Gibbs Example1: Posterior for parameters of the Normal distribution

Patrick Graham

3rd October 2021

Here we provide an illustration of the mechanics of the Gibbs sampler in simple setting: Simulating the posterior mean and precision (inverse variance) of a simple Normal model.

We will work with simulated data from a $\text{Normal}(10, 5)$ (mean 10 and precision 5).

```
mu <- 10
tau <- 0.1
Y <- rnorm(n=100,mean=mu,sd=1/sqrt(tau))
summary(Y)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   2.410   7.707   9.684   9.987  12.303  17.412
```

```
##get sample mean and sample size
Ybar <- mean(Y)
n <- length(Y)
```

We will treat Y as the observed data, assumed to be sampled from a Normal distribution with unknown mean and precision. We adopt the conditionally conjugate priors:

$$\begin{aligned}\mu &\sim \text{Normal}(m_{prior}, \kappa_{prior}) \\ \tau &\sim \text{Gamma}(a, b)\end{aligned}$$

where we assume $p(\mu, \tau) = p(\mu)p(\tau)$.

The full conditionals are:

$$\begin{aligned}[\mu|\tau, \mathbf{Y}] &\sim \text{Normal}\left(\frac{\tau n\overline{Y} + \kappa_{prior} m_{prior}}{\tau n + \kappa_{prior}}, \tau n + \kappa_{prior}\right) \\ [\tau|\mu, \mathbf{Y}] &\sim \text{Gamma}\left(a + n/2, b + 0.5\sum_{i=1}^{i=n}(Y_i - \mu)^2\right)\end{aligned}$$

where n is the sample size and $\overline{Y} = \frac{1}{n}\sum_{i=1}^{i=n} Y_i$.

Set priors

```
#its made up data anyway so # we'll just use conventional #uninformative priors
m_prior <- 0      #prior mean for mu
kappa_prior <- 0.001 #prior precision for mu

a <- 0.01  #prior shape parameter for tau
b <- 0.01  #second parameter of Gamma prior for tau (R calls this "rate")

##set-up for Gibbs sampler

nchains <- 5
nsim <- 2000

burnin <- 1000

##set-up objects for storing simulated values of mu and tau
store_mu <- matrix(NA, nrow=nsim,ncol=nchains)
store_tau <- matrix(NA,nrow=nsim,ncol=nchains)
```

We now have everything need we need to set the Gibb sampler running. As usual we run multiple chains, so the code consists of an outer loop over chains and an inner loop over iterations.

```
for (j in 1:nchains) {
##draw staring value for mu. Usually it would be good practice to
##find a reasonable approximation and draw staring values from an       #over-dispers
ed version of this. However in this simple problem any reasonable approximation would
virtually solve the problem. So instead
#I will just draw from modified version of the prior; modified to lessen  the probabi
lityof obtaining extreme starting points.

  mu <- rnorm(n=1,mean=m_prior,sd=1/sqrt(0.01))

  for (i in 1:nsim) {

    ###update tau
    a_post <- a + n/2
    b_post <- b + 0.5*sum((Y-mu)^2)
    tau <- rgamma(n=1,shape=a_post,rate=b_post)

  ##update mu
    m_post <- (tau*n*Ybar + kappa_prior*m_prior) / (tau*n + kappa_prior)
    kappa_post <- tau*n + kappa_prior
    mu <- rnorm(n=1,mean=m_post,sd=1/sqrt(kappa_post))

  ##store current values of mu and tau
    store_mu[i,j] <- mu
    store_tau[i,j] <- tau


  }

}
```

Discard burn-in samples:

```
post_mu <- store_mu[(burnin+1):nsim,]
post_tau <- store_tau[(burnin+1):nsim,]
```

Posterior convergence checking and inference for μ.

```
##Chunk each  posterior  chains into  two pieces
## and reassemble as a matrix

possize <- nrow(post_mu) #actual posterior sample size

n1 <- round(possize/2)  ##size of first chunk

chunk1_mu <- post_mu[1:n1,]
chunk2_mu <- post_mu[(round(possize/2)+1):possize,]

postmu_chunked <- cbind(chunk1_mu,chunk2_mu)

str(postmu_chunked)
```

```
##  num [1:500, 1:10] 9.68 10.19 9.87 10.28 10.23 ...
```

```
chain_mean_mu <- colMeans(postmu_chunked)
#obtain the within chain variance
chain_sd_mu <- apply(postmu_chunked,MARGIN=2,FUN=sd)
chain_var_mu <- chain_sd_mu^2
W_mu = mean(chain_var_mu)


##get between chain variance

possize_chunked <- nrow(postmu_chunked)

B_mu <- possize_chunked*(sd(chain_mean_mu))^2

##Build Vplus
Vplus_mu <- ( ((possize_chunked-1)/possize_chunked) * W_mu + (1/possize_chunked)*B_mu
)

Rhat_mu <- sqrt(Vplus_mu/W_mu)

Rhat_mu
```

```
## [1] 0.9997532
```

```
##compute  estimated effective sample size, using Coda

postmu.mcmc <- coda::as.mcmc(postmu_chunked )

codaNeff_mu_chains <- coda::effectiveSize(postmu.mcmc)

codaNeff_mu_chains
```

```
##     var1     var2     var3     var4     var5     var6     var7     var8
## 500.0000 736.5340 500.0000 500.0000 463.5703 389.9392 500.0000 969.4263
##     var9    var10
## 500.0000 500.0000
```

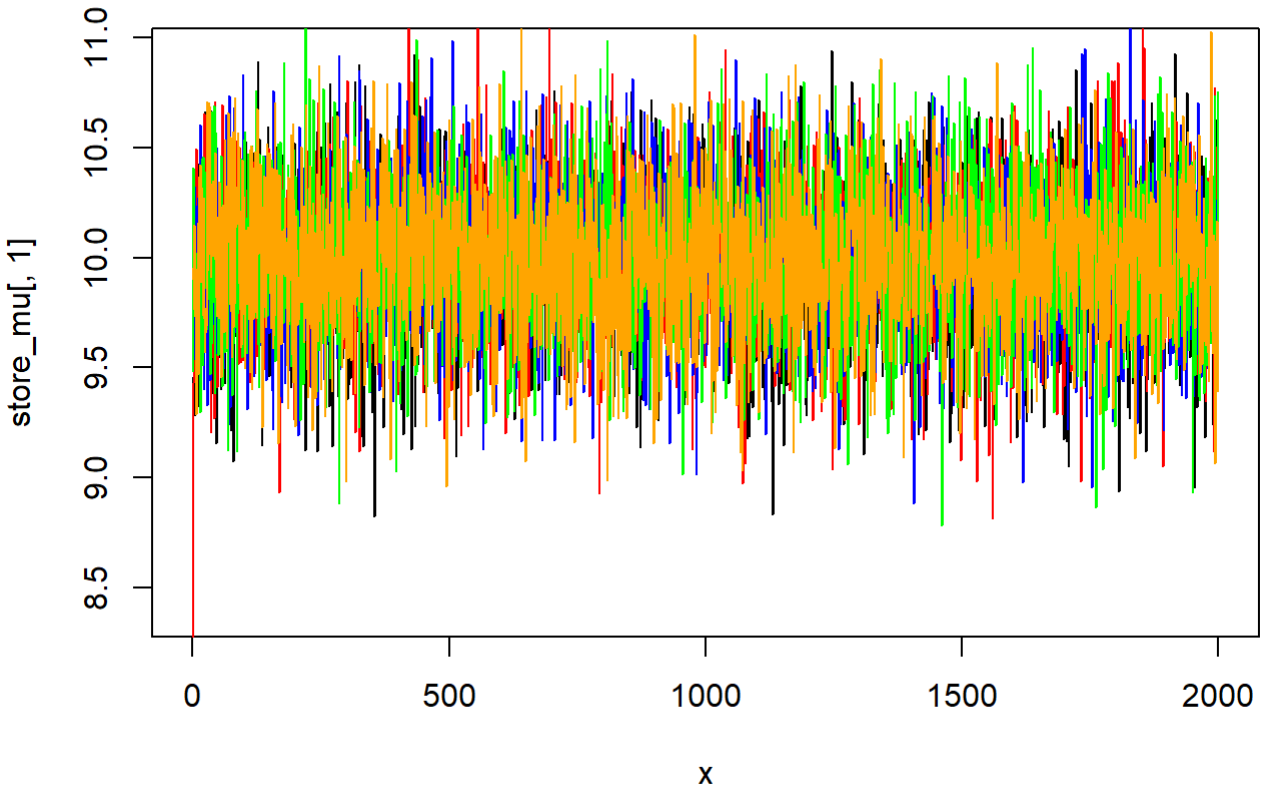
```
codaNeff_mu <-  sum(codaNeff_mu_chains)
codaNeff_mu
```

```
## [1] 5559.47
```

Note that $\hat{R}$ is slightly less than one. This occurs when the between variance is very small, so $Vplus$ is just less then W. This reflects the approximate nature of the between and within variance calculations. The effective sample size being slightly greater than the nominal Monte Carlo sample size is reflective of the same phenomenon. All the indications are that there is no problem with convergence for μ and the posterior samples do not exhibit correlation.

For completeness let's take a look at traceplots.

```
###traceplots - all iterations
x <- seq(from=1,to=nsim,by=1)
plot(x,store_mu[,1],type="l")
lines(x,store_mu[,2],col="red")
lines(x,store_mu[,3],col="blue")
lines(x,store_mu[,4],col="green")
lines(x,store_mu[,5],col="orange")
```



It looks like convergence is virtually immediate - which is not too surprising, given, the toy nature of the problem.

Posterior convergence checking and inference for τ

```
##Chunk each  posterior  chains into  two pieces
## and reassemble as a matrix

possize <- nrow(post_tau) #actual posterior sample size

n1 <- round(possize/2)  ##size of first chunk

chunk1_tau <- post_tau[1:n1,]
chunk2_tau <- post_tau[(round(possize/2)+1):possize,]

posttau_chunked <- cbind(chunk1_tau,chunk2_tau)

str(posttau_chunked)
```

```
##  num [1:500, 1:10] 0.1191 0.115 0.0918 0.0941 0.116 ...
```

```
chain_mean_tau <- colMeans(posttau_chunked)
#obtain the within chain variance
chain_sd_tau <- apply(posttau_chunked,MARGIN=2,FUN=sd)
chain_var_tau <- chain_sd_tau^2
W_tau = mean(chain_var_tau)


##get between chain variance

possize_chunked <- nrow(posttau_chunked)

B_tau <- possize_chunked*(sd(chain_mean_tau))^2

##Build Vplus
Vplus_tau <- ( ((possize_chunked-1)/possize_chunked) * W_tau + (1/possize_chunked)*B_
tau )

Rhat_tau <- sqrt(Vplus_tau/W_tau)

Rhat_tau
```

```
## [1] 0.9996033
```

```
##compute  estimated effective sample size, using Coda

posttau.mcmc <- coda::as.mcmc(posttau_chunked )

codaNeff_tau_chains <- coda::effectiveSize(posttau.mcmc)

codaNeff_tau_chains
```

```
##     var1     var2     var3     var4     var5     var6     var7     var8
## 500.0000 500.0000 500.0000 500.0000 500.0000 612.9011 500.0000 500.0000
##     var9    var10
## 500.0000 500.0000
```

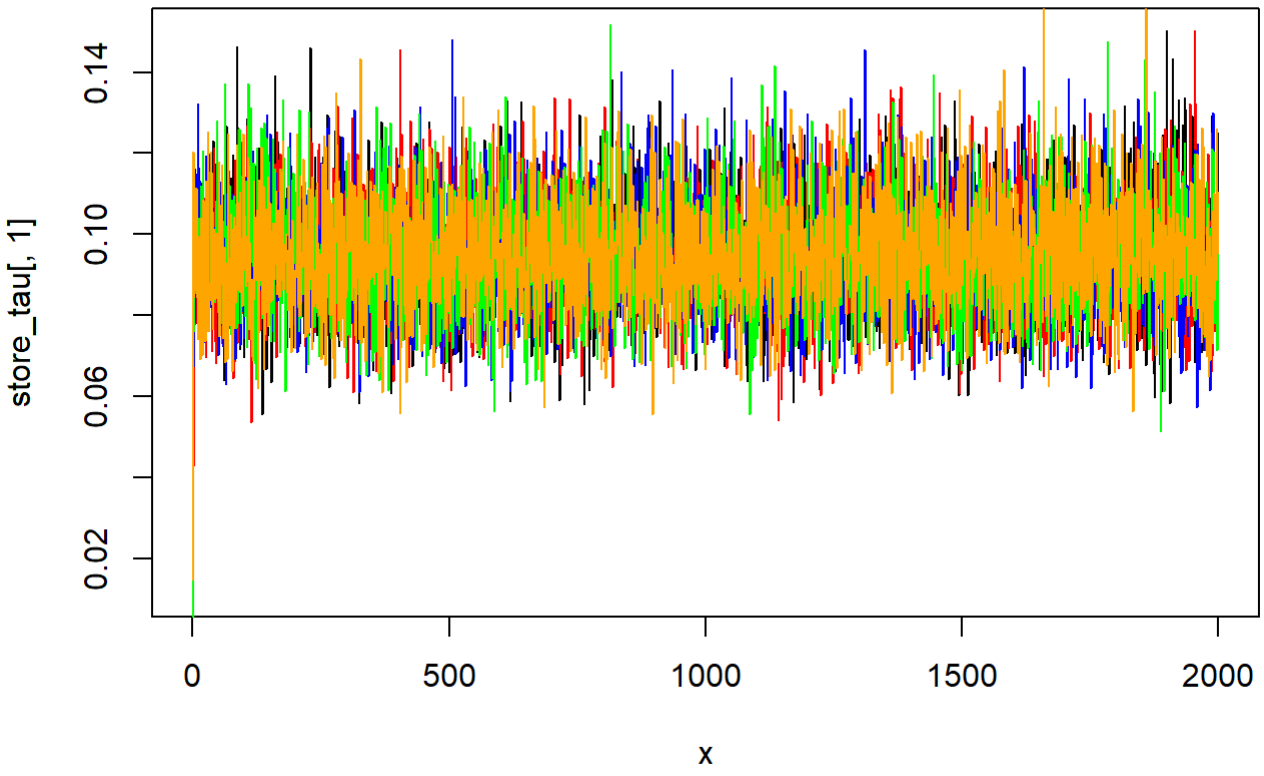
```
codaNeff_tau <-  sum(codaNeff_tau_chains)
codaNeff_tau
```

```
## [1] 5112.901
```

Again $\hat{R}$ is slightly less than 1, and Neff is slightly greater than the nominal Monte Carlo sample size.

Look at the traceplots:

```
###traceplots - all iterations
x <- seq(from=1,to=nsim,by=1)
plot(x,store_tau[,1],type="l")
lines(x,store_tau[,2],col="red")
lines(x,store_tau[,3],col="blue")
lines(x,store_tau[,4],col="green")
lines(x,store_tau[,5],col="orange")
```



Obtain posterior summaries

```
probs <- c(0.025,0.5,0.975)

quantile(post_mu,probs)
```

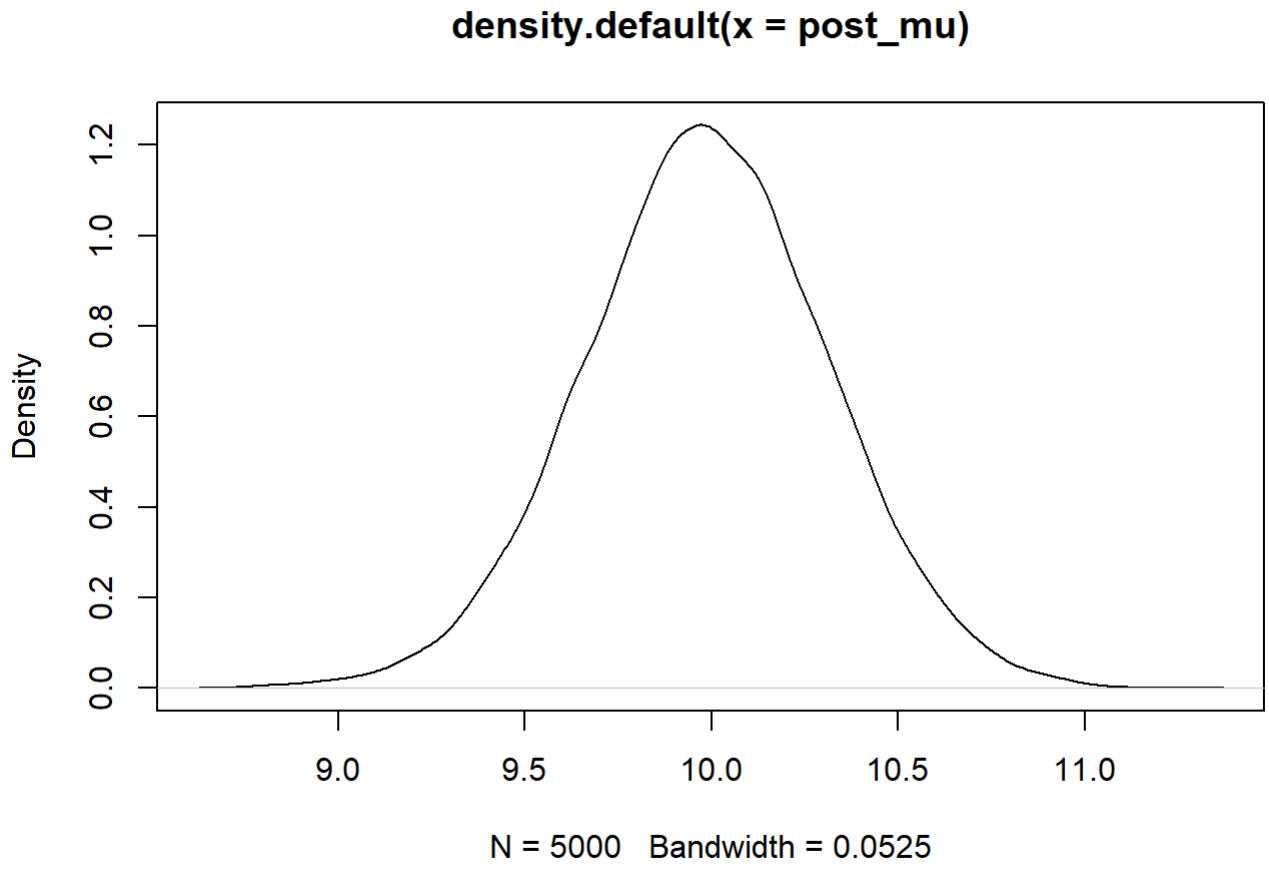
```
##      2.5%       50%     97.5%
##  9.342744  9.986458 10.625605
```

```
quantile(post_tau,probs)
```

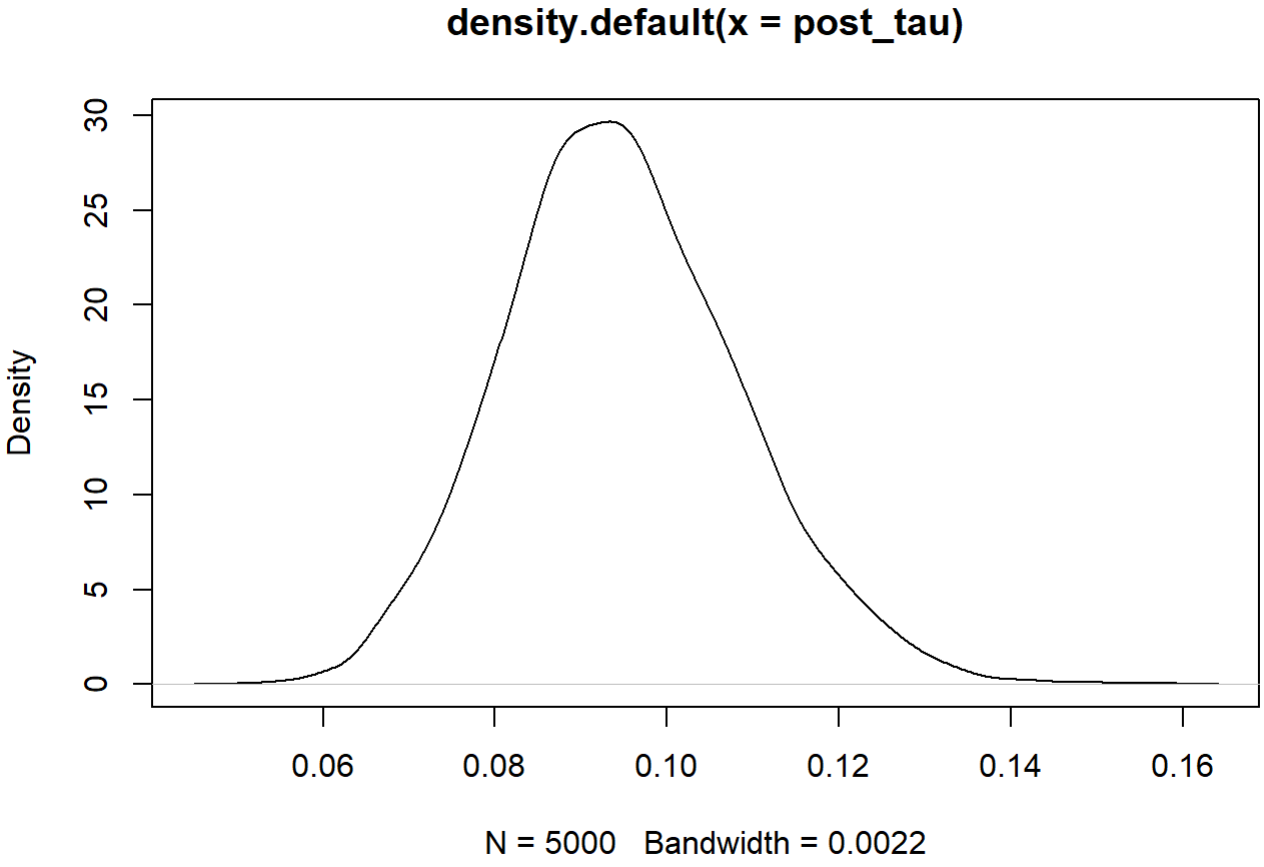
```
##       2.5%        50%      97.5%
## 0.06958775 0.09409444 0.12404772
```

Posterior density plots

```
plot(density(post_mu))
```




```
plot(density(post_tau))
```


density.default(x = post_tau)
Density
0
5
10
15
20
25
30
0.06
0.08
0.10
0.12
0.14
0.16
N = 5000 Bandwidth = 0.0022