Week 5 BDA

## Monte Carlo error

Let’s construct a table of parameters leading to different Markov chains, each having the same marginal distribution at the limit of large number of samples but each also having different amount of autocorrelation between the samples.

varTheta = 1  
sigma2.1 = 1  
sigma2.3 = 0.2  
phi.2 = 0.5  
phi.4 = 0.1  
phi.1 = sqrt(1-sigma2.1/varTheta)  
phi.3 = sqrt(1-sigma2.3/varTheta)  
sigma2.2 = varTheta\*(1-phi.2^2)  
sigma2.4 = varTheta\*(1-phi.4^2)  
table.entries = matrix(nrow=4, ncol=4, data=c(  
 varTheta, phi.1, sigma2.1, phi.1,  
 varTheta, phi.2, sigma2.2, phi.2,  
 varTheta, phi.3, sigma2.3, phi.3,  
 varTheta, phi.4, sigma2.4, phi.4  
))  
table.entries <- t(table.entries) # take transpose since matrix fills in the elements in columnwise  
colnames(table.entries) <- c("var(theta)", "phi", "sigma2","corr")  
print(table.entries)

## var(theta) phi sigma2 corr  
## [1,] 1 0.0000000 1.00 0.0000000  
## [2,] 1 0.5000000 0.75 0.5000000  
## [3,] 1 0.8944272 0.20 0.8944272  
## [4,] 1 0.1000000 0.99 0.1000000

Let’s then construct a function to perform Markov chain sampling

# let's first define a function to conduct the sampling  
MarkovChain <- function(phi,sigma2,initial,m){  
 theta = vector(length=m)  
 theta[1] = initial  
 for (i1 in seq(1,m-1,1)){  
 theta[i1+1] = phi\*theta[i1] + rnorm(1,0,sqrt(sigma2))  
 }  
 return(theta)  
}

For this exercise it is handy to use multidimensional arrays to store the results (not necessary but saves some lines of code). Below an example:

set.seed(123)  
arr = array(dim=c(2000,100,4))  
dim(arr)

## [1] 2000 100 4

Now we need to sample 100 independent realizations of length 2000 chains from the Markov chain defined in exercise 3.1 (that is; ) using each of the combinations of and in the rows of the above table.

With each of the chains we approximate , and using Monte Carlo with the , and last samples. Hence, we will construct 100 independent Monte Carlo approximations for the mean and two probabilities of corresponding to Markov chain sample sizes 10, 100 and 1000.

For example the below rows would construct two independent Markov chains of lenght 2000 and calculate the Monte Carlo approximation for the mean with the last 10 samples

## [1] 0.07540569

## [1] -0.1409847

## [1] 2000

#taking the samples  
for (i in 1:4){  
 for (j in 1:100){  
 arr[,j, i] = MarkovChain(table.entries[i,"phi"],table.entries[i,"sigma2"],initial,m)  
 }  
}

res = array(dim=c(3,100,3,4))  
ns = c(10, 100, 1000)  
for (i in 1:4){  
 for (j in 1:100){  
 for (n in 1:3){  
 temp = arr[,j,i][(m-ns[n]+1):m]  
 res[1,j,n,i] = mean(temp)   
 res[2,j,n,i] = length(temp[temp > 0.5])/ns[n] #prob>.5  
 res[3,j,n,i] = length(temp[temp > 2])/ns[n] #prob>2  
 }  
 }  
}

Now, we need to repeat the above steps 100 times, calculate the mean and asked probabilities for each of the 100 chains and then examine how these Monte Carlo estimates behave and match with the exact results as we vary the row of the table and .

**Task 1**

The estimate of the expected value with respect to autocorrelation and amount of samples:

## [1] "Autocorrelation: 0"  
## [1] "=============================================="  
## [1] "Sample-size: 10"  
## [1] "----------------------------------------------"  
## [1] "Mean: 0.0610876401924915"  
## [1] "----------------------------------------------"  
## [1] "Sample-size: 100"  
## [1] "----------------------------------------------"  
## [1] "Mean: 0.00814040015463319"  
## [1] "----------------------------------------------"  
## [1] "Sample-size: 1000"  
## [1] "----------------------------------------------"  
## [1] "Mean: 0.00228571100193841"  
## [1] "----------------------------------------------"  
## [1] "Summary:"  
## [1] "-----------------------------------------------"  
## [1] "Development of difference between ~mean and true mean:"  
## [1] 0.061087640 0.008140400 0.002285711  
## [1] "=============================================="  
## [1] ""  
## [1] "Autocorrelation: 0.5"  
## [1] "=============================================="  
## [1] "Sample-size: 10"  
## [1] "----------------------------------------------"  
## [1] "Mean: -0.0622460198416144"  
## [1] "----------------------------------------------"  
## [1] "Sample-size: 100"  
## [1] "----------------------------------------------"  
## [1] "Mean: -0.0199698950552745"  
## [1] "----------------------------------------------"  
## [1] "Sample-size: 1000"  
## [1] "----------------------------------------------"  
## [1] "Mean: 0.000125640106616392"  
## [1] "----------------------------------------------"  
## [1] "Summary:"  
## [1] "-----------------------------------------------"  
## [1] "Development of difference between ~mean and true mean:"  
## [1] 0.0622460198 0.0199698951 0.0001256401  
## [1] "=============================================="  
## [1] ""  
## [1] "Autocorrelation: 0.894427190999916"  
## [1] "=============================================="  
## [1] "Sample-size: 10"  
## [1] "----------------------------------------------"  
## [1] "Mean: -0.0488198003080058"  
## [1] "----------------------------------------------"  
## [1] "Sample-size: 100"  
## [1] "----------------------------------------------"  
## [1] "Mean: -0.0292209050307855"  
## [1] "----------------------------------------------"  
## [1] "Sample-size: 1000"  
## [1] "----------------------------------------------"  
## [1] "Mean: -0.00266966715070692"  
## [1] "----------------------------------------------"  
## [1] "Summary:"  
## [1] "-----------------------------------------------"  
## [1] "Development of difference between ~mean and true mean:"  
## [1] 0.048819800 0.029220905 0.002669667  
## [1] "=============================================="  
## [1] ""  
## [1] "Autocorrelation: 0.1"  
## [1] "=============================================="  
## [1] "Sample-size: 10"  
## [1] "----------------------------------------------"  
## [1] "Mean: -0.0120943479305464"  
## [1] "----------------------------------------------"  
## [1] "Sample-size: 100"  
## [1] "----------------------------------------------"  
## [1] "Mean: -0.00757476749442646"  
## [1] "----------------------------------------------"  
## [1] "Sample-size: 1000"  
## [1] "----------------------------------------------"  
## [1] "Mean: -0.00309460176673157"  
## [1] "----------------------------------------------"  
## [1] "Summary:"  
## [1] "-----------------------------------------------"  
## [1] "Development of difference between ~mean and true mean:"  
## [1] 0.012094348 0.007574767 0.003094602  
## [1] "=============================================="  
## [1] ""

From above we can see that a bigger sample size leads to an estimate which is closer to the true mean (which is 0), this is in line with the law of large numbers, as the variance of the sample will become smaller as the sample size grows. However, due to the nature of the Markov chain sampling, if the autocorrelation is bigger, it will take a longer time for the chain to converge to the real distribution, and this is visible in this material. In these particular samples, this effect tends to be most visible in the samples with n=100, at n=1000 all the chains seem to be quite usable.

**Task 2**

Now let’s take a look at the estimates for Prob(>0.5):

## [1] "Autocorrelation: 0"  
## [1] "=============================================="  
## [1] "Sample-size: 10"  
## [1] "----------------------------------------------"  
## [1] "Prob( > 0.5): 0.314"  
## [1] "----------------------------------------------"  
## [1] "Sample-size: 100"  
## [1] "----------------------------------------------"  
## [1] "Prob( > 0.5): 0.3137"  
## [1] "----------------------------------------------"  
## [1] "Sample-size: 1000"  
## [1] "----------------------------------------------"  
## [1] "Prob( > 0.5): 0.30969"  
## [1] "----------------------------------------------"  
## [1] "Summary:"  
## [1] "-----------------------------------------------"  
## [1] "Development of difference between ~prob(>.5) and true prob(>.5):"  
## [1] 0.005462461 0.005162461 0.001152461  
## [1] "=============================================="  
## [1] ""  
## [1] "Autocorrelation: 0.5"  
## [1] "=============================================="  
## [1] "Sample-size: 10"  
## [1] "----------------------------------------------"  
## [1] "Prob( > 0.5): 0.288"  
## [1] "----------------------------------------------"  
## [1] "Sample-size: 100"  
## [1] "----------------------------------------------"  
## [1] "Prob( > 0.5): 0.3016"  
## [1] "----------------------------------------------"  
## [1] "Sample-size: 1000"  
## [1] "----------------------------------------------"  
## [1] "Prob( > 0.5): 0.30828"  
## [1] "----------------------------------------------"  
## [1] "Summary:"  
## [1] "-----------------------------------------------"  
## [1] "Development of difference between ~prob(>.5) and true prob(>.5):"  
## [1] 0.0205375387 0.0069375387 0.0002575387  
## [1] "=============================================="  
## [1] ""  
## [1] "Autocorrelation: 0.894427190999916"  
## [1] "=============================================="  
## [1] "Sample-size: 10"  
## [1] "----------------------------------------------"  
## [1] "Prob( > 0.5): 0.305"  
## [1] "----------------------------------------------"  
## [1] "Sample-size: 100"  
## [1] "----------------------------------------------"  
## [1] "Prob( > 0.5): 0.2976"  
## [1] "----------------------------------------------"  
## [1] "Sample-size: 1000"  
## [1] "----------------------------------------------"  
## [1] "Prob( > 0.5): 0.30895"  
## [1] "----------------------------------------------"  
## [1] "Summary:"  
## [1] "-----------------------------------------------"  
## [1] "Development of difference between ~prob(>.5) and true prob(>.5):"  
## [1] 0.0035375387 0.0109375387 0.0004124613  
## [1] "=============================================="  
## [1] ""  
## [1] "Autocorrelation: 0.1"  
## [1] "=============================================="  
## [1] "Sample-size: 10"  
## [1] "----------------------------------------------"  
## [1] "Prob( > 0.5): 0.307"  
## [1] "----------------------------------------------"  
## [1] "Sample-size: 100"  
## [1] "----------------------------------------------"  
## [1] "Prob( > 0.5): 0.3078"  
## [1] "----------------------------------------------"  
## [1] "Sample-size: 1000"  
## [1] "----------------------------------------------"  
## [1] "Prob( > 0.5): 0.30688"  
## [1] "----------------------------------------------"  
## [1] "Summary:"  
## [1] "-----------------------------------------------"  
## [1] "Development of difference between ~prob(>.5) and true prob(>.5):"  
## [1] 0.0015375387 0.0007375387 0.0016575387  
## [1] "=============================================="  
## [1] ""

Above we see the different results for Prob(>0.5) compared to the true value for a N(0,1) distribution (~0.31). Again, the estimates of the probability are closer to the true value as the sample size grows. The convergence is faster if autocorrelation is lower, but at n=1000 the chains are still quite usable regardless of autocorrelation.

**Task 3**

Now let’s take a look at the estimates for Prob(>2):

## [1] "Autocorrelation: 0"  
## [1] "=============================================="  
## [1] "Sample-size: 10"  
## [1] "----------------------------------------------"  
## [1] "Prob( > 2): 0.03"  
## [1] "----------------------------------------------"  
## [1] "Sample-size: 100"  
## [1] "----------------------------------------------"  
## [1] "Prob( > 2): 0.0225"  
## [1] "----------------------------------------------"  
## [1] "Sample-size: 1000"  
## [1] "----------------------------------------------"  
## [1] "Prob( > 2): 0.02292"  
## [1] "----------------------------------------------"  
## [1] "Summary:"  
## [1] "-----------------------------------------------"  
## [1] "Development of difference between ~prob(>2) and true prob(>2):"  
## [1] 0.0072498681 0.0002501319 0.0001698681  
## [1] "=============================================="  
## [1] ""  
## [1] "Autocorrelation: 0.5"  
## [1] "=============================================="  
## [1] "Sample-size: 10"  
## [1] "----------------------------------------------"  
## [1] "Prob( > 2): 0.023"  
## [1] "----------------------------------------------"  
## [1] "Sample-size: 100"  
## [1] "----------------------------------------------"  
## [1] "Prob( > 2): 0.0193"  
## [1] "----------------------------------------------"  
## [1] "Sample-size: 1000"  
## [1] "----------------------------------------------"  
## [1] "Prob( > 2): 0.02242"  
## [1] "----------------------------------------------"  
## [1] "Summary:"  
## [1] "-----------------------------------------------"  
## [1] "Development of difference between ~prob(>2) and true prob(>2):"  
## [1] 0.0002498681 0.0034501319 0.0003301319  
## [1] "=============================================="  
## [1] ""  
## [1] "Autocorrelation: 0.894427190999916"  
## [1] "=============================================="  
## [1] "Sample-size: 10"  
## [1] "----------------------------------------------"  
## [1] "Prob( > 2): 0.033"  
## [1] "----------------------------------------------"  
## [1] "Sample-size: 100"  
## [1] "----------------------------------------------"  
## [1] "Prob( > 2): 0.0216"  
## [1] "----------------------------------------------"  
## [1] "Sample-size: 1000"  
## [1] "----------------------------------------------"  
## [1] "Prob( > 2): 0.02088"  
## [1] "----------------------------------------------"  
## [1] "Summary:"  
## [1] "-----------------------------------------------"  
## [1] "Development of difference between ~prob(>2) and true prob(>2):"  
## [1] 0.010249868 0.001150132 0.001870132  
## [1] "=============================================="  
## [1] ""  
## [1] "Autocorrelation: 0.1"  
## [1] "=============================================="  
## [1] "Sample-size: 10"  
## [1] "----------------------------------------------"  
## [1] "Prob( > 2): 0.024"  
## [1] "----------------------------------------------"  
## [1] "Sample-size: 100"  
## [1] "----------------------------------------------"  
## [1] "Prob( > 2): 0.0243"  
## [1] "----------------------------------------------"  
## [1] "Sample-size: 1000"  
## [1] "----------------------------------------------"  
## [1] "Prob( > 2): 0.02256"  
## [1] "----------------------------------------------"  
## [1] "Summary:"  
## [1] "-----------------------------------------------"  
## [1] "Development of difference between ~prob(>2) and true prob(>2):"  
## [1] 0.0012498681 0.0015498681 0.0001901319  
## [1] "=============================================="  
## [1] ""

And no surprises here. Again we see convergence with bigger amounts of samples, and slower convergence at higher autocorrelation.

**Task 4**

The general conclusion is already mentioned as a part of the answers above, but in accordance to the law or large numbers, the estimates based on the samples converges towards the true distribution when the sample-size grows. Autocorrelation slows down the convergence, as the values tend to be more linked to each other, and therefore do not represent as random a sample as they would without autocorrelation, so you need bigger samples. At sample-size 1000, the values are quite reliable, so you would want to take a big enough amount of samples when using Markov chains.

## Mauna Loa CO2 data

**1. variable transformations**

Some formulas to find out how to transform the samples back to original scale:

So reducing will change the intercept, and needs to be corrected:

$b = - ax\_m $

What happens when reducing from :

So we get that:

$b = + y\_m $

Let’s look at the division with :

So we get that:

And division with :

So we get that:

.

So combining the above:

**2. Build and analyze Stan model**

Load the needed libraries.

library(ggplot2)  
library(StanHeaders)  
library(rstan)

##   
## rstan version 2.26.13 (Stan version 2.26.1)

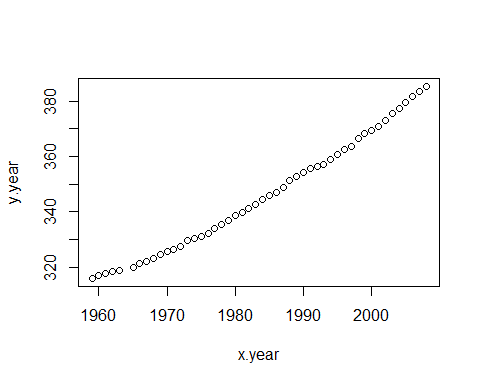
## For execution on a local, multicore CPU with excess RAM we recommend calling  
## options(mc.cores = parallel::detectCores()).  
## To avoid recompilation of unchanged Stan programs, we recommend calling  
## rstan\_options(auto\_write = TRUE)  
## For within-chain threading using `reduce\_sum()` or `map\_rect()` Stan functions,  
## change `threads\_per\_chain` option:  
## rstan\_options(threads\_per\_chain = 1)

## Do not specify '-march=native' in 'LOCAL\_CPPFLAGS' or a Makevars file

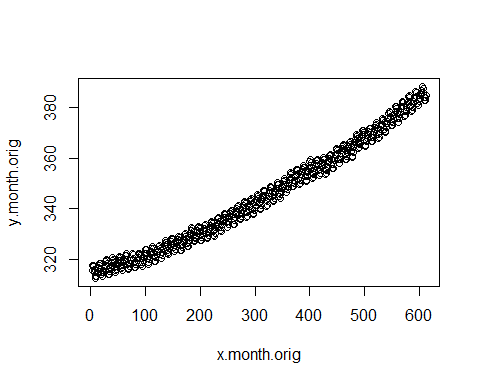
options(mc.cores = parallel::detectCores())  
rstan\_options(auto\_write = TRUE)

Load the data and explore its properties

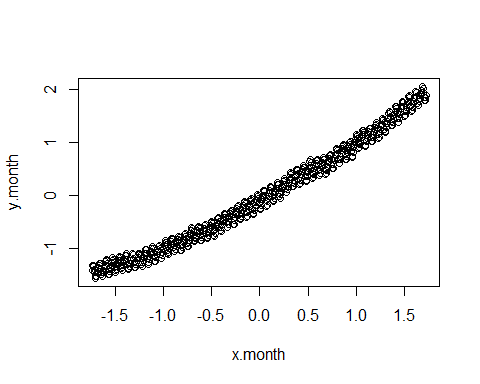
# Load the data and explore it visually  
maunaloa.dat = read.table("maunaloa\_data.txt", header=FALSE, sep="\t")  
# The columns are   
# Year January February ... December Annual average  
  
# Notice! values -99.99 denote NA  
  
# Let's take the yearly averages and plot them  
x.year = as.vector(t(maunaloa.dat[,1]))  
y.year = as.vector(t(maunaloa.dat[,14]))  
# remove NA rows  
x.year = x.year[y.year>0]  
y.year = y.year[y.year>0]  
plot(x.year,y.year)



# Let's take the monthy values and construct a "running month" vector  
y.month.orig = as.vector(t(maunaloa.dat[,2:13]))  
x.month.orig = as.vector(seq(1,length(y.month.orig),1))  
  
# remove NA rows  
x.month.orig = x.month.orig[y.month.orig>0]  
y.month.orig = y.month.orig[y.month.orig>0]  
plot(x.month.orig,y.month.orig)



# standardize y and x  
my = mean(y.month.orig) # mean of y values  
stdy = sd(y.month.orig) # std of y values  
y.month = (y.month.orig - my)/stdy # standardized y values  
  
mx = mean(x.month.orig) # mean of y values  
stdx = sd(x.month.orig) # std of y values  
x.month = (x.month.orig- mx)/stdx # standardized y values  
  
plot(x.month,y.month)



The model description and setting data into list

mauna\_loa\_c02\_model = "  
data{  
 int<lower=0> n;  
 real y[n];  
 real x[n];  
 real sdy;  
 real sdx;  
 real meany;  
 real meanx;  
}  
parameters{  
 real alpha;  
 real beta;  
 real<lower=0> sigma2;  
}  
transformed parameters{  
 real alpha\_t;  
 real beta\_t;  
 real sigma2\_t;  
 alpha\_t = sdy \* (alpha - beta \* meanx / sdx) + meany;  
 beta\_t = beta \* sdy / sdx;  
 sigma2\_t = square(sdy \* sqrt(sigma2));  
}  
  
model{  
 alpha ~ normal(0,sqrt(10));  
 beta ~ normal(1,sqrt(10));  
 sigma2 ~ inv\_gamma(0.001, 0.001);  
 for (i in 1:n){  
 y[i] ~ normal(alpha + beta\*x[i], sqrt(sigma2));  
 }  
}"  
  
data <- list (n=length(x.month), y=y.month, x=x.month,meany=mean(y.month.orig),sdy=sd(y.month.orig), meanx=mean(x.month.orig),sdx=sd(x.month.orig))

Now we will start the analysis. Define parameters and set initial values for them. We are going to sample four chains so we need four starting points. It is good practice to set them far apart from each others. We build linear regression model on data in order to get some reasonable initial values for our model parameters. Examine the convergence.

# Initial values  
init1 <- list(alpha=0.7, beta=1.7, sigma2=0.2)  
init2 <- list(alpha=-0.5, beta=0.3, sigma2=0.01)  
init3 <- list(alpha=0, beta=1, sigma2=0.001)  
init4 <- list(alpha=0.5, beta=1.5, sigma2=0.02)  
#inits <- list(init1)  
inits <- list(init1, init2, init3, init4)  
  
post=stan(model\_code=mauna\_loa\_c02\_model,data=data,warmup=500,iter=2000,chains=4,thin=1,init=inits,control = list(adapt\_delta = 0.8,max\_treedepth = 10))

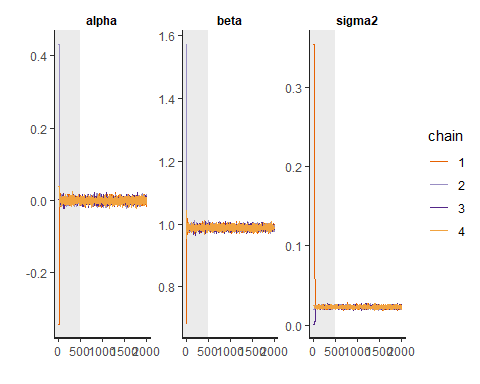
# Check for convergence, see PSRF (Rhat in Stan)  
print(post,pars=c("alpha","beta","sigma2"))

## Inference for Stan model: anon\_model.  
## 4 chains, each with iter=2000; warmup=500; thin=1;   
## post-warmup draws per chain=1500, total post-warmup draws=6000.  
##   
## mean se\_mean sd 2.5% 25% 50% 75% 97.5% n\_eff Rhat  
## alpha 0.00 0 0.01 -0.01 0.00 0.00 0.00 0.01 5730 1  
## beta 0.99 0 0.01 0.98 0.98 0.99 0.99 1.00 6597 1  
## sigma2 0.02 0 0.00 0.02 0.02 0.02 0.02 0.03 3879 1  
##   
## Samples were drawn using NUTS(diag\_e) at Fri Dec 2 10:52:27 2022.  
## For each parameter, n\_eff is a crude measure of effective sample size,  
## and Rhat is the potential scale reduction factor on split chains (at   
## convergence, Rhat=1).

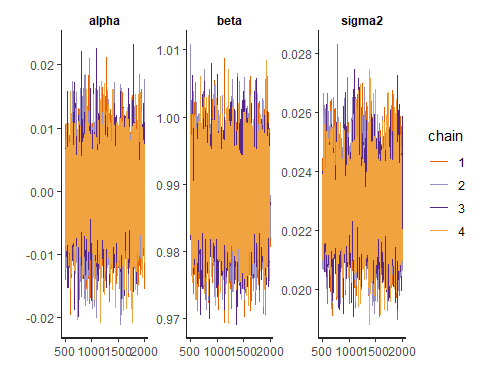
print(post)

## Inference for Stan model: anon\_model.  
## 4 chains, each with iter=2000; warmup=500; thin=1;   
## post-warmup draws per chain=1500, total post-warmup draws=6000.  
##   
## mean se\_mean sd 2.5% 25% 50% 75% 97.5% n\_eff Rhat  
## alpha 0.00 0.00 0.01 -0.01 0.00 0.00 0.00 0.01 5730 1  
## beta 0.99 0.00 0.01 0.98 0.98 0.99 0.99 1.00 6597 1  
## sigma2 0.02 0.00 0.00 0.02 0.02 0.02 0.02 0.03 3879 1  
## alpha\_t 308.72 0.00 0.26 308.21 308.55 308.73 308.89 309.24 6591 1  
## beta\_t 0.12 0.00 0.00 0.12 0.12 0.12 0.12 0.12 6597 1  
## sigma2\_t 10.13 0.01 0.59 9.03 9.72 10.11 10.51 11.31 3879 1  
## lp\_\_ 843.97 0.02 1.24 840.67 843.42 844.29 844.88 845.39 2956 1  
##   
## Samples were drawn using NUTS(diag\_e) at Fri Dec 2 10:52:27 2022.  
## For each parameter, n\_eff is a crude measure of effective sample size,  
## and Rhat is the potential scale reduction factor on split chains (at   
## convergence, Rhat=1).

plot(post, pars=c("alpha","beta","sigma2"),plotfun= "trace", inc\_warmup = TRUE)



plot(post, pars=c("alpha","beta","sigma2"), plotfun= "trace", inc\_warmup = FALSE)



As we can see from the plot and the summary, the chains have converged quite well and Rhat is 1 for all parameters.

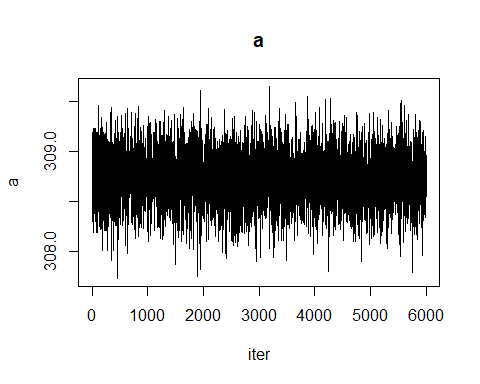
post\_sample=as.matrix(post, pars =c("alpha","beta","sigma2"))

a\_dot=post\_sample[,1]  
b\_dot=post\_sample[,2]  
sigma2\_dot=post\_sample[,3]

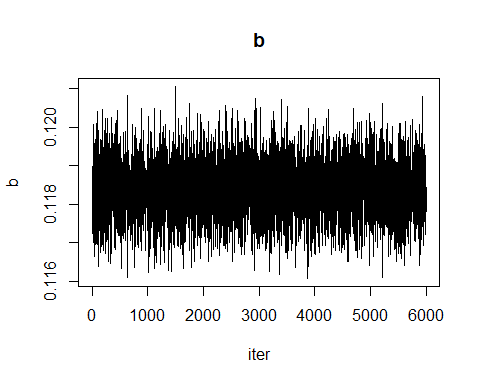
a = stdy \* (a\_dot - b\_dot \* mx / stdx) + my;  
b = b\_dot \* stdy / stdx;  
sigma2 = (stdy\*sqrt(sigma2\_dot))^2

Now parameter a contains a sample from the posterior and parameter b contains sample from the posterior . We can now plot sample chains and histograms of them and do the required summaries.

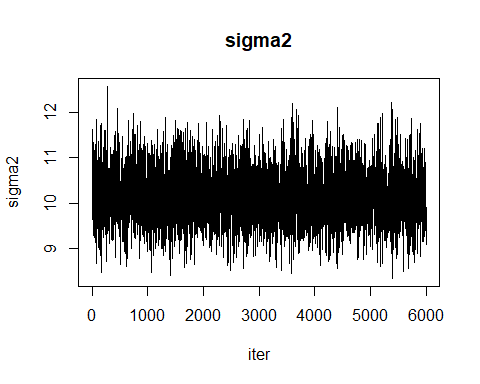
#Trace plot of MCMC output to see if the chains have converged for the original parameters  
plot(a, main="a", xlab="iter",type="l")



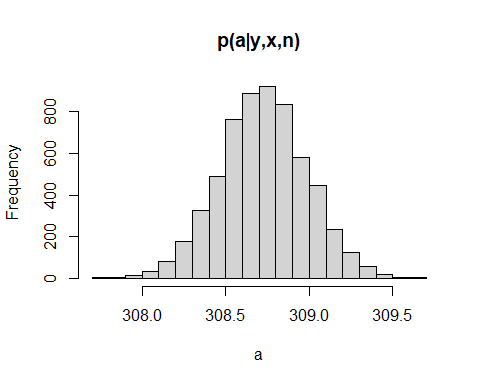
plot(b, main="b", xlab="iter",type="l")



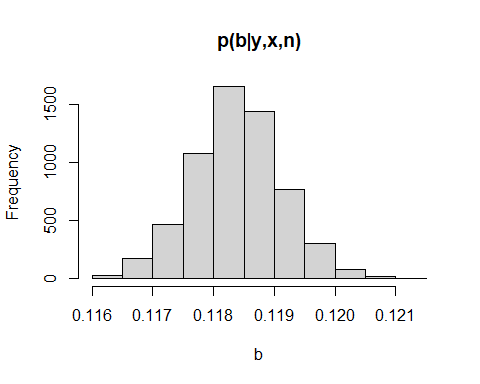
plot(sigma2, main="sigma2", xlab="iter",type="l")



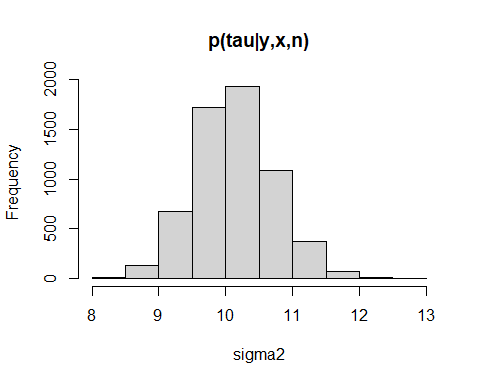
hist(a, main="p(a|y,x,n)", xlab="a")



hist(b, main="p(b|y,x,n)", xlab="b")



hist(sigma2, main="p(tau|y,x,n)", xlab="sigma2")



#calculate the required summaries  
#mean and 95% confidence interval of alpha  
mean(a)

## [1] 308.7246

quantile(a,probs=c(0.025,0.975))

## 2.5% 97.5%   
## 308.2094 309.2370

#mean and 95% confidence interval of beta  
mean(b)

## [1] 0.1183882

quantile(b,probs=c(0.025,0.975))

## 2.5% 97.5%   
## 0.1169239 0.1198362

#mean and 95% confidence interval of sigma^2  
mean(sigma2)

## [1] 10.13177

quantile(sigma2,probs=c(0.025,0.975))

## 2.5% 97.5%   
## 9.027005 11.311355

The chains look well converged.

**3. Interpretation of and**

Linear mean function – is the response from the covariate and some correlation plus the starting point from the beginning of measurements, e.g., the correlation between the covariate added to where the measurement and the target value y. This is the epistemic uncertainty that a sufficient amount of data and a correctly done modelling can alleviate, the “true” relationship between the measured x values (in this case, months that passes by) and the measured response y value (co2-amount that grows over time).

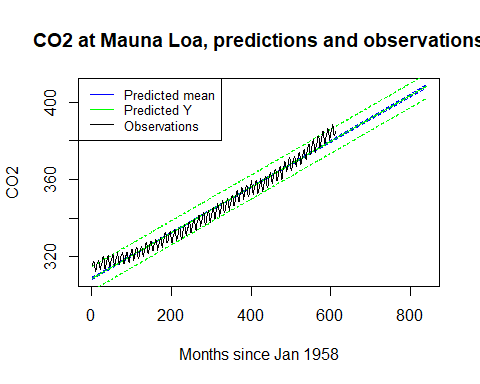
Epsilon is a combination of some epistemic uncertainty left in our model and the irreduceable error. The irreduceable error is the error that can not be reduced, and is an aleatory uncertainty, a variation in the measurement or the system being measured which we can not alleviate by studying the data, we can only estimate its size.

**4. visualization of the regression curve**

Data covers years from 1958 to 2008. Therefore, we need to construct prediction points and predict the historical and future next 20 years of CO2 concentrations

for (i in 1:length(x.pred)) {  
 #remember mu\_i = a + b\*x\_i  
 mu[i,] = a + b\*(i)  
 mean\_mu[i] = mean(mu[i,])  
 int\_mu[i,] = quantile(mu[i,],probs=c(0.025,0.975))  
 #y\_i = mu\_i + e\_i and e\_i ~ N(0,sigma2)  
 y.tilde[i,] = rnorm(mu[i,],mu[i,], sqrt(sigma2))  
 mean\_y[i] = mean(y.tilde[i,])  
 int\_y[i,] = quantile(y.tilde[i,],probs=c(0.025,0.975))  
}

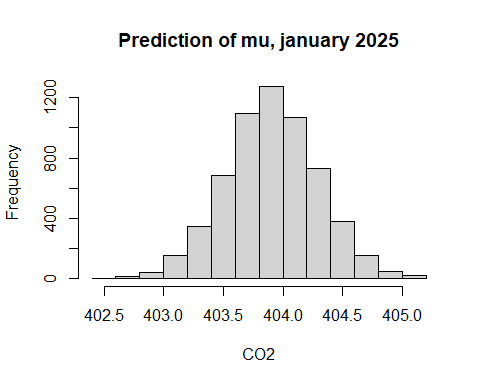
# plot the mean and quantiles for mean function and (replicate) observations and the real observations  
plot(x.pred,mean\_mu, col="blue", main="CO2 at Mauna Loa, predictions and observations", type="l", xlab="Months since Jan 1958", ylab="CO2")  
lines(x.pred, int\_mu[,1], col="blue", lty=2, type='l')  
lines(x.pred, int\_mu[,2], col="blue", lty=2, type="l")  
lines(x.pred,mean\_y, col="green",type="l")  
lines(x.pred, int\_y[,1], col="green", lty=2, type='l')  
lines(x.pred, int\_y[,2], col="green", lty=2, type="l")  
lines(x.month.orig,y.month.orig, col="black", type="l")  
legend(x="topleft", legend=c("Predicted mean", "Predicted Y", "Observations"),  
 col=c("blue", "green", "black"), lty=1:1:1, cex=0.8)



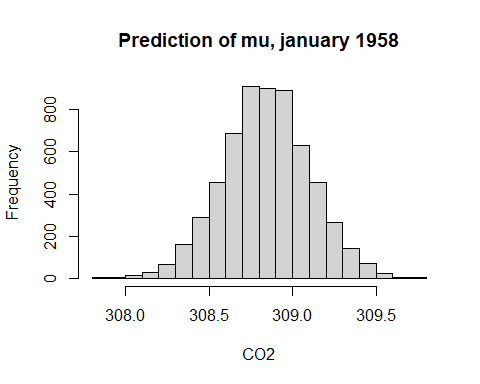
**5. CO2 concentration in January 2025 and 1958**

Posterior predictive distribution of the mean function in January 2025, January 1958 and the difference between these. Notice! x=1 corresponds to January 1958

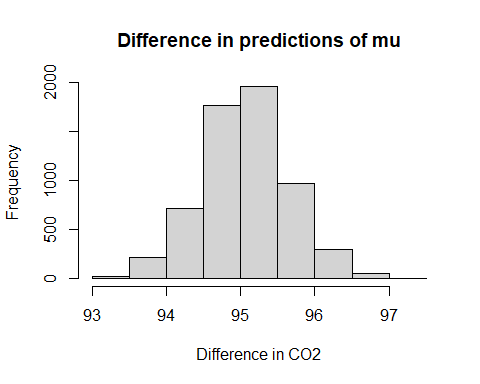
jan2025 <- (2025-1958)\*12  
jan2025\_mu <- a + b\*jan2025  
jan1958\_mu <- a + b\*1  
jan2025\_tilde <- rnorm(b, jan2025\_mu, sqrt(sigma2))  
jan1958\_tilde <- rnorm(b, jan1958\_mu, sqrt(sigma2))  
hist(jan2025\_mu, main="Prediction of mu, january 2025", xlab = "CO2")



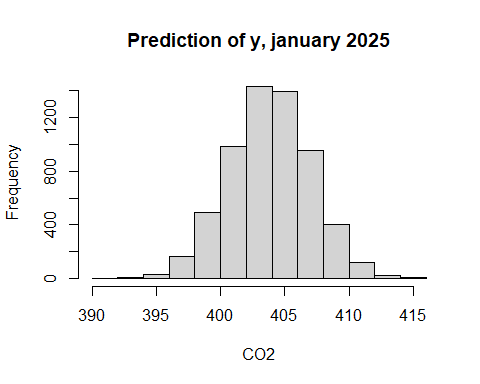
hist(jan1958\_mu, main="Prediction of mu, january 1958", xlab = "CO2")



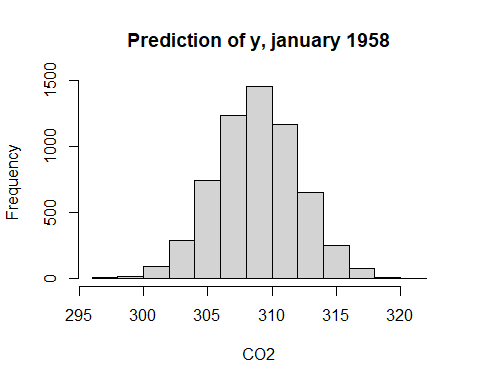
diff\_mu <- jan2025\_mu - jan1958\_mu  
hist(diff\_mu, main="Difference in predictions of mu", xlab = "Difference in CO2")



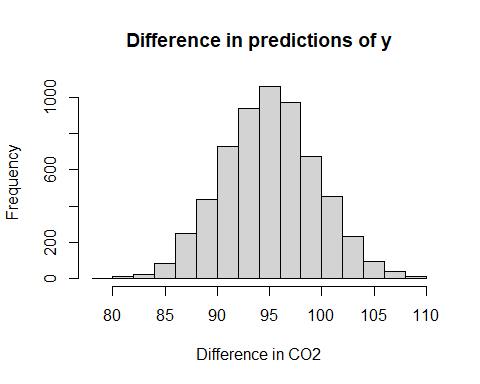
hist(jan2025\_tilde, main="Prediction of y, january 2025", xlab = "CO2")



hist(jan1958\_tilde, main="Prediction of y, january 1958", xlab = "CO2")



diff <- jan2025\_tilde - jan1958\_tilde  
hist(diff, main="Difference in predictions of y", xlab = "Difference in CO2")

 .

As we can see from above, the distribution of differs from in that it is wider, i.e., that there is a bigger variation in the values. This comes from the fact that when we draw predictive samples, we add the , which contains the irreducible error as well as the model bias, we do not, after all, know that the true response is strictly linear. So the later is a sample drawn out of a normal distribution on top of the distribution of that we estimated from the model. We could say, that in the predictive y we double the uncertainty: there is the epistemic uncertainty inherent in our modeling of the linear response, and there is the aleatory uncertainty inherent as the irreducible error, modelled as a normal distribution with variation .