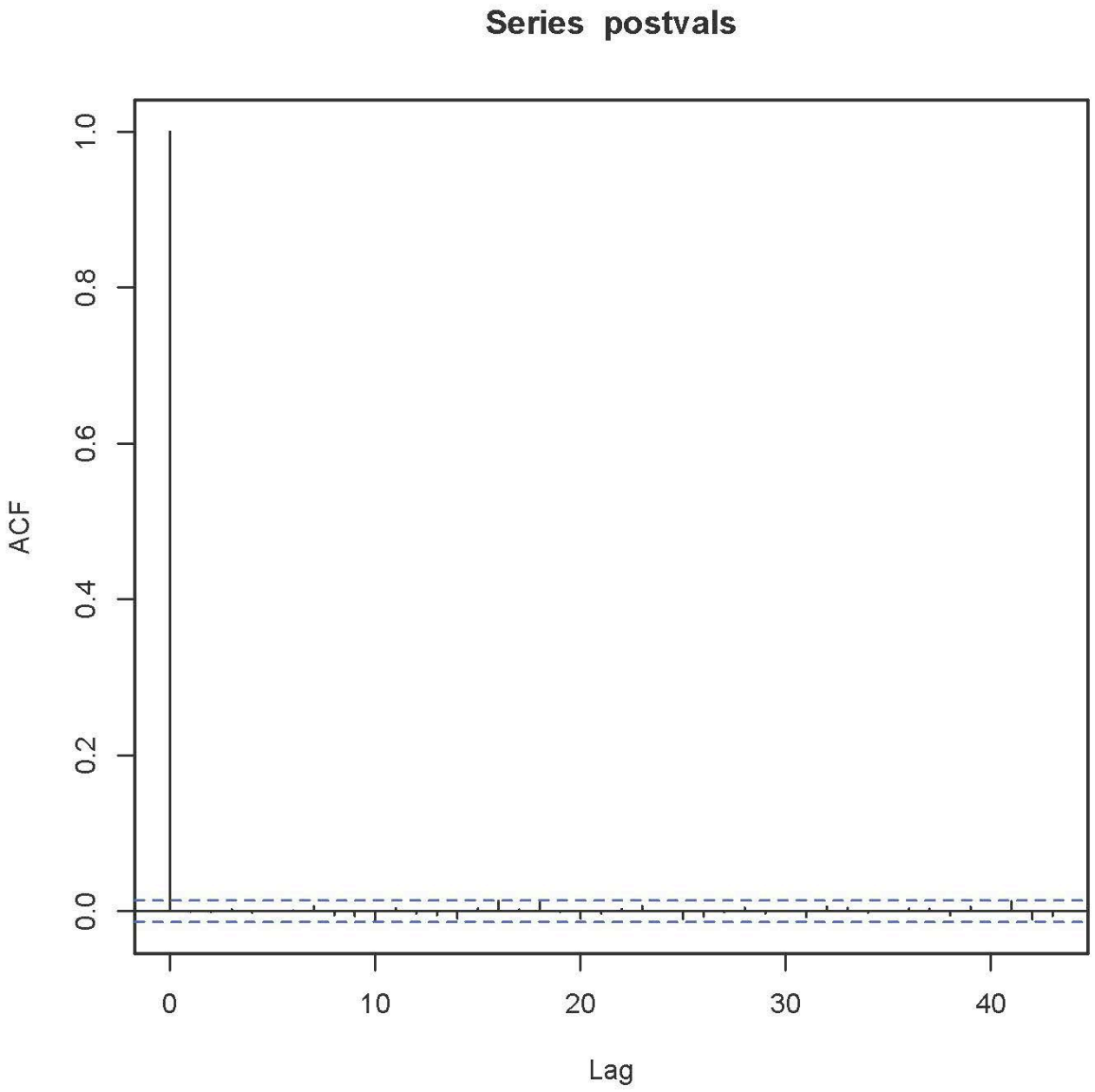
- Downloaded and installed JAGS.
- In the R package installer, downloaded `rjags` and dependencies.

```
> rm(list=ls())
> library(rjags)
Loading required package: coda
Linked to JAGS 3.4.0
Loaded modules: basemod,bugs
> # Try the coffee taste test. 60 chose the new blend.
> # Uniform prior, posterior is Beta(61,41)
> x = 60
> # The model specification
> model_string <- "model{
+ X ~ dbinom(theta,100) # p(x|theta)
+ theta ~ dbeta(1,1)    # pi(theta)
+ }"
> model <- jags.model(textConnection(model_string),
+                     data = list(X=x))
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 4

Initializing model

> update(model, 10000); # Burnin for 10000 samples
  |**************************************************| 100%
> post <- jags.samples(model,
+         variable.names=c("theta"),
+         n.iter=20000)
  |**************************************************| 100%
> summary(post)
      Length Class   Mode
theta 20000  mcarray numeric
> postvals = as.numeric(post[[1]])
> hist(postvals,probability=T,xlab=expression(theta),xlim=c(0,1), main='Posterior
distribution')
```

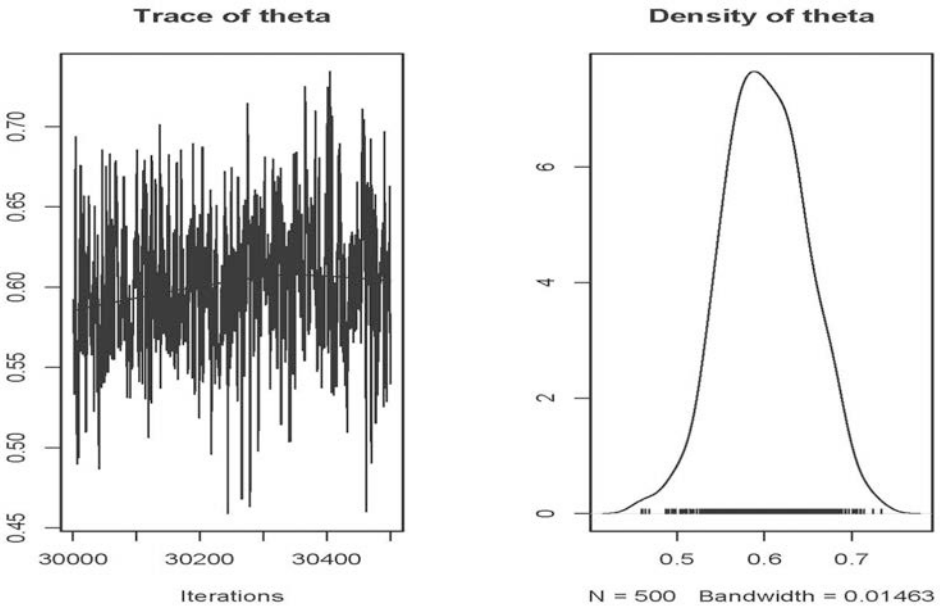
>
>


Posterior distribution
Density
0
2
4
6
8
0.0
0.2
0.4
0.6
0.8
1.0
θ

```
>
> acf(postvals) # Auto-correlation function
```



```
> a = 61; b = 41
> a/(a+b) # Expected value of theta given X
[1] 0.5980392
> mean(postvals)
[1] 0.5987098
> a*b/((a+b)^2*(a+b+1)) # Variance of theta given X
[1] 0.002333867
> var(postvals)
[1] 0.002318528
>
> # It totally worked.
> # Coda samples come with diagnostics as "methods."
> # This seems to be more popular.
>
> samp <- coda.samples(model,
+         variable.names=c("theta"),
+         n.iter=500, progress.bar="none")
>
> summary(samp)

Iterations = 30001:30500
Thinning interval = 1
Number of chains = 1
Sample size per chain = 500

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

          Mean             SD       Naive SE Time-series SE
      0.601243       0.047832       0.002139       0.002540

2. Quantiles for each variable:

  2.5%    25%    50%    75%  97.5%
0.5079 0.5684 0.5995 0.6336 0.6893

> plot(samp)
```

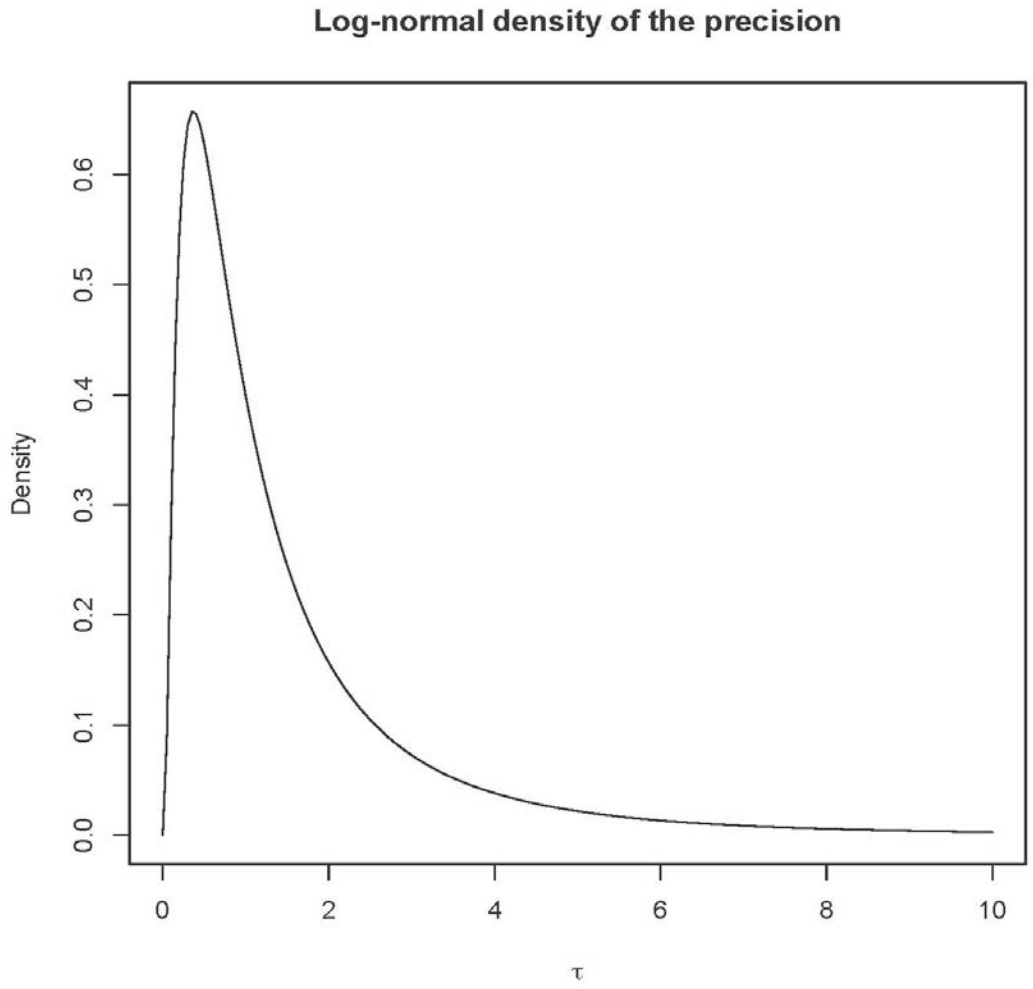


Maybe JAGS knows too much. Try normal data with a non-conjugate prior.

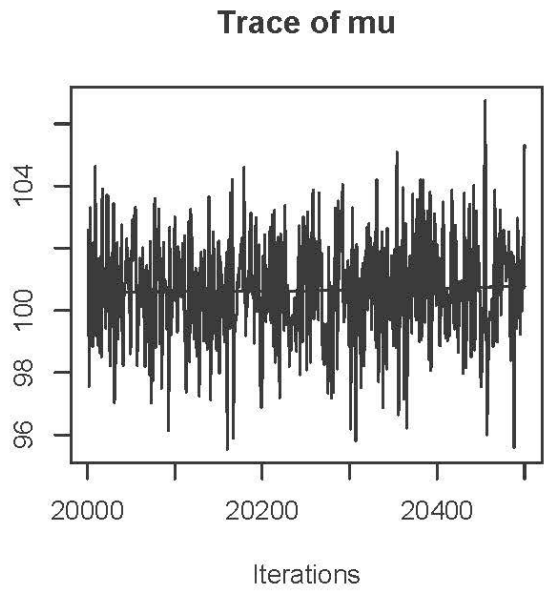
- y_i~Normal(mu,tau) # Tau is precision = 1/variance
- mu~Normal(0,100)
- tau~LogNormal(0,1)

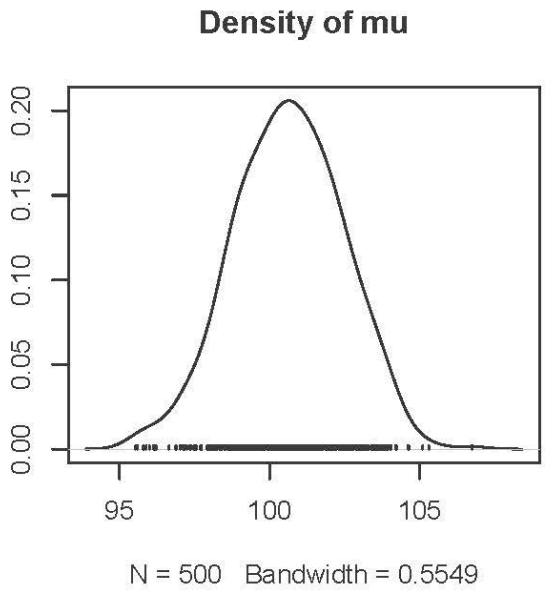
Quit R and re-start. First see what LogNormal(0,1) looks like.

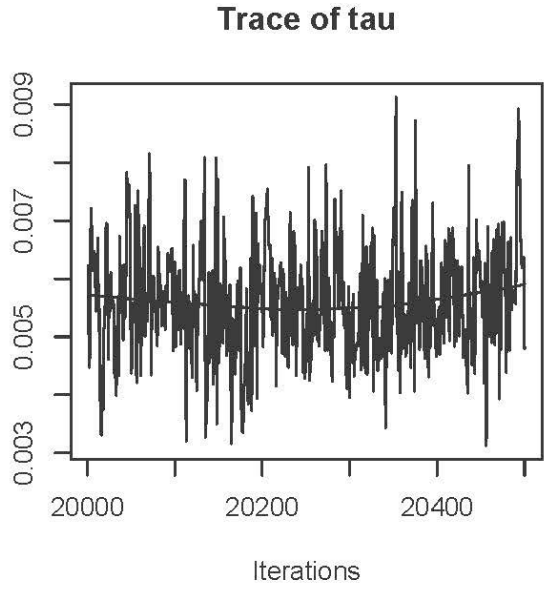
```
> rm(list=ls())
> tau = seq(from = 0,to=10,length=200)
> Density = dlnorm(tau, 0,1)
> plot(tau,Density,type='l',xlab=expression(tau),main="Log-normal density of the
precision")
```

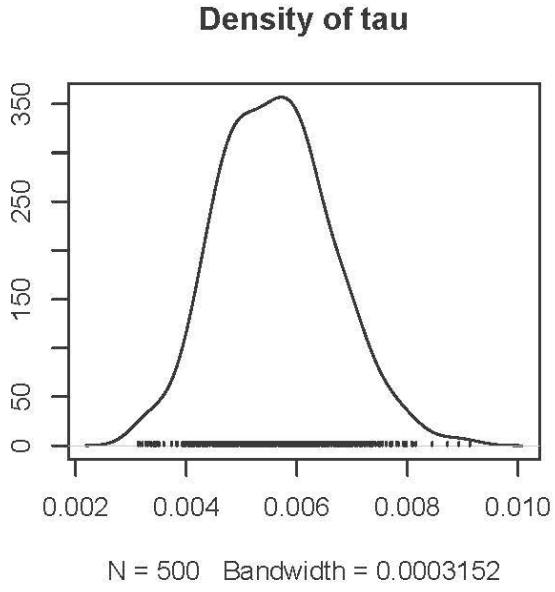


```
> rm(list=ls())
> set.seed(9999)
> y <- rnorm(n = 50, mean = 100, sd = 15) # True precision = 1/15^2 = 0.004444444
> c(mean(y),sqrt(var(y)))
[1] 100.71545  14.73441
> library(rjags)
Loading required package: coda
Linked to JAGS 3.4.0
Loaded modules: basemod,bugs
>
> # The model specification
> model_string <- "model{
+   for(i in 1:length(y)) {
+     y[i] ~ dnorm(mu, tau)
+   }
+   mu ~ dnorm(0, 0.0001) # A diffuse prior, but still proper
+   tau ~ dlnorm(0, 1)
+ }"
>
> # Running the model
> model <- jags.model(textConnection(model_string), data = list(y = y), n.chains =
1, n.adapt= 10000)
  |++++++++++++++++++++++++++++++++++++++++++++++++++| 100%
> # n.adapt:  "Initial sampling phase during which the samplers adapt their
> # behaviour to maximize their efficiency"
> update(model, 10000); # Burnin for 10000 samples
  |**************************************************| 100%
> normal_samples <- coda.samples(model, variable.names=c("mu", "tau"), n.iter=500)
  |**************************************************| 100%
> # normal_samples is a list of matrices -- one item long.
> summary(normal_samples); plot(normal_samples)

Iterations = 20001:20500
Thinning interval = 1
Number of chains = 1
Sample size per chain = 500

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

         Mean       SD  Naive SE Time-series SE
mu  1.006e+02 1.814130 8.113e-02      7.576e-02
tau 5.611e-03 0.001031 4.609e-05      6.406e-05

2. Quantiles for each variable:

         2.5%       25%       50%       75%     97.5%
mu  97.072864 99.378764 100.64321 1.019e+02 103.91211
tau  0.003735  0.004855   0.00558 6.259e-03   0.00776

>
```



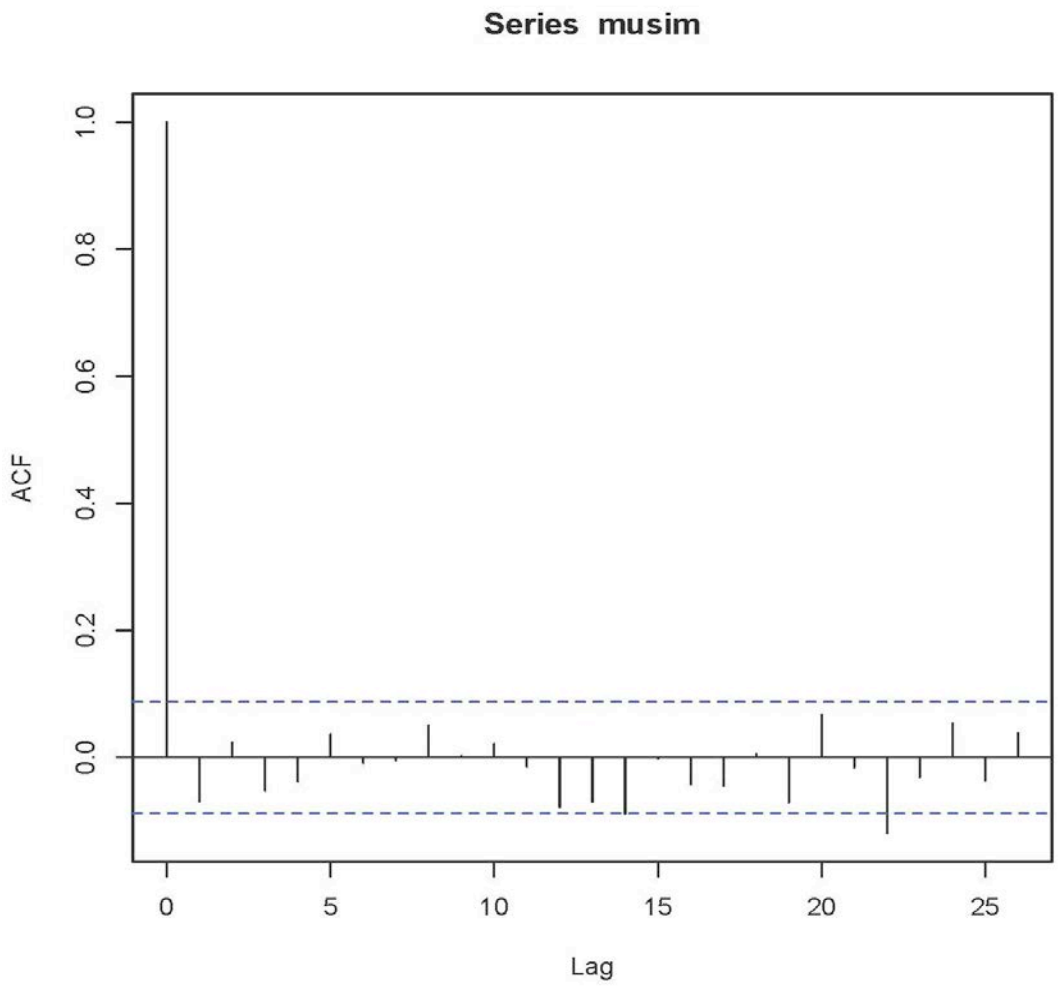




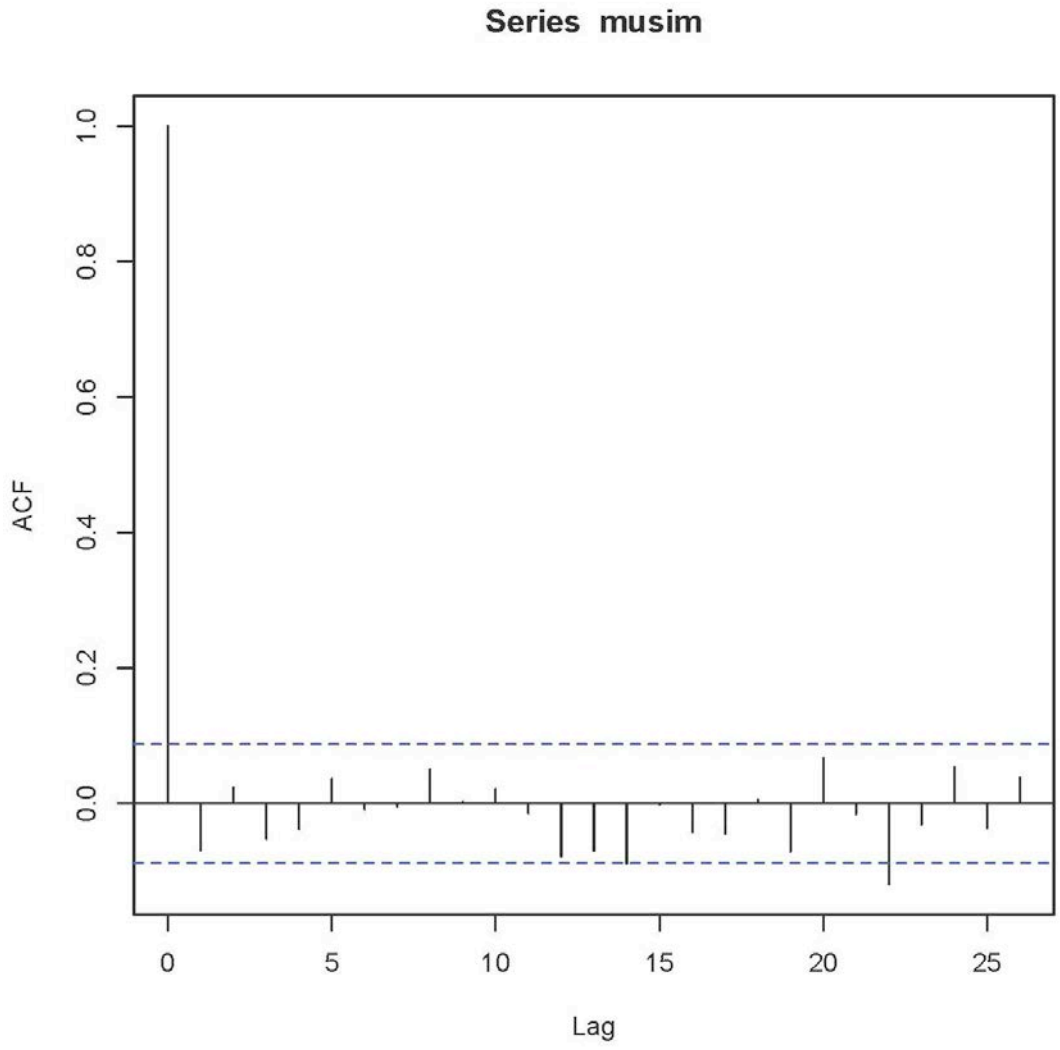



```
> normal_samples[[1]][1:10,] # First 10 rows
             mu          tau
 [1,] 102.59245 0.006222967
 [2,]  97.55598 0.004473895
 [3,] 103.30301 0.004714871
 [4,] 100.41467 0.007215652
 [5,] 100.27086 0.006870191
 [6,]  98.83192 0.006146054
 [7,] 102.17174 0.006398674
 [8,]  99.81384 0.006466838
 [9,] 104.63802 0.006029851
[10,]  99.17965 0.005442836
```

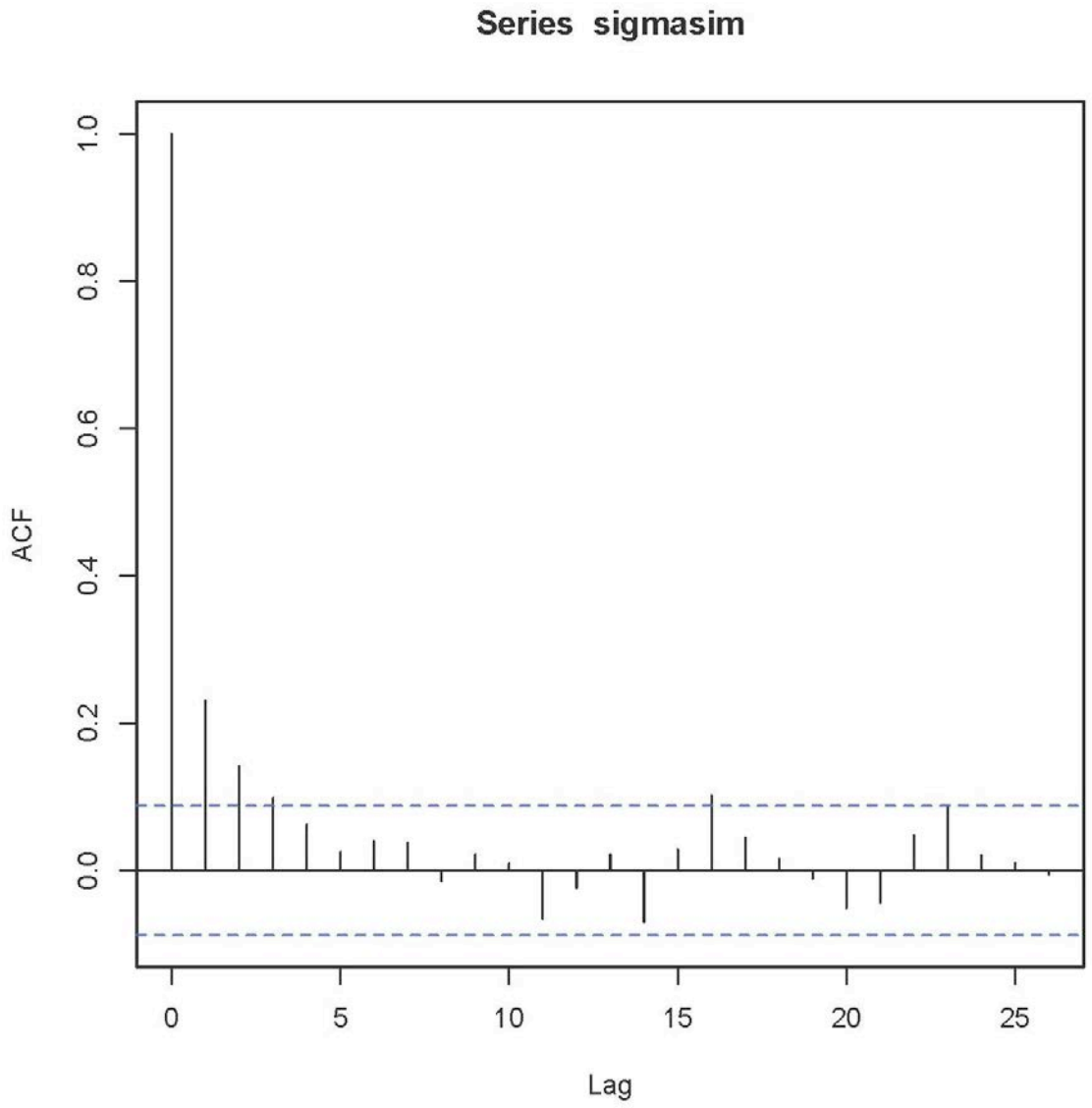
```
> musim = normal_samples[[1]][,1] # First column
> is.numeric(musim)
[1] TRUE
> summary(musim)

Iterations = 20001:20500
Thinning interval = 1
Number of chains = 1
Sample size per chain = 500

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

          Mean             SD       Naive SE Time-series SE
     100.62916        1.81413         0.08113        0.07576

2. Quantiles for each variable:

  2.5%    25%    50%    75%  97.5%
97.7    99.38 100.64 101.90 103.91


> acf(musim)
```



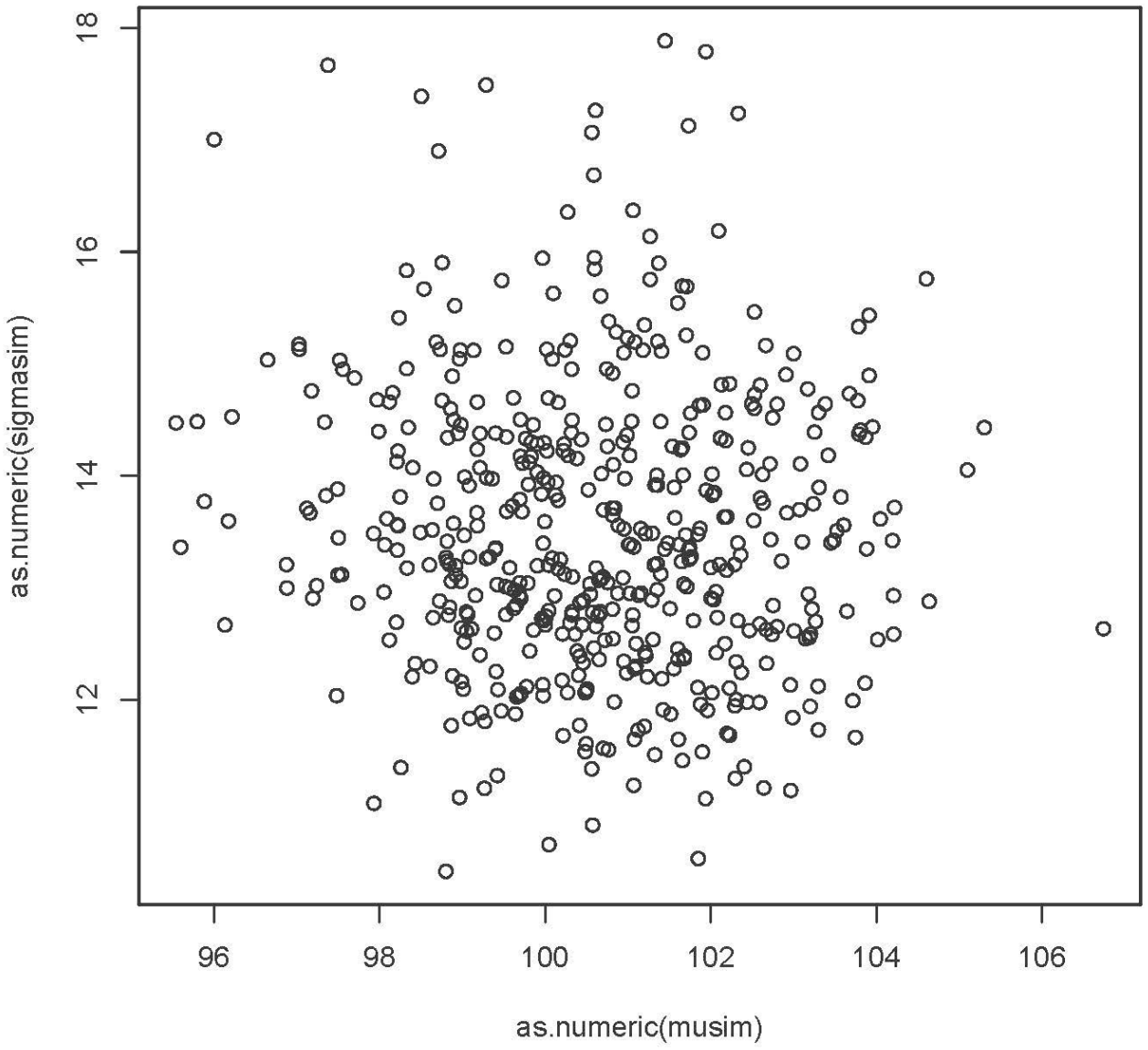
```
> tausim = normal_samples[[1]][,2] # Second column
> summary(tausim)

Iterations = 20001:20500
Thinning interval = 1
Number of chains = 1
Sample size per chain = 500

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

          Mean             SD       Naive SE Time-series SE
     5.611e-03      1.031e-03      4.609e-05      6.406e-05

2. Quantiles for each variable:

    2.5%      25%      50%      75%    97.5%
0.003735 0.004855 0.005580 0.006259 0.007760

> acf(tausim)
```



```
> sigmasim = 1/sqrt(tausim) # Convert to standard deviations -- more familiar
> summary(sigmasim)

Iterations = 20001:20500
Thinning interval = 1
Number of chains = 1
Sample size per chain = 500

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

          Mean             SD       Naive SE Time-series SE
      13.52374        1.28035         0.05726        0.07964

2. Quantiles for each variable:

 2.5%   25%   50%   75% 97.5%
11.35 12.64 13.39 14.35 16.36

> acf(sigmasim)
```

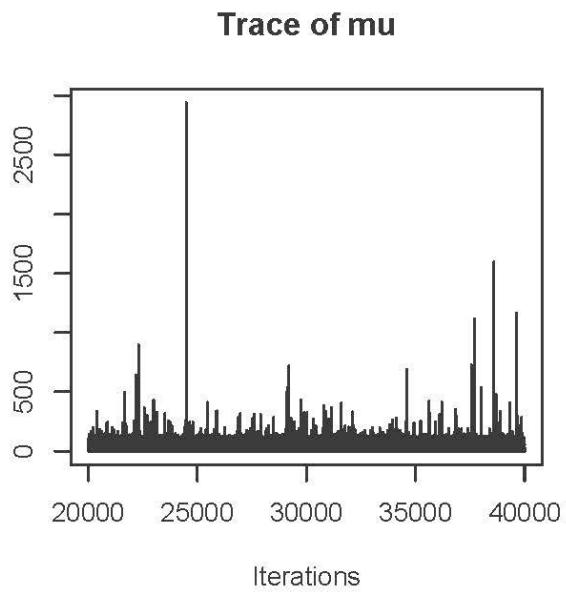


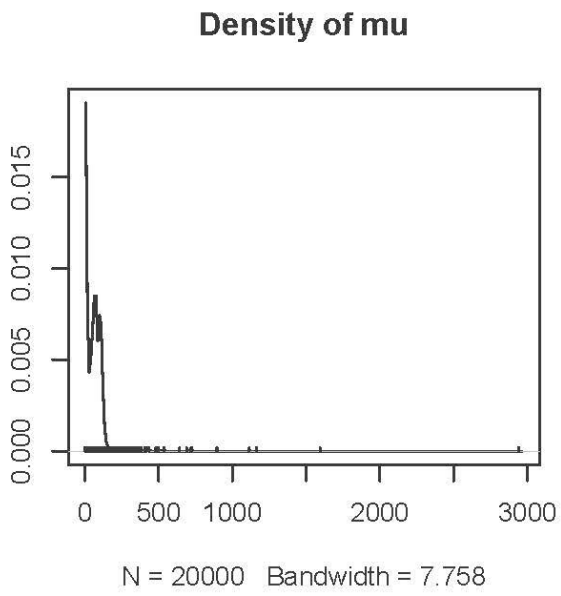
```
> # plot(musim,sigmasim) # Error
> plot(as.numeric(musim),as.numeric(sigmasim))
```

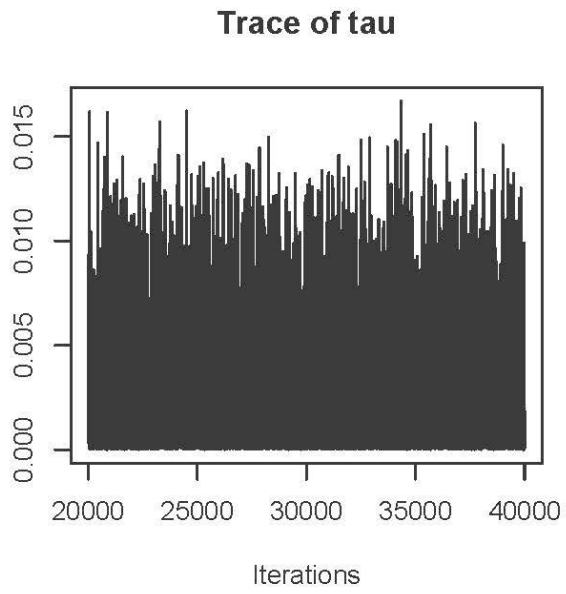


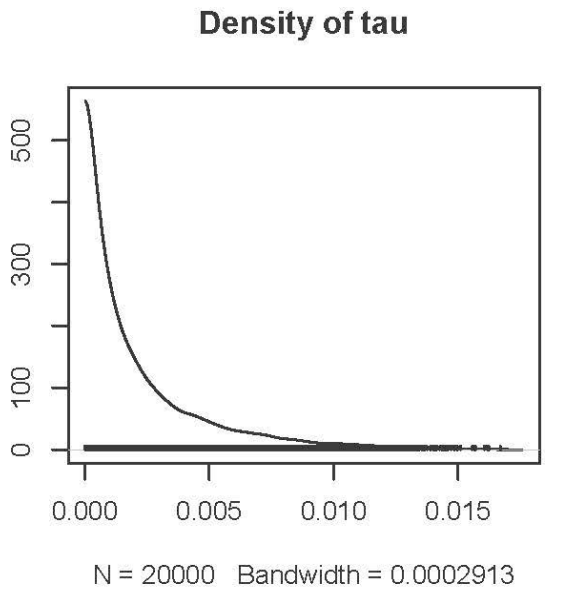

```
> cor(musim,sigmasim)
[1] -0.07504498
```

```
> # Experiment and try to break something
> rm(list=ls())
> set.seed(9999)
> y <- rnorm(n = 500, mean = 100, sd = 15) # Much bigger sample size this time
> c(mean(y),sqrt(var(y)))
[1] 99.55504 15.03604
> library(rjags)
>
> # The model specification
> model_string <- "model{
+   for(i in 1:length(Y)) {
+     Y[i] ~ dnorm(mu, tau)
+   }
+   mu ~ dexp(1) # Standard exponential (Just guessing at the notation)
+   tau ~ dunif(0, 1) # Uniform prior
+ }"
>
> # Running the model
> model <- jags.model(textConnection(model_string), data = list(Y = y), n.chains =
1, n.adapt= 10000)
  |++++++++++++++++++++++++++++++++++++++++++++++++++| 100%
> update(model, 10000); # Burnin for 10000 samples
  |**************************************************| 100%
> normal_samples <- coda.samples(model, variable.names=c("mu", "tau"),
n.iter=20000)
  |**************************************************| 100%
> summary(normal_samples)

Iterations = 20001:40000
Thinning interval = 1
Number of chains = 1
Sample size per chain = 20000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

         Mean        SD  Naive SE Time-series SE
mu  56.288513 53.043303 3.751e-01      0.2471644
tau  0.002273  0.002605 1.842e-05      0.0000232

2. Quantiles for each variable:

         2.5%       25%       50%       75%     97.5%
mu  2.575e-03 4.0041199 59.838811 91.208123 126.90597
tau 5.202e-05 0.0004476  0.001287  0.003117   0.00986

> plot(normal_samples)
```

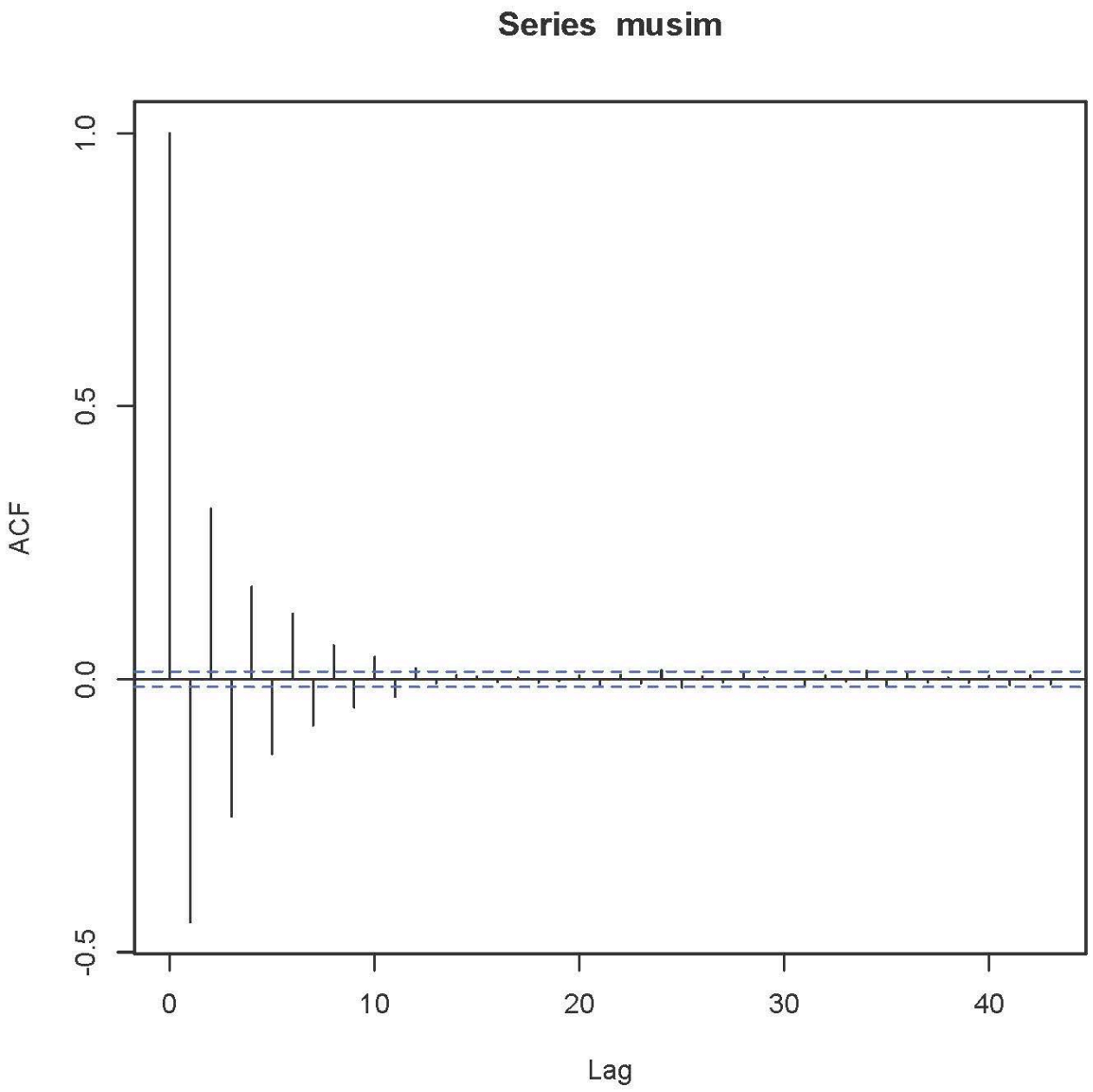









```
> musim = normal_samples[[1]][,1] # First column
> max(musim)
[1] 2940.103
> sort(musim)[19995:20000]
[1]  726.1195  895.2462 1113.2263 1162.9859 1595.2501 2940.1031



> acf(musim)
```


Series musim
ACF
1.0
0.5
0.0
-0.5
0
10
20
30
40
Lag

```
> tausim = normal_samples[[1]][,2] # Second column
> sigmasim = 1/sqrt(tausim) # Convert to standard deviations -- more familiar
> summary(sigmasim)

Iterations = 20001:40000
Thinning interval = 1
Number of chains = 1
Sample size per chain = 20000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

          Mean             SD       Naive SE Time-series SE
       41.1316        49.5001         0.3500         0.4489

2. Quantiles for each variable:

  2.5%    25%    50%    75%  97.5%
 10.07  17.91  27.87  47.27 138.65

> acf(sigmasim)
```