Computational Statistics Assignment 3

Kyriacos Xanthos
CID: 01389741

December 10, 2021

Question 1

In this question we consider a model of the form:

$$Y_t = \alpha t + \beta \cos(\omega t) + \epsilon_t, \quad \epsilon_t \sim_{\text{i.i.d}} \mathcal{N}\left(0, \sigma^2\right) \quad \text{for } t = 1, \dots 50 \tag{1}$$

where α and β are fixed unknown parameters and $\omega = \frac{\pi}{10}$ and $\sigma^2 = 0.1$ are known. To obtain the posterior distribution $p(\alpha, \beta|\mathcal{D})$ where $\mathcal{D} = \{y_t\}_{t=1}^{50}$ is the dataset we have available to us, we consider the priors of α and β which follow a Student's t-distribution with 5 degrees of freedom independently. Noting that responses y_i are independent from each other, we can obtain the likelihood as the product of each likelihood $p(y_i|t_i, \alpha, \beta)$. We then apply Bayes' rule to derive the posterior distribution:

$$p(\theta \mid \mathcal{D}) \propto g(\alpha; 5) g(\beta; 5) \prod_{t=1}^{50} \Phi\left(Y_t; \alpha t + \beta \cos(\omega t), \sigma^2\right) := f(\alpha, \beta) \tag{2}$$

where $\theta = (\alpha, \beta)$, $g(\alpha, s)$ and $g(\beta, s)$ are the probability density functions of a Student's t-distribution with s degrees of freedom for α and β, and $\Phi(y; m, \sigma^2)$ is the pdf of a normal distribution with mean m and variance σ^2.

Question 2

We have chosen to use the following proposal: given the current estimate of the parameter $\theta = (\alpha, \beta)$, we propose $\theta' = (\alpha', \beta')$ such as $\alpha' \sim \mathcal{N}\left(\alpha, \sigma_\alpha^2\right)$, $\beta' \sim \mathcal{N}\left(\beta, \sigma_\beta^2\right)$ (an additive random walk proposal). Then since we have a symmetric proposal around all the positions the term $\frac{q(\theta', \theta)}{q(\theta, \theta')}$ (where q is the proposal distribution) is equal to 1. Then the Metropolis Hastings algorithm used is outlined below:

Algorithm 1 Metropolis-Hastings algorithm

Input: Starting value (α, β) ; variance of proposals $\sigma_\alpha^2, \sigma_\beta^2$; number of iterations n

for *t=1,2,...* **do**

- Let $\alpha' \sim \mathcal{N}\left(\alpha^{t-1}, \sigma_\alpha^2\right)$
- Let $\beta' \sim \mathcal{N}\left(\beta^{t-1}, \sigma_\beta^2\right)$
- Let $U \sim U(0, 1)$
- **if** $U \leq \min\left(\frac{f(\alpha', \beta')}{f(\alpha^{t-1}, \beta^{t-1})}, 1\right)$ **then**
  - $\alpha^t = \alpha', \quad \beta^t = \beta'$
- **else**
  - $\alpha^t = \alpha^{t-1}, \quad \beta^t = \beta^{t-1}$
- **end**

end

Return: $(\alpha^t, \beta^t)_{t=1}^n$

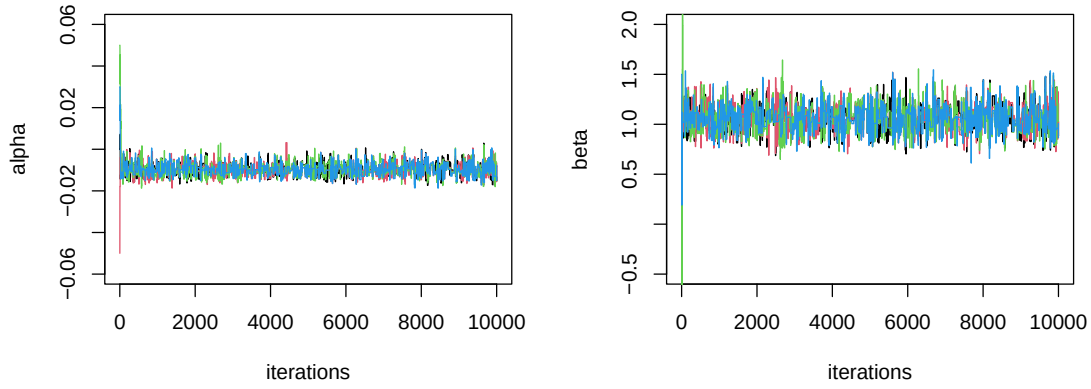

Figure 1: Traceplot for the values of α (left) and β (right) for $10,000$ iterations. We used $\sigma_\alpha = 0.01, \sigma_\beta = 0.8$ with four different initial conditions.

We then run the MCMC sampler for 1×10^4 iterations with $\sigma_\alpha = 0.01, \sigma_\beta = 0.8$ four times with different initial conditions, we obtain the traceplots shown in Figure 1.

Note that we have tried different combinations for the values of σ_α and σ_β but the combination we ended up using gave the best largest decrease in the Autocorrelation plots which qualitatively show the decrease in correlation between the samples in the Markov Chain. An example of the Autocorrelation plot is shown in Figure 2 with initial conditions of $\alpha = 0, \beta = 0$. Furthermore, we checked that the effective sample size [1] was large enough (in the order of $\sim 8 \times 10^2$) for our given σ_α and σ_β. This means that the variance of a MC estimator based on 1×10^4 samples will be approximately equal to the variance of a MC estimator using $\sim 8 \times 10^2$ iid samples from the target distribution.

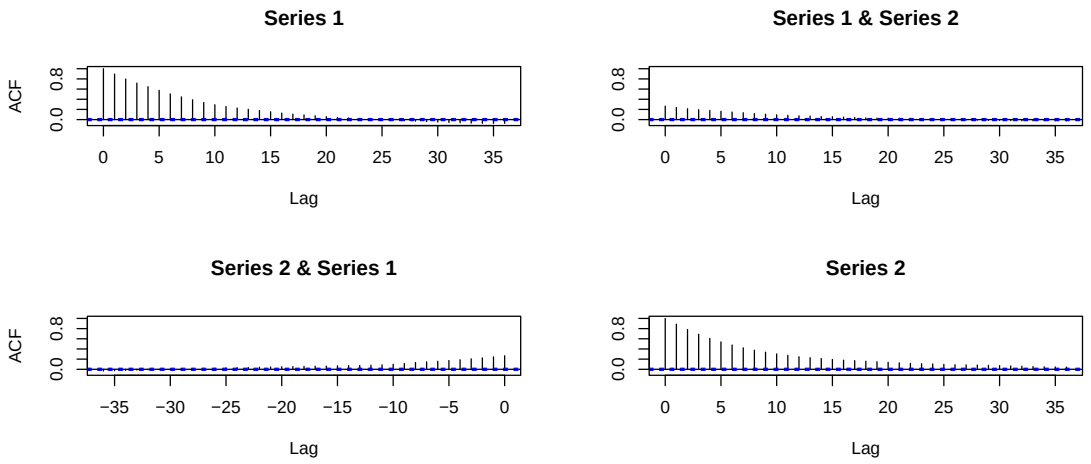

Figure 2: Autocorrelation plots for α, β (diagonal plots) and their cross terms (off-diagonal) with initial conditions with initial conditions of $\alpha = 0, \beta = 0$.

The traceplots in 1 show that our MCMC algorithm has reached convergence for both α and β. The mixing of the chains is good as seen from the ACF plot 2 since we have a quick decrease in the correlation for both parameters α and β.

Question 3

The approximate sample from the joint distribution $p(\alpha, \beta|\mathcal{D})$ is shown in Figure 3

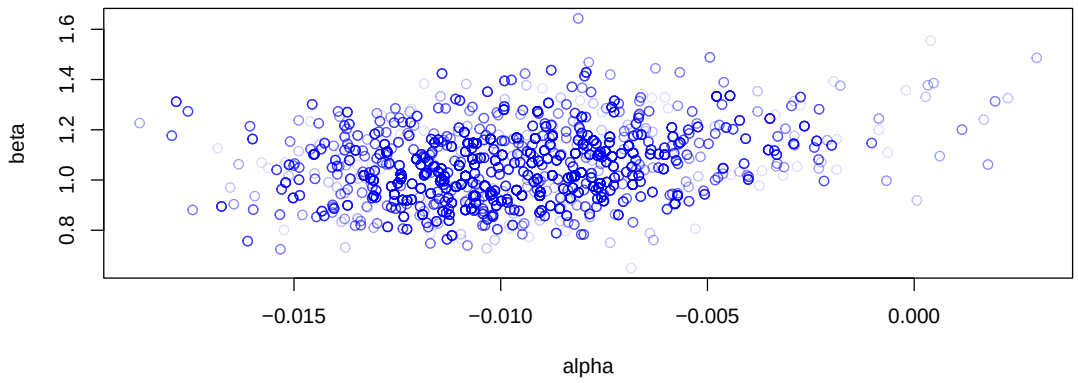

Figure 3: Scatter plot of the approximate sample from the distribution $p(\alpha, \beta|\mathcal{D})$.

Question 4

We can calculate the posterior mean of α by $E(\alpha \mid \mathcal{D}) = \int \alpha p(\alpha \mid \mathcal{D})\, d\alpha$ and similar for β. We can estimate the posterior mean using the output of our MCMC algorithm $(\alpha^t, \beta^t)_t$ in the following way:

$$E(\alpha \mid \mathcal{D}) = \int \alpha p(\alpha \mid \mathcal{D})\, d\alpha \approx \frac{1}{n - n_0} \sum_{t=n_0+1}^{n} \alpha^t \tag{3}$$

where n_0 is the burn-in period (the period we wait until the markov chain is somewhat stable). We have used a burn-in period of $n_0 = 1,000$ and $n = 10 \times 10^4$ for our calculations and obtained the following estimates: $\alpha = 0.0088, \beta = 1.0726$.

Question 5

In this question we consider the predictive distribution of y^* of the form:

$$p(y_t^* \mid \mathcal{D}, t) = \iint p(y_t^* \mid \alpha, \beta, t)\, p(\alpha, \beta \mid \mathcal{D}) d\alpha d\beta \tag{4}$$

a)

Using the α and β estimates from question 4, we can use Monte Carlo integration to obtain the Posterior Predictive Distribution since the second term in the integral in equation 4 is just the posterior distribution we calculated in question 2. Plotting this against y we obtain Figure 4.

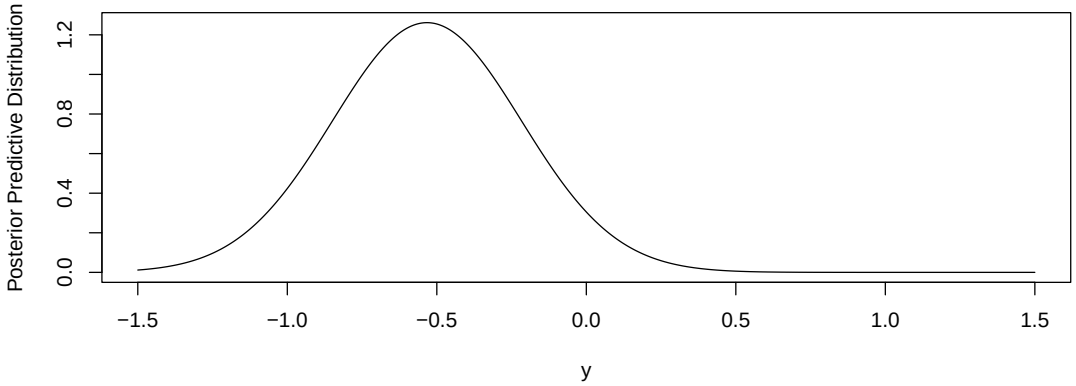

Figure 4: Posterior predictive distribution defined in equation 4, with t=55.

b)

When Y_t^* is a random variable following the distribution with pdf $p\left(y_t^* \mid \mathcal{D}, t\right)$ we can estimate $E(Y_t^*)$ by multiplying the posterior predictive distribution of $\boldsymbol{y}$ with $\boldsymbol{y}$, then divide by the number of points in $\boldsymbol{y}$ and multiply by the length of the sequence of y_is used. This is identical to taking the expected value of a variable with discrete points. This function is taking different values for each t and we plot the resulting predictive distribution in Figure 5 from $t = 1$ to $t = 60$.

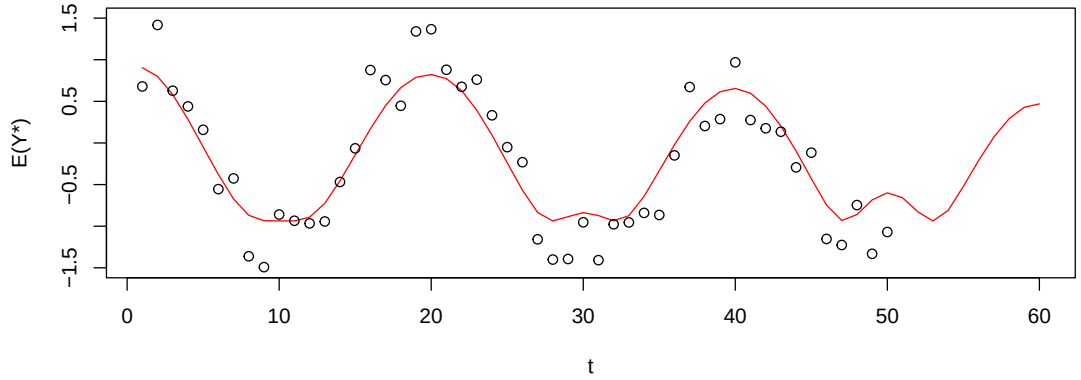

Figure 5: Posterior predictive distribution (red line) defined in equation 4 against t from from $t = 1$ to $t = 60$. The points represent the data we were given (note that we were only given data up to $t = 50$.)

Question 6

a)

Using the output of the MCMC sampler in question 2 we compute the 95% confidence interval for α using a One Sample t-test [2] and we obtain $[-0.00876 - 0.00867]$. This mean that 0 is not included in the interval which suggest that model $\mathcal{M}_1$ might be a better choice given the dataset $\mathcal{D}$. We will explore this further in the next part.

b)

In this part we implement Reversible Jump MCMC as defined the the lecture notes page 80 [1]. The algorithm used is identical to the one outlined in the notes so here I will simply define all the terms we used. The two dimensional target density $f_1\ (\mathcal{M}_1)$ is defined in 2 and the one dimensional target density is defined by:

$$p(\beta \mid \mathcal{D}) \propto g(\beta; 5) \prod_{t=1}^{50} \Phi\left(Y_t; \beta\cos(\omega t), \sigma^2\right) := f\left(\alpha, \beta\right) \tag{5}$$

We also have $\varphi_{12} = 1$ (no augmentation) and $\varphi_{21} = \Phi(x, 0, \sigma^2)$ where $\Phi(x, 0, \sigma^2)$ is the density of a normal distribution for a variable x with 0 mean and σ^2 variance. The mappings T_{mn} will simply be the identity mappings which also means that $\left|\frac{\partial T_{mn}(\theta,u)}{\partial(\theta,u)}\right| = 1$. We are also using an additive random walk so the proposals q_n cancel out because of symmetry. For the model proposal kernel we use $\kappa(1,2) = r_1 = \frac{1}{2}$ and $\kappa(2,1) = r_2 = \frac{1}{2}$.

Finally we can implement the Reversible Jump MCMC algorithm to obtain the distribution of α and β in Figure 6. The acceptance rate varies from 47% to 53% for different chains which is what we except since we have both one and two dimensional models. We calculate that the model spends 9967 iterations in $\mathcal{M}_1$ and only 33 iterations in $\mathcal{M}_1$. This gives $p(\mathcal{M}_1|\mathcal{D}) = 0.9967$. This was expected from part a) since the confidence interval suggested that it was better to include α. This means our algorithm prefers to stay in model $\mathcal{M}_1$ as long as possible which is why we only have 33 data points coming from $\mathcal{M}_2$. We use burn-in for the first 500 iterations (out of $10,000$) to wait for the chains to mix well. We assess convergence by looking at the Gelman-Rubin Diagnostic [3] which gives estimates

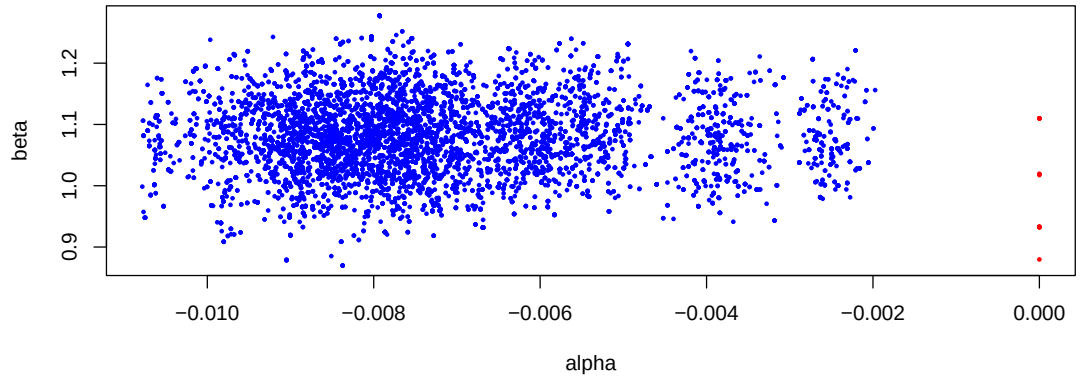

Figure 6: Scatter plot of the values for α and β as predicted from the RJMCMC algorithm. Blue dots represent points that the algorithm spent in model $\mathcal{M}_1$ and red dots represent points that the algorithm spent in model $\mathcal{M}_2$. For $\mathcal{M}_2$ we simply set the α parameter to 0 so that we can visually differentiate between the two models in the plot

of 1.07. This shows a relatively good convergence since it is close to 1. However, we notice that our algorithm is strongly dependent on the variances we use which meant a lot of fine tuning came in to play to get good convergence and this is a possible shortcoming of the implementation of the algorithm. The variances used in the model presented were $\sigma_\alpha^2 = 0.0001, \sigma_\beta^2 = 0.006, \sigma_u^2 = 0.0001$ which are used for the random walk of α, β and u respectively.

Question 7

In this question we will assume that we do not know the value of ω in equation 1 and we will try to infer it using a random walk Metropolis-Hasting algorithm sampling for $p(\omega|\alpha, \beta, \mathcal{D})$. We will use the α and β values we found in question 4.

The posterior $p(\omega|\alpha, \beta, \mathcal{D})$ will be of the form:

$$p(\omega \mid \alpha, \beta, \mathcal{D}) \propto g(\omega; 0, \pi) \prod_{t=1}^{50} \Phi\left(Y_t; \alpha t + \beta \cos(\omega t), \sigma^2\right) := f(\omega) \tag{6}$$

where all the definitions are the same as equation 2 except of $g(\omega; 0, \pi)$ which in our case is the uniform prior distribution between 0 and π.

We alter algorithm 1 to the 1 dimensional case since the only unknown parameter of interest now is ω. Since ω is taking positive values only we can take $\omega' = \omega \exp(\gamma)$ with $\gamma \sim \mathcal{N}\left(0, \sigma^2\right)$ with σ^2 the variance of this random walk for our proposal.

Running the algorithm for $\sigma^2 = 0.001$ we obtain the traceplot shown in figure 7

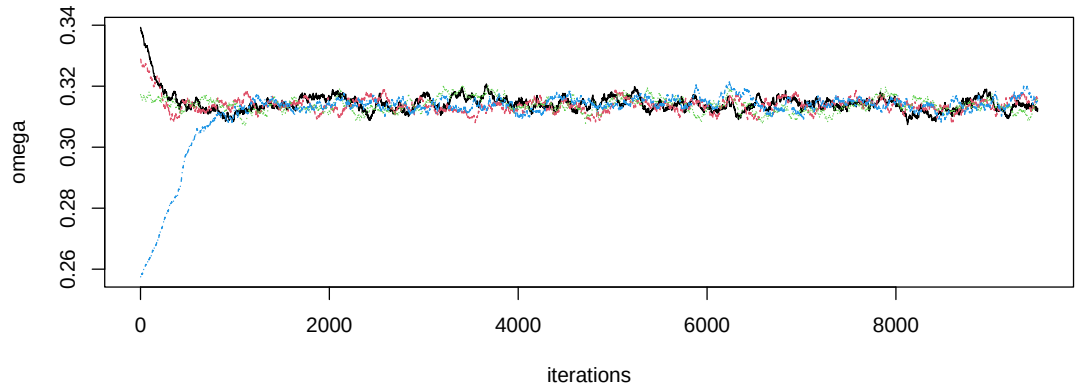

Figure 7: Traceplot for omega found using MCMC Metropolis Hasting algorithm for density 6 with 4 different chains with different initial conditions

We can see that the chains mix well and we have a quick convergence. We can see that the mixing is good from the Autocorrelation plots in Figure 8. In particular we compare in

Figure 8 the mixing between 2 different values for the variance of the proposal and we see that the mixing of the chains depends on the variance, with smaller variance giving better mixing.

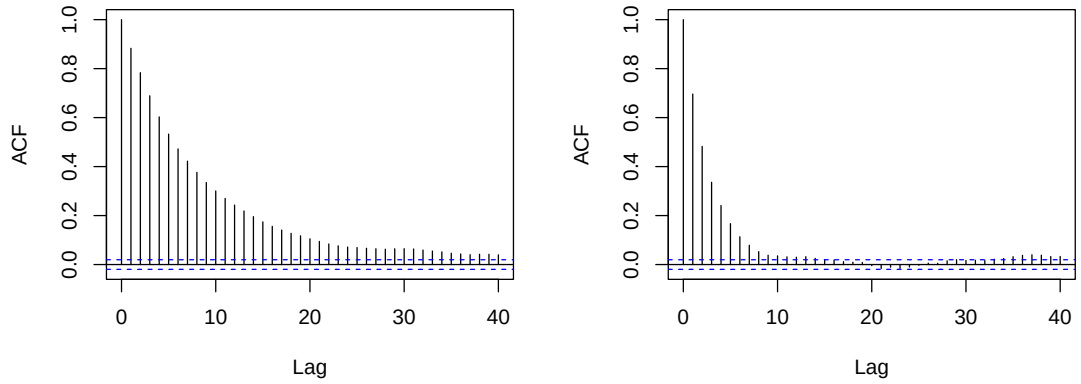

Figure 8: Example of Autocorrelation plot for the MCMC algorithm with posterior distribution 6 with $\sigma^2 = 0.01$ (right) and $\sigma^2 = 0.001$ (left)

We can also assess the convergence of the chains using the Gelman-Rubin Diagnostic [3]. Running different simulations we obtain a diagnostic of 1.02(4) for $\sigma^2 = 0.001$, 1.0 for $\sigma^2 = 0.001$ and 1.17(23) for $\sigma^2 = 0.005$ (the bracket next to the number indicates the 95% confidence interval). This shows that if our variance is too small, we do not have convergence since the optimal value of the diagnostic is 1. The performance of a simple random-walk Metropolis Hasting algorithm is not optimal since we have the parameter ω taking only positive values so most of the moves would be rejected. In contrast the model used produces a 78% acceptance rate whereas the simple Normal Linear Random Walk had an acceptance rate of only 24% showing that a lot more computational power was used to show the convergence of the estimate to 0.3145.

References

[1] Dr Sarah Filippi. Lecture notes. *Imperial College London*, page 69, October 2021.

[2] Wikipedia. Student's t-test — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Student's%20t-test&oldid=1055830420, 2021. [Online; accessed 09-December-2021].

[3] Andrew Gelman and Donald B. Rubin. Inference from Iterative Simulation Using Multiple Sequences. *Statistical Science*, 7(4):457 – 472, 1992. doi: 10.1214/ss/1177011136. URL https://doi.org/10.1214/ss/1177011136.

A Code for all Questions

```
dat <- read.csv("01389741.csv")

y <- dat$y
t <- dat$t
omega <- pi/10

# Q2
f <- function(theta, omega=pi/10) {
  sum(dnorm(y, theta[1]*t + theta[2]*cos(omega*t), sd=sqrt(0.1), log=T)) +
    dt(theta[1], 5, log=TRUE) + dt(theta[2], 5, log=TRUE)
  }

sigma_a = 0.01
```

```
sigma_b = 0.8
rho = 0


rq <- function(x) {
  rmvnorm(1, mean=c(x[1],x[2]), sigma=matrix(
  c(sigma_a^2, rho*sigma_a*sigma_b,  rho*sigma_a*sigma_b, sigma_b^2), 2
))}
q <- function(x,y) {
  dmvnorm(c(y[1],y[2]), mean=c(x[1], x[2]), sigma=matrix(c(sigma_a^2,
          rho*sigma_a*sigma_b,  rho*sigma_a*sigma_b, sigma_b^2),2),log=TRUE)
}

set.seed(222)
my_MH <- function(n, init){
  xall <- matrix(NA,nrow=2,ncol=n+1)
  x <- init
  # print(x)
  xall[,1] <- x
  for(i in 1:n){
    y <- rq(x)
    # print(q(y,x)/q(x,y))
    if (runif(1)<exp(f(y) - f(x))*(q(y,x)/q(x,y))){
      x <- y
    }
    xall[,i+1] <- x
  }
  return(xall)
}
N = 1e4

my_chains_a <- matrix(rep(NA,(N+1)*4), nrow = N+1, ncol = 4)
my_chains_b <- matrix(rep(NA,(N+1)*4), nrow = N+1, ncol = 4)
count <- 0
stat_values = list(c(0,0), c(-0.05, 0.4), c(0.05, 1.2), c(0.03, 1.5))
for (init in stat_values){
  count = count +1
  set.seed(222)
  xall <- my_MH(N,init)
  cat('Chain ', count, 'ESS: ', effectiveSize(t(xall)), '\n')
  print(dim(xall))
  acf(t(xall))
  my_chains_a[,count] <- xall[1,]
  my_chains_b[,count] <- xall[2,]
  par(mfrow=c(1,1))
  plot(xall[1,500:10000], xall[2,500:10000],
  col=rgb(red=0, green=0, blue=1, alpha=0.14),
       xlab='alpha', ylab= 'beta')
  cat('Finished stat value ', count, '\n')

}

par(mfrow=c(1,2))
plot(my_chains_a[,1],type='l', ylim=c(-0.06, 0.06), ylab='alpha',
     xlab='iterations')
for (i in 2:4){
  print(i)
```

```
  lines(my_chains_a[,i], col=i)
}


plot(my_chains_b[,1],type='l', ylim=c(-0.5, 2), ylab='beta', xlab='iterations')
for (i in 2:4){
  print(i)
  lines(my_chains_b[,i], col=i)
}

post <- data.frame(my_chains_a[,2], my_chains_b[,2])
colnames(post) <- c('alpha','beta')

#Q4
ahat <- mean(post$alpha[-seq(1000)])
bhat <- mean(post$beta[-seq(1000)])

#Q5
t=55
p_y_star <- function(y,t){
  mean(dnorm(y,mean=ahat*t +bhat*cos(omega*t),  sd=sqrt(0.1)))
}

all_y <- seq(-1.5,1.5,by=0.005)
t=55
post_y_star <- sapply(all_y, p_y_star, t)
print(post_y_star)
par(mfrow=c(1,1))
plot(all_y, post_y_star, type='l',
     ylab='Posterior Predictive Distribution', xlab='y')


#b

e_y_star <- function(y,t){
  (y*dnorm(y,mean=ahat*t +bhat*cos(omega*t), sd=sqrt(0.1)))
}

eew <- function(t){
  ys <- seq(-1.5,1.5,by=0.05)
  all_p <- c()
  count <- 0
  e_y <- sapply(ys, e_y_star, t)
  return(3*sum(e_y)/length(ys))
}

all_t <- seq(1,60,1)
emetos <- sapply(all_t, eew)
# plot(all_t,y)
plot(all_t, emetos, col='red', type='l', ylim=c(-1.5, 1.5),
     ylab='E(Y*)', xlab='t')
points(y)

#Q6
#a
t.test(post$alpha,conf.level = 0.95)
#b
```

```
f2 <- function(theta, omega=pi/10) {
  sum(dnorm(y,  theta[1]*cos(omega*t), sd=sqrt(0.1), log=TRUE)) +
    dt(theta[1], 5, log=TRUE)
}

RJ <- function(N,r1,r2, init, s1, s2, s3 ){
  n_acc <- 0 #acceptance rate

  #M2
  pos <- c(1, init) #Starting value; first component: dimension-1
  path <- list()
  for (i in 1:N){

    if (pos[1]==1){ # M2
      if (runif(1)>r1){
        # this is beta
        x <- pos[2]
        #simple 1d proposal
        y <- rnorm(1, pos[2], s2)
        if (runif(1) <= min(exp(f2(y) - f2(x)),1)){
          n_acc <- n_acc+1
          x<-y
        }
        pos[2] <- x

      }else{ ##trans-dimensional move
        # new param (alpha) proposal
        u <- rnorm(1, mean=0, sd= s3)
        # proposal state
        y <- c(u, pos[2])
        phi <- dnorm(u,0, s3)
        alpha <- min(1, (exp(f(y) - f2(pos[2])))*(r2/r1)*(1/phi))
        if (runif(1)<alpha){
          n_acc <- n_acc+1
          pos <- c(2,u,pos[2])
        }
      }
    }else{ # M1

      if (runif(1)>r2){
        # now like question 2
        sigma_a <- s1
        sigma_b <- s2
        y <- rq(pos[2:3])
        x <- c(pos[2], pos[3]) # prev state RV

        if (runif(1) <= min(exp(f(y) - f(x)),1)){
          n_acc <- n_acc+1
          x<-y}
        pos[c(2,3)] <- x

      }else{
        y <- pos[3]
        # alpha, beta
        x <- c(pos[2], pos[3])
        phi <- dnorm(y, 0, s3 )
```

```
        alpha <- min(1, (r1/r2)*(exp(f2(y) - f(x)))*(phi))

        if (runif(1)<alpha)
          n_acc <- n_acc+1

          pos <- c(1,y)
      }
    }
    path[[length(path)+1]] <- pos
  }
  print(n_acc/N)
  return(path)
}

set.seed(222)
path <- RJ(1e4,0.5,0.5,1,0.0001,0.08, 0.0009)

#set the unknown aalpha to just 0.
pos <- sapply(path, function(x) if(x[1] == 1) c(0,x[2]) else x[-1])
par(mfrow=c(1,1))
plot(pos[1,], pos[2,], pch=16, xlab="alpha",ylab="beta", cex=0.5,
     col= ifelse(pos[1,] >= -0.0001, "red", 'blue'))

#proportion in Model 2
sum(pos[1,] >= -0.0001)/N

set.seed(222)
stat_values <- c(1,0.5,0.8,0.3)
chains <- lapply(stat_values, function(i) mcmc(RJ(
1e4,0.5,0.5,i,0.0001,0.08, 0.0009)[-(1:500)]))
chains <- mcmc.list(chains)
gelman.diag(chains)

#Q7

rm(list=ls())  #clear loadspace
dat <- read.csv("01389741.csv")

y <- dat$y
t <- dat$t

ahat <- -0.008788731
bhat <- 1.072566


ssss <- seq(0,pi,0.001)
set.seed(222)
f <- function(theta) {
  # print(theta*t)
  my_thingy <- sum(dnorm(y, ahat*t + bhat*cos(theta*t),
        sd=sqrt(0.1), log=TRUE)) + dunif(theta, 0, pi, log=TRUE)
  # return(exp(my_thingy))
  return(my_thingy)
  }
plot(ssss,sapply(ssss, f), type='l')

q <- function(x,y) dnorm(log(y),log(x),sdprop)/y
```

```
rq <- function(x) x*exp(rnorm(1,0,sdprop))

my_MH <- function(n, init){
  xall <- matrix(NA,nrow=1,ncol=n+1)
  x <- init
  n_acc <- 0 #acceptance rate
  # print(x)
  xall[,1] <- x
  for(i in 1:n){
    y <- rq(x)
    # print(q(y,x)/q(x,y))
    if (runif(1)<exp(f(y) - f(x))*(q(y,x)/q(x,y))){
      x <- y
      n_acc <- n_acc+1
    }
    xall[,i+1] <- x
  }
  print(n_acc/n)
  return(xall)
}

#both work
sdprop = 0.005
# sdprop = 0.01
set.seed(222)
par(mfrow=c(1,2))
N = 1e4
xall0 <- my_MH(N,0.4)
xall1 <- my_MH(N,0.38)
xall2 <- my_MH(N,0.33)
xall3 <- my_MH(N,0.31)

# acf(t(xall3), main='')
# acf(t(xall2), main='')

par(mfrow=c(1,1))
plot(xall0[1,], type='l', ylim=c(00.25,0.4), xlab='iterations', ylab='omega')
lines(xall1[1,], col=2)
lines(xall2[1,], col=3)
lines(xall3[1,], col=4)

set.seed(222)
stat_values <- c(0.4,0.38,0.33,0.23)
chains <- lapply(stat_values, function(i) mcmc(my_MH(N,i)[-(1:500)]))
traceplot(chains, xlab='iterations', ylab='omega')
chains <- mcmc.list(chains)

gelman.diag(chains)

effectiveSize(t(xall1))
par(mfrow=c(1,2))
acf(t(xall1), main='')
acf(t(xall2), main='')

gelman.plot(chains)
```