Radiata Pine Example (Lecture 1)

Leah South

Our running example is a simple linear regression model about strength of radiata pine trees.

Below are the R packages we'll be using in this document.

```
library(MASS) # multivariate normal
library(coda) # assessing convergence and sample quality
library(psych) # bivariate plots
```

The MCMC Algorithm

We'll be using RW-MH MCMC for this example. Our focus is on post-processing so we won't focus on this algorithm. You can use an R package for this part if you like.

```
MCMC <- function(theta0,sigma_rw,its,varNames){
  # Setting up storage
  theta <- matrix(NaN,nrow=its+1,ncol=length(theta0))
  loglikes <- rep(NaN,its+1)
  logpriors <- rep(NaN,its+1)

  # Initialisation
  theta[1,] <- theta0
  loglikes[1] <- loglike(theta0,radiata)
  logpriors[1] <- logprior(theta0,radiata)
```

```
  for (i in 2:(its+1)){
    # Proposal
    theta_prop <- mvrnorm(1,theta[i-1,],sigma_rw)

    loglike_prop <- loglike(theta_prop,radiata)
    logprior_prop <- logprior(theta_prop,radiata)

    # MH ratio calculation
    mh <- exp(loglike_prop + logprior_prop -
                loglikes[i-1] - logpriors[i-1])

    # Accept/reject step
    if(runif(1)< mh){
      theta[i,] <- theta_prop
      loglikes[i] <- loglike_prop
      logpriors[i] <- logprior_prop
    } else{
      theta[i,] <- theta[i-1,]
      loglikes[i] <- loglikes[i-1]
      logpriors[i] <- logpriors[i-1]
    }
  }

  # Getting the main thing of interest (the samples)
  # into coda's "mcmc" format.
  samples <- as.mcmc(theta)
  varnames(samples) <- varNames

  # We'll also return the log likelihood and log prior
  # calculations which will occasionally be helpful.
  return(list(samples = samples, loglike = loglikes, logprior = logpriors))
}
```

The Statistical Model

The goal is to estimate the posterior for $\theta = (\alpha, \beta, \sigma^2)$ given

$$y^{(i)} = \alpha + \beta(x^{(i)} - \bar{x}) + \epsilon^{(i)} \text{ where } \epsilon^{(i)} \sim \mathcal{N}(0, \sigma^2),$$

y is the maximum compression strength parallel to the grain and x is the density. Our priors are $\alpha \sim \mathcal{N}(3000, 10^6)$, $\beta \sim \mathcal{N}(185, 10^4)$ and $\sigma^2 \sim \text{InvGamma}(3, (2 \times 300^2)^{-1})$.

We will actually sample for $\log \sigma^2$ rather than σ^2 in the MCMC algorithm. This helps us to avoid the possibility of a higher rejection rate due to a bounded parameter. A transformation of variables is included in the log prior function.

The data is in `radiata.rda` and the functions for computing the log likelihood and log prior are below. There's also a function here for simulating draws from the prior. This can be a nice way to initialise an MCMC run.

```
# Loading in the data
load("radiata.rda")

# A function to compute the log likelihood for this example
loglike <- function(part_vals,options){
```

```
  #Getting data from list
  x <- options$x1
  y <- options$y

  #Log likelihood for the inputted data and parameters
  mean_reg <- part_vals[1]+part_vals[2]*(x-mean(x))
  loglike <- sum(dnorm(y,mean_reg,sqrt(exp(part_vals[3])),log = TRUE))
  return(loglike)
}

# A function to compute the log prior density for this example
logprior <- function(part_vals,options){

  #Parameters of inverse gamma (the prior for parameter 3 in this example)
  a <- 3
  b <- (2*300^2)^(-1)

  #Log of the prior
  logprior <- dnorm(part_vals[1],3000,10^3,log = TRUE) +
    dnorm(part_vals[2],185,10^2,log =TRUE) +
    part_vals[3]-1/b/exp(part_vals[3]) -
    log(gamma(a))-a*log(b)-part_vals[3]*(a+1)

  return(logprior)
}

# A function to simulate data from the prior
# (for overdispersed initial starts in MCMC)
simprior <- function(N,options){

  #Drawing N samples from the prior for each parameter
  part_vals <- matrix(rep(0,N*3),nrow=N,ncol=3)
  part_vals[,1] <- rnorm(N,mean=3000,sd=10^3)
  part_vals[,2] <- rnorm(N,mean=185,sd=10^2)
  part_vals[,3] <- log(rgamma(N,3,(2*300^2))^(-1))

  return(part_vals)

}

# Names of the three variables
varNames <- c("alpha","beta","sigma^2")

# An efficient random walk covariance
sigma_rw <- c(2660,134,0.044)*diag(3)
```

Running MCMC with a Good Start

This section of the document is looking at very basic post-processing: how do we use samples for estimation when we have complete faith in them?

Running MCMC

Below we run MCMC for 1000 iterations. Remember we have to exponentiate the final parameter because we're sampling in terms of $\log \sigma^2$ but we're actually interested in σ^2.

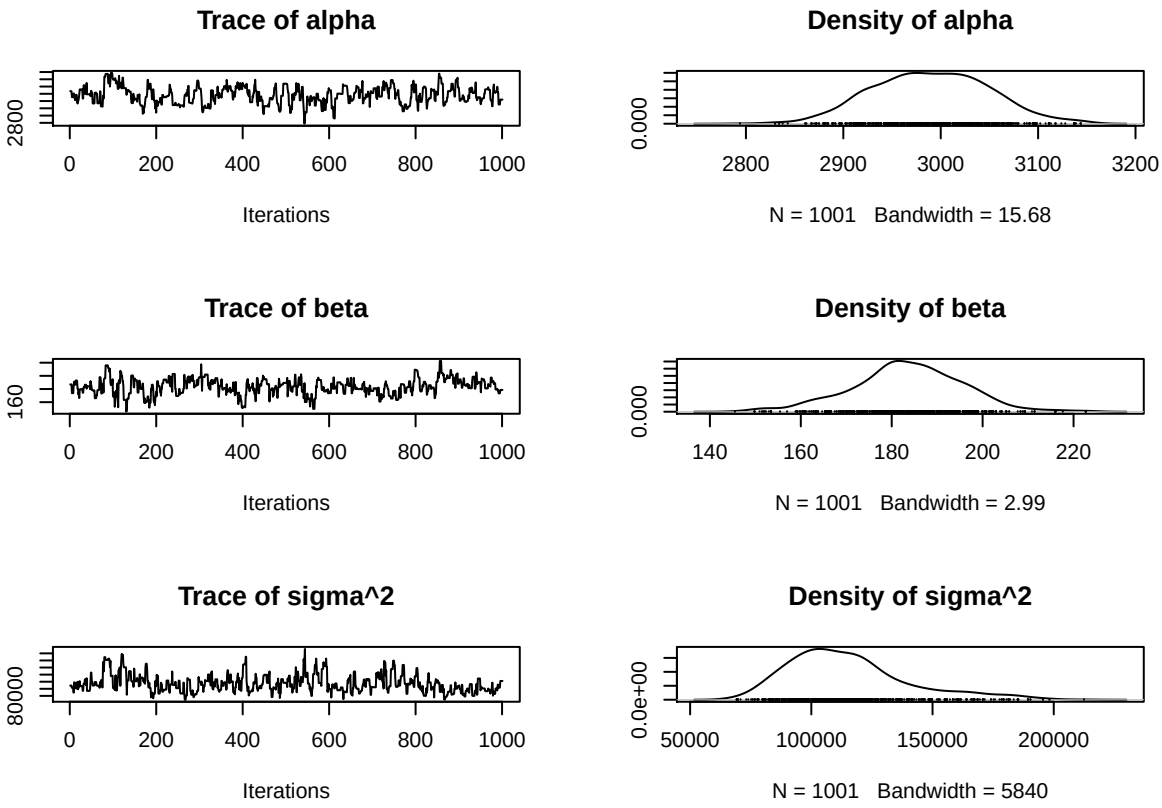
```
set.seed(1)

its <- 10^3 # number of MCMC ierations
theta0 <- c(3018,187,11.6) # starting value

# Running MCMC
chain <- MCMC(theta0,sigma_rw,its,varNames)

# Getting the samples, transforming log(sigma^2) to sigma^2
# and naming each dimension for plotting purposes
samples <- chain$samples
samples[,3] <- exp(samples[,3])
```

Visualisation

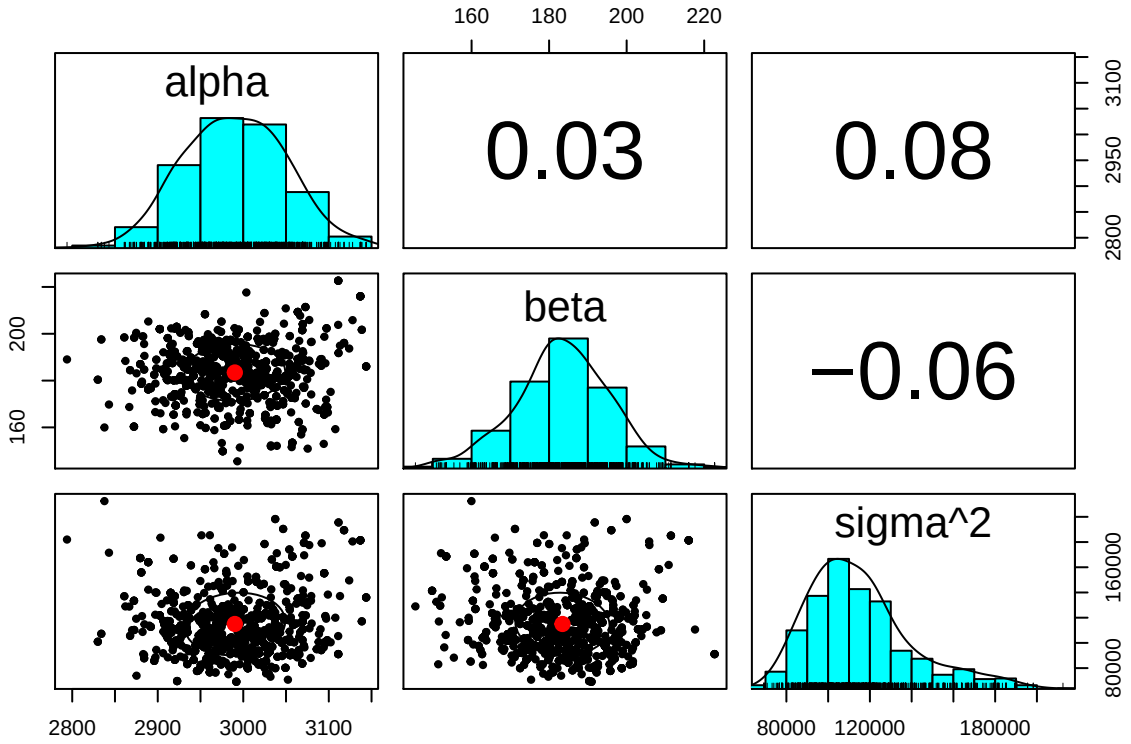
The plots on the left are called trace plots. They show the progression of the samples at each step. These aren't really estimating anything but they'll be useful later on when we're assessing convergence and sample quality.

The right hand side shows estimates of marginal posterior distributions using kernel density estimation.

```
# Plotting the results
plot(samples)
```




```
# Plotting bivariate distributions
pairs.panels(as.matrix(samples), smooth = FALSE)
```



Estimating expectations and quantiles

It is relatively easy to estimate posterior expectations and credible intervals.

```
# Estimating some expectations
colMeans(samples) # Marginal posterior means
```

```
##       alpha        beta     sigma^2
##   2990.1522    183.4497 115006.3897
```

```
mean(sqrt(samples[,3])) # Posterior mean of sigma
```

```
## [1] 337.2695
```

```
var(samples[,2]) # posterior variance of beta
```

```
## [1] 147.8838
```

```
# Estimated 95% credible interval for beta
quantile(samples[,2],c(0.025,0.975))
```

```
##     2.5%    97.5%
## 159.2512 205.8778
```

Single Chain

Here we run a single MCMC chain initialised at a “bad" starting point.

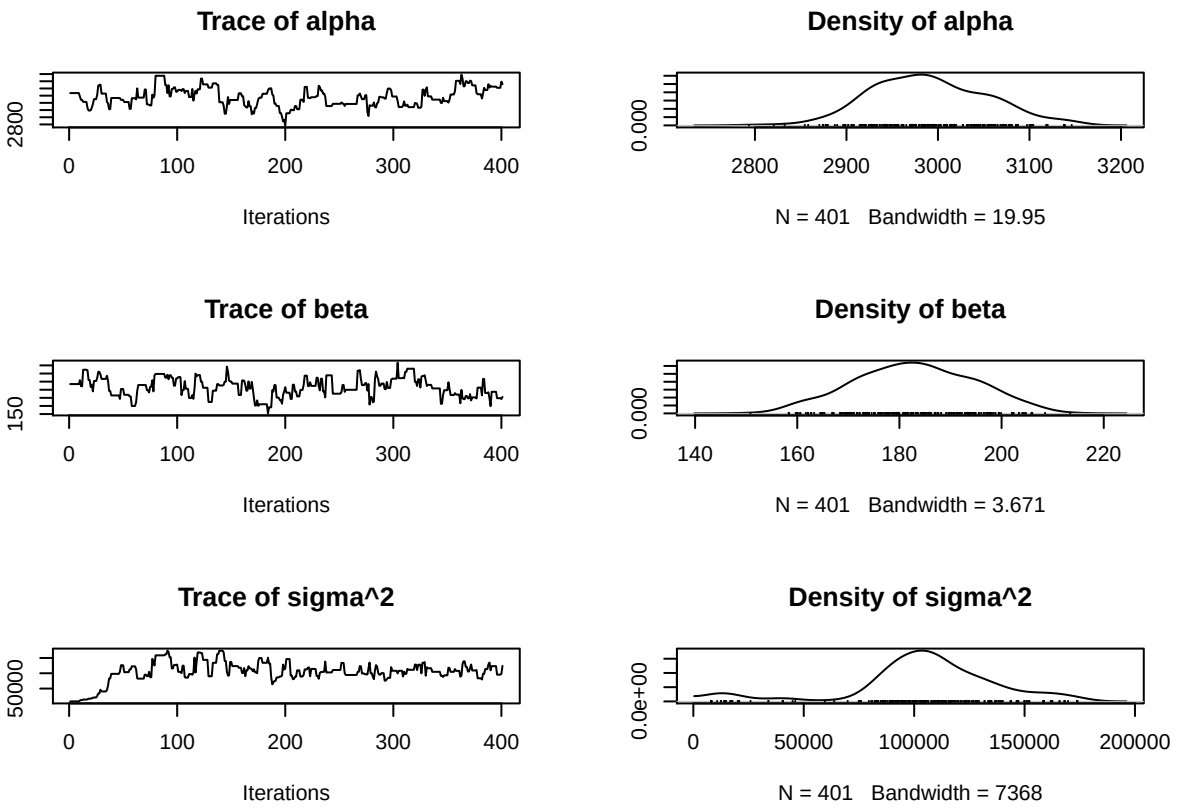
Running the MCMC

```
set.seed(1)
its <- 400 # number of MCMC iterations

# Running MCMC
chain_badstart <- MCMC(c(3018,187,9),sigma_rw,its,varNames)

# Getting the samples, transforming log(sigma^2) to sigma^2
# and naming each dimension for plotting purposes
samples_badstart <- chain_badstart$samples
samples_badstart[,3] <- exp(samples_badstart[,3])
```

Trace Plots

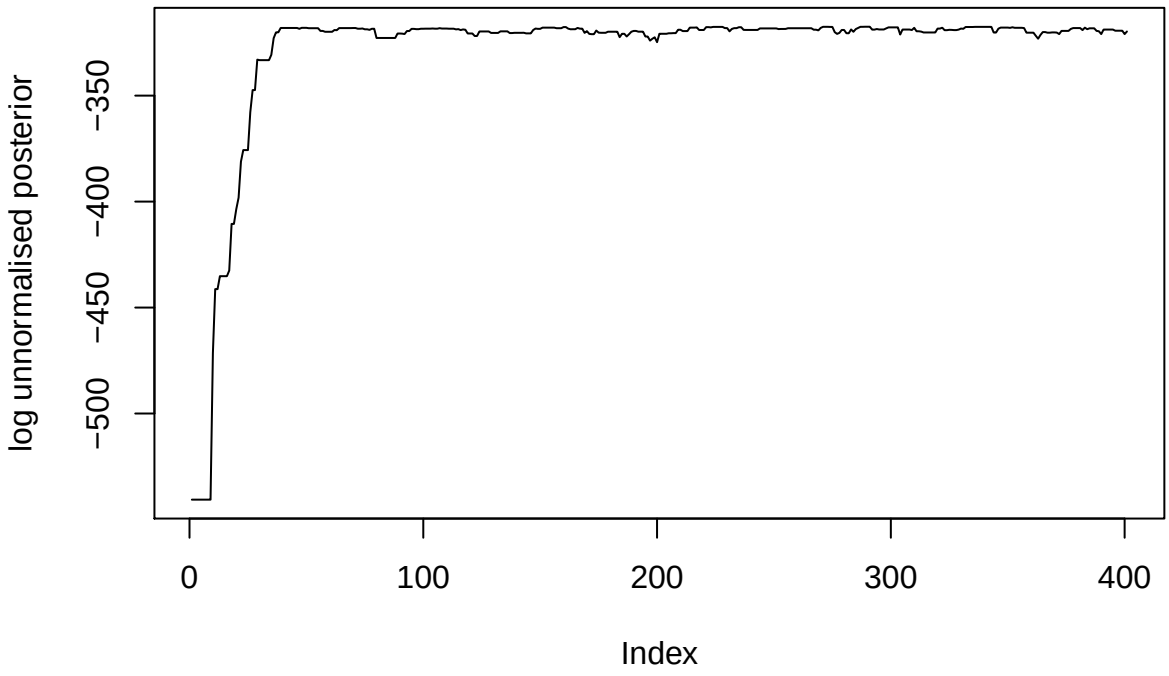
Let’s produce trace plots for each dimension.

```
plot(samples_badstart)
```



Notice that the initial non-convergence is not visible for all parameters. Now let’s produce a trace plot for the log unnormalised posterior.

```
plot(chain_badstart$loglike + chain_badstart$logprior,
     type='l',ylab="log unnormalised posterior")
```




```
length(chain_badstart$loglike + chain_badstart$logprior)
```

```
## [1] 401
```

This can be a nicer approach than looking at all d marginals, but I recommend doing both.

Geweke

Let's do Geweke's formal test of convergence. Notice that there is evidence of non-convergence ($|Z| > 2$) if the first section consists of the first 10% (40 samples), whereas we have insufficient evidence of non-convergence if the first section consists of the first 15% (60 samples). This suggests that 60 samples is enough burn-in.

```
geweke.diag(samples_badstart,frac1=0.1,frac2=0.5)
```

```
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##   alpha    beta sigma^2
## -0.1299  1.7509 -4.3102
```

```
geweke.diag(samples_badstart,frac1=0.15,frac2=0.5)
```

```
##
## Fraction in 1st window = 0.15
```

```
## Fraction in 2nd window = 0.5
##
##   alpha    beta sigma^2
## -0.4152  0.1682 -1.2407
```

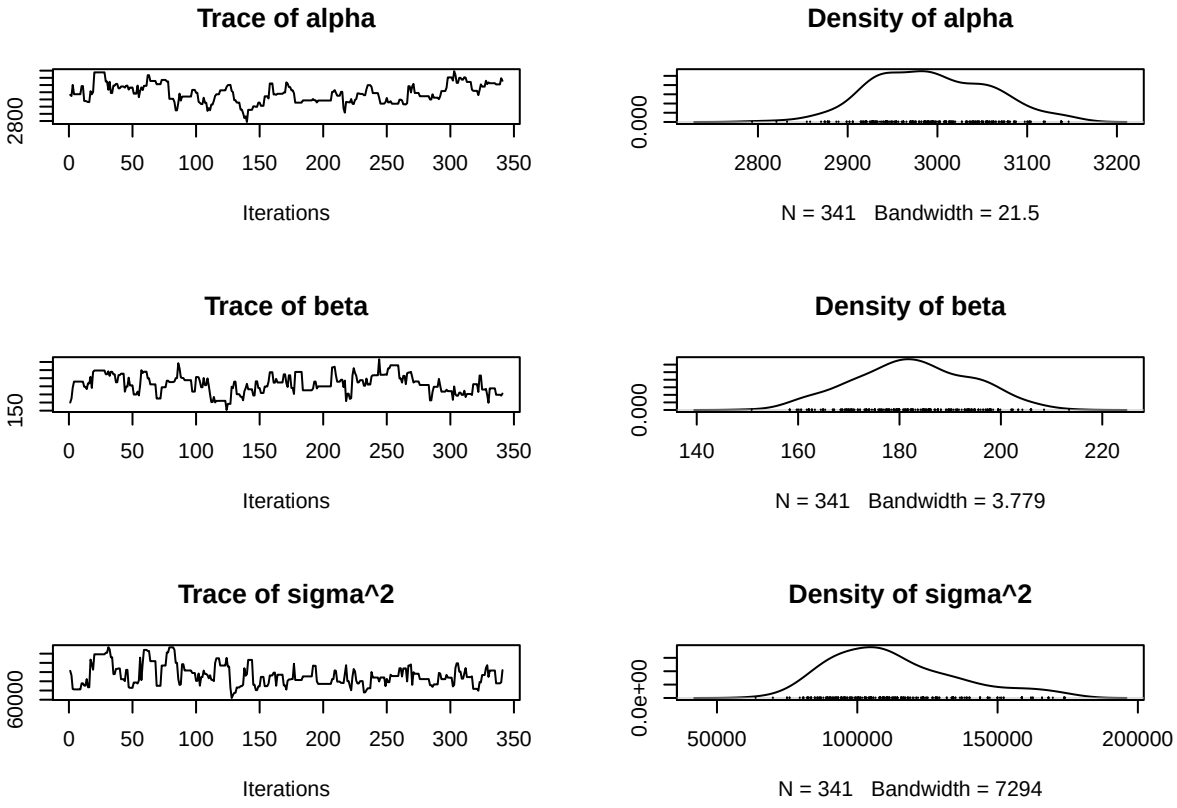
Burn-in

Let's estimate the posterior mean and plot the samples with a burn-in of 60 samples.

```
samples <- mcmc(samples_badstart[61:(its+1),])
colMeans(samples)
```

```
##       alpha        beta     sigma^2
##   2992.6687    182.6055 112911.0741
```

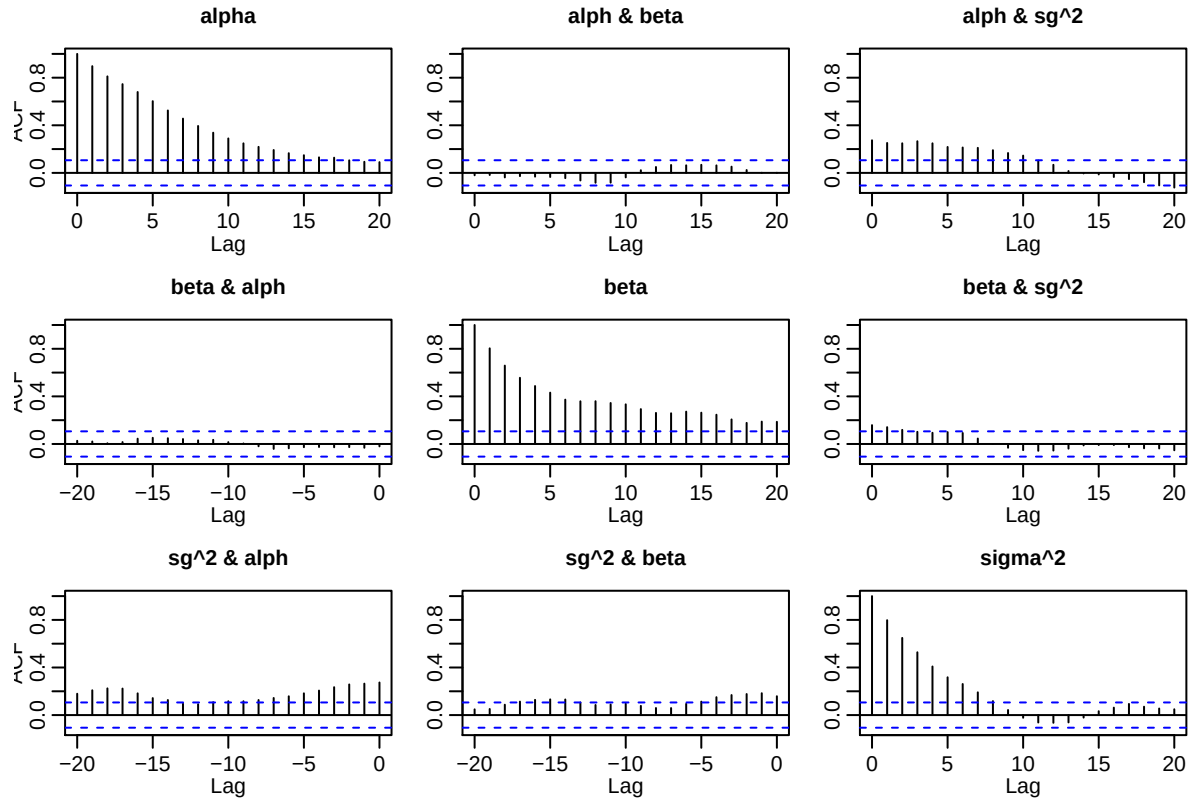
```
plot(samples)
```



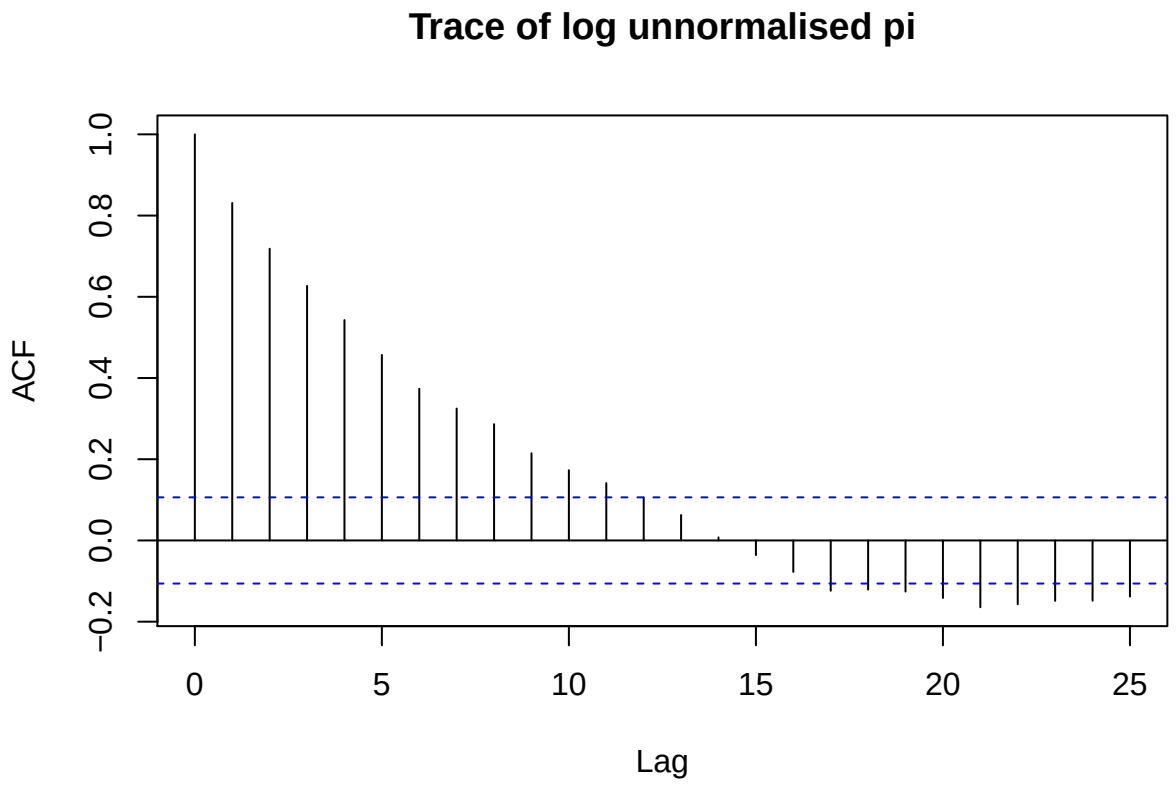
Autocorrelation

How bad is the autocorrelation for this example? It looks like the autocorrelation drops off relatively quickly, which is a good sign. This is more useful as a relative measure.

```
acf(samples)
```




```
acf(chain_badstart$loglike[61:(its+1)] + chain_badstart$logprior[61:(its+1)],
     main="Trace of log unnormalised pi")
```



Effective Sample Size

We can estimate the equivalent number of independent proposals from the posterior by looking at different functions or using Vats et al's multivariate effective sample size.

```
effectiveSize(samples)
```

```
##    alpha     beta  sigma^2
## 18.47982 36.93652 38.31848
```

```
effectiveSize(chain_badstart$loglike[61:(its+1)] + chain_badstart$logprior[61:(its+1)])
```

```
##     var1
## 26.12555
```

```
multiESS(samples)
```

```
## [1] 26.45081
```

Multiple Chains

Running multiple chains from over-dispersed starting points can help us pick up problems with convergence.

Running the MCMC

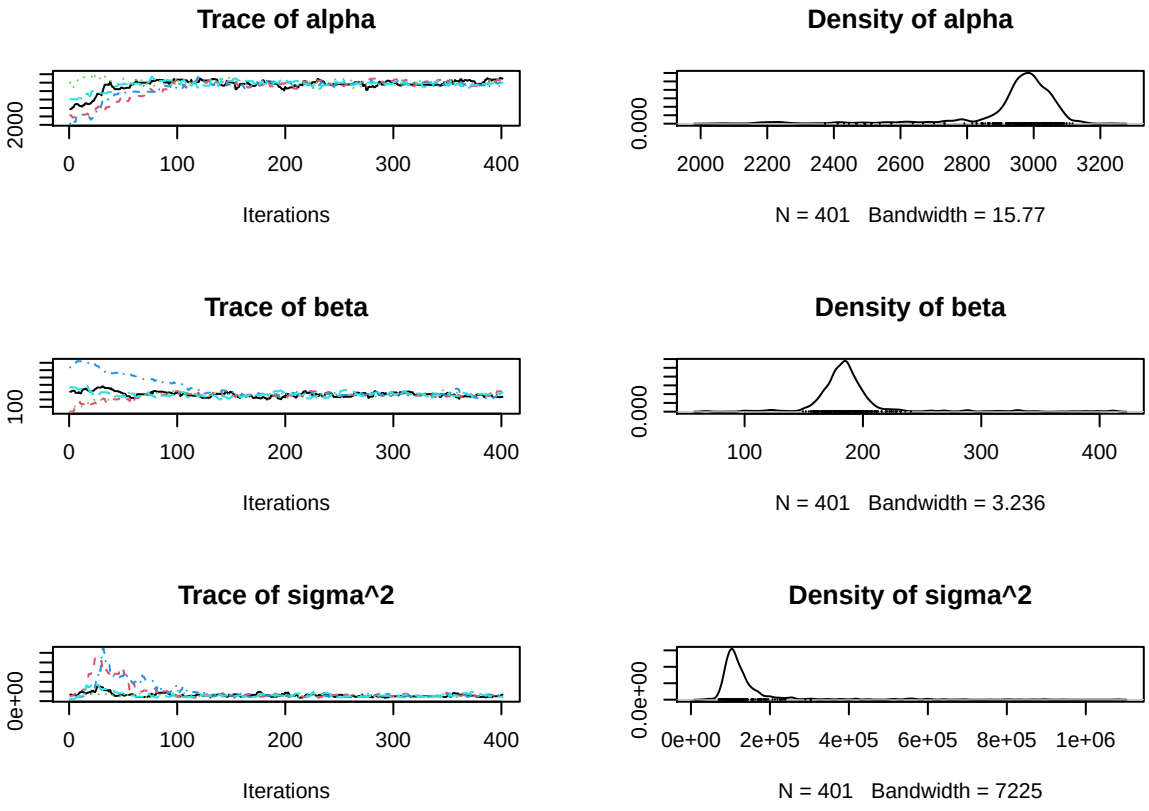
Let's run 5 MCMC chains with different starting points drawn from the posterior distribution.

```
set.seed(1)
n_reps <- 5 # number of chains
its <- 400 # number of MCMC iterations
chains <- list()
for (i in 1:n_reps){
  theta0 <- simprior(1,radiata) # initialising at a prior draw
  chains[[i]] <- MCMC(theta0,sigma_rw,its,varNames)$samples
  chains[[i]][,3] <- exp(chains[[i]][,3])
}
chains_noburnin <- as.mcmc.list(chains)
```

Trace Plots

It looks like the chains have all converged by around iteration 100. Next, we'll investigate this using Gelman & Rubin's $\hat{R}$ diagnostic.

```
plot(chains_noburnin,smooth=FALSE)
```



Rhat

The $\hat{R}$ diagnostic is not < 1.1 for all dimensions, so we have evidence of non-convergence.

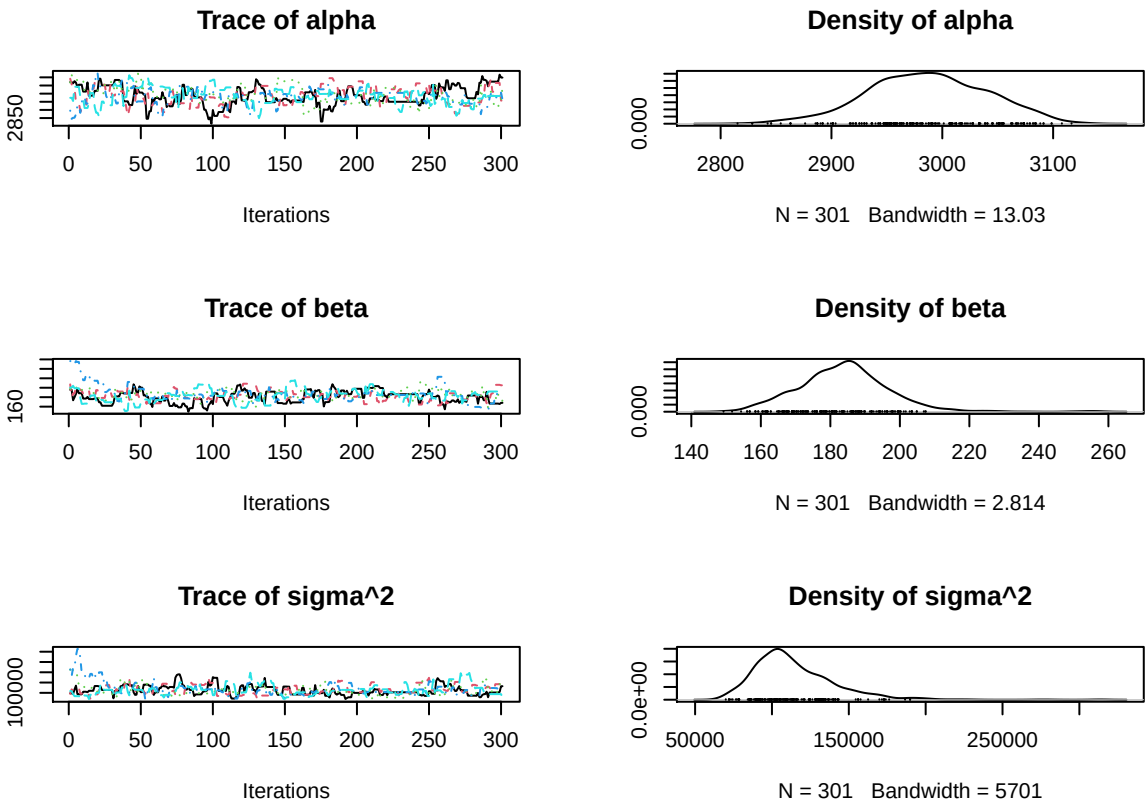
```
gelman.diag(chains_noburnin,autoburnin=FALSE)
```

```
## Potential scale reduction factors:
##
##         Point est. Upper C.I.
## alpha         1.11       1.27
## beta          1.36       2.40
## sigma^2       1.16       1.40
##
## Multivariate psrf
##
## 1.19
```

Removing Burn-In

Let's see if we still have evidence of non-convergence if we remove the first 100 samples as burn-in.

```
# Removing burn-in
burnin <- 100
chains_burnin <- list()
for (i in 1:n_reps){
  chains_burnin[[i]] <- mcmc(chains[[i]][(burnin+1):(its+1),])
}
chains_burnin <- as.mcmc.list(chains_burnin)

# Plotting chains
plot(chains_burnin,smooth=FALSE)
```




```
# Calculating Rhat
gelman.diag(chains_burnin,autoburnin=FALSE)
```

```
## Potential scale reduction factors:
##
##         Point est. Upper C.I.
## alpha         1.01       1.02
## beta          1.05       1.12
## sigma^2       1.02       1.03
##
## Multivariate psrf
##
## 1.03
```

Now we have no evidence of non-convergence.