MATH48122 - Coursework 4

Student ID: 10876957

Question 1

```r
data <- read.table("angina.txt", header = FALSE) #read the data in R

prob=function(y,t,a,b)
{
((exp(a+b*t)/(1+exp(a+b*t)))**y)*((1/(1+exp(a+b*t)))**(1-y))
}
angina=function(y,t,sigma,tau,N) #t - cholesterol level, sigma - proposal sd, tau - prior sd.
{
alpha=0
beta=0
output=matrix(ncol=2,nrow=N)
for(i in 1:N)
{
alphanew=rnorm(1,alpha,sigma)
betanew=rnorm(1,beta,sigma)
prior=(exp(-(alphanew**2+betanew**2)/(2*tau**2)))/(exp(-(alpha**2+beta**2)/(2*tau**2)))
ratio=0
ratio=prod(prob(y,t,alphanew,betanew)/prob(y,t,alpha,beta))
accept=ratio*prior
u=runif(1)
if(u<accept)
{
alpha=alphanew
beta=betanew
}
output[i,]=c(alpha,beta)
}
output
}
RWMangina=angina(data[,1], data[,2], 0.5, 100, 11000) #run the RWM algorithm
RWMangina <- RWMangina[1001:11000,] #discard the first 1000 iterations as burn-in

mean(RWMangina[,1]) #posterior mean for alpha using RWM algorithm
sd(RWMangina[,1]) #posterior sd for alpha using RWM algorithm
mean(RWMangina[,2]) #posterior mean for beta using RWM algorithm
sd(RWMangina[,2]) #posterior sd for beta using RWM algorithm

N=10000
alpha=RWMangina[,1] #samples obtained for alpha when using the RWM algorithm
beta=RWMangina[,2] #samples obtained for beta when using the RWM algorithm
cor(alpha,beta) #correlation between alpha and beta
lag1alpha=cor(alpha[1:(N-1)], alpha[2:N]) #lag-1 autocorrelation for alpha
lag1beta=cor(beta[1:(N-1)], beta[2:N]) #lag-1 autocorrelation for beta
```

Figure 1: R code for using RWM algorithm to obtain samples for α and β

First, we will write and run an RWM algorithm for the data. Then, we will find the posterior means, standard deviations, the correlation between the parameters α and β and the lag-1 autocorrelations. This is done by the code shown in Figure 1.

Code implementation: On Line 1 we read the data stored in "angina.txt" in R. On Lines 3 to 29 we use the code given for the RWM algorithm. On Line 30 we apply the algorithm to the data, where we set y as the first column vector in the data file, since it represents whether or not the patient has angina and t as the second column vector in the data file, giving the patients' cholesterol level. On Line 31 we discard the first 1000 iterations as a burn-in period. On Lines 33 and 34 we calculate the posterior mean and standard deviation for α and on Lines 35 and 36 we do the same for the parameter β. Since the output for α and β is of length 10000, we define N as this number on Line 38. On Lines 39 and 40, we define two vectors to store the output samples for α and β, respectively, for convenience. On Line 41 we calculate the correlation between the two parameters, and on Lines 42 and 43, we calculate the lag-1 autocorrelation for each of them.

Running the code gives us the output shown in Figure 2:

```
1  > mean(RWMangina[,1]) #posterior mean for alpha using RWM algorithm
2  [1] -1.897168
3  > sd(RWMangina[,1]) #posterior sd for alpha using RWM algorithm
4  [1] 1.026537
5  > mean(RWMangina[,2]) #posterior mean for beta using RWM algorithm
6  [1] 0.2790834
7  > sd(RWMangina[,2]) #posterior sd for beta using RWM algorithm
8  [1] 0.20285
9  > cor(alpha,beta) #correlation between alpha and beta
10 [1] -0.9310947
11 > lag1alpha #lag-1 autocorrelation for alpha
12 [1] 0.9840607
13 > lag1beta #lag-1 autocorrelation for beta
14 [1] 0.9605878
```

Figure 2: R code for running the RWM algorithm

The calculations we need are therefore shown in Table 1:

posterior means $(\hat{\alpha}, \hat{\beta})$	posterior sds $(\hat{\sigma_\alpha}, \hat{\sigma_\beta})$	$\hat{\rho} = cor(\alpha, \beta)$	$cor(\alpha_0, \alpha_1)$	$cor(\beta_0, \beta_1)$
$(-1.897168, 0.2790834)$	$(1.026537, 0.20285)$	-0.9310947	0.9840607	0.9605878

Table 1: Posterior summary statistics after running the RWM algorithm

Question 2

In this question, we will reparameterise the model by standardising the predictor c_j. This is done by setting the predictor variable to have zero mean and standard deviation, equal to 1. In order to obtain a standardised variable c_j^{new}, we should subtract the mean of c_j from c_j and then divide by its standard deviation. Therefore, after the transformation the new variable is the one shown in Equation 1:

$$c_j^{new} = \frac{c_j - mean(c_j)}{sd(c_j)} \tag{1}$$

In our example, a transformation of just centering the predictor variable c_j may not be enough. The first reason is the high negative correlation between the parameters α and β /-0.9310947 from Question 1/. That means that when one of the parameters increases, the other decreases. Therefore, our MCMC algorithm may have difficulty exploring the parameter space effectively, since it extends over the line of this negative correlation and may lead to poor mixing due to their different scales. The other reason is the high lag-1 autocorrelation for α and β /approximately 1 from Question 1/. This implies that the samples obtained from the MCMC algorithm are highly correlated with each other and this may cause problems in the convergence. Hence, centering may still lead to slow convergence, whereas standardising the variable would lead to a decrease in the posterior correlation between the parameters and would be more efficient.

If α^{new} and β^{new} are the parameters in the reparameterised model and α and β are the ones in the original model, then the relationship between them is presented in Equations 2 and 3:

$$\alpha^{new} = \alpha + \beta \cdot \bar{c} \tag{2}$$

$$\beta^{new} = \beta \cdot sd(c_j) \tag{3}$$

Question 3

Now, we will run the RWM algorithm for the reparameterised model by the code shown in Figure 3:

```
cj <- data[,2] #extracting the vector of the predictor variable from the data
cjscaled <- (cj-mean(cj))/sd(cj) #standardising

RWMnew=angina(data[,1], cjscaled, 0.5, 100, 11000) #run the RWM algorithm for the
    reparameterised model
RWMnew <- RWMnew[1001:11000,] #discard the first 1000 iterations as burn-in

mean(RWMnew[,1]) #posterior mean for alpha for the reparameterised model
sd(RWMnew[,1]) #posterior sd for alpha for the reparameterised model
mean(RWMnew[,2]) #posterior mean for beta for the reparameterised model
sd(RWMnew[,2]) #posterior sd for beta for the reparameterised model

N=10000
alphanew=RWMnew[,1] #samples obtained for alpha for the reparameterised model
betanew=RWMnew[,2] #samples obtained for beta for the reparameterised model
cor(alphanew,betanew) #correlation between alpha and beta for the reparamterised model
lag1alphanew=cor(alphanew[1:(N-1)], alphanew[2:N]) #lag-1 autocorrelation for alpha for the
    reparameterised model
lag1betanew=cor(betanew[1:(N-1)], betanew[2:N]) #lag-1 autocorrelation for beta for the
    reparameterised model
```

Figure 3: R code for running the RWM algorithm for the reparameterised model

After running the code in Figure 3, we get the output shown in Figure 4:

```
> mean(RWMnew[,1]) #posterior mean for alpha using the reparameterised RWM algorithm
[1] -0.5805086
> sd(RWMnew[,1]) #posterior sd for alpha using the reparameterised RWM algorithm
[1] 0.3767833
> mean(RWMnew[,2]) #posterior mean for beta using the reparameterised RWM algorithm
[1] 0.5027479
> sd(RWMnew[,2]) #posterior sd for beta using the reparameterised RWM algorithm
[1] 0.3911134
> cor(alphanew,betanew) #correlation between alpha and beta for the reparamterised model
[1] -0.009286376
> lag1alphanew #lag-1 autocorrelation for alpha for the reparameterised model
[1] 0.7768222
> lag1betanew #lag-1 autocorrelation for beta for the reparameterised model
[1] 0.7704139
```

Figure 4: R code for running the RWM algorithm for the reparameterised model

Putting these in Table 2, we get:

posterior means $(\hat{\alpha}, \hat{\beta})$	posterior sds $(\hat{\sigma_\alpha}, \hat{\sigma_\beta})$	$\hat{\rho} = cor(\alpha, \beta)$	$cor(\alpha_0, \alpha_1)$	$cor(\beta_0, \beta_1)$
$(-0.5805086, 0.5027479)$	$(0.3767833, 0.3911134)$	-0.009286376	0.7768222	0.7704139

Table 2: Posterior summary statistics after running the RWM algorithm for the reparameterised model

Comparing these results with the ones obtained from the original model in Table 1, we observe that the performance of the reparameterised model is better, because there is a very low correlation between the parameters α and β, whereas in the original model this correlation was a strong negative one. Moreover, the autocorrelations for α and β are also quite lower in the new model. The standard deviation of α shows a decrease, indicating that the accuracy of the estimates increases.

Now, let us do a visual inspection of diagnostic plots. We will have a look at the traceplots, autocorrelation and correlation plots of the original and the reparameterised model. These can be produced after running the code in Figure 5:

```
plot.ts(RWMangina) #traceplot for the original model
plot.ts(RWMnew) #traceplot for the reparameterised model

acf(RWMangina) #autocorrelation plot for the original model
acf(RWMnew) #autocorrelation plot for the reparameterised model

plot(alpha,beta) #correlation plot for alpha and beta in the original model
plot(alphanew,betanew) #correlation plot for alpha and beta in the reparameterised model
```

Figure 5: R code for diagnostic plots for the original and the reparameterised model

The plots we obtain are shown in Figures 6, 7, 8, 9, 10 and 11:

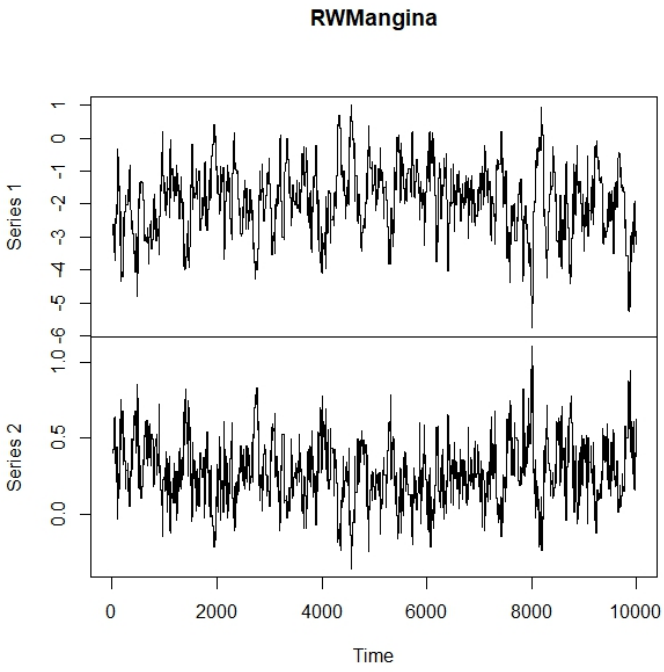

Figure 6: Traceplots for α and β - original model

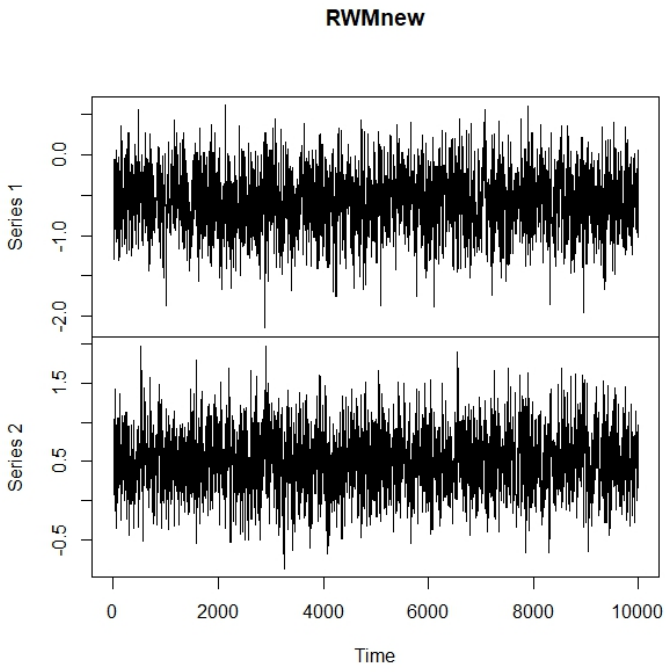

Figure 7: Traceplots for α and β - reparameterised model

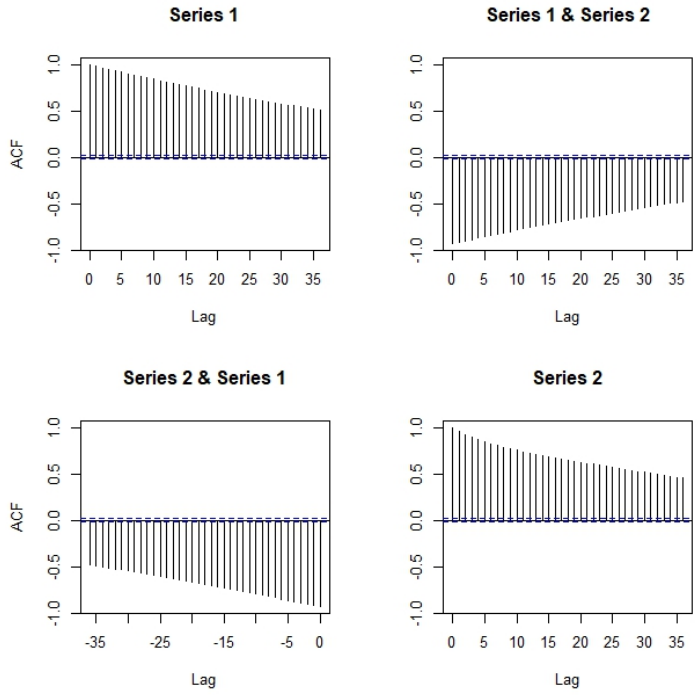

Figure 8: Acf plots for the original model

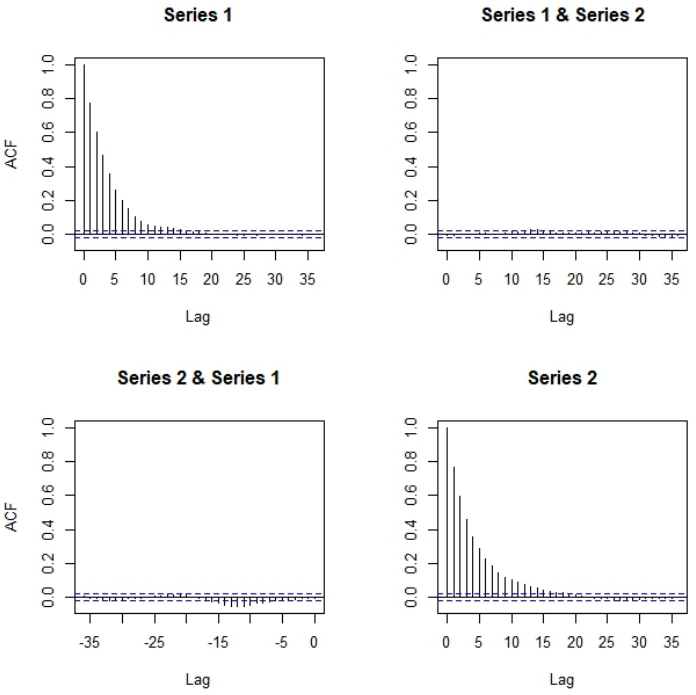

Figure 9: Acf plots for the reparameterised model

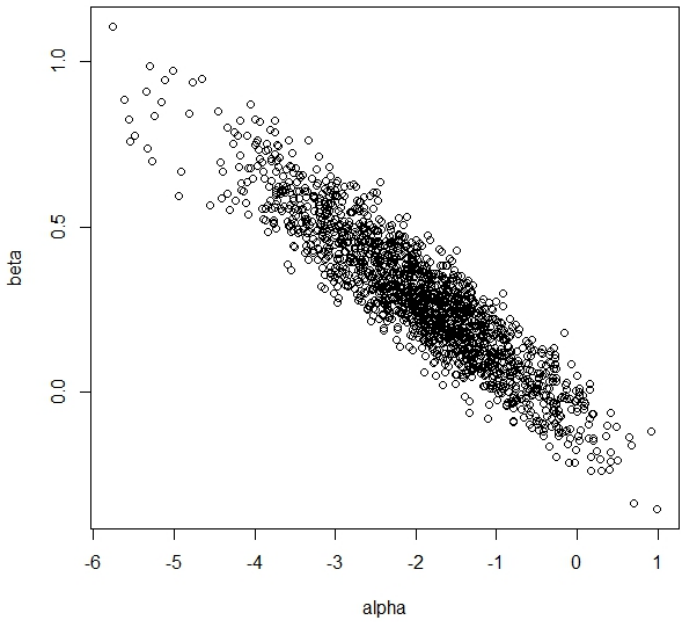

Figure 10: Correlation plot for the original model

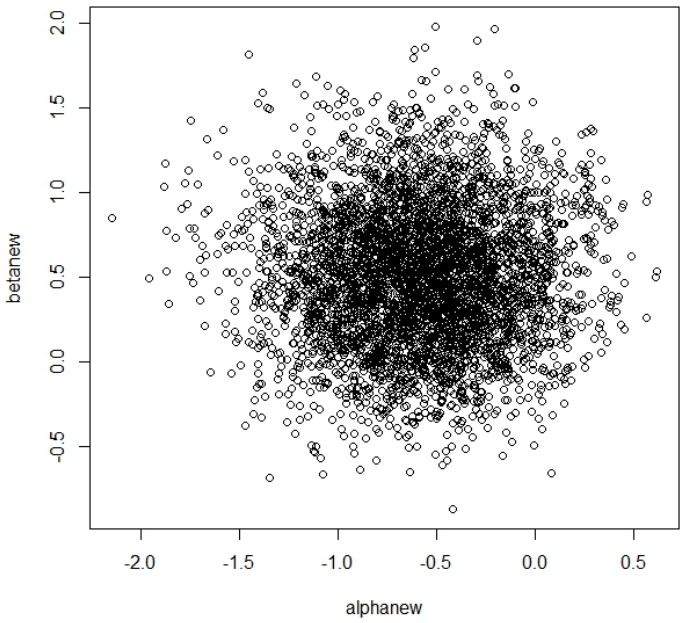

Figure 11: Correlation plot for the reparameterised model

All the figures clearly show that the reparameterised model is much better. From the traceplots in Figures 6 and 7, we observe that when running the RWM for the new model, the algorithm moves much faster around the parameter space and does not stay at one state for long. When comparing the acf plots in Figures 8 and 9, we see that the dependence between observations has vanished after approximately 10 to 15 iterations for the reparameterised model, whereas for the original one, it takes more than 35 observations for it to vanish. Finally, the correlation plot in Figures 10 shows a high negative correlation between the parameters in the original model, whereas Figure 11 indicates that there is not any obvious correlation between the parameters in the reparameterised model.

Both the posterior summary statistics and the plots we have seen indicate that the performance of the new proposed algorithm is much more efficient, regarding the convergence and correlation problems we had before in the original one.

Question 4

In order to find the partial derivatives, let us start with finding the posterior distribution $\pi^*(\alpha, \beta|\mathbf{y}, \mathbf{c})$. Since it should be up to proportionality, we can find it by Equation 4:

$$\pi^*(\alpha, \beta|\mathbf{y}, \mathbf{c}) \propto likelihood \cdot prior \tag{4}$$

Since $Y_j \sim Bernoulli\left(\frac{exp(\alpha+\beta c_j)}{1+exp(\alpha+\beta c_j)}\right)$, the likelihood function we need to use in Equation 4 is shown in Equation 5:

$$\prod_{j=1}^{35}\left[\left(\frac{exp(\alpha+\beta c_j)}{1+exp(\alpha+\beta c_j)}\right)^{y_j}\cdot\left(\frac{1}{1+exp(\alpha+\beta c_j)}\right)^{1-y_j}\right] \tag{5}$$

Since, apriori, $\alpha \sim N(0, \tau^2)$ and $\beta \sim N(0, \tau^2)$, the priors for α and β are shown in Equations 6 and 7, respectively:

$$\pi(\alpha) \propto \frac{1}{\tau} exp\left(-\frac{\alpha^2}{2\tau^2}\right) \tag{6}$$

$$\pi(\beta) \propto \frac{1}{\tau} exp\left(-\frac{\beta^2}{2\tau^2}\right) \tag{7}$$

Substituting Equations 5, 6 and 7 into Equation 4, we get Equation 8 for the posterior distribution of (α, β):

$$\pi^*(\alpha, \beta|\mathbf{y}, \mathbf{c}) \propto \prod_{j=1}^{35} \left[\left(\frac{exp(\alpha+\beta c_j)}{1+exp(\alpha+\beta c_j)}\right)^{y_j} \cdot \left(\frac{1}{1+exp(\alpha+\beta c_j)}\right)^{1-y_j}\right] \cdot \frac{1}{\tau^2} exp\left(-\frac{\alpha^2+\beta^2}{2\tau^2}\right) \tag{8}$$

Next, we will find the potential function by the formula in Equation 9:

$$\begin{aligned} &U(\alpha, \beta) = -log(\pi^*(\alpha, \beta|\mathbf{y}, \mathbf{c})) = \\ &= -\left(\sum_{j=1}^{35}[y_j(\alpha+\beta c_j) - log(1+exp(\alpha+\beta c_j))] - 2log(\tau) - \frac{\alpha^2}{2\tau^2} - \frac{\beta^2}{2\tau^2}\right) \\ &= -\left(\sum_{j=1}^{35}[y_j(\alpha+\beta c_j) - log(1+exp(\alpha+\beta c_j))]\right) + 2log(\tau) + \frac{\alpha^2}{2\tau^2} + \frac{\beta^2}{2\tau^2} \end{aligned} \tag{9}$$

Now, taking the derivatives with respect to α and β, we get Equations 10 and 11, respectively:

$$\frac{\partial U(\alpha, \beta)}{\partial \alpha} = -\left(\sum_{j=1}^{35}\left[y_j - \frac{exp(\alpha+\beta c_j)}{1+exp(\alpha+\beta c_j)}\right]\right) + \frac{\alpha}{\tau^2} \tag{10}$$

$$\frac{\partial U(\alpha, \beta)}{\partial \beta} = -\left(\sum_{j=1}^{35}\left[y_j c_j - \frac{c_j exp(\alpha+\beta c_j)}{1+exp(\alpha+\beta c_j)}\right]\right) + \frac{\beta}{\tau^2} \tag{11}$$

Therefore, we can now complete the code for the HMC algorithm, where we added the code for the potential function from Equation 9 on Line 8 in Listing 1. On Line 12 we defined the partial derivative of the potential function with respect to α from Equation 10 and on Line 16 we encoded Equation 11. Since we needed to sample a varying ϵ from the uniform distribution $U(0.8\epsilon_0, \epsilon_0)$, we did that on Line 29 in our code. The resulting full code is shown in Listing 1 below:

```
1  library(MASS)
2  #HMC for the logistic model
3  HMClogit=function(y, t, tau, q.curr, epsilon0, L, M, run) #q.curr is the starting value for (
       alpha,beta)
4  {
5  n=length(y)
6  U=function(q) #q=c(alpha,beta)
7  {
8  -sum(y*(q[1]+q[2]*t)-log(1+exp(q[1]+q[2]*t)))+2*log(tau)+(((q[1])^2)/(2*(tau^2)))+(((q[2])^2)/
       (2*(tau^2)))
9  }
10 DUa=function(q)
11 {
```

```
-sum(y-((exp(q[1]+q[2]*t))/(1+exp(q[1]+q[2]*t))) ) + ((q[1])/(tau^2))
}
DUb=function(q)
{
-sum( (y*t) -((t*exp(q[1]+q[2]*t))/(1+exp(q[1]+q[2]*t))) ) + ((q[2])/(tau^2))
}
gradU=function(q){c(DUa(q),DUb(q))}
output=matrix(0,nrow=run,ncol=2)
#Hamiltonian and Leapfrog trajectories for q and p
ham=matrix(0,nrow=run,ncol=L)
patha=matrix(0,nrow=run,ncol=L)
pathb=matrix(0,nrow=run,ncol=L)
moma=matrix(0,nrow=run,ncol=L)
momb=matrix(0,nrow=run,ncol=L)
Minv=solve(M) #inverse of mass matrix M
for(i in 1:run)
{
epsilon=runif(1,(0.8)*epsilon0,epsilon0)
q = q.curr
p = mvrnorm(1,c(0,0),M)
p.curr = p
####Simulate Hamiltonian Dynamics####
p = p - epsilon * gradU(q) / 2 #half step for p
for (j in 1:L)
{
q = q + epsilon * Minv %*% p
if (j!=L){p = p - epsilon * gradU(q)}
patha[i,j]=q[1]
pathb[i,j]=q[2]
ham[i,j]=U(q)+t(p)%*%Minv%*%p/2
if (j!=L)
{
moma[i,j]=p[1]
momb[i,j]=p[2]
}
}
p = p - epsilon * gradU(q) / 2 #half step for p at end
moma[i,j]=p[1]
momb[i,j]=p[2]
####Acc/rej Metropolis step####
U.curr = U(q.curr)
K.curr = t(p.curr)%*%Minv%*%p.curr/2
U.prop = U(q)
K.prop = t(p)%*%Minv%*%p/2
if (runif(1) < exp(U.curr-U.prop+K.curr-K.prop))
{
q.curr=q
}
output[i,]=q.curr
}
return(list(output,ham,moma,momb,patha,pathb))
}
```

Listing 1: R code for the HMC algorithm

Question 5

In this question, we will apply the HMC algorithm to the data by the code shown in Figure 12. Here, we set the mass matrix M to be the identity matrix for simplicity. We also set $L = 20$ and $\epsilon_0 = 0.045$. We will also plot the traceplots to check the efficiency of the algorithm, i.e. if we have tuned the parameters well.

```
result=HMClogit(data[,1], data[,2], 100, c(-2,0), 0.045, 20, diag(c(1,1)), 10000) #apply the
    HMC algorithm to the data
HMC <- result[[1]] #store the output the estimates for alpha and beta

plot.ts(HMC) #traceplots
```

Figure 12: R code for running the HMC algorithm

After applying the HMC algorithm to the data with the chosen parameters, we get the traceplots shown in Figure 13:

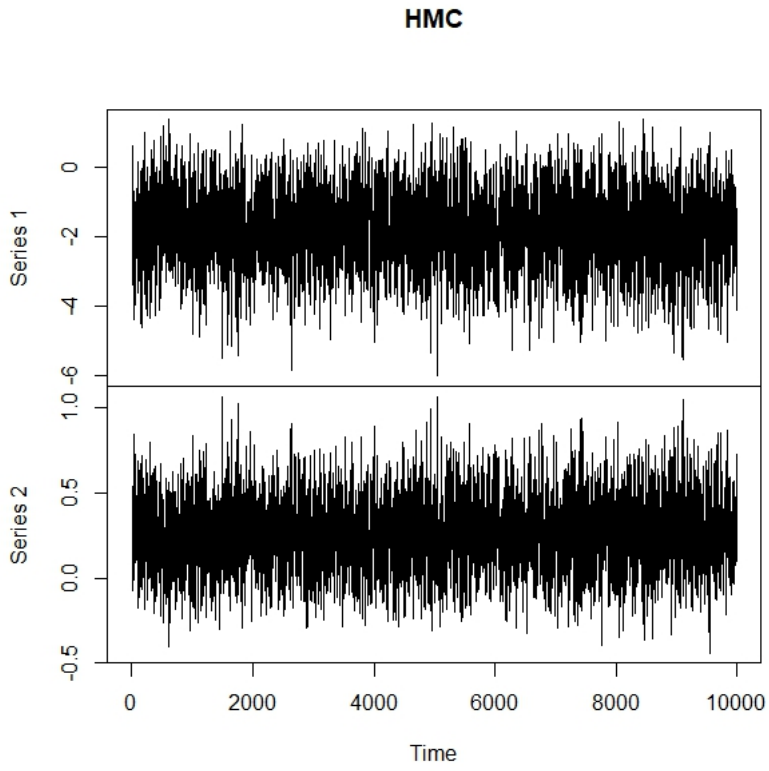

Figure 13: Trace plot

Therefore, our HMC algorithm seems to have great efficiency. Comparing it with the traceplots from the two RWM algorihtms in Figures 6 and 7, we can conclude that HMC algorithm appears to have the best performance. There is a rapid convergence and efficient exploration of the space. Now let us also compare the performance of the two RWM algorithms we used and the HMC algorithm by calculating the effective sample size. This could be done by the code in Figure 14:

```
1 library(coda)
2 effectiveSize(RWMangina)
3 effectiveSize(RWMnew)
4 effectiveSize(HMC)
```

Figure 14: R code for calculating the ESS of all the three models

The output we get is presented in Figure 15:

```
1 > effectiveSize(RWMangina)
2      var1      var2
3  97.78883 103.64478
4 > effectiveSize(RWMnew)
5     var1     var2
6 1214.392 1138.531
7 > effectiveSize(HMC)
8     var1     var2
9 1551.342 1668.188
```

Figure 15: R code output for calculating the ESS of all the three models

As we can see on Line 3 in Figure 15, the ESS for our first RWM algorithm are approximately 98 and 104 for α and β, whereas for the RWM algorithm with the parameterised model, these values are 1214 and 1139 /Line 6/. Lastly, for the HMC algorithm, ESS values are 1551 and 1668. Since the higher the ESS is, the better the accuracy is, we can conclude that the HMC algorithm is the most accurate and efficient one, whereas the first RWM algorithm we used in Question 1 is the least accurate one. The RWM algorithm, in which we reparameterised the model is still quite accurate, but not as much as the HMC one.

In conclusion, HMC is an algorithm, which can be really efficient and accurate in practice. As long as the parameters are tuned accordingly, it can even outperform other algorithms such as RWM.