Q1

```r
# read data
WOOL <- read.csv("Warpbreaks.csv")
```

```r
# model poisson regression
mod<-glm(breaks~., WOOL, family = poisson(link = "log"))
summary(mod)
```

```
##
## Call:
## glm(formula = breaks ~ ., family = poisson(link = "log"), data = WOOL)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.6871  -1.6503  -0.4269   1.1902   4.2616
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  3.17347    0.05567  57.002  < 2e-16 ***
## woolB       -0.20599    0.05157  -3.994 6.49e-05 ***
## tensionL     0.51849    0.06396   8.107 5.21e-16 ***
## tensionM     0.19717    0.06833   2.885  0.00391 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 297.37  on 53  degrees of freedom
## Residual deviance: 210.39  on 50  degrees of freedom
## AIC: 493.06
##
## Number of Fisher Scoring iterations: 4
```

```r
vcov(mod)
```

```
##              (Intercept)         woolB      tensionL      tensionM
## (Intercept)  0.003099518 -1.193312e-03 -2.564099e-03 -2.564099e-03
## woolB       -0.001193312  2.659585e-03  5.034078e-19  2.454025e-19
## tensionL    -0.002564099  5.034078e-19  4.090810e-03  2.564099e-03
## tensionM    -0.002564099  2.454025e-19  2.564099e-03  4.669354e-03
```

```r
confint(mod,level=0.995)
```

```
## Waiting for profiling to be done...
```

```
##                    0.3 %      99.8 %
## (Intercept)  3.013926429  3.32660462
## woolB       -0.351173215 -0.06152152
## tensionL     0.340192963  0.69951267
## tensionM     0.005811379  0.38973193
```

Part c

```r
# Extracting the design matrix
X <- model.matrix(mod)
```

```
# betaest<-modest$coef
sigma  <-vcov(mod)
y <- WOOL$breaks
p <-dim(X)[2]   #number of parameters
```

```
#Part one: function for performing Metropolis sampling for poisson regression normal random walk.
#Inputs:
#y: vector of responses
#n: vector (or scalar) of trial sizes.
#X: predictor matrix including intercept.
#c: rescaling for variance-covariance matrix, scalar J(lambda*|lambda(t-1)) = N(lambda(t-1), c^2*Sigma)
#Sigma is variance covariance matrix for parameters in J()
#iter: number of iterations
#burnin: number of initial iterations to throw out.
Metropolis.fn<-function(y,X,c,Sigma,iter,burnin){
p <-dim(X)[2]   #number of parameters
library(mvtnorm)
beta0<-rnorm(p) #initial values.
beta.sim<-matrix(0,iter,p) #matrix to store iterations
beta.sim[1,]<-beta0
for(i in 1:(iter-1)){
beta.cand <-rmvnorm(1,mean=beta.sim[i,],sigma=c^2*Sigma) #draw candidate (jointly)
beta.cand <-as.numeric(beta.cand)
xbc       <-X%*%beta.cand
lambda.c       <-exp(xbc)   #Calculating probability of success for candidates.
xb        <-X%*%beta.sim[i,]
lambda.b       <-exp(xb)     #Calculating probability of success for lambda(t-1).
#difference of log joint distributions.
r<-sum( dpois(y,lambda.c,log=TRUE)-dpois(y,lambda.b ,log=TRUE) )
#Draw an indicator whether to accept/reject candidate
ind<-rbinom(1,1,exp( min(c(r,0)) ) )
beta.sim[i+1,]<- ind*beta.cand + (1-ind)*beta.sim[i,]
}

#Removing the iterations in burnin phase
results<-data.frame(beta.sim[-c(1:burnin),])
names(results)<-c('beta0','beta1','beta2','beta3') #column names
return(results)
}
```

```
library(coda)
```

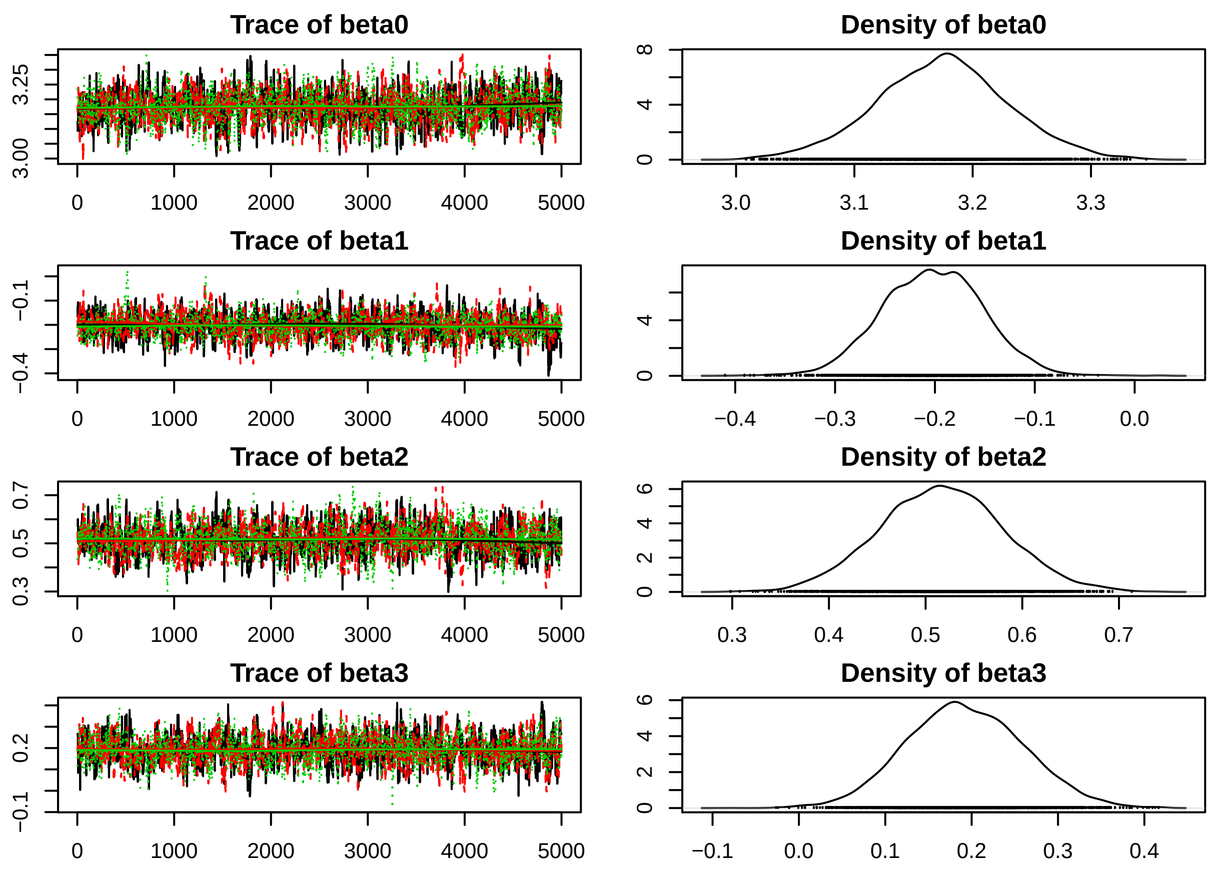
```
## Warning: package 'coda' was built under R version 3.5.2
```

```
# c =1.6
par(mar=c(2,2,2,2))
results16_1 <- Metropolis.fn(y,X,c=1.6/sqrt(p),Sigma = sigma,iter=10000,burnin =5000)
```

```
## Warning: package 'mvtnorm' was built under R version 3.5.2
```

```
results16_2 <- Metropolis.fn(y,X,c=1.6/sqrt(p),Sigma = sigma,iter=10000,burnin =5000)
results16_3 <- Metropolis.fn(y,X,c=1.6/sqrt(p),Sigma = sigma,iter=10000,burnin =5000)
c16_1 <- mcmc(results16_1[c('beta0','beta1','beta2','beta3')])
c16_2 <- mcmc(results16_2[c('beta0','beta1','beta2','beta3')])
c16_3 <- mcmc(results16_3[c('beta0','beta1','beta2','beta3')])
c16 <- mcmc.list(c16_1,c16_2,c16_3)
```

```r
plot(c16)
```



```r
# Acceptance Rate
1 - rejectionRate(c16)
```

```
##     beta0     beta1     beta2     beta3
## 0.4659599 0.4659599 0.4659599 0.4659599
```

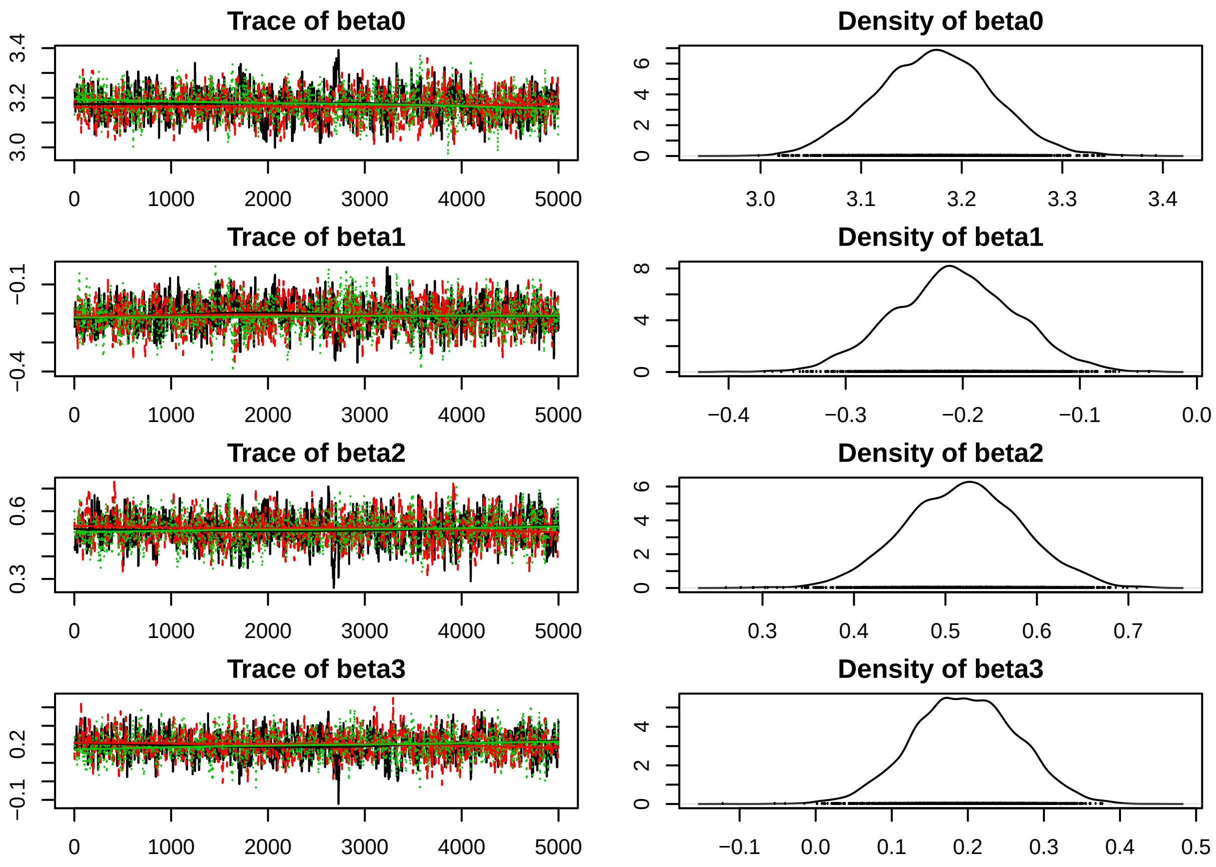
```r
# Gelman and Rubin diagnostic
gelman.diag(c16)
```

```
## Potential scale reduction factors:
##
##       Point est. Upper C.I.
## beta0       1.00       1.00
## beta1       1.01       1.01
## beta2       1.00       1.01
## beta3       1.00       1.00
##
## Multivariate psrf
##
## 1.01
```

```r
# Effective sample size
effectiveSize(c16)
```

```
##    beta0    beta1    beta2    beta3
## 939.5267 971.9721 917.5420 969.9256
```

```
# c = 2.4
par(mar=c(2,2,2,2))
results24_1 <- Metropolis.fn(y,X,c=2.4/sqrt(p),Sigma = sigma,iter=10000,burnin =5000)
results24_2 <- Metropolis.fn(y,X,c=2.4/sqrt(p),Sigma = sigma,iter=10000,burnin =5000)
results24_3 <- Metropolis.fn(y,X,c=2.4/sqrt(p),Sigma = sigma,iter=10000,burnin =5000)
c24_1 <- mcmc(results24_1[c('beta0','beta1','beta2','beta3')])
c24_2 <- mcmc(results24_2[c('beta0','beta1','beta2','beta3')])
c24_3 <- mcmc(results24_3[c('beta0','beta1','beta2','beta3')])
c24 <- mcmc.list(c24_1,c24_2,c24_3)
plot(c24)
```



```
# Acceptance Rate
1 - rejectionRate(c24)
```

```
##     beta0     beta1     beta2     beta3
## 0.2945923 0.2945923 0.2945923 0.2945923
```

```
# Gelman and Rubin diagnostic
gelman.diag(c16)
```

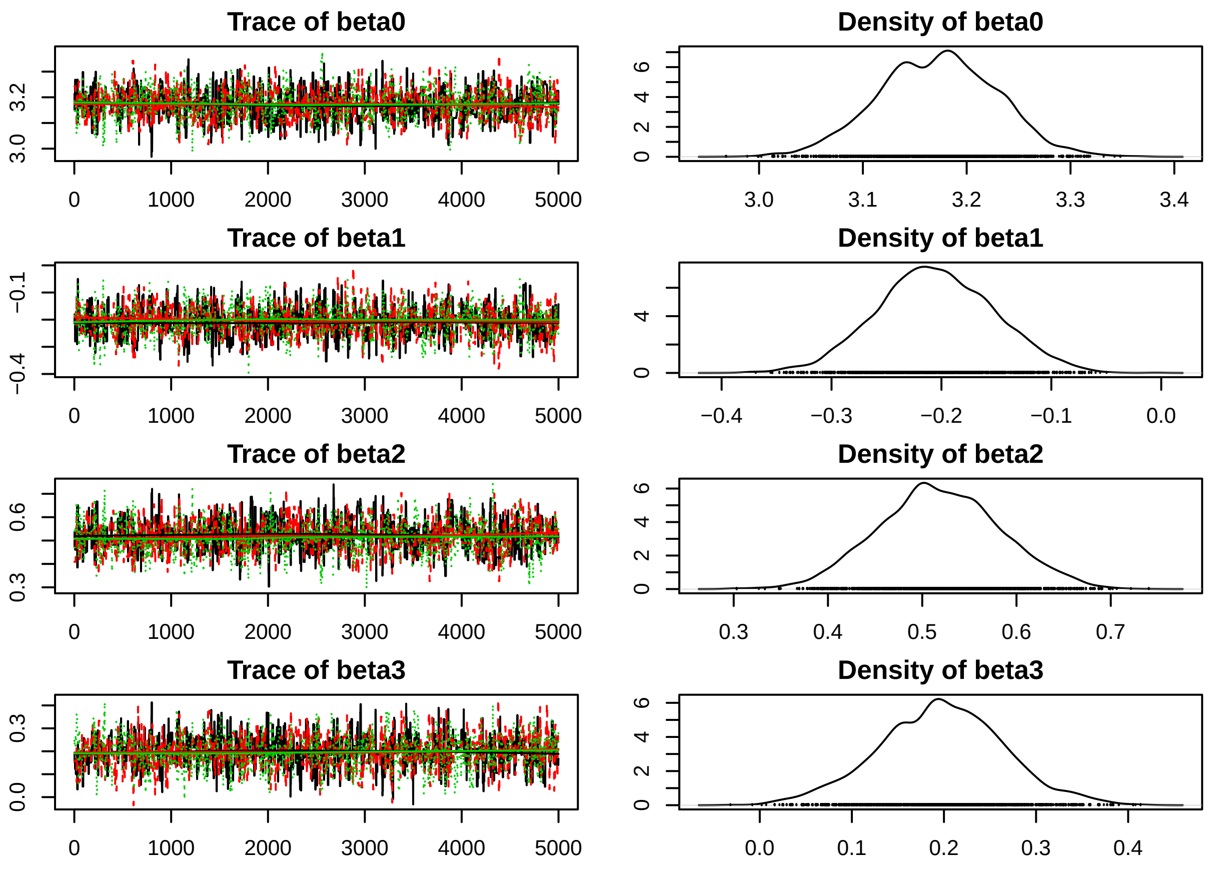
```
## Potential scale reduction factors:
##
##       Point est. Upper C.I.
## beta0       1.00       1.00
## beta1       1.01       1.01
## beta2       1.00       1.01
## beta3       1.00       1.00
##
```

```
## Multivariate psrf
##
## 1.01
```

```
# Effective sample size
effectiveSize(c24)
```

```
##    beta0    beta1    beta2    beta3
## 1048.558 1104.525 1132.244 1059.152
```

```
# c = 3.2
par(mar=c(2,2,2,2))
results32_1 <- Metropolis.fn(y,X,c=3.2/sqrt(p),Sigma = sigma,iter=10000,burnin =5000)
results32_2 <- Metropolis.fn(y,X,c=3.2/sqrt(p),Sigma = sigma,iter=10000,burnin =5000)
results32_3 <- Metropolis.fn(y,X,c=3.2/sqrt(p),Sigma = sigma,iter=10000,burnin =5000)
c32_1 <- mcmc(results32_1[c('beta0','beta1','beta2','beta3')])
c32_2 <- mcmc(results32_2[c('beta0','beta1','beta2','beta3')])
c32_3 <- mcmc(results32_3[c('beta0','beta1','beta2','beta3')])
c32 <- mcmc.list(c32_1,c32_2,c32_3)
plot(c32)
```



```
# Acceptance Rate
1 - rejectionRate(c32)
```

```
##     beta0     beta1     beta2     beta3
## 0.1899713 0.1899713 0.1899713 0.1899713
```

```
# Gelman and Rubin diagnostic
gelman.diag(c16)
```

```
## Potential scale reduction factors:
##
##       Point est. Upper C.I.
## beta0       1.00       1.00
## beta1       1.01       1.01
## beta2       1.00       1.01
## beta3       1.00       1.00
##
## Multivariate psrf
##
## 1.01
```

```
# Effective sample size
effectiveSize(c32)
```

```
##    beta0    beta1    beta2    beta3
## 1087.530 1034.897 1132.564 1165.365
```

```
# comments
AcceptanceRate <- c(1 - rejectionRate(c16),1 - rejectionRate(c24),1 - rejectionRate(c32))
EffectiveSampleSize <- c(effectiveSize(c16),effectiveSize(c24),effectiveSize(c32))
rown <- c("c=1.6 beta0","c=1.6 beta1","c=1.6 beta2","c=1.6 beta3",
          "c=2.4 beta0","c=2.4 beta1","c=2.4 beta2","c=2.4 beta3",
          "c=3.2 beta0","c=3.2 beta1","c=3.2 beta2","c=3.2 beta3")
summarychain <- data.frame(AcceptanceRate,EffectiveSampleSize,row.names = rown)
summarychain
```

```
##             AcceptanceRate EffectiveSampleSize
## c=1.6 beta0      0.4659599            939.5267
## c=1.6 beta1      0.4659599            971.9721
## c=1.6 beta2      0.4659599            917.5420
## c=1.6 beta3      0.4659599            969.9256
## c=2.4 beta0      0.2945923           1048.5580
## c=2.4 beta1      0.2945923           1104.5254
## c=2.4 beta2      0.2945923           1132.2441
## c=2.4 beta3      0.2945923           1059.1521
## c=3.2 beta0      0.1899713           1087.5298
## c=3.2 beta1      0.1899713           1034.8968
## c=3.2 beta2      0.1899713           1132.5639
## c=3.2 beta3      0.1899713           1165.3651
```

As we increase c the Acceptance rate decreases. With a larger c we have that the jumping distribution can draw samples from a larger range as seen from the density plots above.

For the case c=2.4/sqrt(p) the is reported to be 0.2903247. This is in line with the results from the lectures as

```
# c=1.6/sqrt(p)
grc16 <- gelman.diag(c16)
grc16$psrf
```

```
##       Point est. Upper C.I.
## beta0   1.001018   1.003546
## beta1   1.005790   1.013343
## beta2   1.002093   1.005700
## beta3   1.000863   1.001044
```

```
# c=2.4/sqrt(p)
grc24 <- gelman.diag(c24)
```

```r
grc24$psrf
```

```
##       Point est. Upper C.I.
## beta0   1.001464   1.004556
## beta1   1.002813   1.006706
## beta2   1.000879   1.002053
## beta3   1.003362   1.010520
```

```r
# c=3.2/sqrt(p)
grc32 <- gelman.diag(c32)
grc32$psrf
```

```
##       Point est. Upper C.I.
## beta0   1.002898   1.005915
## beta1   1.001826   1.007033
## beta2   1.003060   1.011108
## beta3   1.001769   1.001854
```