Assignment 3 Question 1 Steven Maharaj 695281

Due: Friday 25 October 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 (17 marks)

To explore some properties of Expectation propagation and Hamiltonian Monte Carlo, consider the dataset `Warpbreaks.csv`, which is on LMS and previously analysed in assignment 2. 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. As the observed data consists of integer counts, it was assumed that a Poisson distribution should be used to model counts. The probability mass function of a Poisson distribution is

$$Pr(y_i|\lambda_i) = \frac{\lambda_i^{y_i} e^{-\lambda_i}}{y_i!}.$$

a) Assuming that the canonical link for observation i can be represented as $\mathbf{X}_i\boldsymbol{\beta}$, determine the following:

- The likelihood, $p(\mathbf{y}|\boldsymbol{\beta})$ and log-likelihood.
- The first derivative of the log-likelihood with respect to $\boldsymbol{\beta}$.

Answer:

Given n observations we have

$$p(\boldsymbol{y}|\boldsymbol{\beta}) = \prod_{i=1}^{n} \frac{\lambda_i^{y_i} e^{-\lambda_i}}{y_i!}$$

thus the log-liklihood

$$\begin{aligned}
\log(p(\boldsymbol{y}|\boldsymbol{\beta})) &= \sum_{i=1}^{n} y_i \log(\lambda_i) - \lambda_i - \log(y_i!) \\
&= \sum_{i=1}^{n} y_i \boldsymbol{X}_i\boldsymbol{\beta} - e^{\boldsymbol{X}_i\boldsymbol{\beta}} - \log(y_i!) && (\text{since } \log(\lambda) = \boldsymbol{X}_i\boldsymbol{\beta})
\end{aligned}$$

Taking the derivate

$$\frac{d\log(p(\boldsymbol{y}|\boldsymbol{\beta}))}{d\boldsymbol{\beta}_j} = \sum_{i=1}^{n} \boldsymbol{X}_{ij} y_i - \boldsymbol{X}_{ij} e^{\boldsymbol{X}_i\boldsymbol{\beta}}.$$

Thus,

$$\frac{d\log(p(\boldsymbol{y}|\boldsymbol{\beta}))}{d\boldsymbol{\beta}} = \boldsymbol{X}'(\boldsymbol{y} - e^{\boldsymbol{X}\boldsymbol{\beta}}).$$

b) If you wish to construct a Bayesian analogue to Poisson regression, what prior(s) would you use?

We choose $\boldsymbol{\beta} \sim \mathcal{N}(0, \Sigma)$ where sigma is the variance co-variance matrix of a fitted glm. Actually it should be flat

c) Fit a Poisson regression to the warpbreak data, with Wool type and tension treated as factors, using Hamiltonian Monte Carlo. To ensure identifiability, make Wool type A and tension type H the reference category. You are expected to code this in `R`, as opposed to fitting the model using `Stan`. Consider the following values for the number of leapfrog steps $L = 2, 3, 4$. Assume the momentum variable ϕ is drawn from a multivariate normal distribution with zero mean and variance-covariance matrix $5\mathbf{X}'\mathbf{X}$. Run a single chain for each choice of L for 10000 iterations, and remove 30 % of iterations as burn-in. Report the following.

- The posterior mean, standard deviations and 90 % credible intervals for all parameters, combining the results for all chains. Interpret the 90 % credible interval.
- The acceptance rate for each choice of L.

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

```
# model poisson regression
mod<-glm(breaks~ ., WOOL, family = poisson(link = "log"))
X <- model.matrix(mod)
# sigma  <-vcov(mod)
y <- WOOL$breaks
p <-dim(X)[2]   #number of parameters
M <- 5*crossprod(X)
```

```
#Part one: function for performing Hamiltonian Monte Carlo for logistic regression.
#Inputs:
#y: vector of responses
#n: vector (or scalar) of trial sizes.
#X: predictor matrix including intercept.
#L: number of leapfrog steps.
#M is variance covariance matrix for normal prior of momentum variable \phi. Ideally diagonal.
#iter: number of iterations
#burnin: number of initial iterations to throw out.
HMC.fn<-function(y,X,L,M,iter,burnin){
p <-dim(X)[2]   #number of parameters
library(mvtnorm)
theta0<-rnorm(p) #initial values of beta
theta.sim<-matrix(0,iter,p+1) #matrix to store iterations plus acceptance.
theta.sim[1,1:p]<-theta0       #initial values in matrix.
epsilon<-1/L                #epsilon assuming epsilon*L =1.
Minv   <-solve(M)


for(i in 1:(iter-1)){
phi       <-rmvnorm(1,mean=rep(0,p),sigma=M)   #draw momentum variable.
phi       <-as.numeric(phi)
phi0      <-phi                              #saving starting phi for calculation of r.
theta     <-theta.sim[i,1:p]                  #current state of theta.

lbd.b         <-exp(X%*%theta)  #calculate lambda.
gradtheta <- crossprod(X,y-lbd.b )   #Gradient of posterior = joint distribution with respect to theta.

#leapfrog steps.
for(j in 1:L){
  phi   <- phi + 0.5*epsilon*gradtheta   #first half step for phi
  theta <- theta + epsilon*(Minv%*%phi)  #full step for theta
```

```
lbd.c         <-exp(X%*%theta) #calculate probabilities of success at candidate (sub) state.
gradtheta <- crossprod(X,y-lbd.c )   #Gradient of posterior = joint distribution with respect to theta.

phi   <- phi + 0.5*epsilon*gradtheta #second half step for phi.
phi   <- as.numeric(phi)
}

#difference of log joint distributions at final iteration of leap.frog vs current state.
r<-sum( dpois(y,lambda = lbd.c,log=TRUE))+dmvnorm(phi,mean=rep(0,p),sigma=M,log=TRUE)-sum(dpois(y,lambda
#Draw an indicator whether to accept/reject candidate
ind<-rbinom(1,1,exp( min(c(r,0)) ) )
theta.sim[i+1,1:p]<- ind*theta + (1-ind)*theta.sim[i,1:p]
theta.sim[i+1,p+1] <- ind
}

#Removing the iterations in burnin phase
results<-theta.sim[-c(1:burnin),]
names(results)<-c('beta0','beta1','beta2','beta3','accept') #column names

return(results)
}
```

```
# L = 2
HMCl2<-HMC.fn(y=y,X=X,L=2,M=M,iter=10000,burnin=3000)

# L = 3
HMCl3<-HMC.fn(y=y,X=X,L=3,M=M,iter=10000,burnin=3000)

# L = 4
HMCl4<-HMC.fn(y=y,X=X,L=4,M=M,iter=10000,burnin=3000)
```

For L = 2

```
#Posterior means of beta0, beta1, beta2, beta3 Acceptance rate comparison

colMeans(HMCl2)
```

```
## [1]  3.1708593 -0.2058539  0.5199441  0.1978643  0.8324286
```

```
#Posterior standard deviations
apply(HMCl2,2,FUN=sd)
```

```
## [1] 0.05514903 0.05127605 0.06304162 0.06656290 0.37351195
```

```
#90 % credible intervals
apply(HMCl2,2,FUN=function(x) quantile(x,c(0.05,0.95)) )
```

```
##         [,1]       [,2]      [,3]      [,4] [,5]
## 5%  3.080528 -0.2897791 0.4157289 0.0861268    0
## 95% 3.261829 -0.1200403 0.6210818 0.3042288    1
```

```
# acceptance rate
colMeans(HMCl2)[5]
```

```
## [1] 0.8324286
```

For L = 3

```r
#Posterior means of beta0, beta1, beta2, beta3 Acceptance rate comparison

colMeans(HMC13)
```

```
## [1]  3.1716852 -0.2064958  0.5193036  0.1979674  0.9230000
```

```r
#Posterior standard deviations
apply(HMC13,2,FUN=sd)
```

```
## [1] 0.05701033 0.05164100 0.06314559 0.06958419 0.26661049
```

```r
#90 % credible intervals
apply(HMC13,2,FUN=function(x) quantile(x,c(0.05,0.95)) )
```

```
##         [,1]       [,2]      [,3]       [,4] [,5]
## 5%  3.078909 -0.2911735 0.4148381 0.08240899    0
## 95% 3.266748 -0.1206554 0.6212428 0.31180073    1
```

```r
# acceptance rate
colMeans(HMC13)[5]
```

```
## [1] 0.923
```

For L = 4

```r
#Posterior means of beta0, beta1, beta2, beta3 Acceptance rate comparison
colMeans(HMC14)
```

```
## [1]  3.1719214 -0.2058379  0.5189682  0.1968202  0.9552857
```

```r
#Posterior standard deviations
apply(HMC14,2,FUN=sd)
```

```
## [1] 0.05552354 0.05217846 0.06379348 0.06736974 0.20669064
```

```r
#90 % credible intervals
apply(HMC14,2,FUN=function(x) quantile(x,c(0.05,0.95)) )
```

```
##         [,1]       [,2]      [,3]       [,4] [,5]
## 5%  3.080650 -0.2937332 0.4143111 0.08504399    1
## 95% 3.261597 -0.1192360 0.6239704 0.30748666    1
```

```r
# acceptance rate
colMeans(HMC14)[5]
```

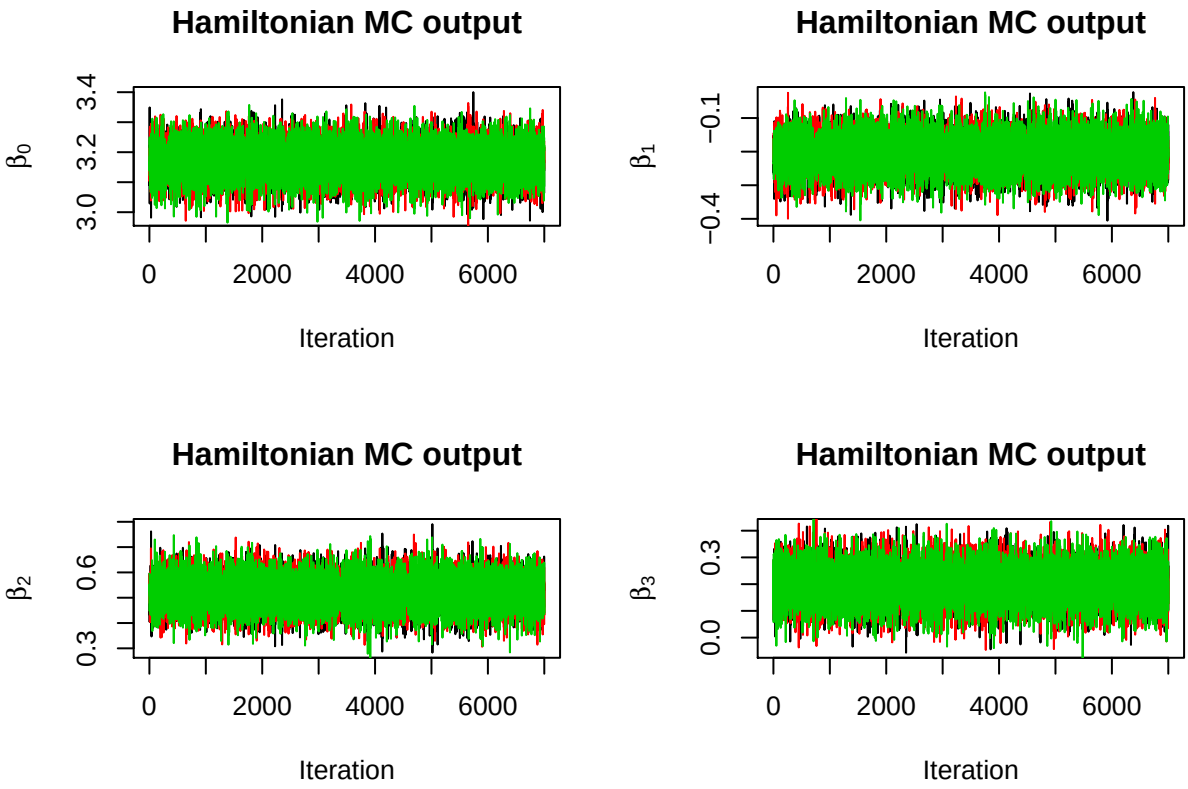
```
## [1] 0.9552857
```

The 90% credible interval tells there is 90% probability that parameter we seek to estimate is between the lower and upper bound of the given interval.

d) Check each chain obtained converged to the same distribution. For each chain and parameter, create acf plots. Based on this, what do you think was the best choice for L?

Answer: We will visually check if the chains converged to the same distribution

```r
#plotting HMC.
par(mfrow=c(2,2))
for (i in 1:4) {
plot(HMC12[,i],type='l',main='Hamiltonian MC output',xlab='Iteration',ylab=bquote( beta[.(i-1)] ))
lines(HMC13[,i],col=2)
lines(HMC14[,i],col=3)
}
```



Checking the Gelman-Rubin diagnostic.

```
library(coda)
hl1<-as.mcmc.list(as.mcmc((HMC12[1:3500,1:4])))
hl2<-as.mcmc.list(as.mcmc((HMC12[1:3500,1:4])))
hl3<-as.mcmc.list(as.mcmc((HMC13[1:3500,1:4])))
hl4<-as.mcmc.list(as.mcmc((HMC13[3500+1:3500,1:4])))
hl5<-as.mcmc.list(as.mcmc((HMC14[3500+1:3500,1:4])))
hl6<-as.mcmc.list(as.mcmc((HMC14[3500+1:3500,1:4])))
hl<-c(hl1,hl2,hl3,hl4,hl5,hl6)

#Gelman-Rubin diagnostic.
gelman.diag(hl)[[1]]
```

```
##      Point est. Upper C.I.
## [1,]  0.9998943   1.000006
## [2,]  1.0001577   1.000182
## [3,]  1.0002284   1.000370
## [4,]  1.0001417   1.000292
```

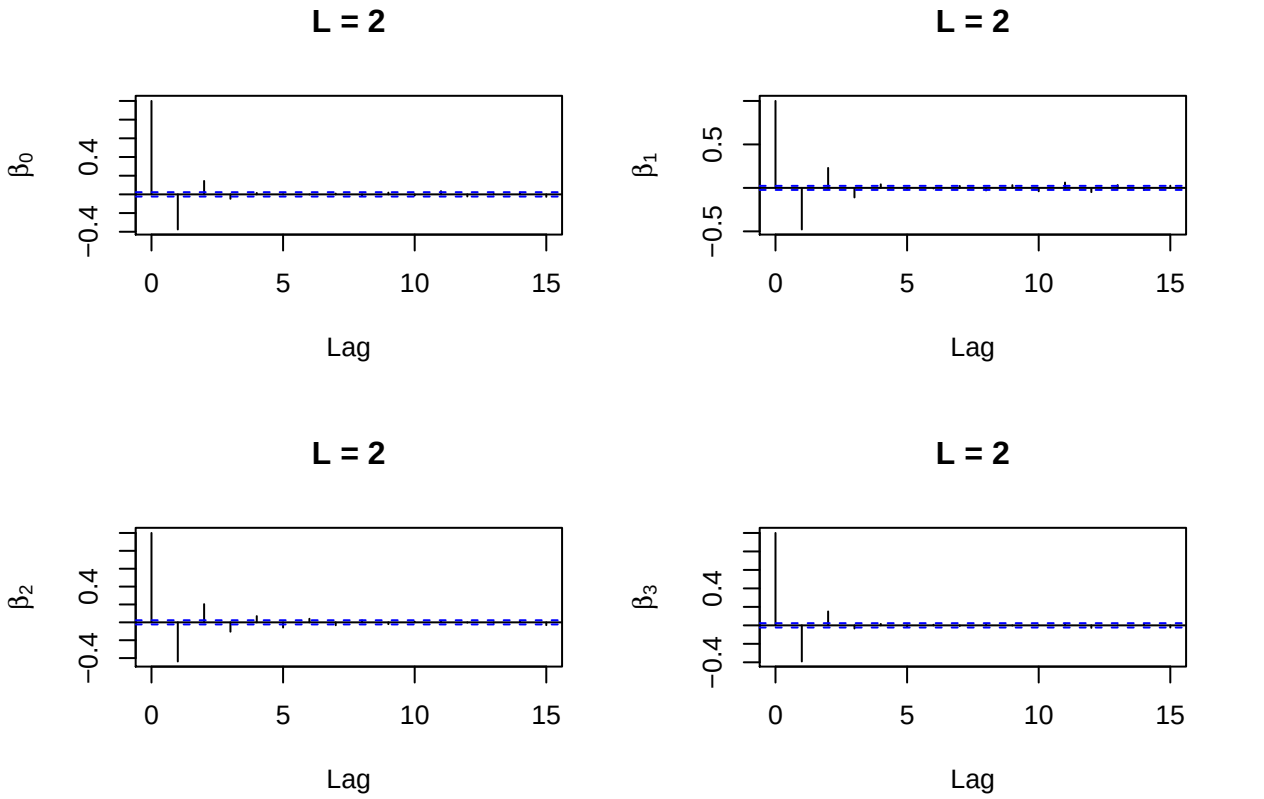
```
#effective sample size.
effectiveSize(hl)
```

```
##     var1     var2     var3     var4
## 55123.72 87829.08 67323.85 56010.16
```

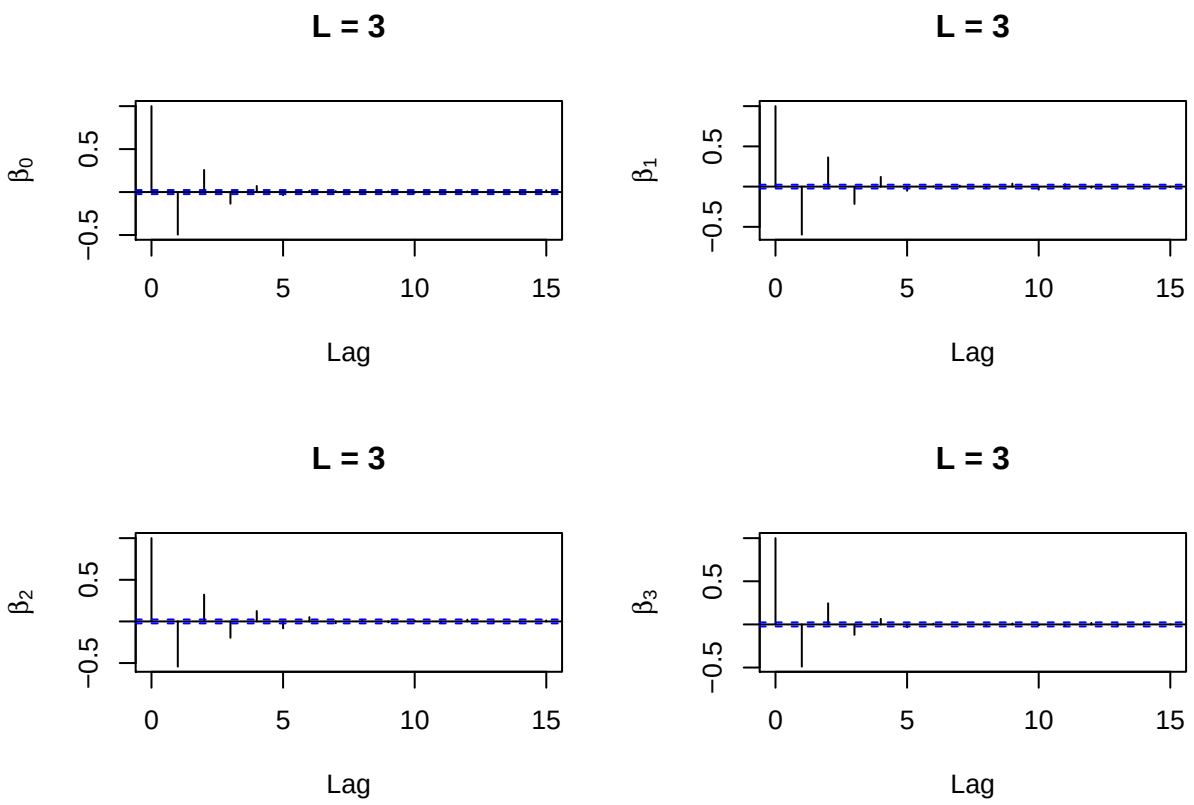
From ploting the iterations and computing the Gelman-Rubin diagnostic we see that all the chains converged to the same distribution.

Now let us check the autocorrelation in each chain using acf plots.

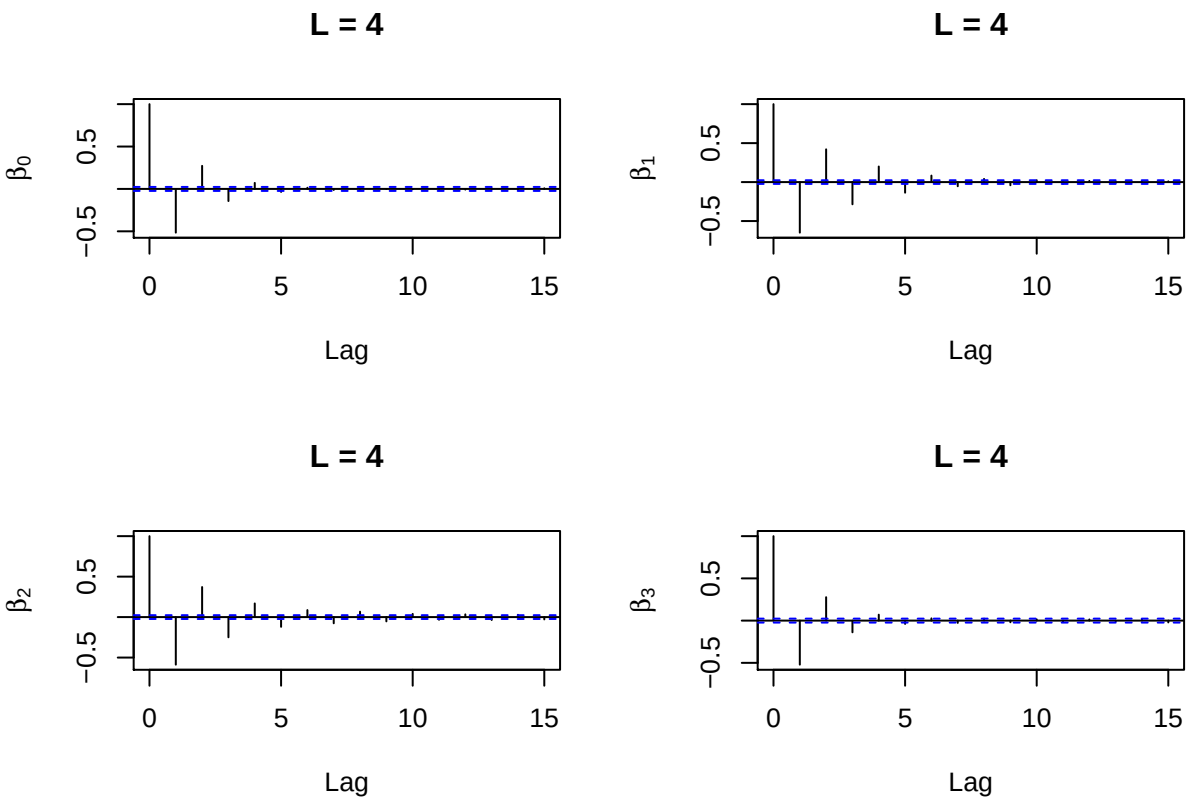
```r
# L=2
par(mfrow=c(2,2))
for (i in 1:4) {
acf(HMCl2[,i],ylab=bquote( beta[.(i-1)] ),lag.max = 15,main="L = 2")
}
```




```r
# L=3
par(mfrow=c(2,2))
for (i in 1:4) {
acf(HMCl3[,i],ylab=bquote( beta[.(i-1)] ),lag.max = 15,main="L = 3")
}
```




```
# L=4
par(mfrow=c(2,2))
for (i in 1:4) {
acf(HMCl4[,i],ylab=bquote( beta[.(i-1)] ),lag.max = 15,main="L = 4")
}
```



So from above we observe that as L increase, the acceptance rate increase and from the acf plots the autocorrelation also increases. Thus one should choose a moderate value of L. In our case one should choose L=3.

e) Fit the same model using an expectation propagation algorithm. Report the approximate posterior means, and 90 % credible interval. Comparing the results obtained using expectation propagation to Hamiltonian Monte Carlo, what 'bias' do you think has been caused by using approximate inference.

```
#Coding for implementing Expectation-propagation for poission regression.

#Arguments.
#response, response vector
#n: vector of trial sizes
#iter: number of rounds to consider
#epsilon: convergence criteria.

EP.logit<-function(response,n,X,iter,epsilon){
N<-length(response) #size of dataset.
p<-dim(X)[2]

Sigmainvmu <-matrix(0,p,N) #natural parameter \Sigma^{-1}mu
Sigmainv  <-rep(list(diag(p)),N)  #natural parameter \Sigma^{-1}, stored as list

#Previous parameters of g.
Sigmainvmu0<-rowSums(Sigmainvmu)
Sigmainv0   <-Reduce("+",Sigmainv)

#g_0(\bm \beta) need not be updated and is assumed flat.

#function for tilted distribution
```

```
tilt.dist      <- function(x){
gnoti          <-dnorm(x,mean=Mnoti,sd=sqrt(Vnoti))
# beta           <- rnorm(p,mean=Mnoti,sd=sqrt(Vnoti))
lbd              <-exp(x)
like           <-dpois(response[i],lbd)
result <-gnoti*like
return(result)
}

#loop for updating g_i(\bm \beta) i= 1, ..., n.
for(j in 1:iter){
for(i in 1:N){
Sigmainvmunoti <-rowSums(Sigmainvmu)  - Sigmainvmu[,i] #Natural parameter \Sigma_{-i}^{-1}\mu_{-i}
Sigmainvnoti   <-Reduce("+",Sigmainv) - Sigmainv[[i]]  #Natural parameter \Sigma_{-i}^{-1}\mu_{-i}
Sigmanoti      <-solve(Sigmainvnoti)                   #parameter \Sigma_{-i}^{-1}
munoti         <-Sigmanoti%*%Sigmainvmunoti            #parameter \mu_{-i}
Mnoti          <-t(X[i,])%*%munoti                     #M_{-i}
Vnoti          <-t(X[i,])%*%Sigmanoti%*%X[i,]          #V_{-i}

#Moment matching.
E0<-integrate(f= function(x) { tilt.dist(x)}, lower=Mnoti-10*sqrt(Vnoti), upper=Mnoti+10*sqrt(Vnoti))
E1<-integrate(f= function(x) {x*tilt.dist(x)}, lower=Mnoti-10*sqrt(Vnoti), upper=Mnoti+10*sqrt(Vnoti))
E2<-integrate(f= function(x) {x^2*tilt.dist(x)}, lower=Mnoti-10*sqrt(Vnoti), upper=Mnoti+10*sqrt(Vnoti))
M <- E1$value/E0$value
V <- E2$value/E0$value - M^2

#Update g_i
MiViinv <- M/V - Mnoti/Vnoti
Viinv   <- 1/V - 1/Vnoti

#transform back to beta scale.
Sigmainvmu[,i] <-X[i,]%*%MiViinv  #natural parameter \Sigma^{-1}mu
Sigmainv[[i]]  <-X[i,]%*%Viinv%*%t(X[i,])
}
#Note by the way the previous lines of code have been written, step six has been implicitly.

#Checking whether to stop iterations.
currentSinvmu <-rowSums(Sigmainvmu)
currentSinv   <-Reduce("+",Sigmainv)

  diff1  <- sqrt((currentSinvmu-Sigmainvmu0)^2)/(abs(currentSinvmu)+0.01)
  diff2  <- sqrt((currentSinv-Sigmainv0)^2)/(abs(currentSinv)+0.01)
  diff.all<-c(diff1,diff2)
  if(max(diff.all) < epsilon) break else Sigmainvmu0 <- currentSinvmu; Sigmainv0 <- currentSinv
}

#Final mean and variance-covariance matrix of g(\beta)
Sigma <-solve(currentSinv)
mu    <-Sigma%*%currentSinvmu


#Storing and returning results.
param<-list(mu,Sigma,j)
```

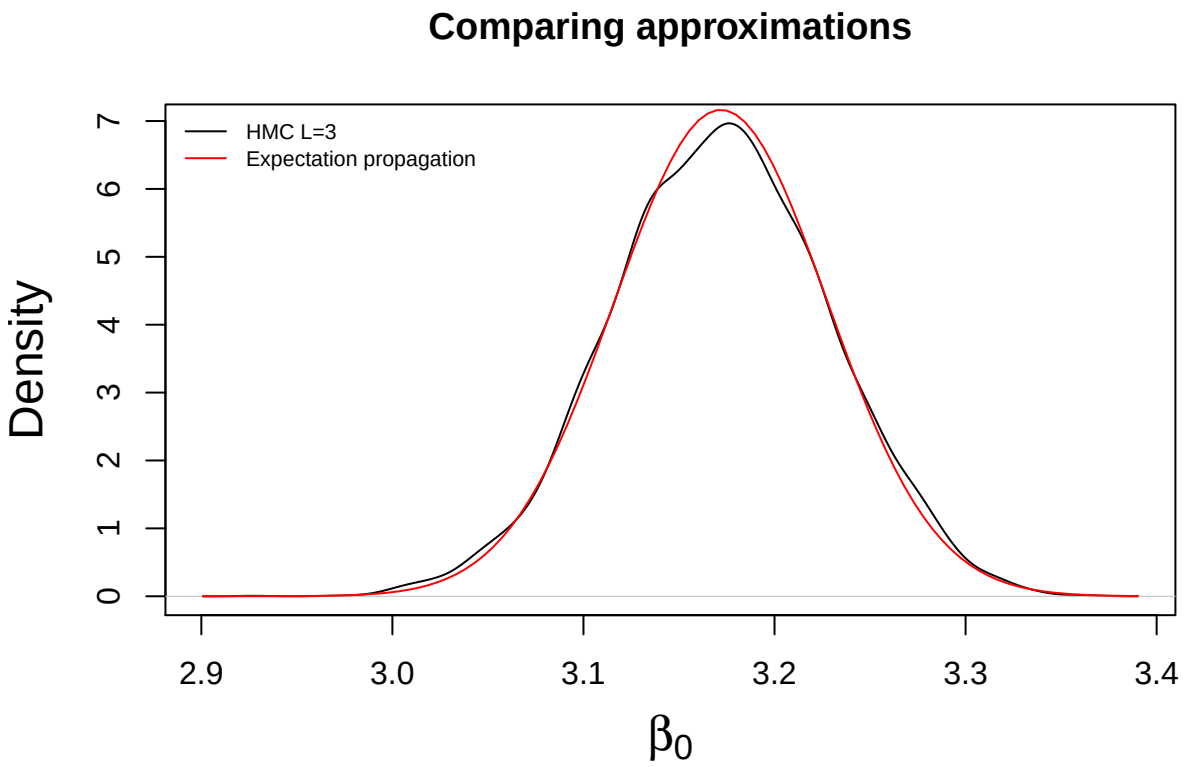
```
names(param)<-c('betahat','Sigma','iter_break')
return(param)
}
```

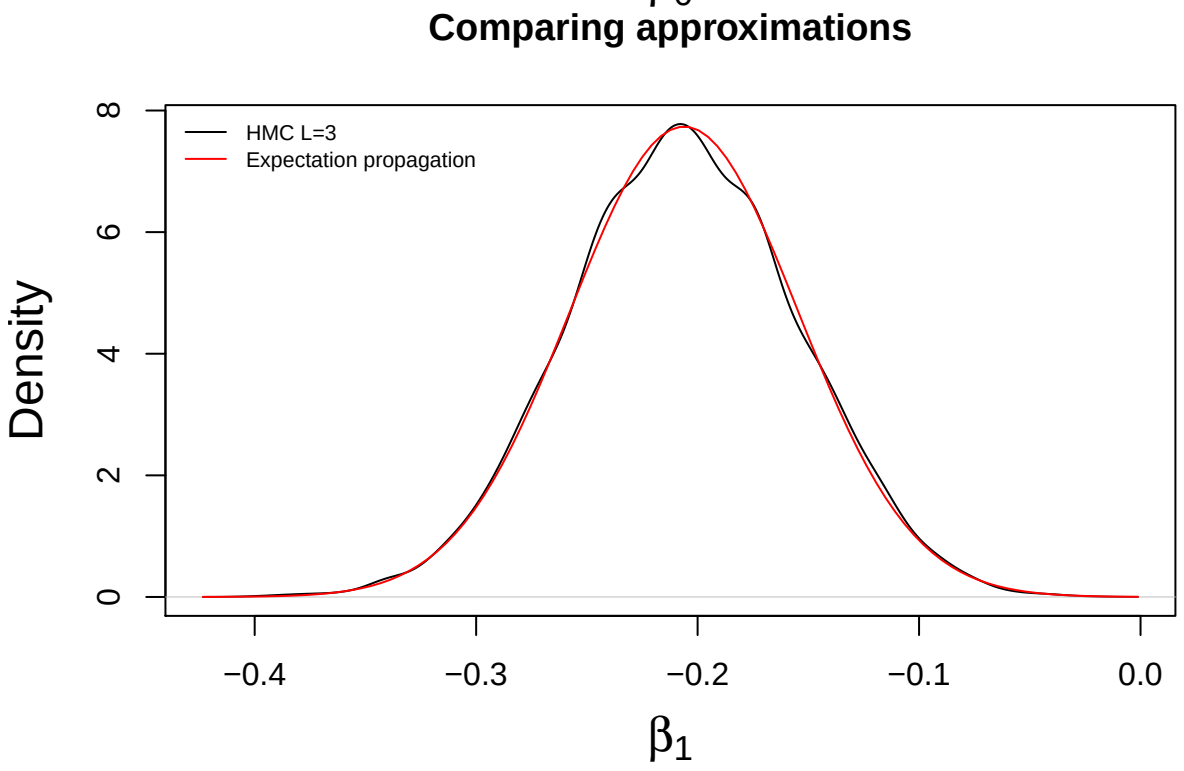
```
N <- 10
n <- length(y)
mresult<-EP.logit(response=y,n=rep(N,n),X=X,iter=100,epsilon=1e-6)
```

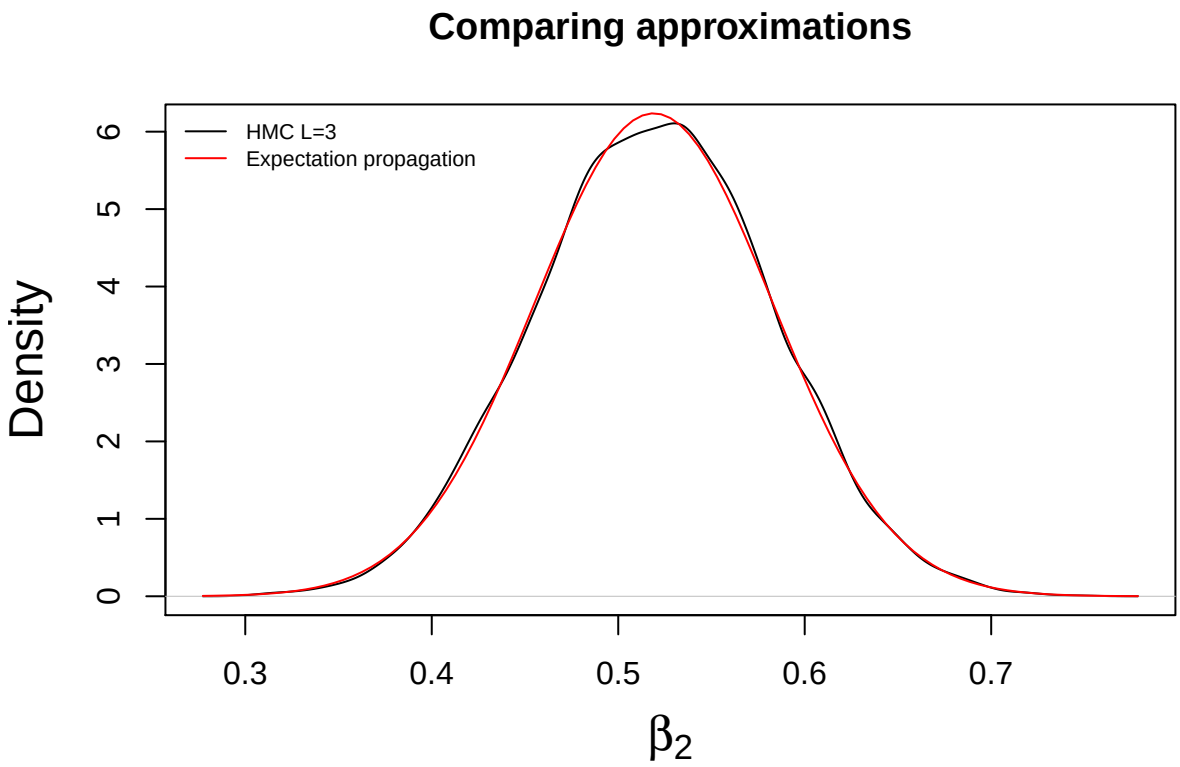
```
#Posterior means
report <- mresult$betahat
ci <- matrix(rep(0,8),nrow = 4)
#90 % credible intervals
std <- diag(mresult$Sigma)
for (i in 1:4) {
  ci[i,] <- qnorm(c(0.05,0.95),mean =mresult$betahat[i] ,sd =std[i] )
}

report <- cbind(report,ci)
colnames(report) <-(c("mean","lower CI" ,"upper CI"))
report
```

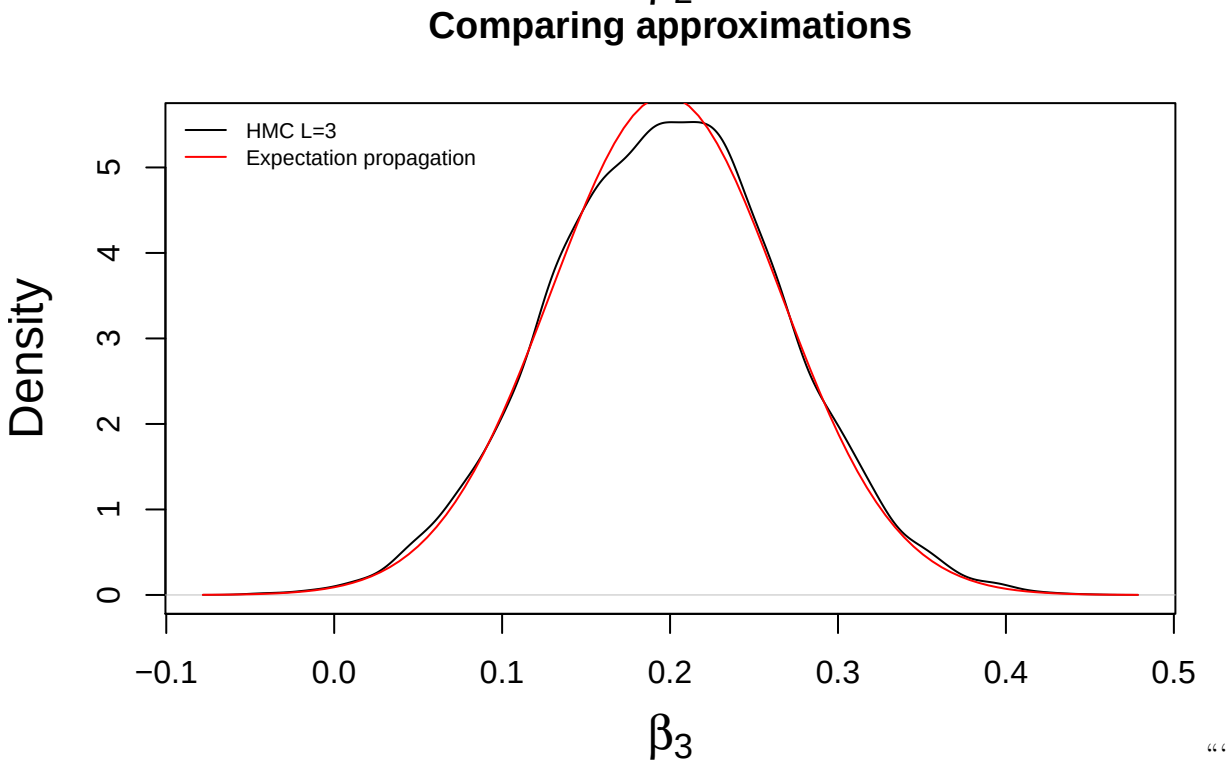
```
##                   mean   lower CI   upper CI
## (Intercept)  3.1719248  3.1668260  3.1770237
## woolB       -0.2061249 -0.2105000 -0.2017499
## tensionL     0.5190072  0.5122777  0.5257367
## tensionM     0.1973975  0.1897162  0.2050787
```

```
# par(mfrow=c(1,2))
for(i in 1:4){
plot(density(HMC13[,i]),xlab=bquote(beta[.(i-1)]),main='Comparing approximations',cex.lab=1.5)
# curve(dnorm(x,mean=mod$coef[i],sd=sqrt(vcov(mod)[i,i])),add=TRUE,col=2 )
curve(dnorm(x,mean=mresult$betahat[i],sd=sqrt(mresult$Sigma[i,i])),add=TRUE,col=2 )
legend('topleft',legend=c('HMC L=3','Expectation propagation'),col=1:2,lty=1,bty='n',cex=0.7)
}
```


Comparing approximations
HMC L=3
Expectation propagation
Density
0
1
2
3
4
5
6
7
2.9
3.0
3.1
3.2
3.3
3.4
β0


Comparing approximations
HMC L=3
Expectation propagation
Density
0
2
4
6
8
−0.4
−0.3
−0.2
−0.1
0.0
β1





“‘

Possible bias - Expectation propagation is data point specific so it may not be a global approximation to the posterior. In other words Expectation propagation may not able to generalise. Therefore, out of sample result my not be as accurate as the in-sample results above.