ST4234: Bayesian Statistics

Tutorial 8 Solution, AY 19/20

Solutions

1.(a) In the importance sampling, we use a multivariate t_4 distribution as a proposal distribution, whose location is equal to the posterior mode $\hat{\theta}$ and scale matrix is $-2[\ell''(\hat{\theta})]^{-1}$, where $\ell(\theta)$ is the log posterior (derived from Tutorial 6 Question 3)

$$\ell(\theta_1, \theta_2) = (1-s)\theta_1 + \theta_2 - \mathrm{e}^{-\theta_1}(t - nt_1 + n\mathrm{e}^{\theta_2}).$$

Let $g(\theta)$ denote the density of this multivariate t_4 distribution. After generating samples $\{\theta^{(s)}|s = 1, \ldots, 1000\}$ from this t distribution, we compute their weights which are given by $f(\theta^{(s)})/g(\theta^{(s)})$, where f is the unnormalized posterior density. These weights are then normalized to obtain the probabilities $W_s = w(\theta^{(s)})/\sum_{s=1}^{S} w(\theta^{(s)})$ for the resampling algorithm. The SIR algorithm can be implemented using the R-code below.

```
require(mvtnorm)

## Loading required package:  mvtnorm

s <- 8
n <- 15
t <- 15962989
t1 <- 237217
df <- 4

logpost <- function(theta, s, n, t, t1){
        theta1 <- theta[1]
        theta2 <- theta[2]
        return((1-s)*theta1 + theta2 -
               (t-n*t1)*exp(-theta1) - n*exp(theta2-theta1))
}
(out <- optim(par=c(14.5,11), fn=logpost, hessian=TRUE,
              control=list(fnscale=-1), s=s, n=n, t=t, t1=t1))

## $par
## [1] 14.54155 11.83311
##
## $value
## [1] -96.95903
```

```
##
## $counts
## function gradient
##       45       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
##
## $hessian
##            [,1]       [,2]
## [1,] -7.0013119  0.9996118
## [2,]  0.9996118 -0.9996118
```

```
(post.mode <- out$par)
```

```
## [1] 14.54155 11.83311
```

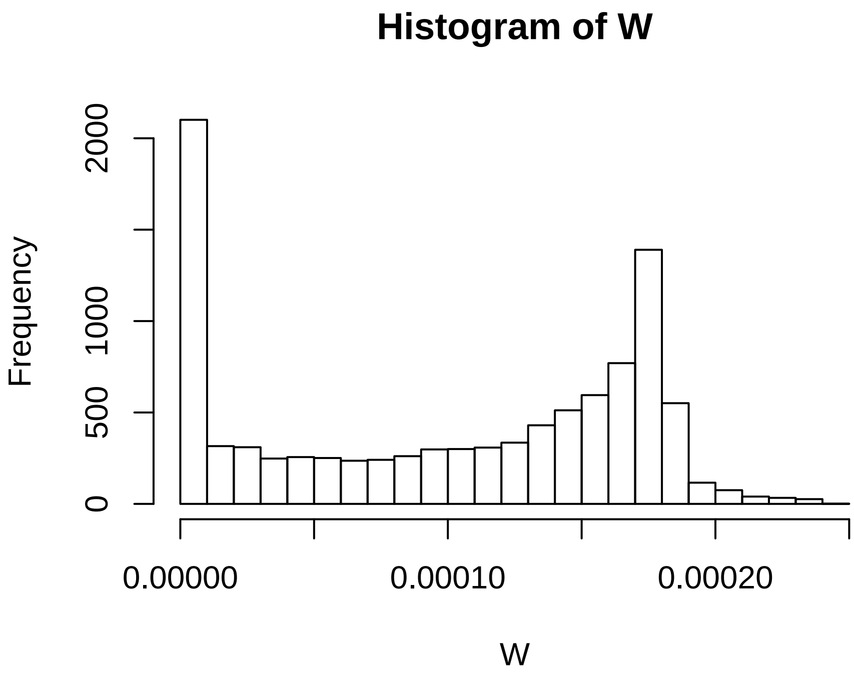
```
(post.cov <- -solve(out$hessian))
```

```
##           [,1]      [,2]
## [1,] 0.1666195 0.1666195
## [2,] 0.1666195 1.1670078
```

```
# define the difference function
diff <- function(theta, s, n, t, t1, post.mode, post.cov, df){
        logpost(theta,s,n,t,t1) -
   dmvt(theta,delta=post.mode,sigma=2*post.cov,df=df)
}

set.seed(4234)
S <- 10^4
# importance sampling
theta.samples <- rmvt(S,delta=post.mode,sigma=2*post.cov,df=df)
logw <- numeric(S)
for (i in 1:S) {
        logw[i] <- diff(theta.samples[i,],
                        s, n, t, t1, post.mode, post.cov, df)
}
w <- exp(logw - max(logw))
```

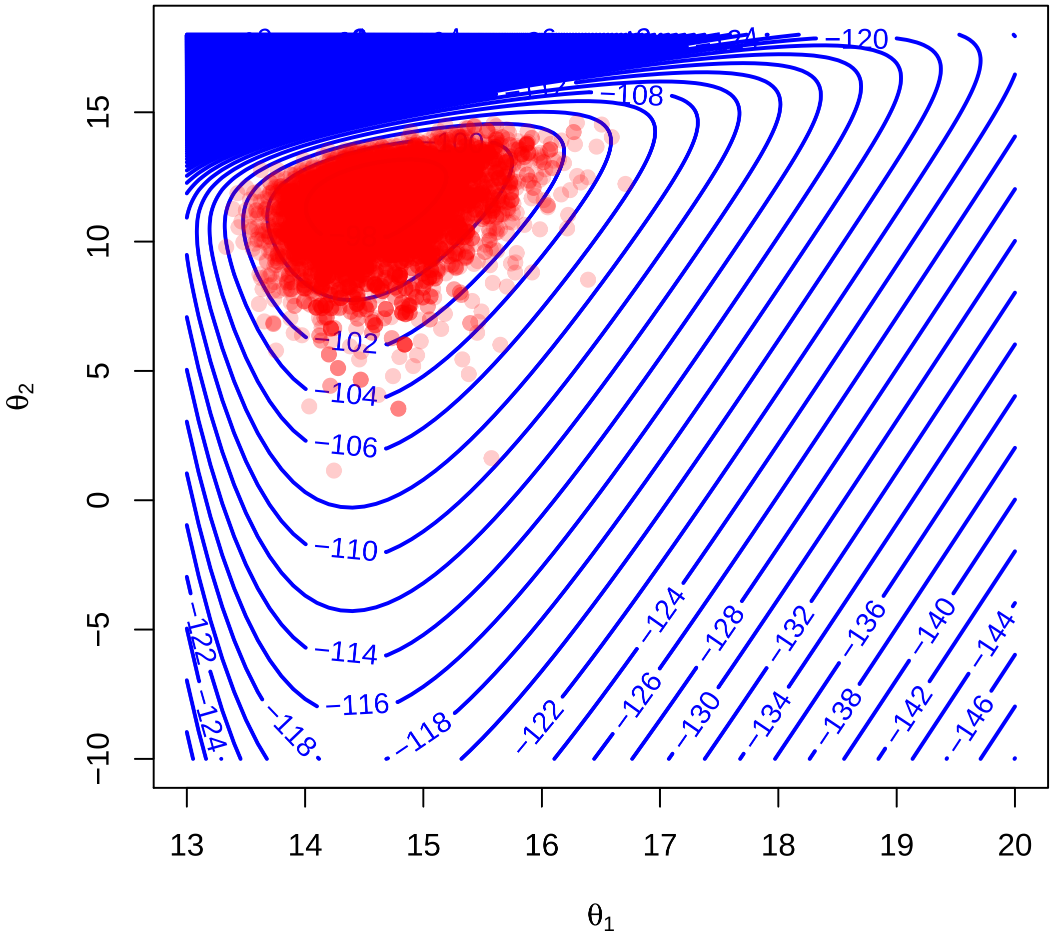
```
W <- w/sum(w)
hist(W, breaks=20)                # histogram of weights
```




```
# SIR
indices <- sample(1:S, size=S, prob=W, replace=TRUE)
theta.SIR <- theta.samples[indices,]
```

We can also plot the contour plot with the SIR samples overlayed.

```
theta1.grid <- seq(from=13, to=20, by=.1)
theta2.grid <- seq(from=-10, to=18, by=.1)
theta.grid <- expand.grid(theta1.grid, theta2.grid)
grid.logpost <- apply(theta.grid, 1,
                      logpost, s=s, n=n, t=t, t1=t1)
contour(theta1.grid, theta2.grid,
        matrix(grid.logpost,nrow=length(theta1.grid),
               ncol=length(theta2.grid)),
        col="blue", nlevels=800, lwd=2, labcex=0.9,
                xlab=expression(theta[1]), ylab=expression(theta[2]))
points(theta.SIR[,1], theta.SIR[,2],
       col=rgb(1,0,0,alpha=0.2),pch=19)
```



(b) We first write the function `Rt0` to compute $R(t_0)$ from values of θ. Then we obtain posterior samples of $R(t_0)$ and compute the mean and standard deviation from these samples. The posterior mean of $R(t_0)$ is 0.656 and posterior standard deviation is 0.103.

```
t0 <- 10^6
Rt0 <- function(theta, t0, t1){
        mu <- t1 - exp(theta[,2])
        beta <- exp(theta[,1])
        R <- exp(-(t0-mu)/beta)
        return(R)
}
R.samples <- Rt0(theta.SIR,t0,t1)
mean(R.samples)

## [1] 0.6555554
```

```
sd(R.samples)

## [1] 0.1030259
```

2.

(a) From Tutorial 7 Question 1, the log posterior function (in terms of η) is

$$\ell(\eta) = \log p(\eta|\boldsymbol{y}) = 125 \log\left(2 + \frac{e^{\eta}}{1+e^{\eta}}\right) - 74\log(1+e^{\eta}) + 35\eta + C,$$

where C is an additive constant not dependent on η. We find the posterior mode and Hessian matrix at the mode by using `optim()` with the option `method="Brent"`. Then, we set the `start` to be equal to the posterior mode, and specify the `var` in `proposal` to be equal to the negative inverse Hessian $-[\ell''(\hat{\theta})]^{-1}$. We can make some trial runs for the `scale` option. `scale=2` gives an acceptance rate 0.502. Since this rate is a little bit high, we further increase to `scale=3` and the acceptance rate is 0.374, which is better.

```r
# log posterior function
logpost <- function(eta) {
        125*log(2+(exp(eta)/(1+exp(eta))))-74*log(1+exp(eta))+35*eta
}

# find the posterior mode using optim() with method="Brent"
(out <- optim(par=0, fn=logpost, hessian=TRUE,
              control=list(fnscale=-1),
              method="Brent", lower=-10, upper=10))

## $par
## [1] 0.5066004
##
## $value
## [1] 65.93283
##
## $counts
## function gradient
##       NA       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
##
## $hessian
##             [,1]
```

```
## [1,] -21.13337
```

```r
(post.mode <- out$par)
```

```
## [1] 0.5066004
```

```r
(post.var <- -1/out$hessian)
```

```
##             [,1]
## [1,] 0.04731853
```

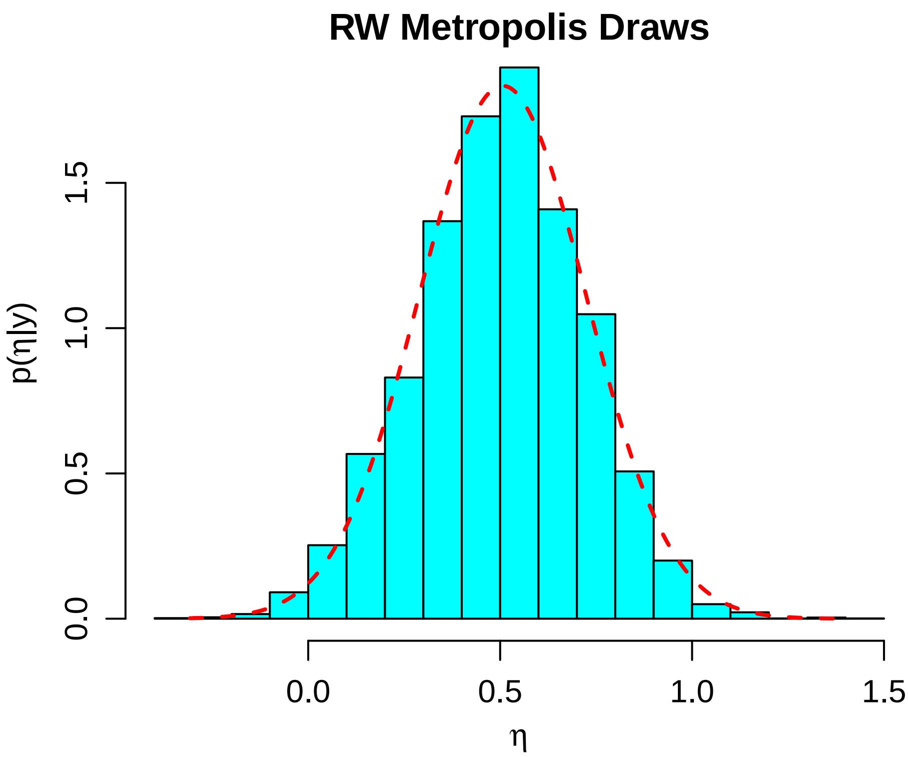
```r
# random walk Metropolis
require(LearnBayes)
```

```
## Loading required package:  LearnBayes
```

```r
T <- 10^4
proposal <- list(var=post.var,scale=3)
set.seed(4234)
fit1 <- rwmetrop(logpost,proposal,start=post.mode,m=T)
fit1$accept
```

```
## [1] 0.3739
```

```r
# histogram of RW Metropolis draws and normal approximation
# normal curve is overlayed in the histogram
eta.grid <- seq(from=min(fit1$par), to=max(fit1$par), length=1000)
par(mar=c(3.5,3.5,1,1))
par(mgp=c(2.1,0.8,0))
hist(fit1$par, freq=F, breaks=20, col="cyan",
     main="RW Metropolis Draws", xlab=expression(eta),
           ylab=expression(paste("p(",eta,"|y)")))
lines(eta.grid, dnorm(eta.grid,mean=post.mode,sd=sqrt(post.var)),
      lty=2,lwd=2,col="red")
```




```r
# 95% Bayesian CI of theta
CI <- quantile(fit1$par[5001:T], probs=c(0.025,0.975))
exp(CI)/(1+exp(CI))
```

```
##      2.5%     97.5%
## 0.5182480 0.7066912
```

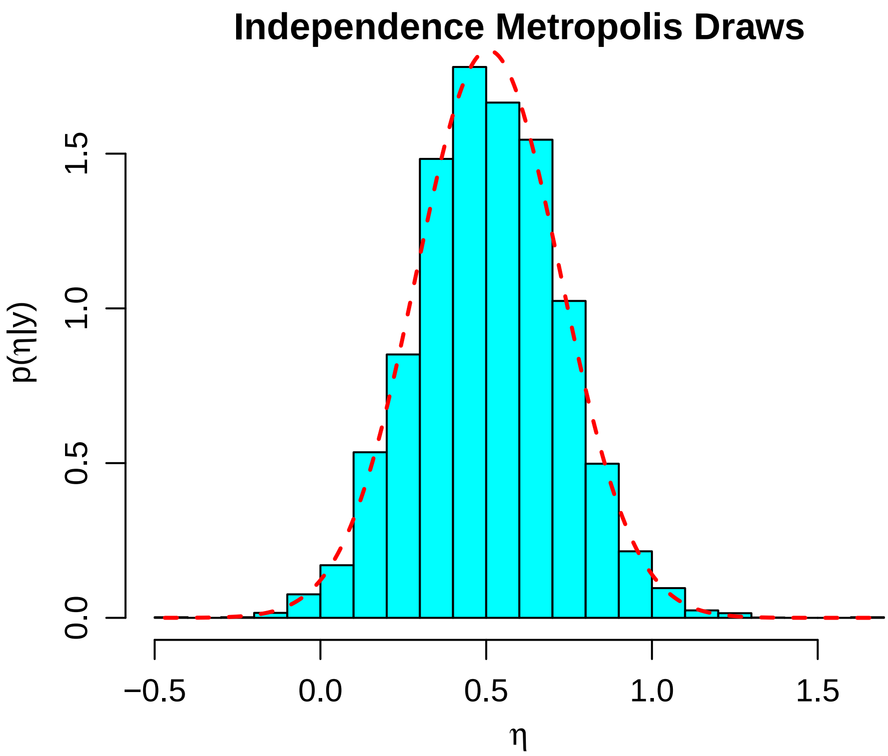
Therefore, with the normal proposal with variance $-9[\ell''(\hat{\theta})]^{-1}$ (note that `scale=3` means that we multiply 9 to the normal variance), we obtain the acceptance rate 0.374 from 10^4 MCMC iterations. A histogram of the posterior draws with the normal approximation density overlayed is shown below. The normal approximation seems to match the posterior density $p(\eta|\boldsymbol{y})$ well. The 95% Bayesian confidence interval for η is $[0.518, 0.707]$.

(b) We repeat Part (a) with the independence Metropolis algorithm. We only need to specify the `proposal` and we use the same configuration as in Part (a), i.e. the normal proposal with mean equal to the posterior mode $\hat{\theta}$ and variance equal to $-9[\ell''(\hat{\theta})]^{-1}$. We choose the `mu=post.mode` since this will be the closest distribution to the true posterior and the chance of accepting proposals will be higher. The acceptance rate of this independence Metropolis algorithm from 10^4 MCMC iterations is about 0.42. A histogram of the posterior draws with the normal approximation density overlayed is shown below. The normal approximation seems to match the posterior density $p(\eta|\boldsymbol{y})$ well. The 95% Bayesian confidence interval for η is $[0.526, 0.718]$.

```r
proposal2 <- list(mu=post.mode,var=9*post.var)
set.seed(4234)
fit2 <- indepmetrop(logpost,proposal2,start=post.mode,m=T)
fit2$accept
```

```
##         [,1]
## [1,] 0.4153
```

```r
# histogram of RW Metropolis draws and normal approximation
# normal curve is overlayed in the histogram
eta.grid <- seq(from=min(fit2$par), to=max(fit2$par), length=1000)
par(mar=c(3.5,3.5,1,1))
par(mgp=c(2.1,0.8,0))
hist(fit2$par, freq=F, breaks=20, col="cyan",
     main="Independence Metropolis Draws", xlab=expression(eta),
           ylab=expression(paste("p(",eta,"|y)")))
lines(eta.grid, dnorm(eta.grid,mean=post.mode,sd=sqrt(post.var)),
      lty=2,lwd=2,col="red")
```




```r
# 95% Bayesian CI of theta
CI <- quantile(fit2$par[5001:T], probs=c(0.025,0.975))
exp(CI)/(1+exp(CI))
```

```
##      2.5%     97.5%
```

```
## 0.5259686 0.7182764
```

3.(a) We have

$$p(y_i|\beta_0,\beta_1) = \frac{\exp(-\lambda_i)\lambda_i^{y_i}}{y_i!} = \frac{\exp(-\mathrm{e}^{\beta_0+\beta_1 i})\mathrm{e}^{y_i(\beta_0+\beta_1 i)}}{y_i!}.$$

Since $p(\beta_0,\beta_1) \propto 1$, the posterior is given by

$$\begin{aligned} p(\beta_0,\beta_1|\boldsymbol{y}) &\propto \left\{\prod_{i=1}^{18} p(y_i|\beta_0,\beta_1)\right\} p(\beta_0,\beta_1) \\ &\propto \left\{\prod_{i=1}^{18} \exp(-\mathrm{e}^{\beta_0+\beta_1 i})\mathrm{e}^{y_i(\beta_0+\beta_1 i)}\right\} \\ \Rightarrow \log p(\beta_0,\beta_1|\boldsymbol{y}) &= \sum_{i=1}^{18}[y_i(\beta_0+\beta_1 i) - \exp(\beta_0+\beta_1 i)] + C \end{aligned}$$

where C is an additive constant not depending on (β_0,β_1).

(b) We define the log posterior function given in Part (a). To help with the optimization of `logpost()`, we can use the R function `glm()` to obtain the maximum likelihood estimator of (β_0,β_1). This MLE can be used as an initial value for searching the posterior mode.

```
y <- c(15,11,14,17,5,11,10,4,8,10,7,9,11,3,6,1,1,4)
n <- length(y)
x <- 1:n

# log posterior function
logpost <- function(theta,y){
        n <- length(y)
        beta0 <- theta[1]
        beta1 <- theta[2]
        lp <- beta0 + beta1*(1:n)
        L <- sum(y*lp - exp(lp))
        return(L)
}

# first fit the generalized linear model (Poisson regression)
fit.glm <- glm(y~x,family=poisson)
summary(fit.glm)

##
## Call:
## glm(formula = y ~ x, family = poisson)
##
```

```
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.9886  -0.9631   0.1737   0.5131   2.0362
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  2.80316    0.14816  18.920  < 2e-16 ***
## x           -0.08377    0.01680  -4.986 6.15e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 50.843  on 17  degrees of freedom
## Residual deviance: 24.570  on 16  degrees of freedom
## AIC: 95.825
##
## Number of Fisher Scoring iterations: 5
```

```
theta.mle <- fit.glm$coefficients       # MLE of theta used as initial values


(out <- optim(par=theta.mle,fn=logpost,hessian=TRUE,
              control=list(fnscale=-1),y=y))
```

```
## $par
## (Intercept)           x
##  2.80315907 -0.08376906
##
## $value
## [1] 174.8451
##
## $counts
## function gradient
##       53       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
##
## $hessian
##             (Intercept)          x
## (Intercept)    -147.000  -1077.024
## x             -1077.024 -11434.597
```

```
(post.mode <- out$par)
```

```
## (Intercept)           x
##  2.80315907 -0.08376906
```

```
(post.cov <- -solve(out$hessian))
```

```
##              (Intercept)             x
## (Intercept)  0.021951407 -0.0020676023
## x           -0.002067602  0.0002822013
```

Hence, the posterior $p(\beta_0, \beta_1 | \boldsymbol{y})$ can be approximated by

$$(\beta_0, \beta_1) \mid \boldsymbol{y} \overset{\text{approx}}{\sim} \mathrm{N}\left(\begin{bmatrix} 2.8032 \\ -0.0838 \end{bmatrix}, \begin{bmatrix} 0.0220 & -0.0021 \\ -0.0021 & 0.0003 \end{bmatrix}\right). \tag{1}$$

(c) (i) To construct the Metropolis random walk algorithm, we define a list `proposal1` that contains the covariance matrix of the normal approximation and a scale factor of 2. We use the posterior mode as a starting value and use the function `rwmetrop` to simulate 10^4 iterates.

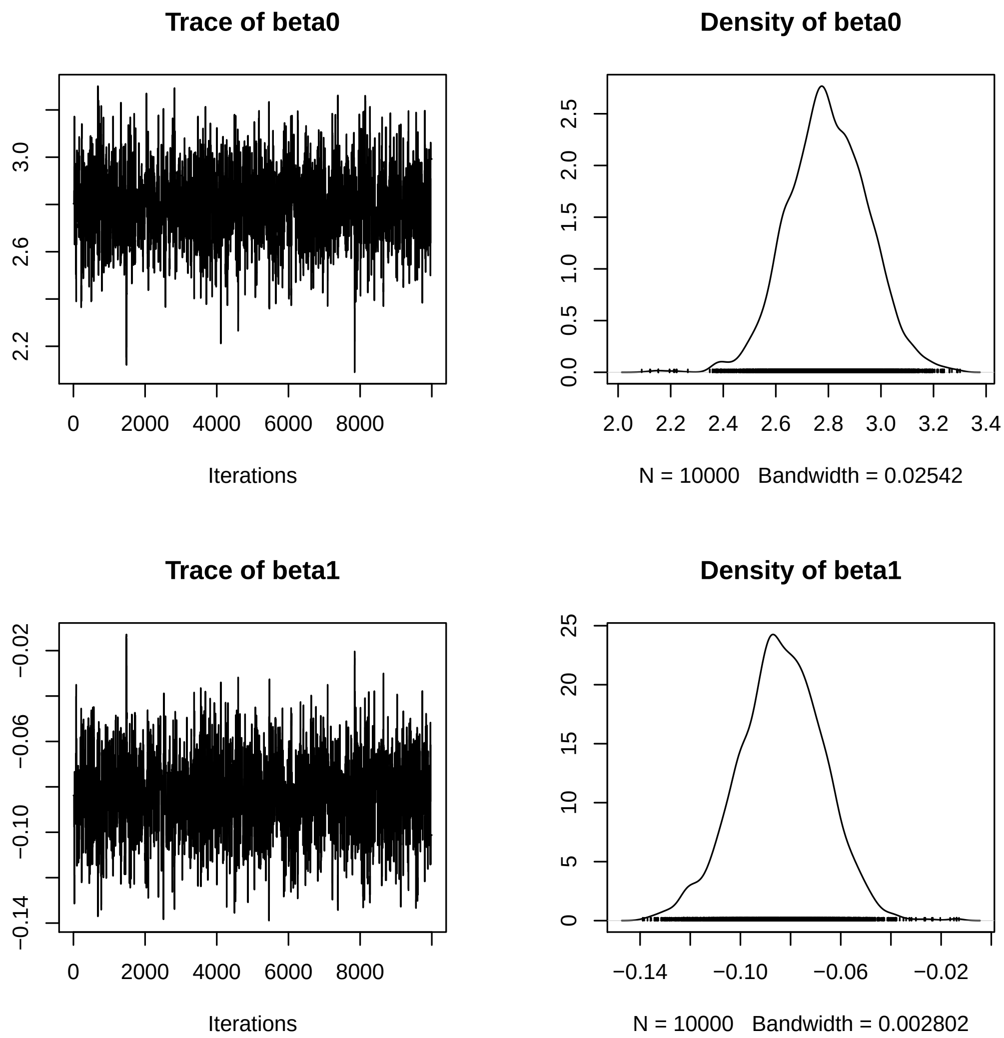
```
require(LearnBayes)
require(coda)
```

```
## Loading required package:  coda
```

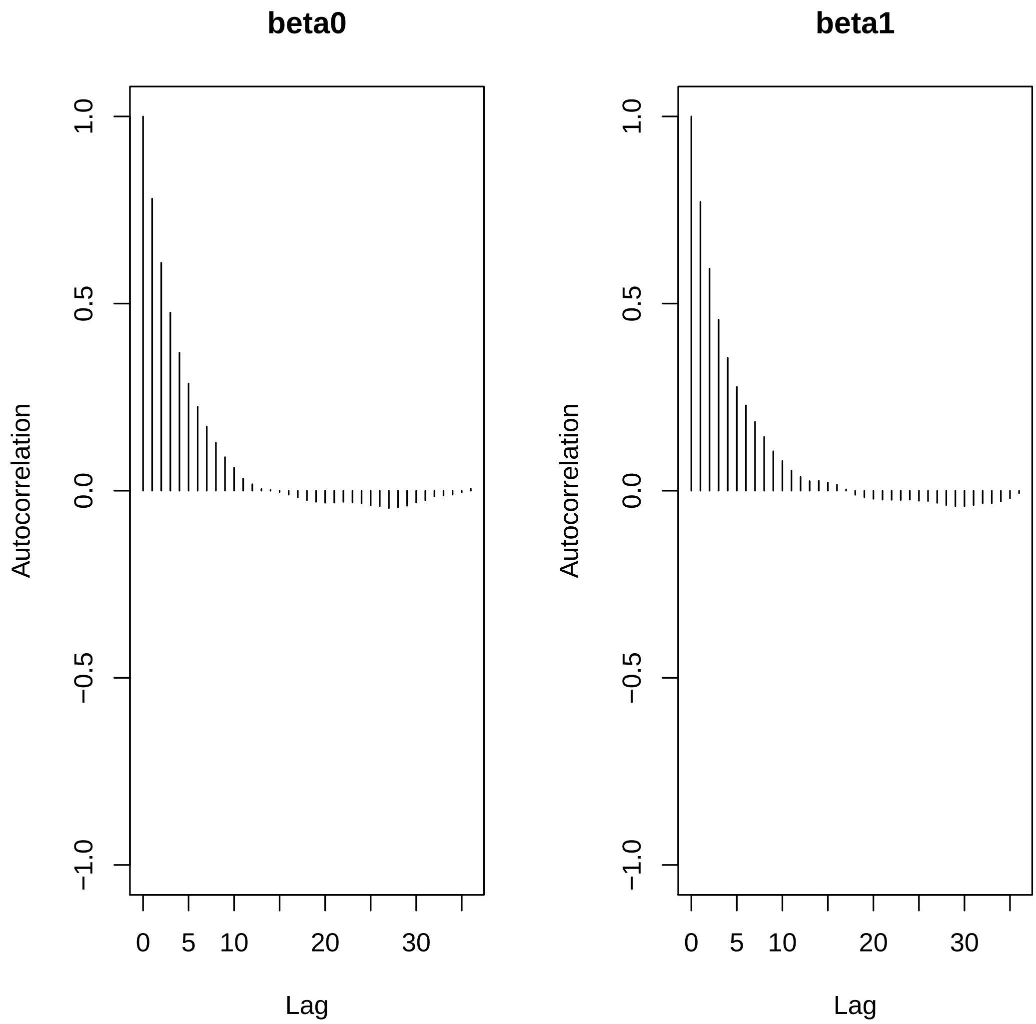
```
T <- 10^4
proposal1 <- list(var=post.cov,scale=2)
start <- post.mode
set.seed(4234)
fit1 <- rwmetrop(logpost,proposal1,start,T,y)
fit1$accept
```

```
## [1] 0.2922
```

```
colnames(fit1$par) <- c("beta0","beta1")
plot(mcmc(fit1$par))
```




```
autocorr.plot(mcmc(fit1$par))
```



The acceptance rate is 0.29. The traceplot indicates good convergence and the autocorrelation plot shows that the autocorrelation decreases fairly quickly as a function of the lag. (It's not necessary to report these since they are not asked by the question. This is just for your reference.)

```
mean(fit1$par[,2])

## [1] -0.08379816

sd(fit1$par[,2])

## [1] 0.01691401
```

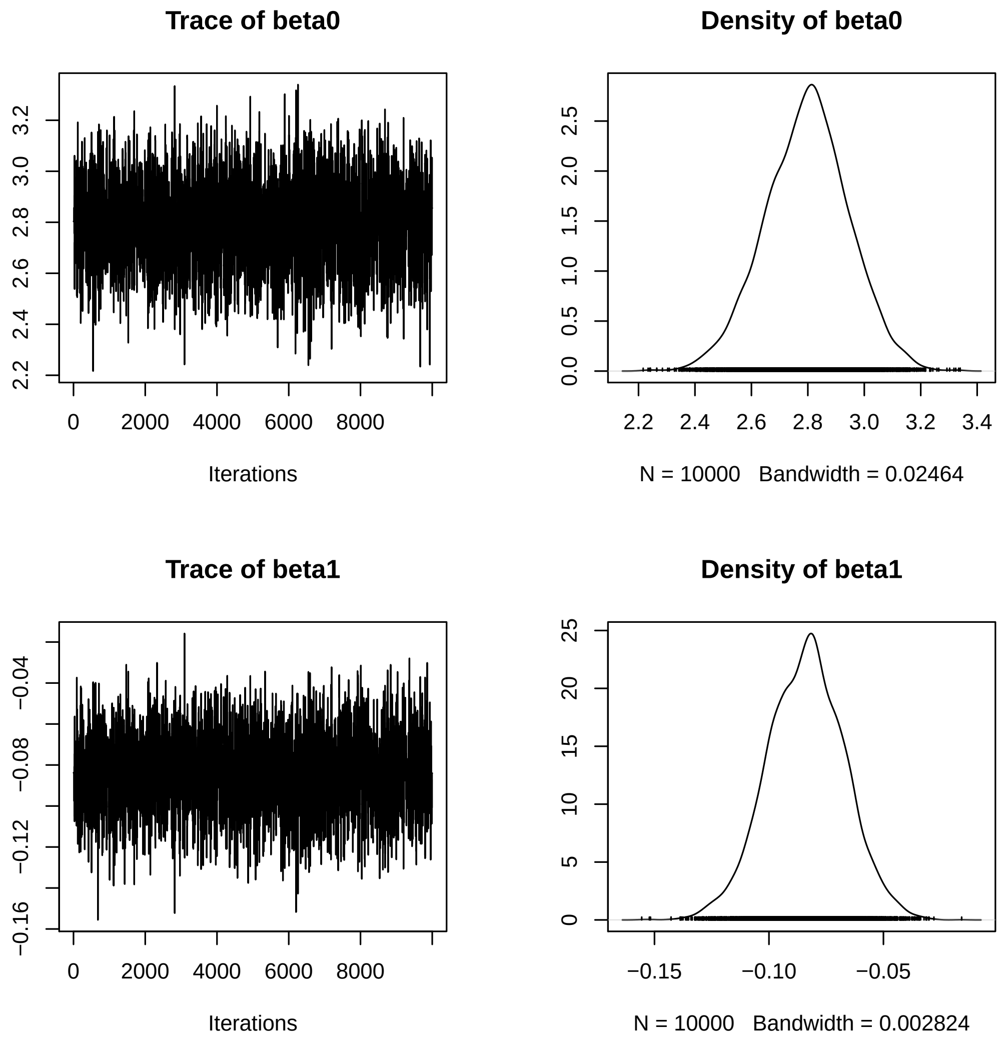
The posterior mean of β_1 is -0.084 and the posterior standard deviation is 0.017.

(ii) To construct a Metropolis independence algorithm, we consider a proposal density that is multivariate normal with mean and covariance matrix given by the normal approximation. We continue to use the posterior mode as a starting value and define a list `proposal2` that contains the mean and covariance matrix of the normal approximation. Then we use the function indepmtrop to simulate 10^4 iterates from the posterior.

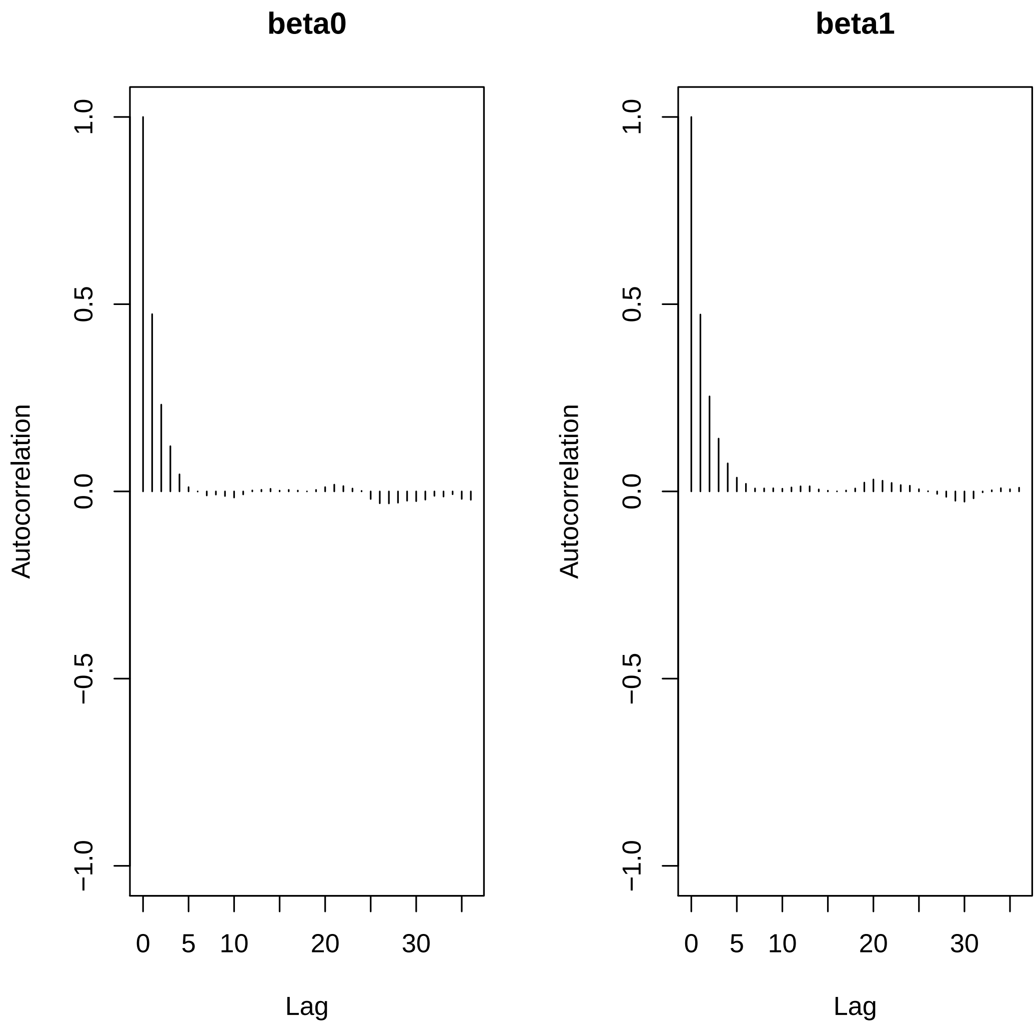
```
proposal2 <- list(mu=post.mode,var=4*post.cov)
set.seed(4234)
fit2 <- indepmetrop(logpost,proposal2,start,T,y)
fit2$accept
```

```
##         [,1]
## [1,] 0.3991
```

```
colnames(fit2$par) <- c("beta0","beta1")
plot(mcmc(fit2$par))
```



```r
autocorr.plot(mcmc(fit2$par))
```



Compared to the Metropolis random walk algorithm in (c), the Metropolis independence algorithm has an acceptance rate of about 0.40. The trace plot indicate better mixing and the autocorrelations also dissipate much more rapidly as a function of the lag.

```r
mean(fit2$par[,2])

## [1] -0.08385069

sd(fit2$par[,2])

## [1] 0.01680854
```

The posterior mean of β_1 is -0.084 and the posterior standard deviation is 0.017.

(d) The table below summarizes the acceptance rates and 5th, 50th and 95th percentiles of β_0 and β_1.

Method	acceptance rate	β_0	β_1
Normal approximation	–	(2.559, 2.803, 3.047)	(-0.111, -0.084, -0.056)
Random walk Metropolis	0.29	(2.560, 2.795, 3.041)	(-0.112, -0.084, -0.057)
Independence Metropolis	0.40	(2.548, 2.802, 3.040)	(-0.111, -0.083, -0.056)

The 5th, 50th and 95th percentiles can be obtained using the following R-code.

```
# normal approximation
post.sd <- sqrt(diag(post.cov))
qnorm(c(0.05,0.5,0.95),mean=post.mode[1],sd=post.sd[1])

## [1] 2.559457 2.803159 3.046861

qnorm(c(0.05,0.5,0.95),mean=post.mode[2],sd=post.sd[2])

## [1] -0.11140071 -0.08376906 -0.05613742

# RW Metropolis
apply(fit1$par,2,quantile,probs=c(0.05,0.5,0.95))

##        beta0       beta1
## 5%  2.559875 -0.11187971
## 50% 2.794621 -0.08376906
## 95% 3.041218 -0.05680467

# Independence Metropolis
apply(fit2$par,2,quantile,probs=c(0.05,0.5,0.95))

##        beta0       beta1
## 5%  2.548381 -0.11116999
## 50% 2.801986 -0.08337727
## 95% 3.039601 -0.05626486
```