Steven Maharaj 695281 Assignment 2, Question 1

Due: Friday 20 September 2019

There are places in this assignment where R code will be required. Therefore set the random seed so assignment is reproducible.

```
set.seed(695281)  #Please change random seed to your student id number.
```

Question One (12 marks)

In generalised linear models, rather than estimating effects from the response data directly, we model through a link function, $\eta(\boldsymbol{\theta})$, and assume $\eta(\boldsymbol{\theta})_i = \mathbf{x}_i'\boldsymbol{\beta}$. The link function can be determined by re-arranging the likelihood of interest into the exponential family format,

$$p(y|\boldsymbol{\theta}) = f(y)g(\boldsymbol{\theta})e^{\eta(\boldsymbol{\theta})'u(y)}. \tag{1}$$

a) Re-arrange the Poisson probability mass function into the exponential family format to determine the canonical link function. The Poisson pmf is

$$Pr(y|\lambda) = \frac{\lambda^y e^{-\lambda}}{y!}.$$

Answer:

We have that the Poisson pmf is

$$\begin{aligned} Pr(y|\lambda) &= \frac{\lambda^y e^{-\lambda}}{y!} \\ &= \frac{1}{y!} e^{y\log(\lambda)} e^{-\lambda} \end{aligned}$$

Hence $f(y) = \frac{1}{y!}, u(y) = y, g(\lambda) = e^{-\lambda}$ and the link function

$$\eta(\lambda) = \log(\lambda).$$

To explore some properties of Metropolis sampling, consider the dataset `Warpbreaks.csv`, which is on LMS. This dataset contains information of the number of breaks in a consignment of wool. In addition, Wool type (A or B) and tension level (L, M or H) was recorded.

b) Fit a Poisson regression to the warpbreak data, with Wool type and tension treated as factors using the function glm in R. Report co-efficient estimates and the variance-covariance matrix.

Answer:

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

```
# 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
```

```
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
```

```
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
```

c) Fit a Bayesian Poisson regression using Metropolis sampling. Assume flat priors for all coefficients. Extract the design matrix $\mathbf{X}$ from the `glm` fitted in a). For the proposal distribution, use a Normal distribution with mean θ^{t-1} and variance-covariance matrix $c^2\hat{\mathbf{\Sigma}}$ where $\mathbf{\Sigma}$ is the variance-covariance matrix from the glm fit. Consider three candidates for c, $1.6/\sqrt{p}, 2.4/\sqrt{p}, 3.2\sqrt{p}$, where p is the number of parameters estimated. Run the Metropolis algorithm for 10,000 iterations, and discard the first 5,000. Report the following:

- Check, using graphs and appropriate statistics, that each chain converges to the same distribution. To do this, you may find installing the R package `coda` helpful.
- The proportion of candidate draws that were accepted.
- The effective sample size for each chain.
- What do you think is the best choice for c. Does this match the results stated in class on efficiency and optimal acceptance rate?

Answer:

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

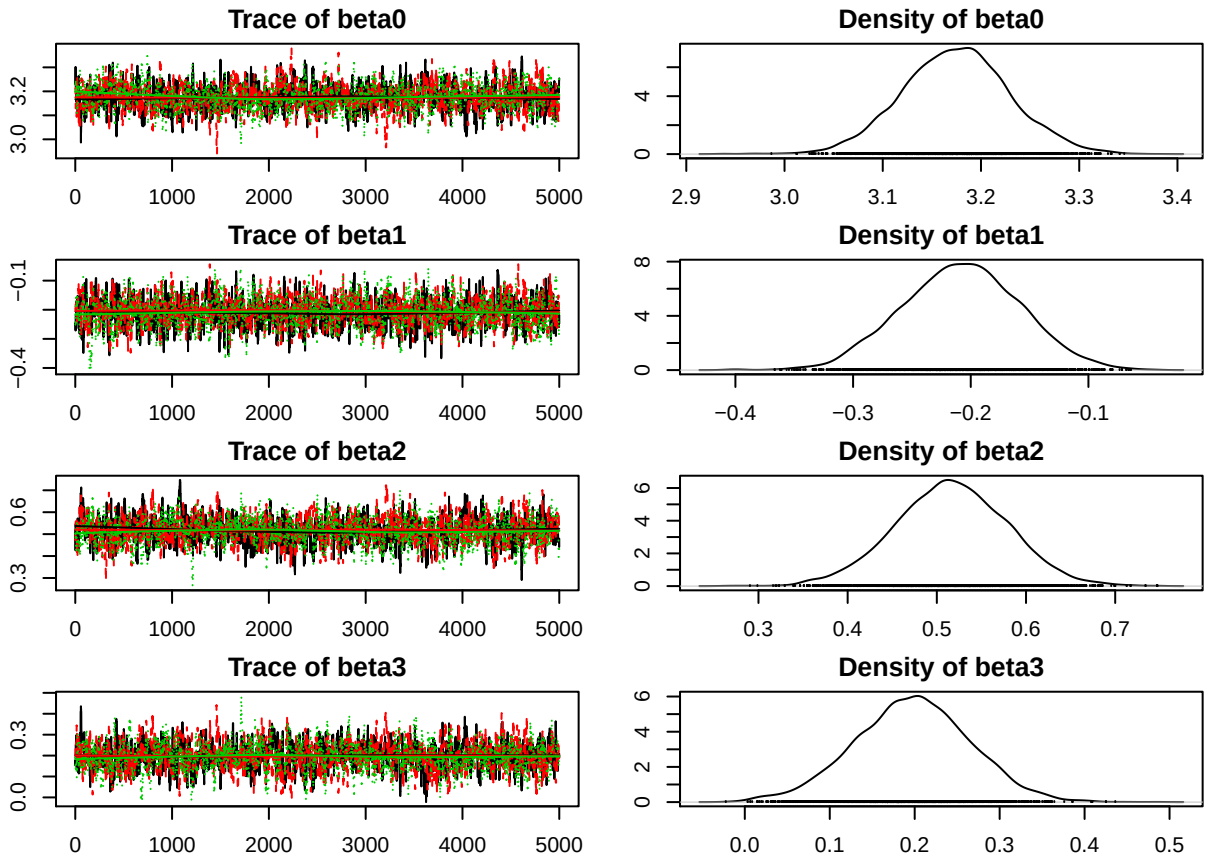
```
#Part one: function for performing Metropolis sampling for poisson regression normal random walk.
#Inputs:
#y: vector of responses
#n: vector (or scalar) of trial sizes.
#X: predictor matrix including intercept.
#c: rescaling for variance-covariance matrix, scalar J(lambda*|lambda(t-1)) = N(lambda(t-1), 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)
```

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

```
plot(c16)
```




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

```
##     beta0     beta1     beta2     beta3
## 0.4743615 0.4743615 0.4743615 0.4743615
```

```
# Gelman and Rubin diagnostic
gelman.diag(c16)[[1]]
```

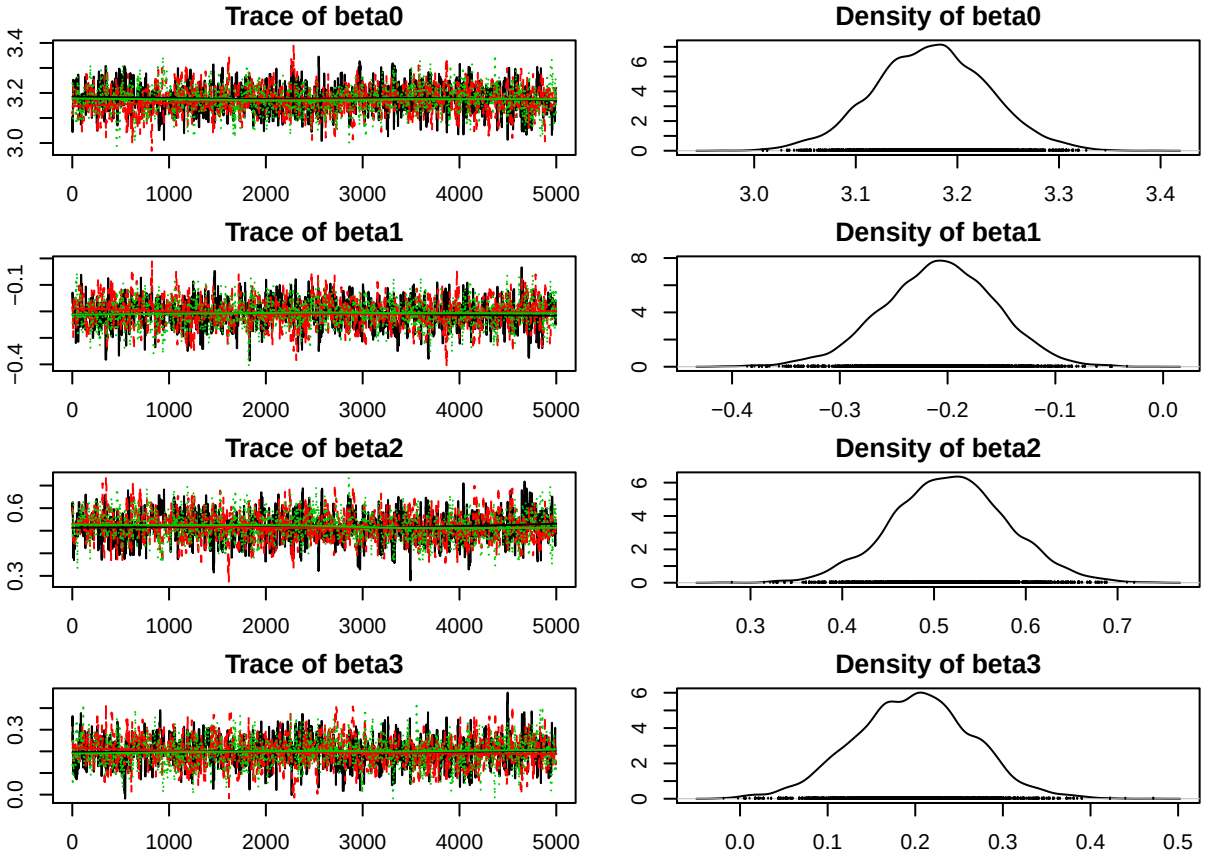
```
##       Point est. Upper C.I.
## beta0   1.002893   1.006415
## beta1   1.002969   1.007820
## beta2   1.005583   1.017900
## beta3   1.006790   1.017059
```

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

```
##     beta0     beta1     beta2     beta3
## 1003.2585 1063.4202  966.5833  944.1443
```

```
# c = 2.4/sqrt(p)
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
## 0.2993265 0.2993265 0.2993265 0.2993265
```

```
# Gelman and Rubin diagnostic
gelman.diag(c24)[[1]]
```

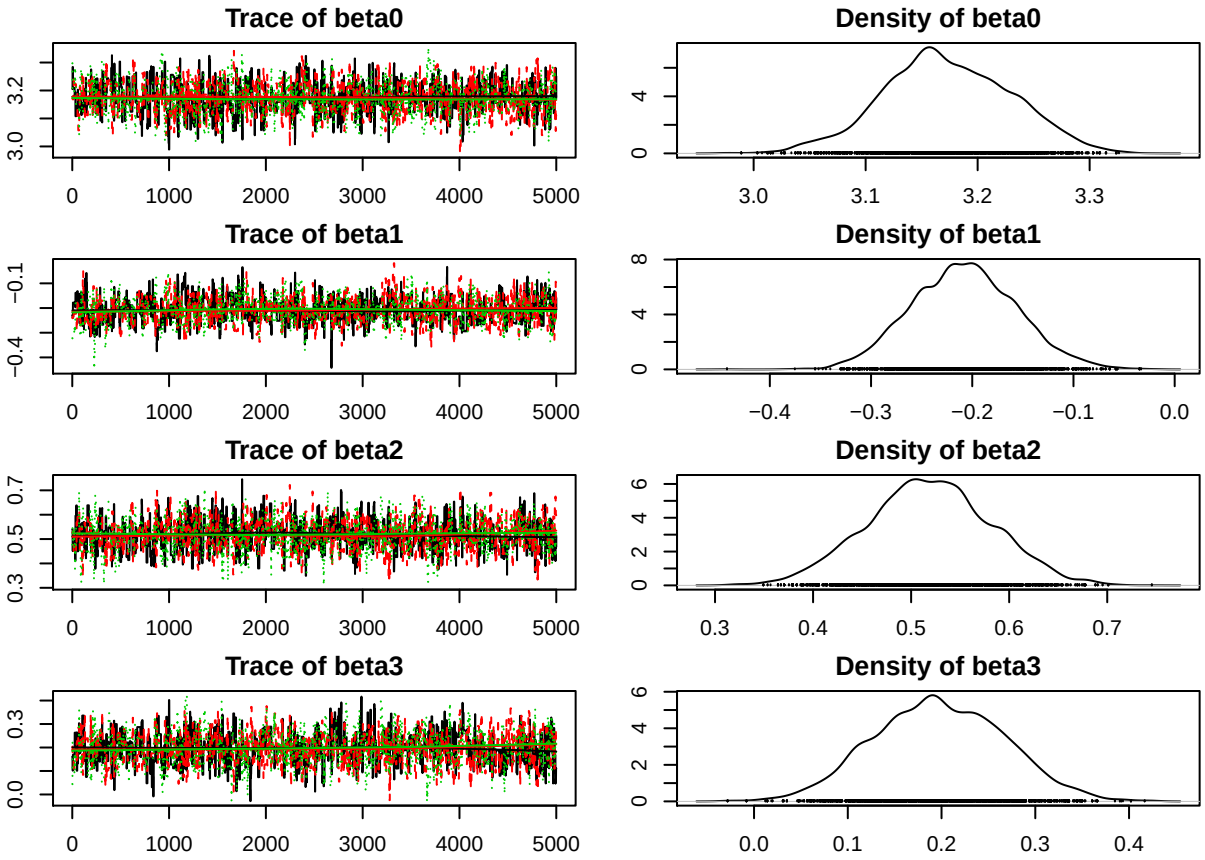
```
##       Point est. Upper C.I.
## beta0   1.002268   1.007975
## beta1   1.003960   1.013431
## beta2   1.001135   1.002572
## beta3   1.002112   1.006229
```

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

```
##    beta0    beta1    beta2    beta3
## 1113.915 1098.547 1145.363 1167.613
```

```
# c = 3.2/sqrt(p)
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
## 0.1844369 0.1844369 0.1844369 0.1844369
```

```
# Gelman and Rubin diagnostic
gelman.diag(c32)[[1]]
```

```
##       Point est. Upper C.I.
## beta0   1.007283   1.021598
## beta1   1.005338   1.011690
## beta2   1.006693   1.016116
## beta3   1.001921   1.004986
```

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

```
##     beta0     beta1     beta2     beta3
##  966.9207 1006.9051  981.5870  975.0611
```

```
# 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      0.4743615           1003.2585
## c=1.6 beta1      0.4743615           1063.4202
## c=1.6 beta2      0.4743615            966.5833
## c=1.6 beta3      0.4743615            944.1443
## c=2.4 beta0      0.2993265           1113.9150
## c=2.4 beta1      0.2993265           1098.5467
## c=2.4 beta2      0.2993265           1145.3633
## c=2.4 beta3      0.2993265           1167.6132
## c=3.2 beta0      0.1844369            966.9207
## c=3.2 beta1      0.1844369           1006.9051
## c=3.2 beta2      0.1844369            981.5870
## c=3.2 beta3      0.1844369            975.0611
```

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. However, if a very small c is choose there would be substantial auto-correlation in the posterior samples. Hence a moderate value of c (c=2.4/sqrt(p)) is the best choice.

For the case c=2.4/sqrt(p) the jumping efficiency from class is

```
p <- dim(X)[2]
0.3/p
```

```
## [1] 0.075
```

For our simulation the jumping efficiency was

```
mean(effectiveSize(c24)/(4*5000))
```

```
## [1] 0.05656798
```

Note that this is the reciprocal of the integrated auto-correlaction. This was used as a measure of efficiency by Gelman et al. [1].

And the acceptance rate from lectures is 0.44 for p=1 and 0.23 for large p. For our simulation we achieved an acceptance rate of

```
1 - rejectionRate(c24)[[1]]
```

```
## [1] 0.2993265
```

Hence, these match the results from lectures.

[1] Gelman, Andrew, Gareth O. Roberts, and Walter R. Gilks. “Efficient Metropolis jumping rules.” Bayesian statistics 5.599-608 (1996): 42.