portfolio_9

2023-05-02

Metropolis-within-Gibbs algorithm

First we will load in and format our data:

```
library(mlbench)
data(PimaIndiansDiabetes)
PID <- PimaIndiansDiabetes

PID["diabetes"] <- ifelse(PID[["diabetes"]] == "pos", 1, 0)
head(PID)
```

```
##   pregnant glucose pressure triceps insulin mass pedigree age diabetes
## 1        6     148       72      35       0 33.6    0.627  50        1
## 2        1      85       66      29       0 26.6    0.351  31        0
## 3        8     183       64       0       0 23.3    0.672  32        1
## 4        1      89       66      23      94 28.1    0.167  21        0
## 5        0     137       40      35     168 43.1    2.288  33        1
## 6        5     116       74       0       0 25.6    0.201  30        0
```

```
X <- as.matrix(PID[, -9])
y <- PID[, 9]
```

Let's now define the prior distribution:

```
prior_mu <- rep(0, 9)    # prior mean
prior_sigma <- diag(9)    # prior covariance
```

Let's now run our Metropolis-within-Gibbs algorithm! The proposal distributions that will be used by the p+1 Metropolis-Hastings kernels are univariate normal distributions, for the i-th parameter its distribution is $N(\mu_i, \Sigma_{i,i})$ where $\mu = \text{argmax}_{(\alpha,\beta)\in\mathbb{R}^{p+1}} \log \pi(\alpha, \beta)|y^0)$ and $\Sigma = -(\mathbf{H}_n(\mu_q))^{-1}$ where $\mathbf{H}_n(\theta) = (\frac{\partial^2}{\partial\theta_j \partial\theta_l} \log \pi(\theta|y^0))_{j,l=1}^{p+1}$, ie. like the proposal distribution from the previous portfolio.

```
library(MASS)
library(mvtnorm)
library(numDeriv)

# The log-likelihood function
log_likelihood <- function(params, X, y) {
  z <- params[1] + X %*% params[2:9]
  sum(y*z - log(1 + exp(z)))
}

# For calculating the proposal ratio
proposal_ratio <- function(current, proposal, proposal_cov){
    dmvnorm(current, mean = proposal, sigma = proposal_cov, log = TRUE) - dmvnorm(proposal, mean = curr
}
```

```r
# We set the number of iterations and burn-in period
num_iterations <- 10000
burn <- 1000

# We set up the proposal distribution for each parameter
#proposal_mu <- rep(0.1, 9)
#proposal_sigma <- diag(9)
n_log_posterior <- function(params) {
  z <- params[1] + X %*% params[2:9]
  ll <- sum(y*z - log(1 + exp(z)))
  log_prior <- dmvnorm(params, mean = prior_mu, sigma =prior_sigma, log = TRUE)
  log_post <- ll + log_prior
  return(-log_post)
}
# The log posterior distribution
log_posterior <- function(params) {
  z <- params[1] + X %*% params[2:9]
  ll <- sum(y*z - log(1 + exp(z)))
  log_prior <- dmvnorm(params, mean = prior_mu, sigma =prior_sigma, log = TRUE)
  log_post <- ll + log_prior
  return(log_post)
}
proposal_mu <- optim(rep(0.1,9), n_log_posterior)$par
proposal_sigma <- -solve(hessian(func=log_posterior, x=proposal_mu))

# Initial parameters
params <- rep(0, 9)
chain <- matrix(0, nrow=num_iterations, ncol=9)
chain[1,] <- params
accept <- rep(0, 9)

# The Metropolis-within-Gibbs algorithm
for (i in 2:num_iterations) {
  for (j in 1:9) {
    # We sample a proposal for the j-th parameter from its conditional distribution
    proposal <- rnorm(1, proposal_mu[j], sqrt(proposal_sigma[j,j]))
    new_params <- params
    new_params[j] <- proposal

    # We compute the acceptance ratio
     log_ratio <- log_likelihood(new_params, X, y) - log_likelihood(params, X, y) + dmvnorm(new_params,

    # We accept or reject the proposal
    if (log(runif(1)) < exp(log_ratio)) {
      params <- new_params
      accept[j] <- accept[j] + 1
    }
  }

  # We store current parameter value in the chain
  chain[i,] <- params
}
```

```r
# We discard the burn in phase
chain <- chain[-c(1:burn),]

# We print the acceptance rate
cat("Acceptance rate for each parameter:", accept / num_iterations, "\n")
```

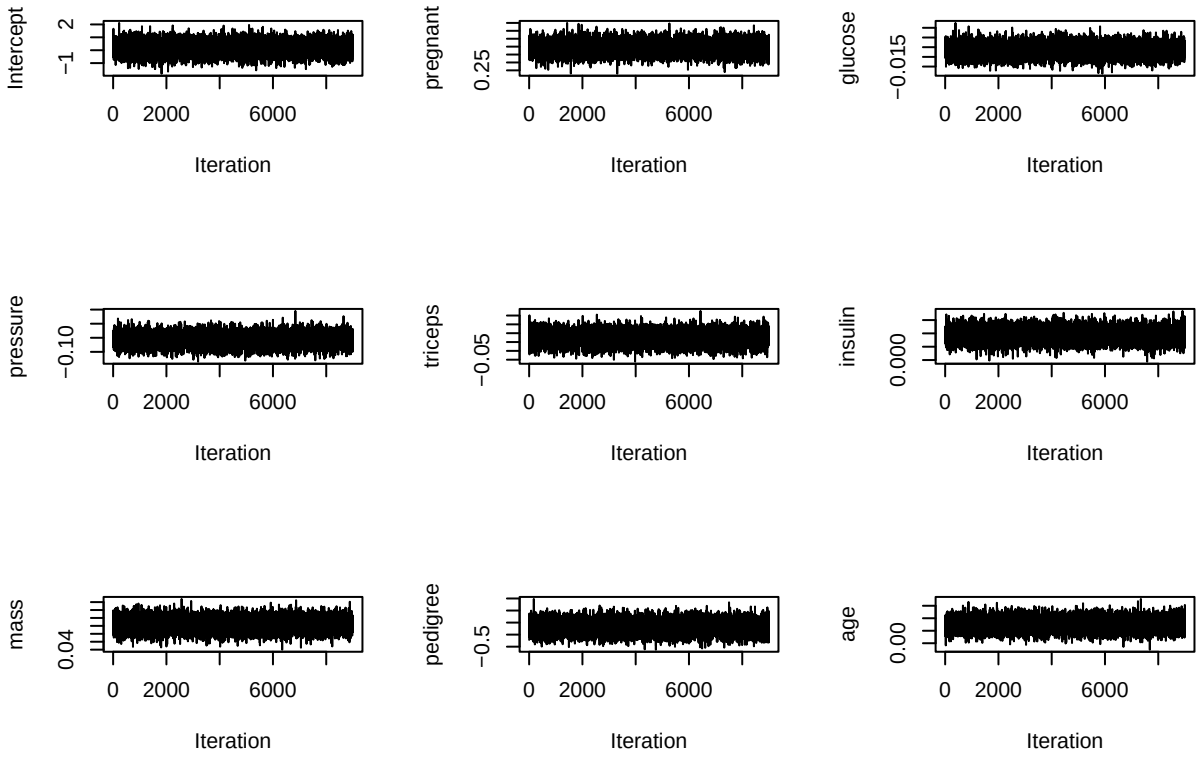
```
## Acceptance rate for each parameter: 0.9999 0.9999 0.9999 0.9999 0.9999 0.9999 0.9999 0.9999 0.9999
```

```r
# We print some summary statistics
post_mean <- apply(chain, 2, mean)
post_sd <- apply(chain, 2, sd)
post_quantiles <- apply(chain, 2, quantile, c(0.025, 0.5, 0.975))
post_summary <- rbind(post_mean, post_sd, post_quantiles)
colnames(post_summary) <- c( "Intercept" , colnames(PID[, -9]) )
rownames(post_summary) <- c("Mean", "SD", "2.5%", "50%", "97.5%")
print(post_summary)
```

```
##        Intercept   pregnant       glucose     pressure      triceps
## Mean   0.1876870 0.39355265 -6.401770e-03 -0.083104582 -0.025509408
## SD     0.5105566 0.04005604  3.268242e-03  0.008860402  0.007397766
## 2.5%  -0.8177064 0.31595026 -1.293050e-02 -0.100130373 -0.040171006
## 50%    0.1855370 0.39371016 -6.345336e-03 -0.083177115 -0.025522797
## 97.5%  1.1742165 0.47315048 -6.281979e-05 -0.065716146 -0.010973541
##            insulin       mass   pedigree         age
## Mean  0.0037861173 0.10389051  0.3156065 0.030450651
## SD    0.0009995941 0.01653100  0.2739305 0.009932581
## 2.5%  0.0018569473 0.07171058 -0.2203042 0.010924932
## 50%   0.0037888924 0.10399399  0.3140483 0.030381728
## 97.5% 0.0057383680 0.13574693  0.8544595 0.050016395
```

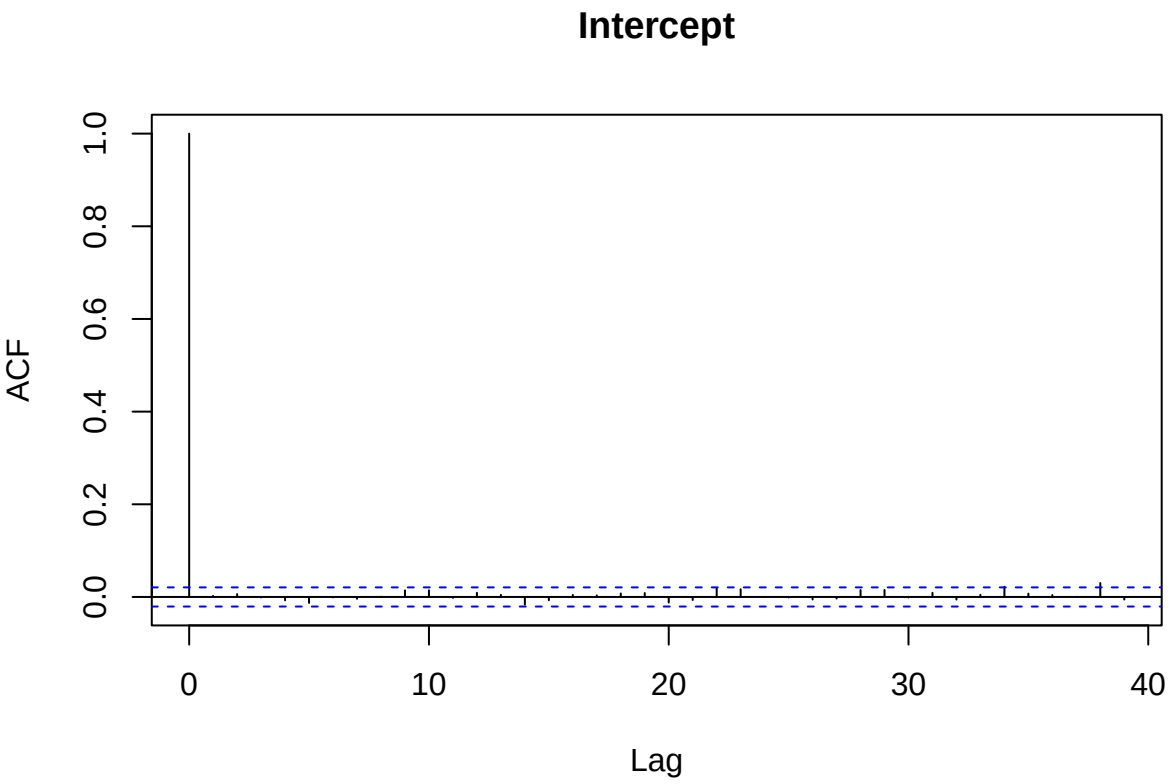
Above we can see some summary statistics on chain for the parameters aswell as the acceptance rate. We note the sd of the chains for the parameters are quite small. We also note that the acceptance rate is extremely high which could mean that our proposal distribution is well tuned. We will now investigate further whether our MCMC chain has properly converged. Let's now produce some trace plots of the parameters:

```r
par(mfrow=c(3,3))
plot(chain[,1], type="l", xlab="Iteration", ylab="Intercept")
plot(chain[,2], type="l", xlab="Iteration", ylab="pregnant")
plot(chain[,3], type="l", xlab="Iteration", ylab="glucose")
plot(chain[,4], type="l", xlab="Iteration", ylab="pressure")
plot(chain[,5], type="l", xlab="Iteration", ylab="triceps")
plot(chain[,6], type="l", xlab="Iteration", ylab="insulin")
plot(chain[,7], type="l", xlab="Iteration", ylab="mass")
plot(chain[,8], type="l", xlab="Iteration", ylab="pedigree")
plot(chain[,9], type="l", xlab="Iteration", ylab="age")
```

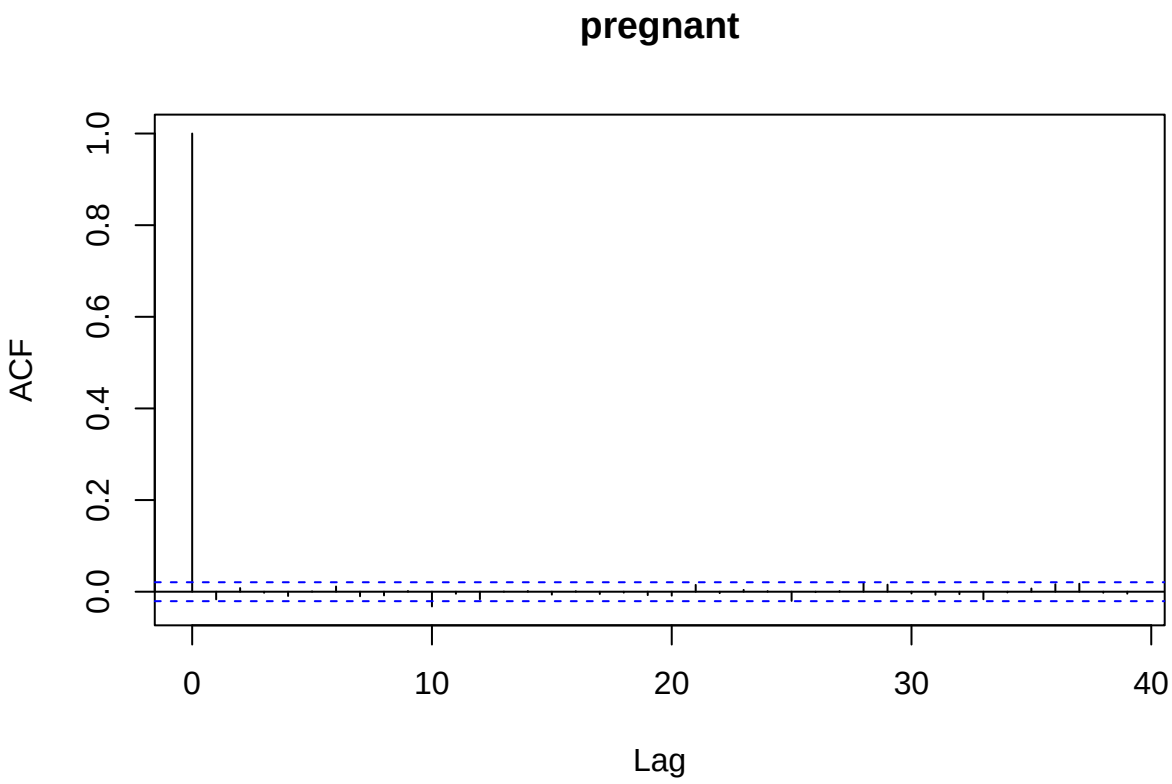


These trace plots look like what we would expect from a converged chain as they appear stationary, look very random and there are no obvious trends in trace. Let's now look at the acf plots for all the parameters:

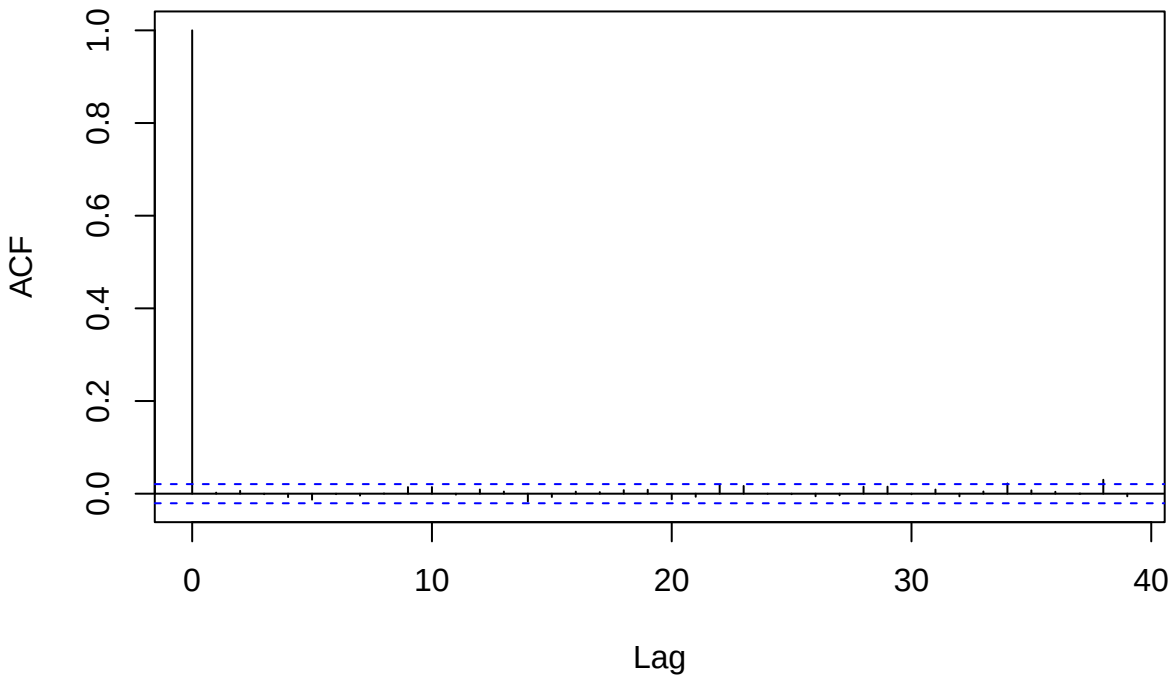
```
acf(chain[,1], main="Intercept")
```



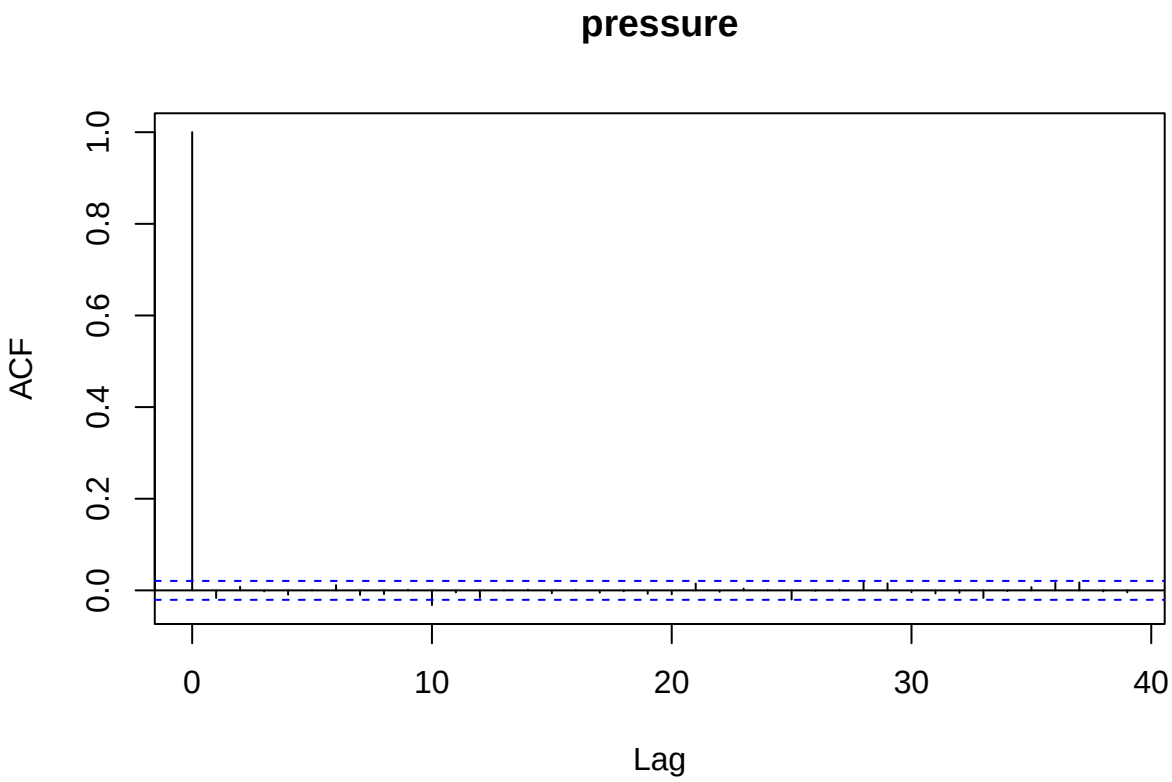
```
acf(chain[,2], main="pregnant")
```



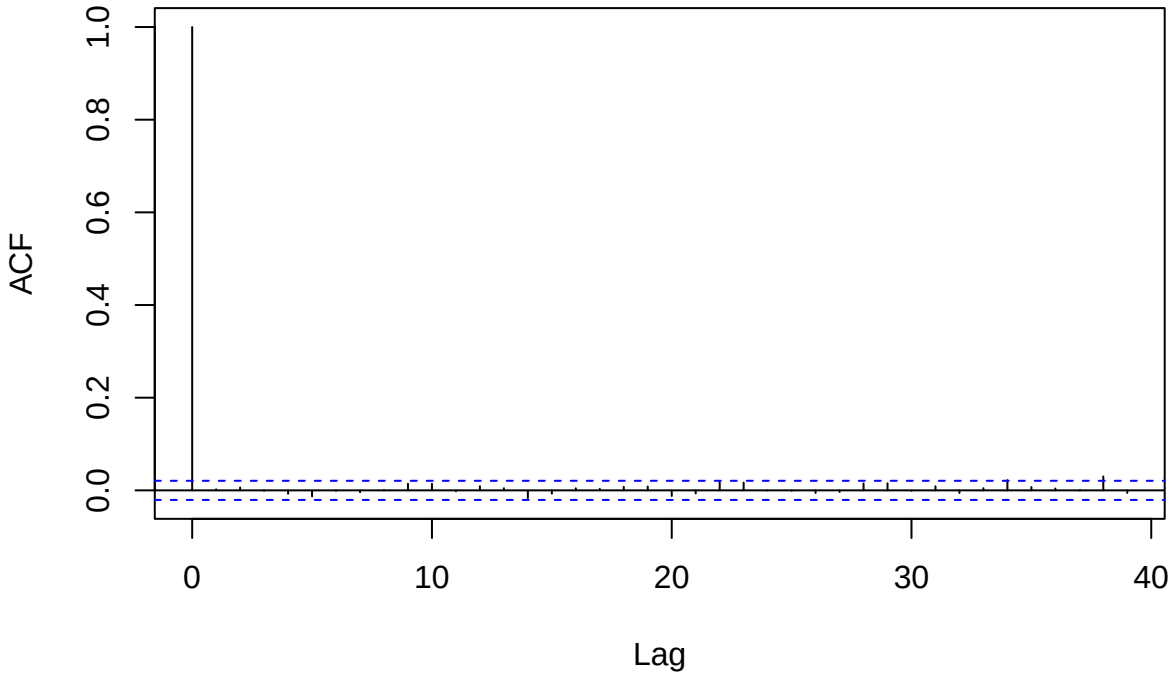

```
acf(chain[,1], main="glucose")
```



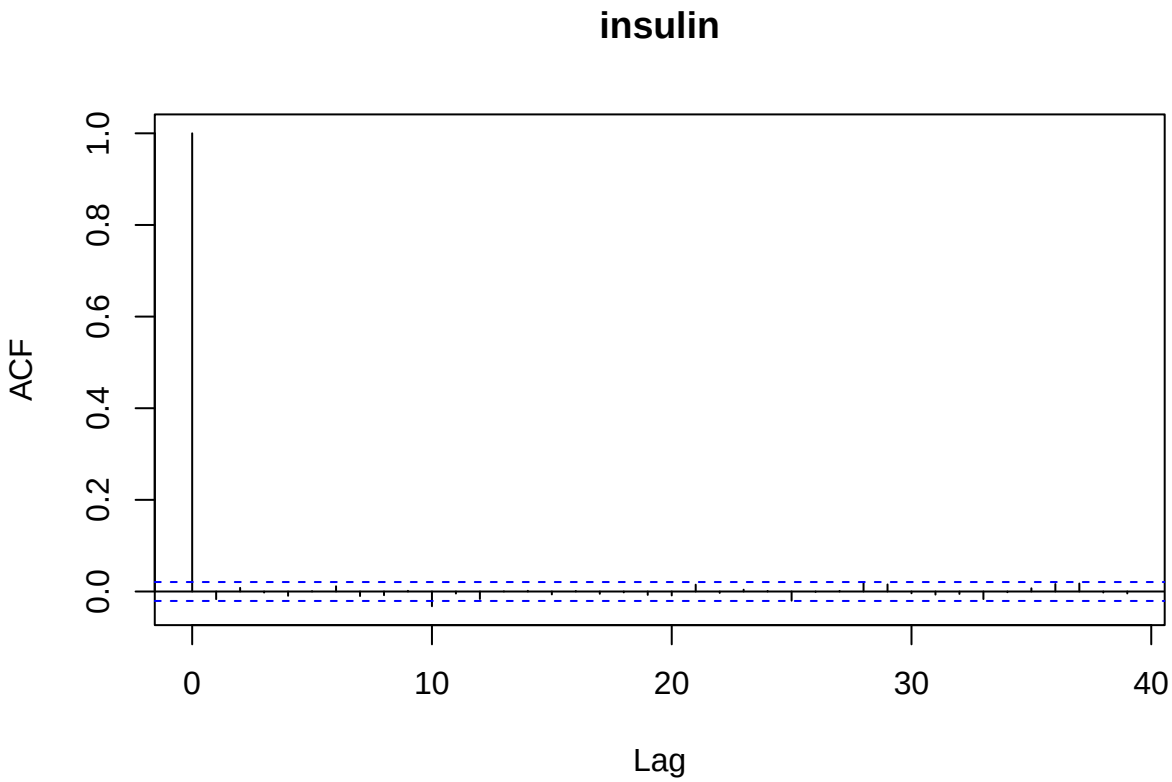
```
acf(chain[,2], main="pressure")
```



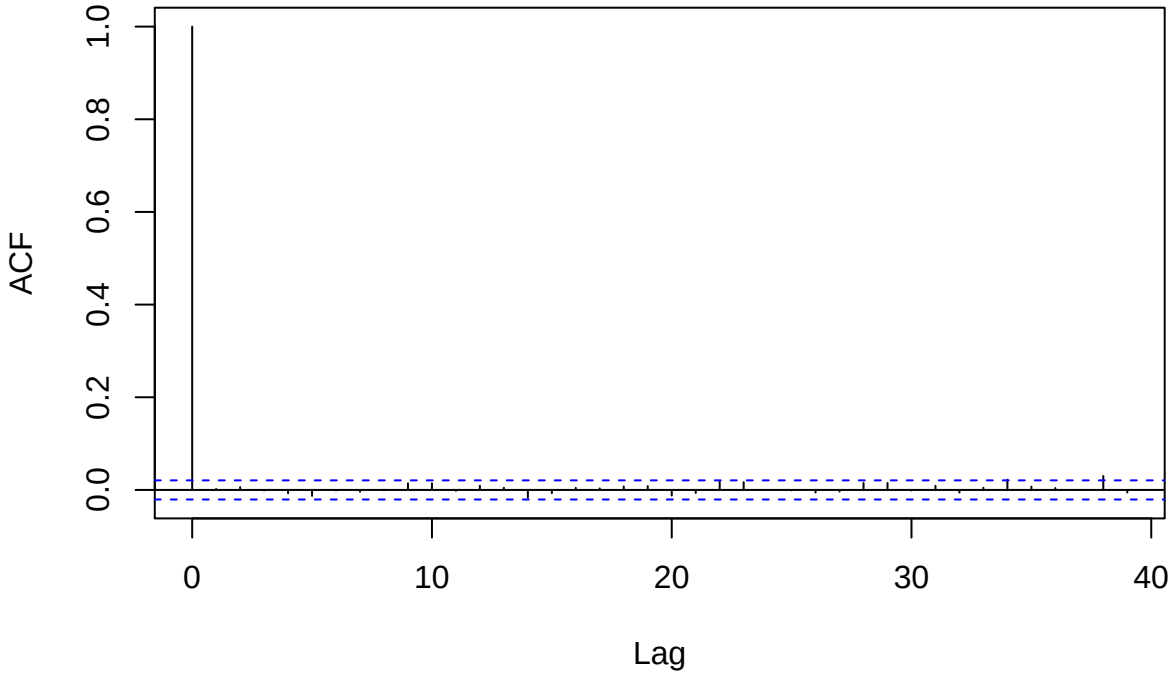

```
acf(chain[,1], main="triceps")
```



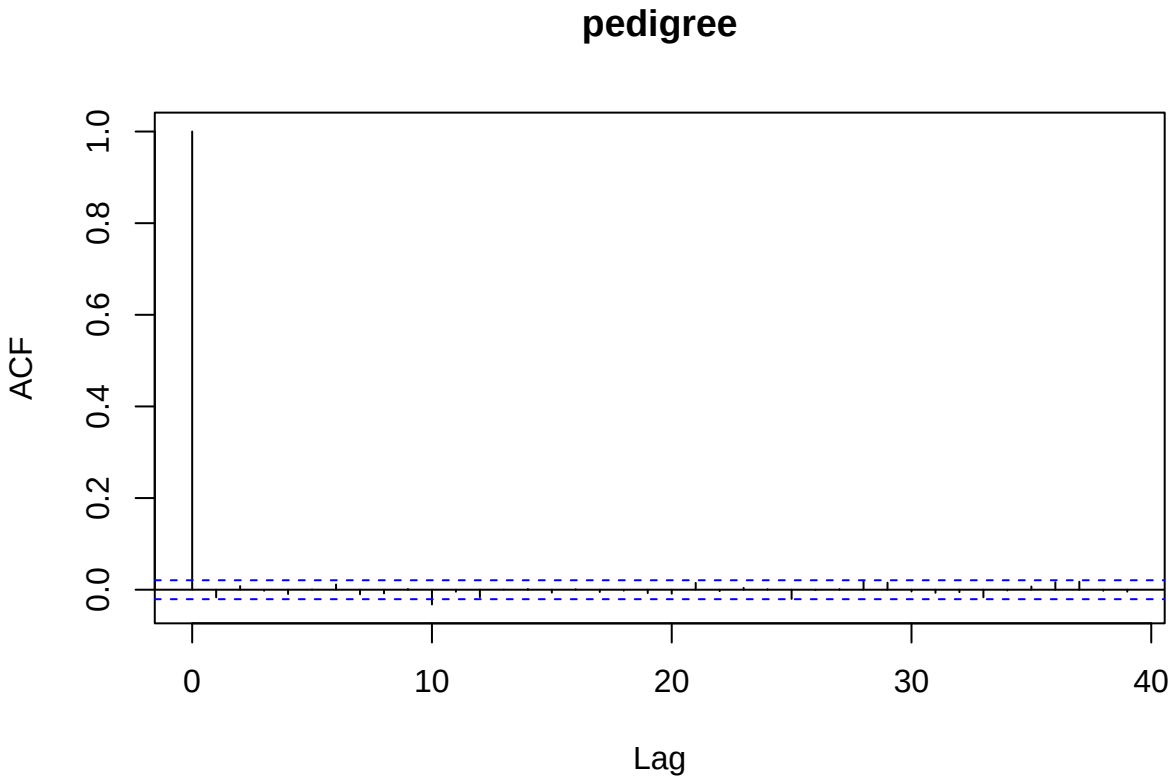
```
acf(chain[,2], main="insulin")
```



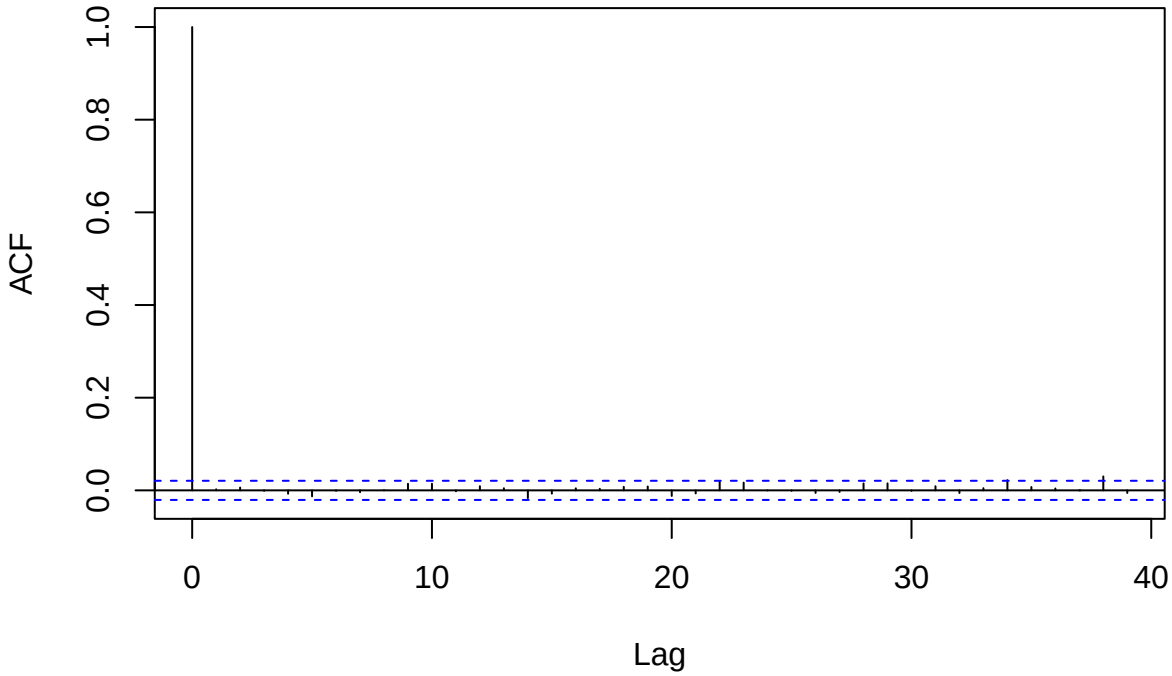

```
acf(chain[,1], main="mass")
```



```
acf(chain[,2], main="pedigree")
```



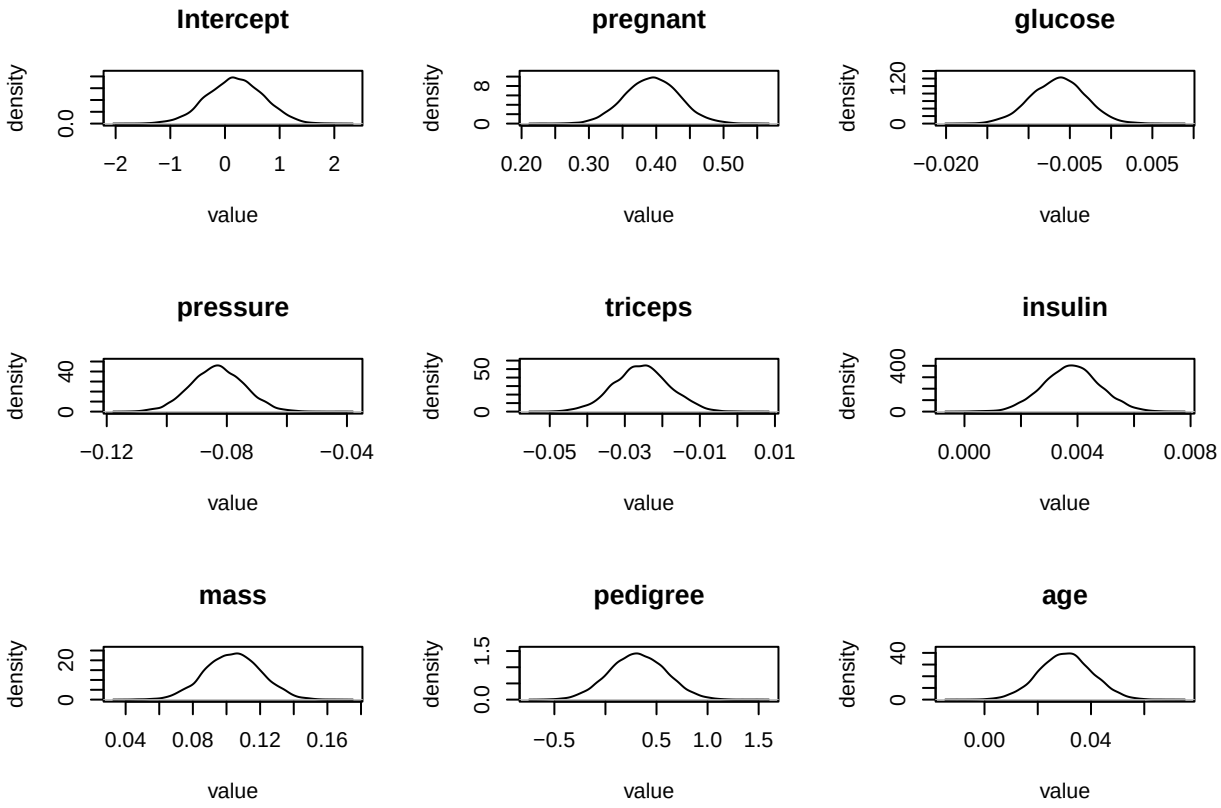

```
acf(chain[,1], main="age")
```



The acf plots also look like what we would expect from a converged chain with the ACF dropping off instantly

for all parameters, ie. the chain shows very low autocorrelation. Finally, let's plot the estimated marginal posterior distribution for each of the parameters:

```
names <- c( "Intercept" , colnames(PID[, -9]) )
# Plot the estimated marginal posterior distribution for each parameter
par(mfrow=c(3,3))
for (i in 1:9) {
  density <- density(chain[, i])
  plot(density, main=names[i], xlab="value", ylab="density", ylim=c(0, max(density$y)*1.1))
}
```



All the parameters have marginal posterior distributions that are very Gaussian and don't have a large standard deviation (ie. are not very spread out), indicating that the proposal distribution is appropriately centered around the true parameter values.

The results we got using Metropolis-within-Gibbs are very similar to those we got with Metropolis-Hastings, therefore I would recommend using the Metroplis-Hastings due to it having smaller compute (you don't have to iterate over each parameter in Metroplis-Hastings). Metropolis-within-Gibbs is more useful in situations where we cant sample from any of the full conditional distributions, in this case finding good lower dimensional proposal distributions in Metropolis-within-Gibbs may be easier than trying to find a good multivariate proposal distribution that would work in Metropolis-Hastings.