Stats4Astro 2017

MCMC Computer Lab

This lab will focus on writing MCMC samplers for high energy spectral analysis. This will largely involve writing code for the samplers described in the notes. We will use a number of simulated data sets rather than real data to avoid technical difficulties that would arise in fully accounting for the actual data generation mechanisms. The simulated data sets are among the files provided for the course. The following Table lists the simulated data sets and calibration products that you will need to complete this lab.

file name	columns	description
`spectral-1.txt`	Energy, Count	a basic spectral data set
`spectral-2.txt`	Energy, Count	a low-count spectral data set
`spectral-3.txt`	Energy, Count	a spectral data set with an emission line (Not provided, see below.)
`spectral-4.txt`	Energy, Count	a spectral data set that includes absorption, effective area, and background
`eff-area.txt`	Energy, Effective Area	an effective area curve

Each of the spectral data sets is a simulated high-energy spectrum and consist of photon counts from a number of energy bins. The first column reports the mean energy (keV) of each bin and the second the photon count. I refer to the mean energy in bin i as E_i and the count in bin i as Y_i for $i = 1, \ldots, n$. The energy range for the first three simulated spectra is [1.905, 10.730] and the energy range for the fourth is [1.905, 7.365].

We model the photon counts as independent Poisson random variables, $Y_i \overset{\text{indep}}{\sim} \text{Poisson}(\Lambda_i)$, where Λ_i is the expected photon count in bin i and is constrained according to a parameterized model. Three parametrized models will be considered. In Model 1 Λ_i follows a simple powerlaw, i.e.,

$$\text{Model 1} : \Lambda_i(\theta_1) = \alpha E_i^{-\beta},$$

where $\theta_1 = (\alpha, \beta)$ is a model parameter of direct scientific interest.

In Model 2, we add a spectral line to Model 1.

$$\text{Model 2} : \Lambda_i(\theta_2) = \alpha E_i^{-\beta} + \gamma I\{i \in \mathcal{L}(\delta)\},$$

where $I\{i \in \mathcal{L}(\delta)\}$ is an indicator function that is one if i is contained in the set $\mathcal{L}(\delta)$ and zero otherwise, $\mathcal{L}(\delta) = \{\delta - 1, \delta, \delta + 1\}$ for $\omega = 1, \ldots, n$, and $\theta_2 = (\alpha, \beta, \gamma, \delta)$ is a parameter of scientific interest. Thus, the spectral line adds a constant γ to the powerlaw in each of three adjacent bins and the location of the spectral line in indexed by δ.

In Model 3, we add absorption, effective area and background contamination to Model 1, but do not include the spectral line.:

$$\text{Model 3} : \Lambda_i(\theta_3) = (\alpha E_i^{-\beta} + \kappa) \cdot e^{-\omega/E_i} \cdot A(E_i)$$

where κ models the background intensity, $e^{-\omega/E_i}$ is the probability of a photon not being absorbed and $A(E_i)$ represents the effective area. Here $\theta_2 = (\alpha, \beta, \kappa, \omega)$ is a parameters of scientific interest.

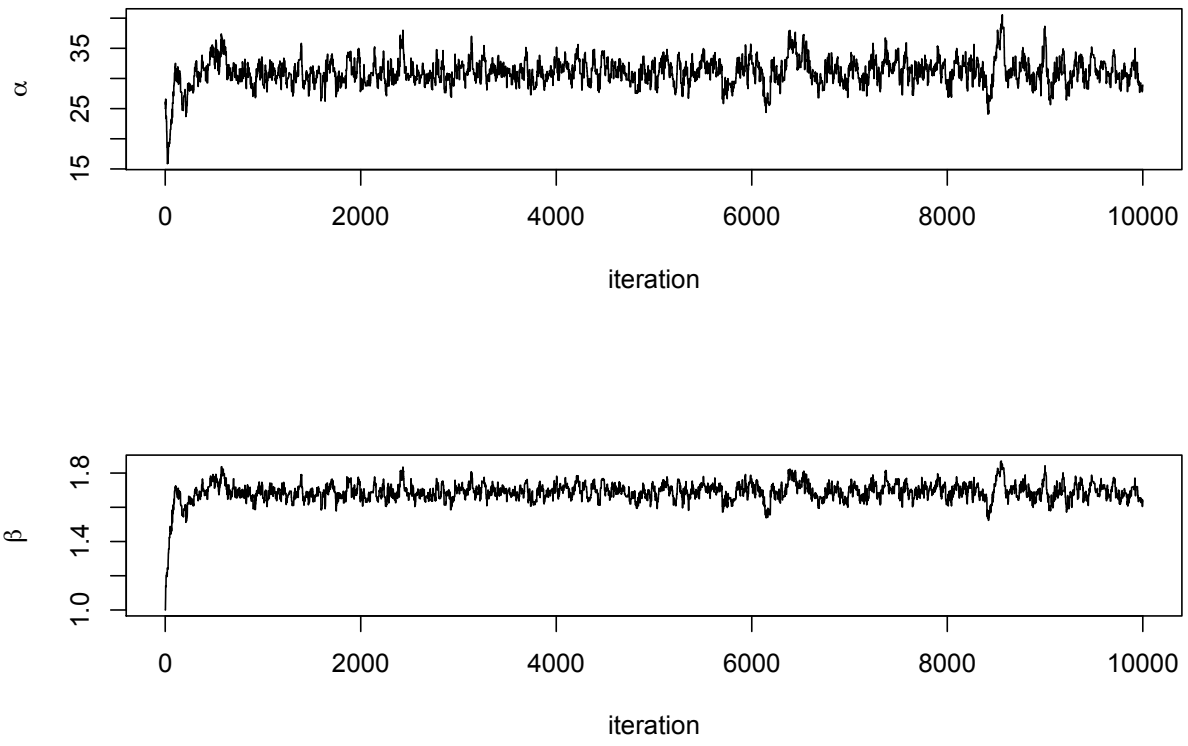
When fitting these models you may use flat priors on all parameters.

1. Write code for a Random Walk Metropolis sampler to fit MODEL 1 to `spectral-1.txt`. What do you consider when you select your jumping rule? How efficiently does your sampler explore the posterior distribution?

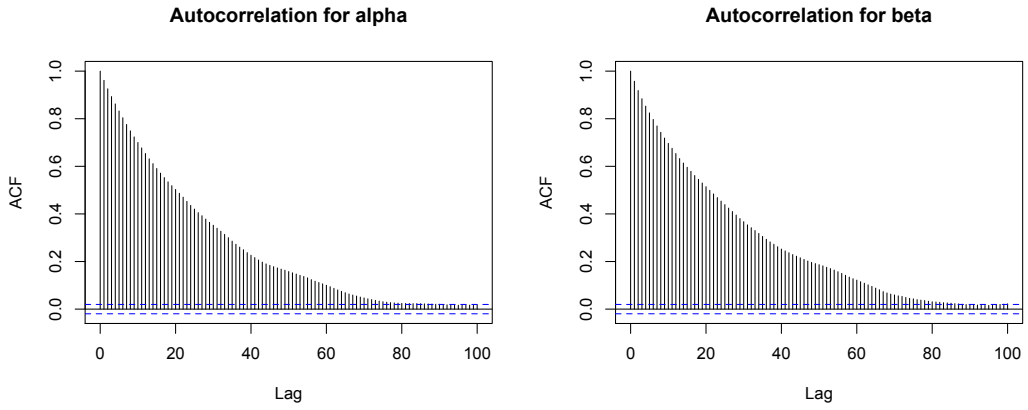
   Solution:

```
runMetropolis <- function(spec, draw.num=10000, start.vals, jump.var){

E = spec[,1]; Y = spec[,2]

Draws = matrix(NA, draw.num, 2)
Draws[1,1] = start.vals[1]; Draws[1,2] = start.vals[2];
accept = 0

log.lkhd = function( y=Y,x=E,alpha,beta ) ##up to a normalizing constant
        return( log(alpha)*sum(y) - beta*sum( y*log(x) ) - alpha*sum( x^(-beta) ) )

### Prior for alpha and beta:: uniform on [0, 100]
cat("Iteration: 1")
for(i in 2:draw.num)
{
        draw.star = rmvnorm( 1,mean=Draws[i-1,],sigma=jump.var*diag(c(1500,1)) )
        if(draw.star[1]<=0||draw.star[1]>=100||draw.star[2]<=0||draw.star[2]>=100){
                Draws[i,] = Draws[i-1,]
        }else{
                log.ratio = log.lkhd(alpha=draw.star[1],beta=draw.star[2] )-
                                        log.lkhd(alpha=Draws[i-1,1],beta=Draws[i-1,2] )
                ratio = exp( min(log.ratio,100) )
                temp = runif(1)
                if(temp < min(ratio,1) ){
                        Draws[i,] = draw.star
                        accept = accept + 1
                }else{
                        Draws[i,] = Draws[i-1,]
                }
        }
        if(i %% 1000 == 0) cat(", ",i)
}
output <- list(Draws,accept/draw.num)
names(output) <- c("Draws","accept.ratio")
return(output)
}
```

```
# loading some necessary packages; use install.packages() if not done previously
library(mvtnorm)
library(coda)

# reading in the spectrum and running the Random Walk Metropolis sampler
spec1 <- read.table("data/spectral-1.txt",header=TRUE )
p1 <- runMetropolis(spec = spec1, start.vals=c(25,1,5),jump.var=0.035^2)

## some plotting routines and examining the results
par(mfrow=c(2,1))
plot(p1$Draws[,1],xlab="iteration",ylab=expression(alpha),type="l",main="MCMC trace plots")
plot(p1$Draws[,2],xlab="iteration",ylab=expression(beta),type="l")

par(mfrow=c(1,2))
acf(p1$Draws[,1],main="Autocorrelation for alpha",lag.max=100)
acf(p1$Draws[,2],main="Autocorrelation for beta",lag.max=100)

print(p1$accept.ratio)
print(effectiveSize(p1$Draws))
```



MCMC trace plots



Autocorrelation plots

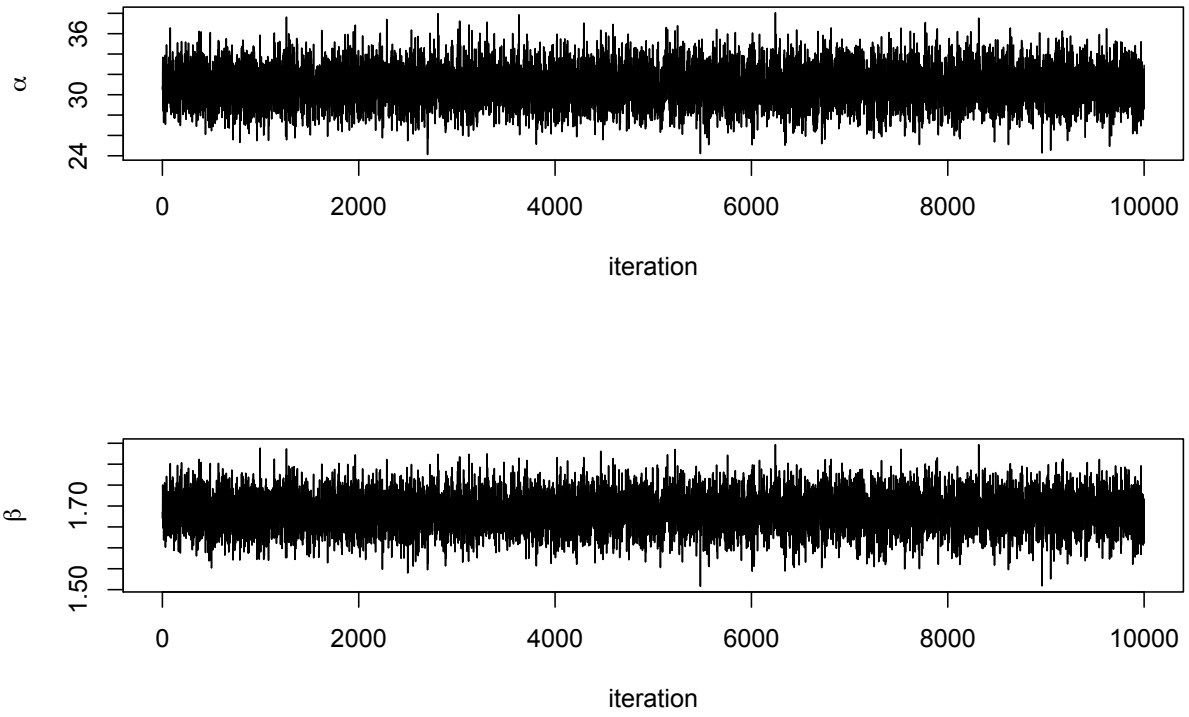
We find that the acceptance rate for the Random Walk Metropolis sampler with the chosen jumping rule is about 36%, which is in the target range of 20% to 40%. There is somewhat high autocorrelation, and the effective sample size is about 160 to 180 for both α and β.

2. Write code for an Independence Sampler to fit MODEL 1 to `spectral-1.txt`. Construct your jumping rule by using the R command `GLM` to derive an approximation to the posterior distribution. How efficiently does your sampler explore the posterior distribution?
`GLM` *hint: You can fit the model with* `GLM` *using*
`glm.fit <- glm( Count ~ I(-log(Energy)),family=poisson(link="log"))`
You can save the fitted values of (α, β) *into* `fit` *with*
`fit <- c(exp(glm.fit$coef[1]), glm.fit$coef[2])`
and compute the asymptotic variance-covariance matrix via the delta method with
`vmat <- diag(c(fit[1],1))%*%vcov(glm.fit)%*%diag(c(fit[1],1))`.

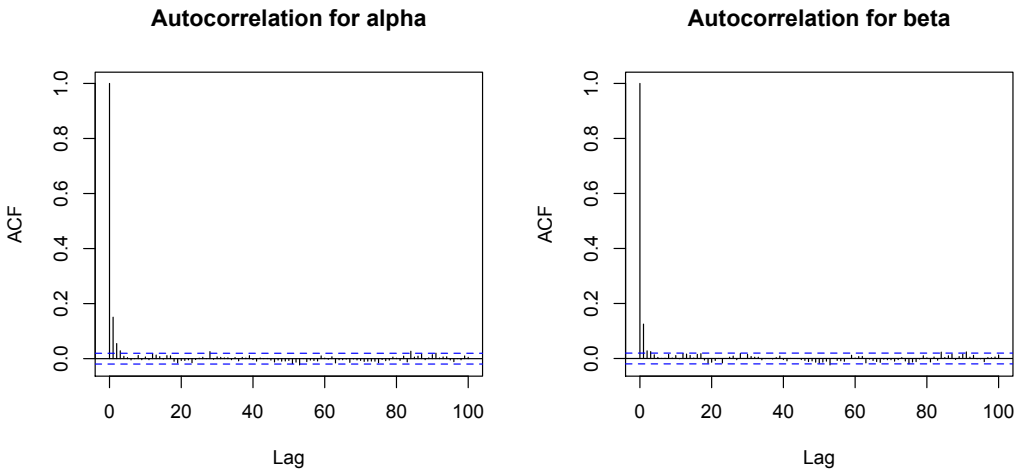
Solution:

```
runIndep <- function(spec,draw.num=10000){
E = spec[,1]; Y = spec[,2]

draw.num = 10000;
accept = 0;
Draws = matrix(NA, draw.num, 2)

log.lkhd = function( y=Y,x=E,alpha,beta ) ##up to a normalizing constant
        return( log(alpha)*sum(y) - beta*sum( y*log(x) ) - alpha*sum( x^(-beta) ) )

glm.fit = glm( Y~I(-log(E)),family=poisson( link="log" ) )
fit <- c( exp( glm.fit$coef[1] ), glm.fit$coef[2] )
vmat <- diag( c(fit[1],1) ) %*% vcov(glm.fit) %*% diag( c(fit[1],1) )

Draws[1,] = fit
## Prior for alpha and beta:: uniform on [0, 100]
```

```
cat("Iteration: 1")
for(i in 2:draw.num)
{
        draw.star = rmvnorm( 1, mean=fit, sigma=vmat )

        log.ratio = ( log.lkhd(alpha=draw.star[1],beta=draw.star[2] )-
                                log.lkhd(alpha=Draws[i-1,1],beta=Draws[i-1,2] ) )-
                                ( log(dmvnorm(draw.star,mean=fit,sigma=vmat ) )-
                                log(dmvnorm(Draws[i-1,],mean=fit,sigma=vmat ) ) )

        ratio = exp( min(log.ratio,100) )
        temp = runif(1)
        if(temp < min(ratio,1) ){
        Draws[i,] = draw.star
        accept = accept + 1
        }else{
                Draws[i,] = Draws[i-1,]
        }
        if(i %% 1000 == 0) cat(", ",i)
}
output <- list(Draws,accept/draw.num)
names(output) <- c("Draws","accept.ratio")
return(output)
}
p2Draws <- runIndep(spec = spec1)

p2 <- runIndep(spec = spec1)

## some plotting routines and examining the results
par(mfrow=c(2,1))
plot(p2$Draws[,1],xlab="iteration",ylab=expression(alpha),type="l")
plot(p2$Draws[,2],xlab="iteration",ylab=expression(beta),type="l")

par(mfrow=c(1,2))
acf(p2$Draws[,1],main="Autocorrelation for alpha",lag.max=100)
acf(p2$Draws[,2],main="Autocorrelation for beta",lag.max=100)

print(p2$accept.ratio)
print(effectiveSize(p2$Draws))
```



MCMC trace plots



Autocorrelation plots

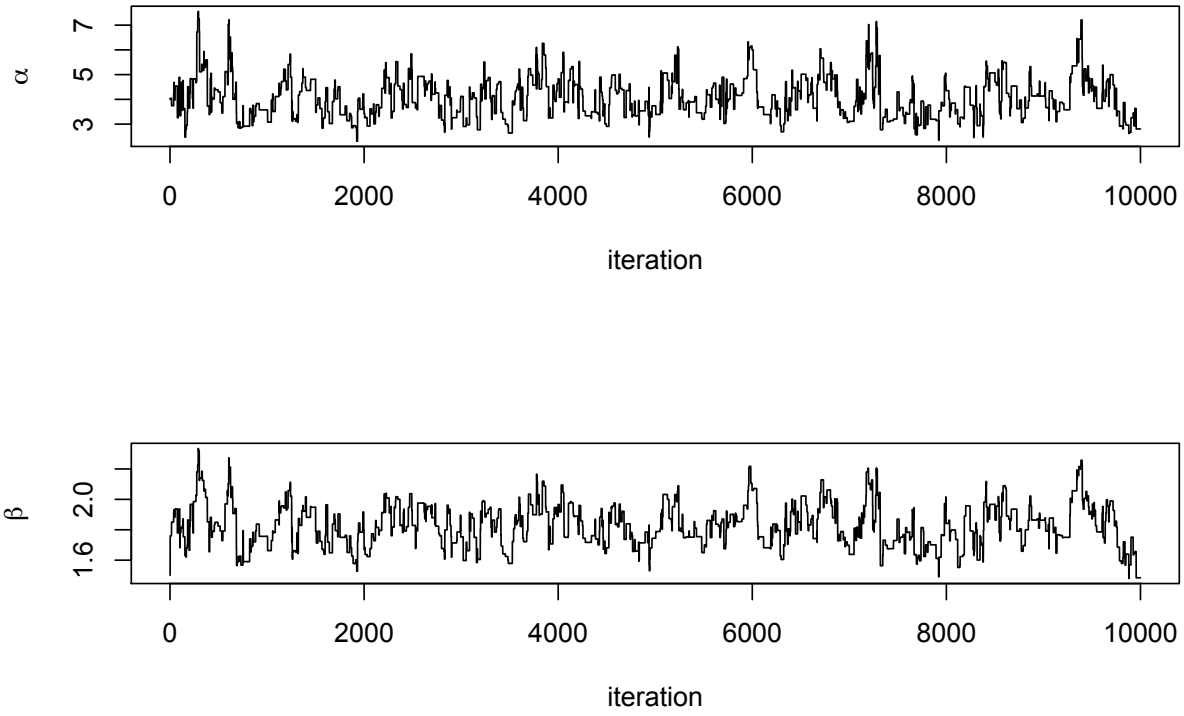
We find that the acceptance rate for the Independence Sampler is about 94%, with very low autocorrelation. The effective sample size is around 6000-8000 for α and β. This sampler is efficiently exploring the posterior distribution.

3. Run the Random Walk Metropolis sampler and the Independence Sampler that you derived in parts 1 and 2 to fit MODEL 1 to `spectral-2.txt`. How does the convergence behavior compare with the two data sets? You may want to reconsider the jumping rules that you settled on in parts 1 and 2 to design a more robust pair of samplers.

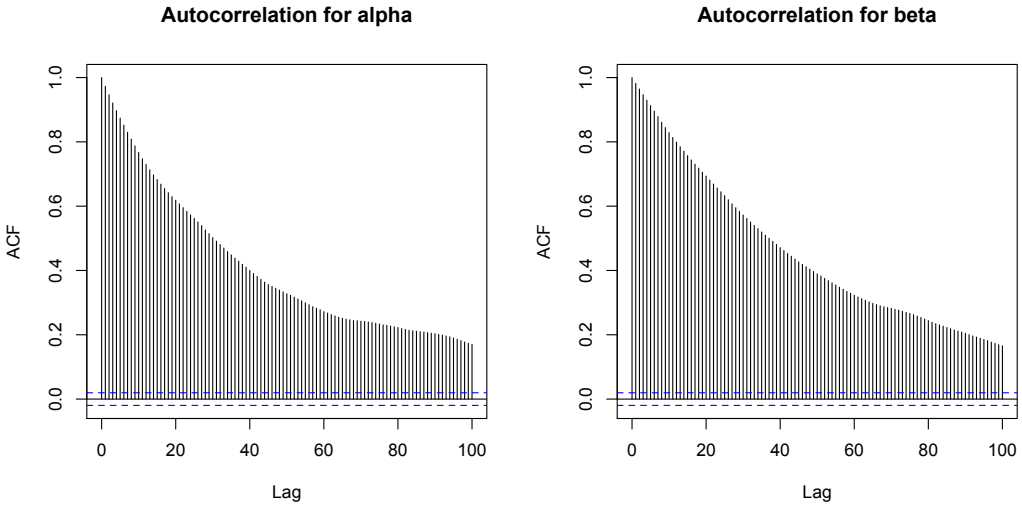
Solution:

```
spec2 <- read.table("data/spectral-2.txt",header=TRUE )

# Running with Metropolis sampler.
# Note that we're adjusting the jumping rule to obtain a more efficient sampler
#     compared to that with the previous jumping rule.
p3_Metrop <- runMetropolis(spec=spec2,start.vals=c(4,1.5),jump.var=0.1^2)

par(mfrow=c(2,1))
plot(p3_Metrop$Draws[,1],xlab="iteration",ylab=expression(alpha),type="l")
plot(p3_Metrop$Draws[,2],xlab="iteration",ylab=expression(beta),type="l")

par(mfrow=c(1,2))
acf(p3_Metrop$Draws[,1],main="Autocorrelation for alpha",lag.max=100)
acf(p3_Metrop$Draws[,2],main="Autocorrelation for beta",lag.max=100)

print(p3_Metrop$accept.ratio)
print(effectiveSize(p3_Metrop$Draws))
```

The convergence behavior for the second data set is clearly poorer compared to the first data set. For the second, the acceptance rate with the adjusted jumping rule is about 8%, there is high autocorrelation, and the effective sample size is about than 80 to 120 for each parameter.

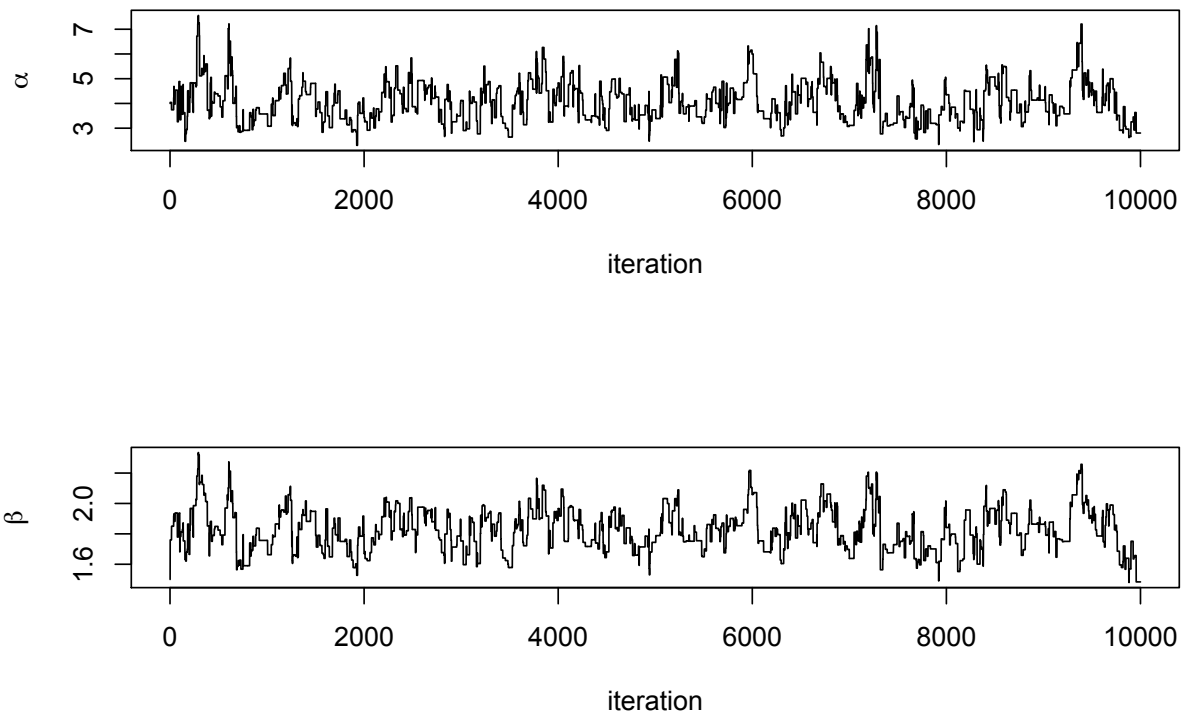


Trace plots: Random Walk Metropolis Sampler

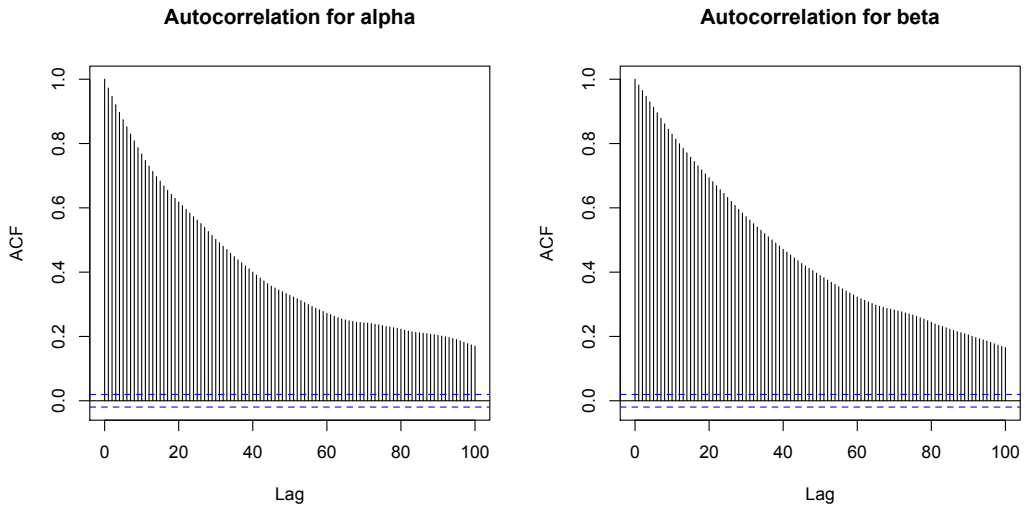


Autocorrelation plots: Random Walk Metropolis Sampler

```
# Running with Independence Sampler
p3_Indep  <- runIndep(spec=spec2)

par(mfrow=c(2,1))
plot(p3_Indep$Draws[,1],xlab="iteration",ylab=expression(alpha),type="l")
plot(p3_Indep$Draws[,2],xlab="iteration",ylab=expression(beta),type="l")

par(mfrow=c(1,2))
acf(p3_Indep$Draws[,1],main="Autocorrelation for alpha",lag.max=100)
acf(p3_Indep$Draws[,2],main="Autocorrelation for beta",lag.max=100)

print(p3_Indep$accept.ratio)
print(effectiveSize(p3_Indep$Draws))
```



Trace plots: Independence Sampler



Autocorrelation plots: Independence Sampler

We also find poorer performance with the Independence Sampler when using the second data set compared to the first data set. Here, the acceptance ratio is about 80%, the effective sample size is in the range of 1000 to 2000 for each parameter, and the autocorrelation is higher than we found previously.

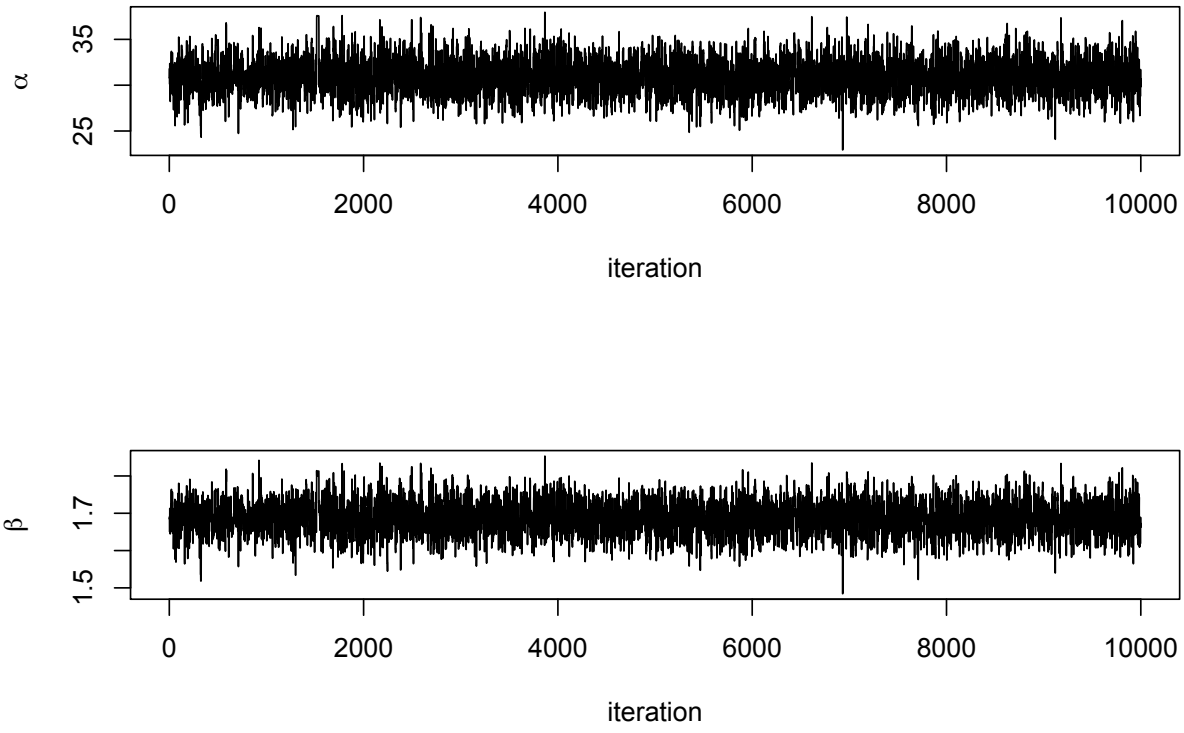
4. Write code for a new MCMC sampler that switches between your best Random Walk Metropolis sampler and your best Independence Sampler at each iteration. That is, write a sampler that uses the Random Walk Metropolis update in even numbered iterations and the Independence Sampler in odd numbered iterations. Run this mixed sampler to fit MODEL 1 to each of `spectral-1.txt` and `spectral-2.txt`. How does the mixed sampler compare with the samplers your derived in ran in parts 1, 2, and 3?

Solution:

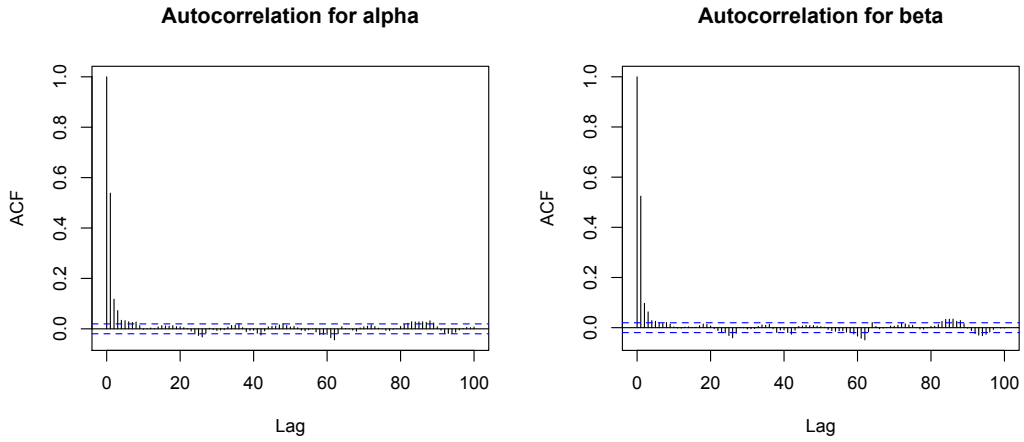
```
runMixed <- function(spec,draw.num=10000,jump.var){

E = spec[,1]; Y = spec[,2]
accept = 0;
Draws = matrix(NA, draw.num, 2)
log.lkhd = function( y=Y,x=E,alpha,beta ) ##up to a normalizing constant
        return( log(alpha)*sum(y) - beta*sum( y*log(x) ) - alpha*sum( x^(-beta) ) )

glm.fit = glm( Y~I(-log(E)),family=poisson( link="log" ) )
fit <- c( exp( glm.fit$coef[1] ), glm.fit$coef[2] )
vmat <- diag( c(fit[1],1) ) %*% vcov(glm.fit) %*% diag( c(fit[1],1) )
Draws[1,] = fit

## Prior for alpha and beta:: uniform on [0, 100]
cat("Iteration: 1")
```

```
for(i in 2:draw.num)
{
     if(i%%2==0){
            draw.star = rmvnorm( 1,mean=Draws[i-1,],sigma=jump.var*diag(c(1500,1)) )
            if(draw.star[1]<=0||draw.star[1]>=100||draw.star[2]<=0||draw.star[2]>=100){
                     Draws[i,] = Draws[i-1,]
             }else{
                     log.ratio = log.lkhd(alpha=draw.star[1],beta=draw.star[2] )-
                                          log.lkhd(alpha=Draws[i-1,1],beta=Draws[i-1,2] )
                     ratio = exp( min(log.ratio,100) )
                     temp = runif(1)
                     if(temp < min(ratio,1) ){
                              Draws[i,] = draw.star
                              accept = accept + 1
                      }else{
                              Draws[i,] = Draws[i-1,]
                      }
             }
     }else{
             draw.star = rmvnorm( 1, mean=fit, sigma=vmat )

             log.ratio = ( log.lkhd(alpha=draw.star[1],beta=draw.star[2] )-
                                  log.lkhd(alpha=Draws[i-1,1],beta=Draws[i-1,2] ) )-
                                  ( log(dmvnorm(draw.star,mean=fit,sigma=vmat ) )-
                                   log(dmvnorm(Draws[i-1,],mean=fit,sigma=vmat ) ) )

             ratio = exp( min(log.ratio,100) )
             temp = runif(1)
             if(temp < min(ratio,1) ){
                     Draws[i,] = draw.star
                     accept = accept + 1
             }else{
                     Draws[i,] = Draws[i-1,]
             }
     }
     if(i %% 1000 == 0) cat(", ",i)
}
output <- list(Draws,accept/draw.num)
names(output) <- c("Draws","accept.ratio")
return(output)
}

p4_spec1 <- runMixed(spec=spec1,jump.var=0.035^2)

par(mfrow=c(2,1))
plot(p4_spec1$Draws[,1],xlab="iteration",ylab=expression(alpha),type="l")
plot(p4_spec1$Draws[,2],xlab="iteration",ylab=expression(beta),type="l")

par(mfrow=c(1,2))
acf(p4_spec1$Draws[,1],main="Autocorrelation for alpha",lag.max=100)
acf(p4_spec1$Draws[,2],main="Autocorrelation for beta",lag.max=100)
```

```
print(p4_spec1$accept.ratio)
print(effectiveSize(p4_spec1$Draws))
```

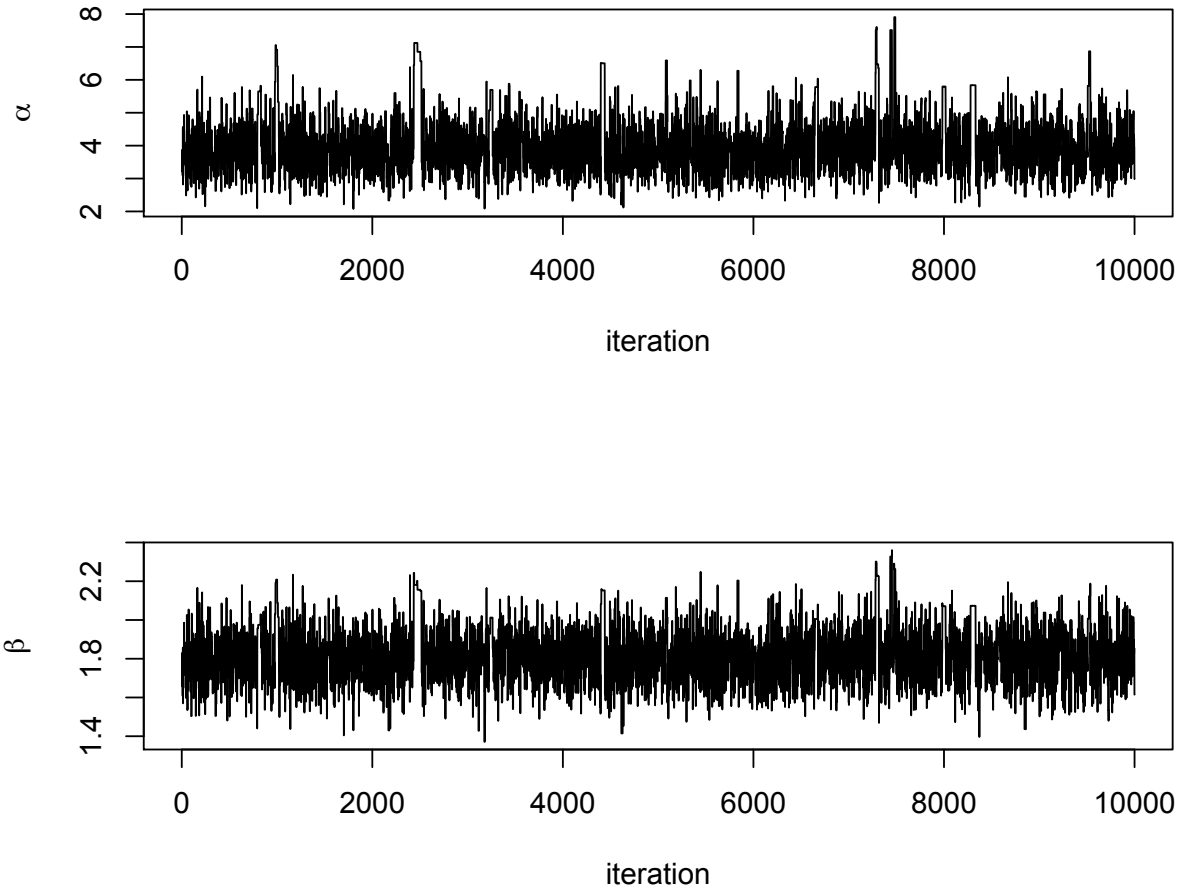


Trace plots: spectrum 1

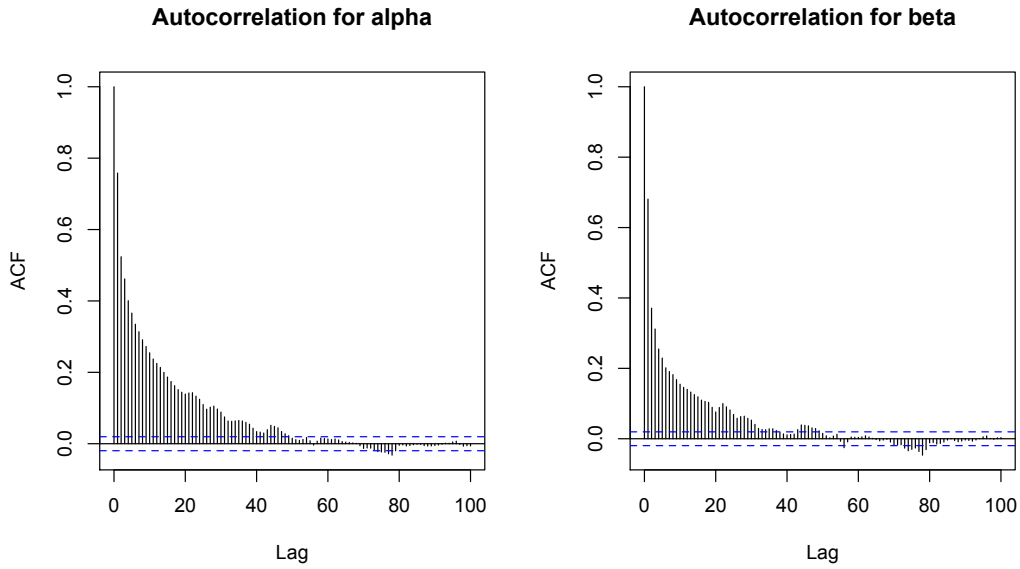


Autocorrelation plots: spectrum 1

```
p4_spec2 <- runMixed(spec=spec2,jump.var=0.1^2)

par(mfrow=c(2,1))
plot(p4_spec2$Draws[,1],xlab="iteration",ylab=expression(alpha),type="l")
plot(p4_spec2$Draws[,2],xlab="iteration",ylab=expression(beta),type="l")

par(mfrow=c(1,2))
acf(p4_spec2$Draws[,1],main="Autocorrelation for alpha",lag.max=100)
acf(p4_spec2$Draws[,2],main="Autocorrelation for beta",lag.max=100)

print(p4_spec2$accept.ratio)
print(effectiveSize(p4_spec2$Draws))
```



Trace plots: spectrum 2



Autocorrelation plots: spectrum 2

The mixed sampler tends to perform somewhere in between the performance of the Random Walk Metropolis Sampler and the Independence Sampler.

5. Based on your findings in parts 1–4. Write an efficient sampler to fit MODEL 3 to the dataset `spectral-4.txt`. You may want to use the method of Data Augmentation to handle the background contamination.

Solution:

```
arf = read.table("data/eff-area.txt",header=T)
spec4 <- read.table("data/spectral-4.txt",header=TRUE )


E = spec4[,1]; Y = spec4[,2];

draw.num = 100000
Draws = matrix(NA, draw.num, 4)
Draws[1,] = c(50, 1.69, 1, 1)
k = 0.04^2; # may need to adjust for more efficient sampling...
accept = 0;
Y.back = Y;

log.lkhd = function( y.source,x=E,alpha,beta,omega ) ##up to a normalizing constant
{
```

```
         return( log(alpha)*sum(y.source) - beta*sum( y.source*log(x) ) -
               omega*sum( y.source/x ) - alpha*sum( x^(-beta)*exp(-omega/x)*arf[,2]/100 ) )
}
cat("Iteration: 1")
for(i in 2:draw.num)
{
        #### Draw Y_back
        Y.back = rbinom( length(Y), Y, Draws[i-1,4] / ( Draws[i-1,1]*E^(-Draws[i-1,2])*
                                        exp(-Draws[i-1,3]/E)*arf[,2]/100 + Draws[i-1,4] ) )
        Draws[i,4] = rgamma(1, shape=(sum(Y.back)+1), scale=1/length(Y) )

        draw.star = rmvnorm( 1,mean=Draws[i-1,1:3],sigma=k*diag(c(2000,1,10) ) )
        if(draw.star[1]<=0||draw.star[1]>=100||draw.star[2]
                              <=0||draw.star[2]>=100||draw.star[3]<=0||draw.star[3]>=100){
                Draws[i,1:3] = Draws[i-1,1:3]
         }else{
                log.ratio = log.lkhd(Y-Y.back,E,draw.star[1],draw.star[2],draw.star[3] )-
                           log.lkhd(Y-Y.back,E,Draws[i-1,1],Draws[i-1,2],Draws[i-1,3] )
                ratio = exp( min(log.ratio,100) )
                temp = runif(1)
                if(temp < min(ratio,1) ){
                        Draws[i,1:3] = draw.star
                        accept = accept + 1
                }else{
                        Draws[i,1:3] = Draws[i-1,1:3]
                }
         }
         if(i == 2) cat("Iteration: ")
         if(i %% 5000 == 0) cat(", ",i)
}
```

6. *Bonus:* Add five counts to `Counts[200]`, `Counts[201]`, and `Counts[202]` in `spectral-1.txt`. Call the new file `spectral-3.txt`. Now write an MCMC sampler to fit MODEL 2 to `spectral-3.txt`. Use the strategy outlined in the notes to fit the model using Data Augmentation and Metropolis within Gibbs. You might try repeating your run with a weaker or stronger line, or with the low-count spectral model in `spectral-2.txt`.

   Solution: See slides for an outline of the solution.