Bayesian_lab3

Akshath Srinivas (akssr921),Yaning Wang (yanwa579)

Question 1

(a)

```r
#1 a
#loading data
data<-readRDS('Precipitation.rds')
ln_y<-log(data)
n<-length(ln_y)

#setting prior parameter values
mu0<-0
tau0<-1
nu0<-1
sigma0<-1

#mean and variance from data
x_bar<-mean(ln_y)
sigma2<-var(ln_y)

#initializing iterations and mu and sigma 2 samples(empty vector to store)
ndraws <- 1000
mu_samples<-rep(NA,ndraws)
sigma2_samples <- numeric(ndraws)

#gibbs sampling for ndraws(number of iteration)
for (i in 1:ndraws) {

  #calculating tau_n and mu_n to sample mu from normal distribution
  taun<-1/((n/sigma2)+(1/tau0))
  W<-(n/sigma2)/((n/sigma2)+(1/tau0))
  mun<-W*x_bar+(1-W)*mu0
  mu <- rnorm(1, mun, sqrt(taun))
  mu_samples[i] <- mu

  #calculating nu_n and sigma_n to sample sigma^2 from inverse chi distribution
  nun<-nu0+n
  sigman<-(nu0*sigma0+sum((ln_y-mu)^2))/nun
  sigma2<-LaplacesDemon::rinvchisq(1,df=nun,scale=sigman)
  sigma2_samples[i]<-sigma2
}

# Calculate the Inefficiency Factors (IFs) for mu_samples
```

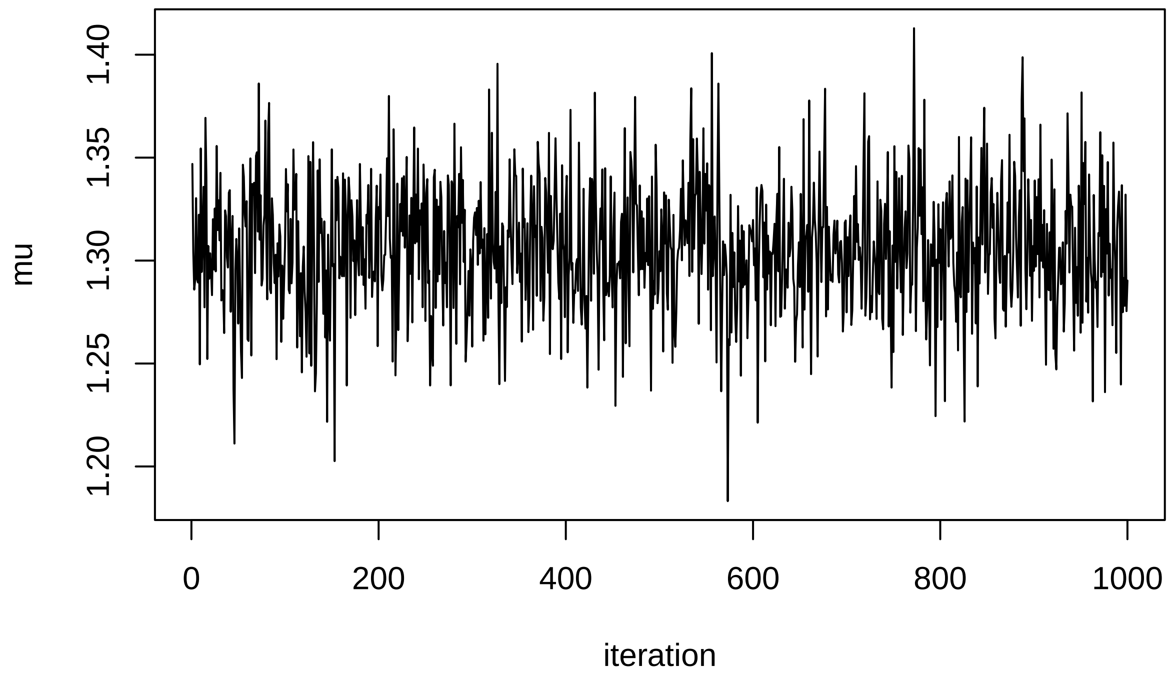
```
acf_mu_IF <- 1+2*sum(acf(mu_samples,plot = FALSE)$acf[-1])

# Calculate the Inefficiency Factors (IFs) for sigma^2_samples
acf_sigma_IF <- 1+2*sum(acf(sigma2_samples,plot = FALSE)$acf[-1])

cat(paste("The Inefficiency Factor of mu", acf_mu_IF))
```

```
## The Inefficiency Factor of mu 0.948531561328679
```

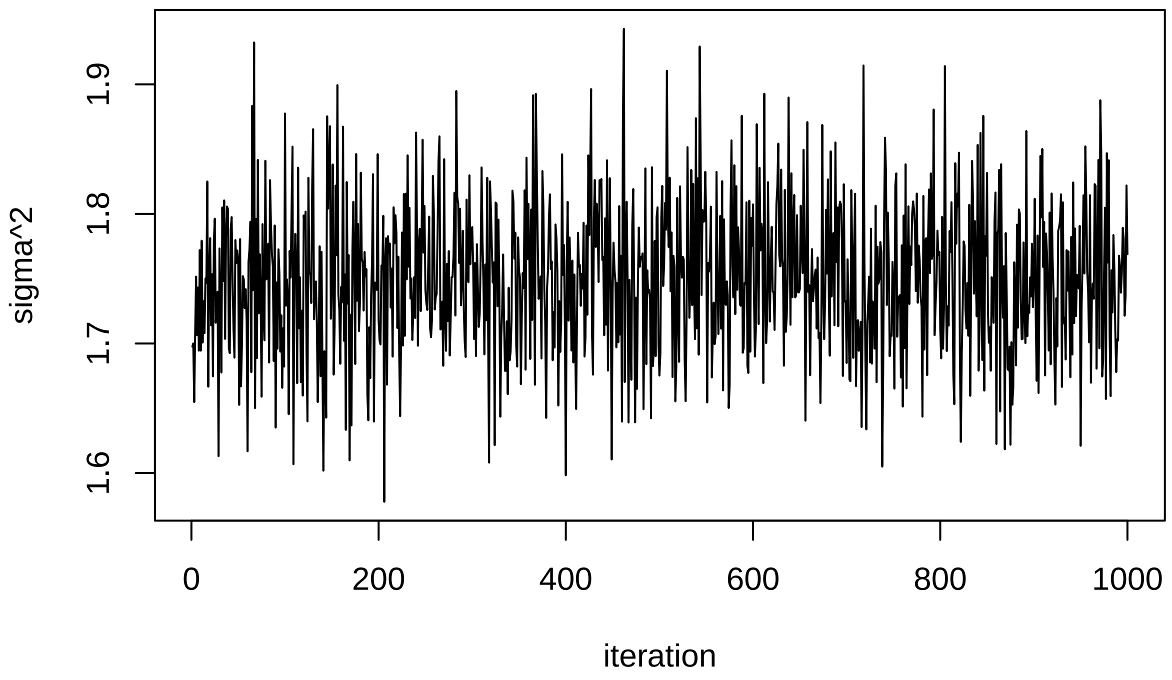
```
cat(paste("The Inefficiency Factor of sigma^2", acf_sigma_IF))
```

```
## The Inefficiency Factor of sigma^2 1.1159409756789
```

```
#plotting drawn mu and sigma^2
plot(mu_samples, type="l", ylab="mu", xlab="iteration")
```



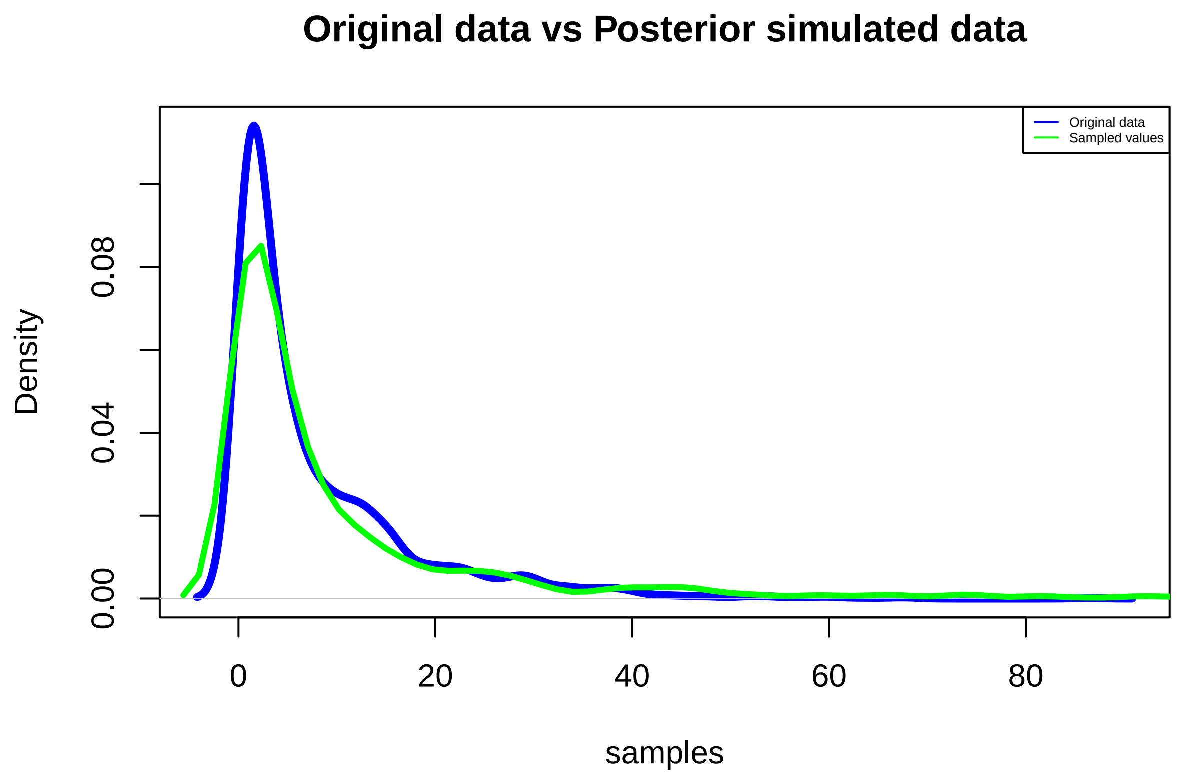
```
plot(sigma2_samples, type="l", ylab="sigma^2", xlab="iteration")
```



From the results above,the IF values of μ and σ^2 are small(it is able to explore the posterior distribution efficiently) and from the plot we can observe that both the parameters converge.

(b)

```
#1b
#plotting the kernel density of data
plot(density(exp(ln_y)),lwd = 4, col='blue',main = 'Original data vs Posterior simulated data',xlab='sa

#simulating posterior predictive samples from the parameteres we got from above question
simulated_posterior<-numeric(length(sigma2_samples))
for (i in 1:length(sigma2_samples)) {
  simulated_posterior[i]<-rnorm(1,mu_samples[i],sigma2_samples[i])
}

#drawing the posterior predictive density line on the existing plot
lines(density(exp(simulated_posterior)),col='green',lwd=3)

legend("topright",legend=c("Original data",'Sampled values'),col=c('blue','green'),
       lty = c(1, 1), cex=0.4)
```



From the plot above, we can conclude that density curve of sampled values matches pretty well to the original data density curve.

Question 2

(a)

```
# Load data set
eBay <- read.table("eBayNumberOfBidderData.dat", header = TRUE)
# delete the covariate const column
data<-eBay[,-2]
model<-glm(nBids~., family = poisson,data=data)
summary(model)
```

```
##
## Call:
## glm(formula = nBids ~ ., family = poisson, data = data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.5800  -0.7222  -0.0441   0.5269   2.4605
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.07244    0.03077  34.848  < 2e-16 ***
## PowerSeller -0.02054    0.03678  -0.558   0.5765
## VerifyID    -0.39452    0.09243  -4.268 1.97e-05 ***
## Sealed       0.44384    0.05056   8.778  < 2e-16 ***
## Minblem     -0.05220    0.06020  -0.867   0.3859    
## MajBlem     -0.22087    0.09144  -2.416   0.0157 *  
## LargNeg      0.07067    0.05633   1.255   0.2096    
## LogBook     -0.12068    0.02896  -4.166 3.09e-05 ***
## MinBidShare -1.89410    0.07124 -26.588  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## (Dispersion parameter for poisson family taken to be 1)
## 
##     Null deviance: 2151.28  on 999  degrees of freedom
## Residual deviance:  867.47  on 991  degrees of freedom
## AIC: 3610.3
## 
## Number of Fisher Scoring iterations: 5
```

From the result we got, base on the P value, we can conclude that:VerifyID,Sealed,MajBlem,LogBook,MinBidShare are significant. Their p values are all in the interval[0,0.05].

(b)

From the question we know the prior of y_i is:

$y_i|\beta \sim Possion[\lambda = exp(x_i^T\beta)]$

To derive the expression for the posterior $p(\beta|y)$, according to Bayesian theorem. We should get the likelihood of the possion model. Based on the model, the likelihood is :

$\prod_{i=1}^{n} p(y_i|\beta) = \prod_{i=1}^{n} \frac{\lambda^{y_i} * exp[-\lambda]}{y_i!} = \frac{\lambda^{\sum_{i=1}^{n} y_i} * exp[-n\lambda]}{\prod_{i=1}^{n} y_i!}$

now get the log likelihood expression:

$log\frac{\lambda^{\sum_{i=1}^{n} y_i} * exp[-n\lambda]}{\prod_{i=1}^{n} y_i!} = \sum_{i=1}^{n} y_i * log\lambda - n\lambda - \sum_{i=1}^{n} log(y_i!) = \sum_{i=1}^{n}(y_i * log\lambda - \lambda) - \sum_{i=1}^{n} log(y_i!)$

```r
# Extract the response variable and features
y<-as.matrix(eBay[,1])
x<-as.matrix(eBay[,-1])
# get the number of observations and features
n<-length(y)
n_fea<-dim(x)[2]
#get the prior variables
mu<-as.matrix(rep(0,n_fea))
sigma<-100*solve((t(x)%*%x))
#return log posterior for the poisson regression
log_postlogis <- function(betas){
  lambda <-exp(x%*%betas)
  #use the log_likelihood formula we get above
  logLik <- sum(y*log(lambda) - lambda)-sum(log(factorial(y)))
  #write the prior formula
  logPrior <- dmvnorm(t(betas), mean=mu, sigma=sigma, log=TRUE)
  return(logLik + logPrior)
}
#initial betas
init_betas <- matrix(0,n_fea,1)
```

```r
#use optim function to get the posterior mode value and hessian value
OptimRes <- optim(par=init_betas,
                  fn=log_postlogis,
                  gr=NULL,
                  method=c("BFGS"),
                  control=list(fnscale=-1),
                  hessian=TRUE)

#Naming the coefficient by features
Xnames<-colnames(x)
posterior_mode<-data.frame(feature_name=Xnames,beta_mode=OptimRes$par)
print('The posterior mode is:')
```

```
## [1] "The posterior mode is:"
```

```r
print(posterior_mode)
```

```
##   feature_name   beta_mode
## 1        Const  1.06984118
## 2  PowerSeller -0.02051246
## 3     VerifyID -0.39300599
## 4       Sealed  0.44355549
## 5      Minblem -0.05246627
## 6      MajBlem -0.22123840
## 7      LargNeg  0.07069683
## 8      LogBook -0.12021767
## 9  MinBidShare -1.89198501
```

```r
approxpost_variance <- solve(-OptimRes$hessian)
print('The variance of the posterior is')
```

```
## [1] "The variance of the posterior is"
```

```r
print(approxpost_variance)
```

```
##                [,1]          [,2]          [,3]          [,4]          [,5]
##  [1,]  9.454625e-04 -7.138972e-04 -2.741517e-04 -2.709016e-04 -4.454554e-04
##  [2,] -7.138972e-04  1.353076e-03  4.024623e-05 -2.948968e-04  1.142960e-04
##  [3,] -2.741517e-04  4.024623e-05  8.515360e-03 -7.824886e-04 -1.013613e-04
##  [4,] -2.709016e-04 -2.948968e-04 -7.824886e-04  2.557778e-03  3.577158e-04
##  [5,] -4.454554e-04  1.142960e-04 -1.013613e-04  3.577158e-04  3.624606e-03
##  [6,] -2.772239e-04 -2.082668e-04  2.282539e-04  4.532308e-04  3.492353e-04
##  [7,] -5.128351e-04  2.801777e-04  3.313568e-04  3.376467e-04  5.844006e-05
##  [8,]  6.436765e-05  1.181852e-04 -3.191869e-04 -1.311025e-04  5.854011e-05
##  [9,]  1.109935e-03 -5.685706e-04 -4.292828e-04 -5.759169e-05 -6.437066e-05
##                [,6]          [,7]          [,8]          [,9]
##  [1,] -2.772239e-04 -5.128351e-04  6.436765e-05  1.109935e-03
##  [2,] -2.082668e-04  2.801777e-04  1.181852e-04 -5.685706e-04
##  [3,]  2.282539e-04  3.313568e-04 -3.191869e-04 -4.292828e-04
##  [4,]  4.532308e-04  3.376467e-04 -1.311025e-04 -5.759169e-05
##  [5,]  3.492353e-04  5.844006e-05  5.854011e-05 -6.437066e-05
##  [6,]  8.365059e-03  4.048644e-04 -8.975843e-05  2.622264e-04
##  [7,]  4.048644e-04  3.175060e-03 -2.541751e-04 -1.063169e-04
##  [8,] -8.975843e-05 -2.541751e-04  8.384703e-04  1.037428e-03
##  [9,]  2.622264e-04 -1.063169e-04  1.037428e-03  5.054757e-03
```

(c)

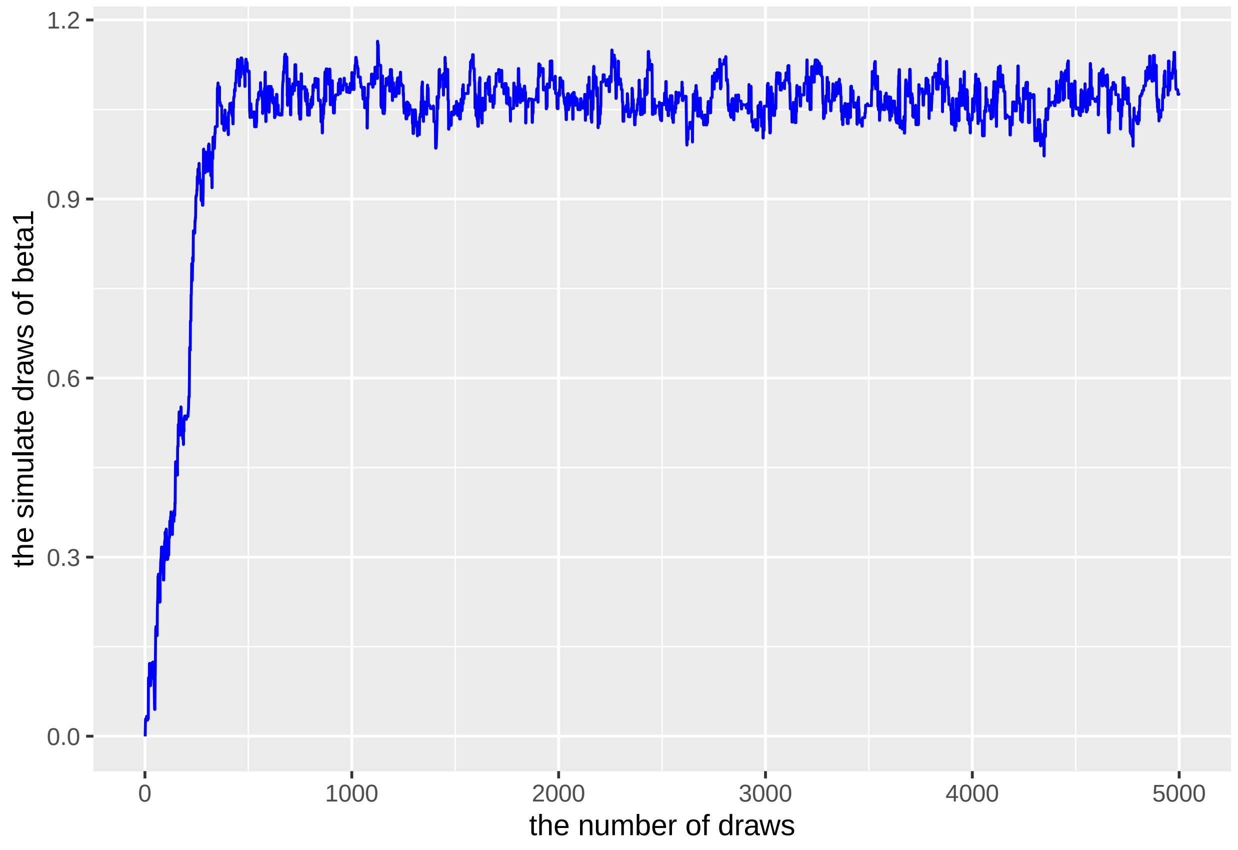
```
#set the number of draws
m<-5000
RWMSampler<-function(c){
  #set the start betas value
  sample_beta<-matrix(0,nrow = m,ncol = n_fea)
  betas<-sample_beta[1,]
  variance<-c*approxpost_variance
  for(i in 1:m){
    #generate new betas
    betas_new<-rmvnorm(1, mean=betas, sigma=variance)
    #change betas_new vector to matrix form with 9 rows and compute the posterior density
    postdensi_new<-log_postlogis(t(betas_new))
    postdensi_old<-log_postlogis(betas)
    #compute the ratio of posterior densities
    ratio<-exp(postdensi_new-postdensi_old)
    #compute the acceptance probability
    a<-min(1,ratio)
    u<-runif(1)
    if(u<=a){
      sample_beta[i,]<-betas_new
      }
    else{
      sample_beta[i,]<-betas
    }
    betas<-sample_beta[i,]
  }
  return(sample_beta)
}


# Now we should compare the results with the approximate results in b
RWMdraws<-RWMSampler(0.5)
mean_betas<-colMeans(RWMdraws)
compare_data<-data.frame(feature_name=Xnames,approx_b=OptimRes$par,RWM_c=mean_betas)
print(compare_data)
```

```
##   feature_name    approx_b       RWM_c
## 1        Const  1.06984118  1.03358991
## 2  PowerSeller -0.02051246 -0.02300957
## 3     VerifyID -0.39300599 -0.38169885
## 4       Sealed  0.44355549  0.43918738
## 5      Minblem -0.05246627 -0.05256232
## 6      MajBlem -0.22123840 -0.19426690
## 7      LargNeg  0.07069683  0.05985966
## 8      LogBook -0.12021767 -0.12619819
## 9  MinBidShare -1.89198501 -1.83513446
```

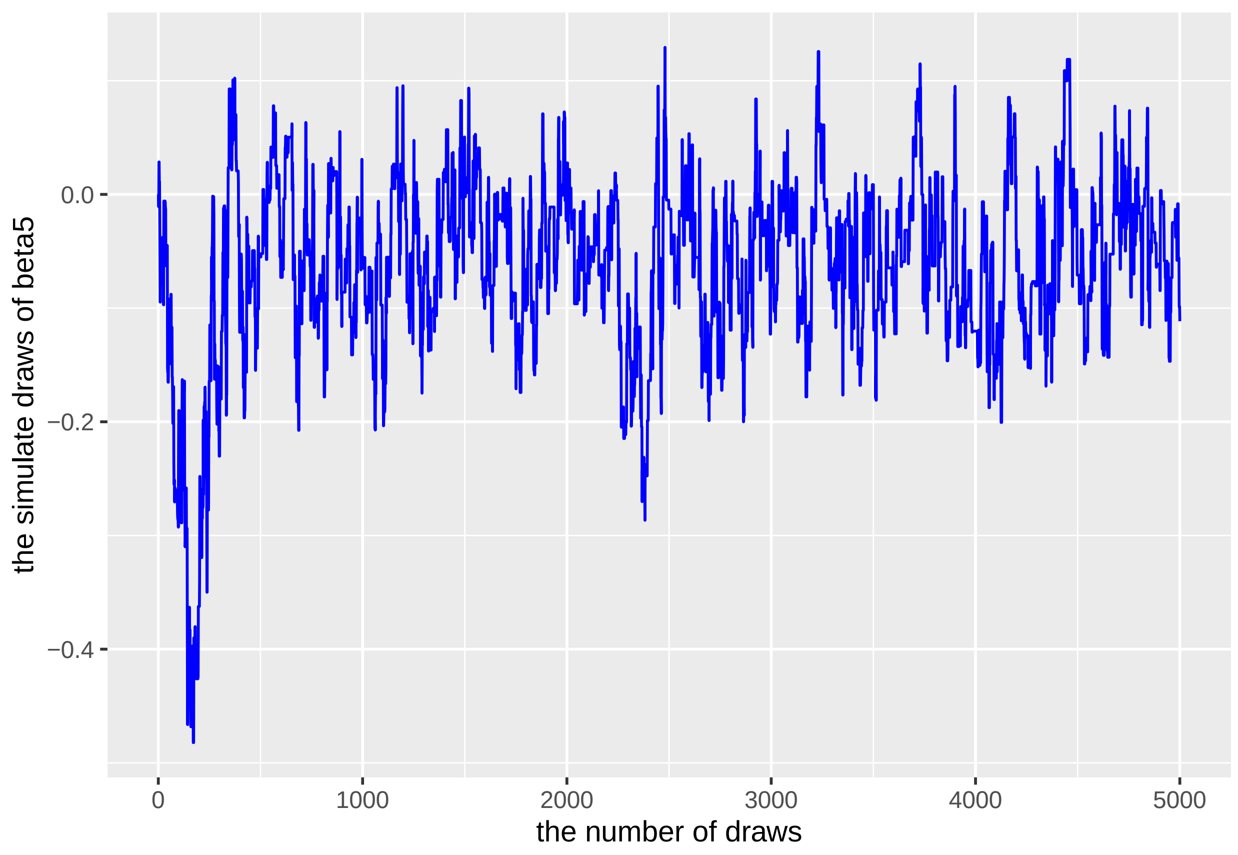
From the table above, we can see the results obtained by these two methods are very close.

Now we should to check if they are convergence. We will check it below. The values we simulate from the actual posterior of beta using the Metropolis algorithm has 9 dimensions, we randomly use the first and second dimension(to save place here) to check.

```r
data<-data.frame(number=c(1:m),beta1=RWMSampler(0.5)[,1],beta5=RWMSampler(0.5)[,5])

ggplot(data)+
  geom_line(aes(x=number,y=beta1),colour="blue")+
  xlab("the number of draws")+
  ylab("the simulate draws of beta1")
```



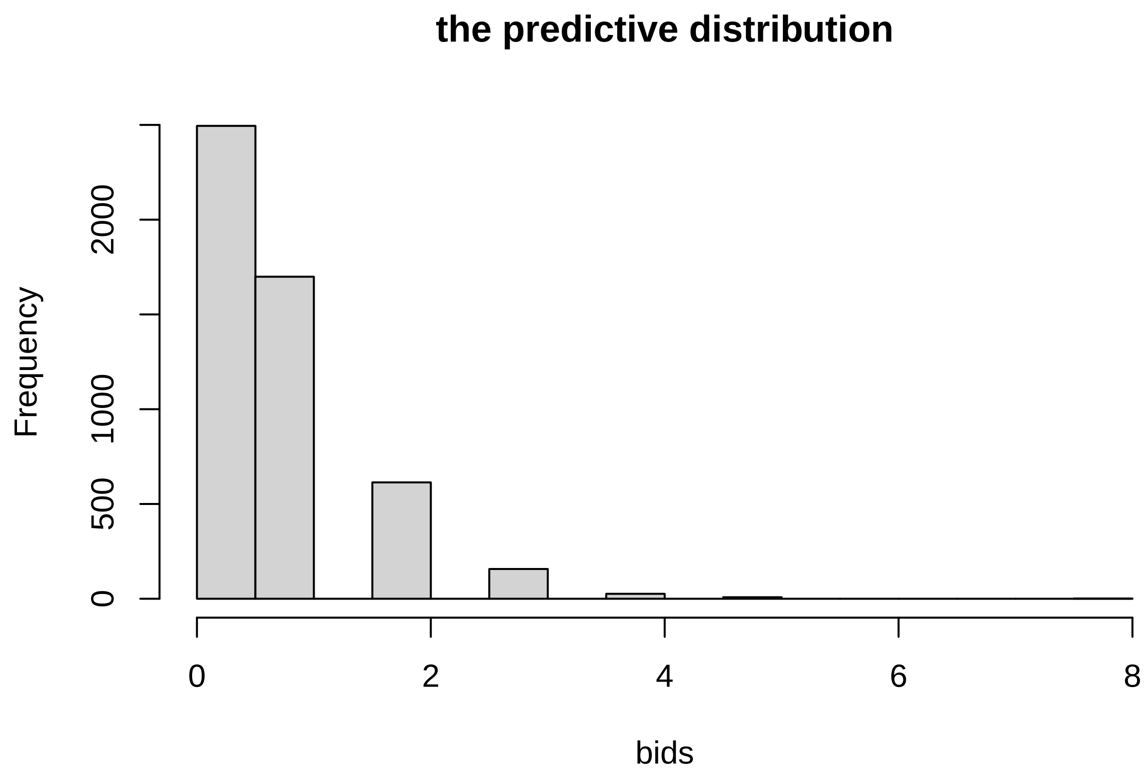
```r
ggplot(data)+
  geom_line(aes(x=number,y=beta5),colour="blue")+
  xlab("the number of draws")+
  ylab("the simulate draws of beta5")
```



From the plot we can see, the values of betas are converging, They are all around some certain values.

(d)

```
#RWMdraws is the samples we get from c,and m is the rows of RWMdraws
xnew_withconst<-matrix(c(1,1,0,1,0,1,0,1.2,0.8),1,n_fea)
lambda<-exp(xnew_withconst%*%t(RWMdraws))
#use the build in function rpois to simulate the number of bidders
bids<-c()
for(i in 1:m){
  bid<-rpois(1, lambda[i])
  bids[i]<-bid
}
#plot the predictive distribution
hist(bids,main="the predictive distribution")
```



```r
cat("The probability of no bidders in this new auction is ",length(which(bids==0))/m)
```

```
## The probability of no bidders in this new auction is  0.499
```

Question 3

(a)

```r
SimuData<-function(mu,phi,sigma2,T){
  sample<-c()
  x<-mu
  sample[1]<-x
  for(t in 2:T){
    #sample from normal distribution to get epsilon value
    epsilon<-rnorm(1,mean=0,sd=sqrt(sigma2))
    #base on the expression to get the new sample
    x_new<-mu+phi*(x-mu)+epsilon
    #collect the samples
    sample<-append(sample,x_new)
    x<-x_new
  }
  return(sample)
}

#get some phi values between -1 and 1
phis<-seq(-1,1,by=0.4)
```

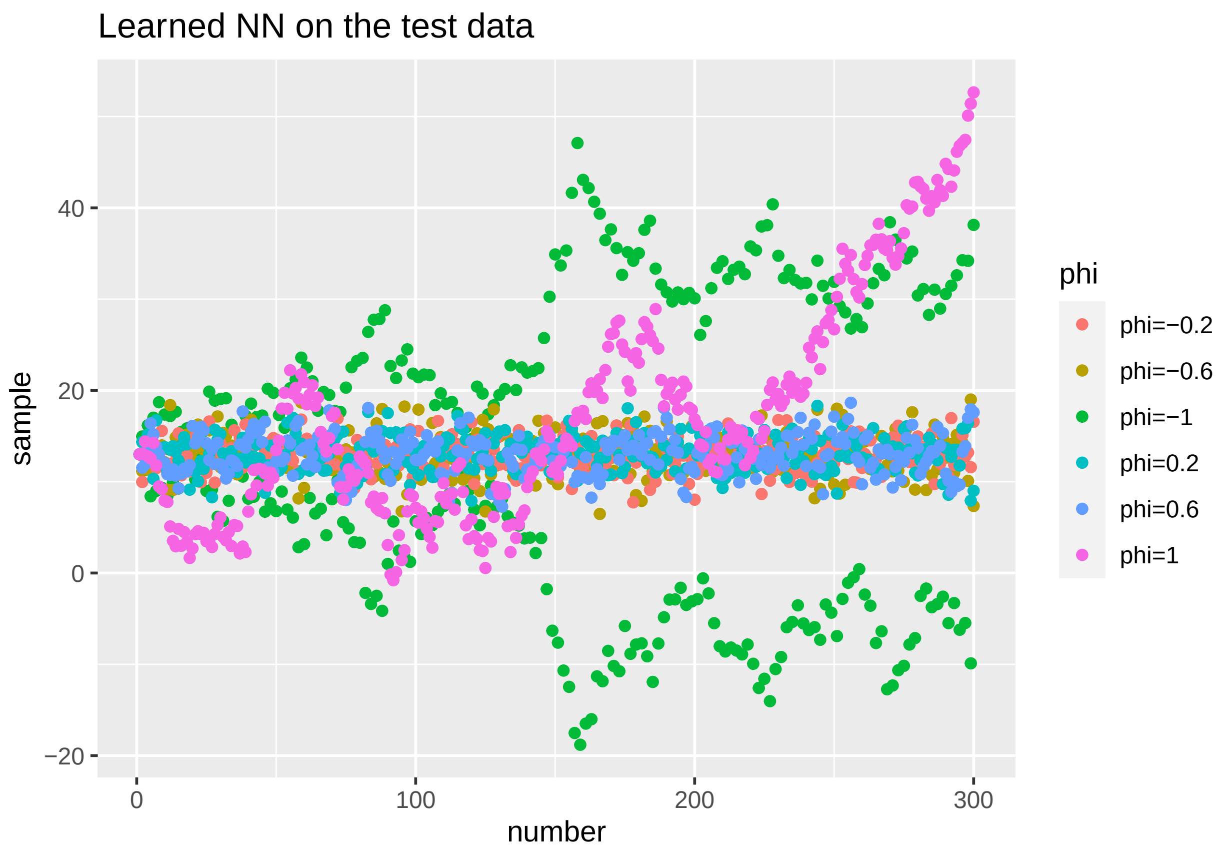
```r
n<-length(phis)
T<-300
data<-data.frame()
mean_sample<-c()
val_sample<-c()
#simulate samples for different phi values
for(i in 1:n){
  sample<-SimuData(mu=13,phi=phis[i],sigma2=3,T=T)
  mean_sample<-c(mean_sample,mean(sample))
  val_sample<-c(val_sample,var(sample))
  name<-rep(paste0("phi=",phis[i]),T)
  samples<-data.frame(number=c(1:300),sample=sample,phi=name)
  data<-rbind(data,samples)
}

cat("The mean of the samples base on different phi is ","\n",mean_sample, "\n")
```

```
## The mean of the samples base on different phi is
##  13.15773 13.02705 12.99739 13.07878 12.9717 17.40895
```

```r
cat("The variance of the samples base on different phi is ","\n",val_sample, "\n")
```

```
## The variance of the samples base on different phi is
##  248.3765 5.031863 3.062339 3.622591 4.80384 144.0543
```

```r
#plot the samples according to different phi
ggplot(data=data)+
  geom_point(aes(x=number,y=sample,color=phi))+
  labs(title="Learned NN on the test data")
```



From the plot, mean and variance we got, the samples simulated based on phi between -1 and 1(not include) are very close.This also proves what is said in the question that in this interval the AR process is stationary.

(b)

i

```
#draw samples from the function
samplex<-SimuData(mu=13,phi=0.2,sigma2=3,T=T)
sampley<-SimuData(mu=13,phi=0.95,sigma2=3,T=T)

StanModel <- '
data{
  int <lower=0> N;  //number of observations,it should be positive
  vector[N] y;  //obeservations at time N
}
parameters{
  real mu;
  real <lower=-1,upper=1> phi; //the constraint for phi to ensure time series stationary
  real <lower=0> sigma2; //the variance should be positive(from teachers slide)
}
model{
  mu ~ normal(0,100); // Normal prior(from teachers slide)
  phi ~ uniform(-1,1); // uniform prior
  sigma2 ~ scaled_inv_chi_square(1,2); // Scaled-inv-chiprior(from teachers slide)
  for (n in 1:N){
    y[n] ~ normal(mu + phi * (y[n-1]-mu), sqrt(sigma2));
  }
}'

datax <- list(N=300, y=samplex)
datay <- list(N=300, y=sampley)
# number of warmup iterations per chain
nwarmup <- 1000
# total number of iterations per chain
niter <- 2000
# number of Markov chains
nchains<-4
fitx <- stan(model_code=StanModel,data=datax, warmup=nwarmup,iter=niter,chains=nchains)
```

```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.0001 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 1 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.137 seconds (Warm-up)
## Chain 1:                0.123 seconds (Sampling)
## Chain 1:                0.26 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 2.3e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.23 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.128 seconds (Warm-up)
## Chain 2:                0.139 seconds (Sampling)
## Chain 2:                0.267 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 2.5e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.25 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.829 seconds (Warm-up)
## Chain 3:                0.134 seconds (Sampling)
## Chain 3:                0.963 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 2.6e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.26 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.144 seconds (Warm-up)
## Chain 4:                0.139 seconds (Sampling)
## Chain 4:                0.283 seconds (Total)
## Chain 4:
```

```
fity <- stan(model_code=StanModel,data=datay, warmup=nwarmup,iter=niter,chains=nchains)
```

```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 2.9e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.29 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.183 seconds (Warm-up)
## Chain 1:                0.159 seconds (Sampling)
## Chain 1:                0.342 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 2.1e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.21 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
```

```
## Chain 2:  Elapsed Time: 0.274 seconds (Warm-up)
## Chain 2:                0.158 seconds (Sampling)
## Chain 2:                0.432 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 2.3e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.23 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.21 seconds (Warm-up)
## Chain 3:                0.165 seconds (Sampling)
## Chain 3:                0.375 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 2.5e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.25 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.226 seconds (Warm-up)
## Chain 4:                0.213 seconds (Sampling)
## Chain 4:                0.439 seconds (Total)
## Chain 4:
```

```r
print("The required values using data x are:")
```

```
## [1] "The required values using data x are:"
```

```r
print(summary(fitx)$summary[1:3,c(1,4,8,9)])
```

```
##              mean        2.5%     97.5%    n_eff
## mu     13.0907328 12.84619811 13.324888 4083.398
## phi     0.1882598  0.08363333  0.289028 3251.080
## sigma2  2.7061517  2.32365977  3.150904 3675.536
```

```r
print("The required values using data y are:")
```

```
## [1] "The required values using data y are:"
```

```r
print(summary(fity)$summary[1:3,c(1,4,8,9)])
```

```
##              mean      2.5%      97.5%    n_eff
## mu     11.6195899 8.6196504 15.0365377 1096.469
## phi     0.9211184 0.8752915  0.9696354 1779.811
## sigma2  2.9691471 2.5075617  3.5094848 1226.889
```

```r
#the post means compared with the true mean using x data
post_valuex<-as.data.frame(summary(fitx)$summary[1:3,c(1,4,8,9)])
post_valuex["true_value"]<-c(13,0.2,3)
print(post_valuex)
```

```
##              mean        2.5%     97.5%    n_eff true_value
## mu     13.0907328 12.84619811 13.324888 4083.398       13.0
## phi     0.1882598  0.08363333  0.289028 3251.080        0.2
## sigma2  2.7061517  2.32365977  3.150904 3675.536        3.0
```

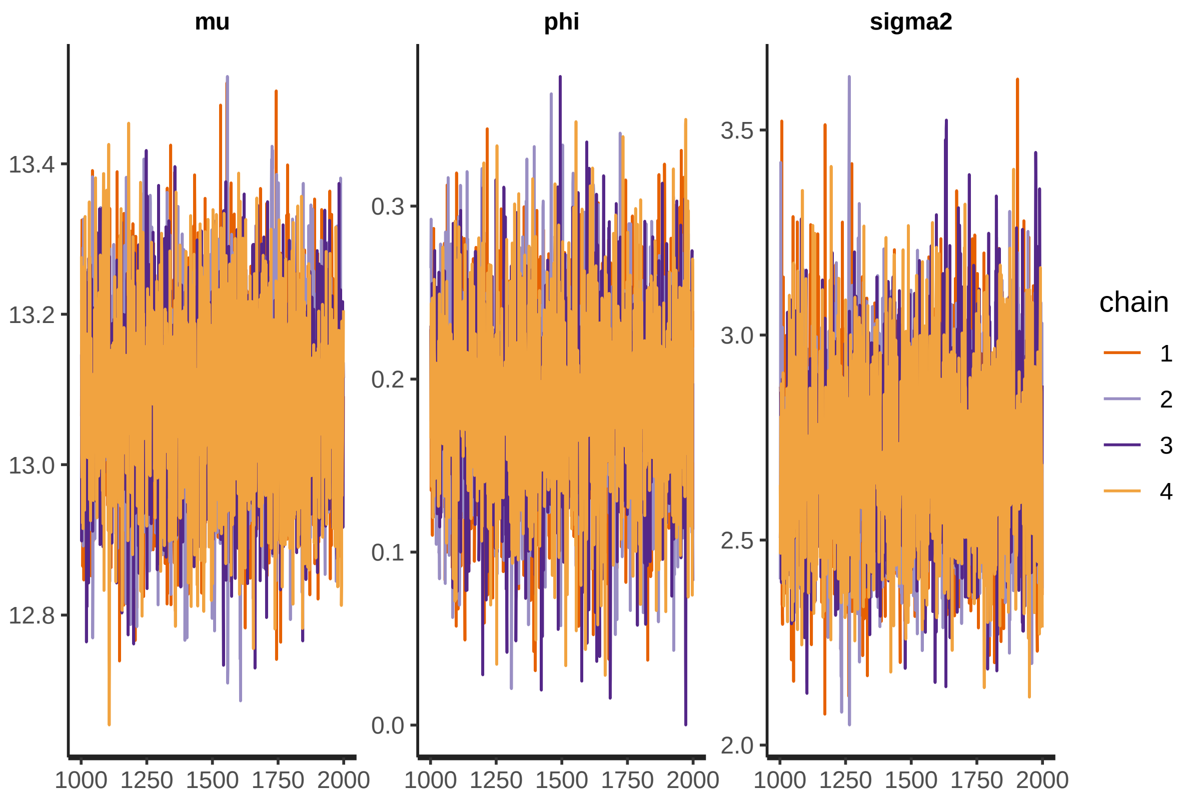
```r
#the post means compared with the true mean using y data
post_valuey<-as.data.frame(summary(fity)$summary[1:3,c(1,4,8,9)])
post_valuey["true_value"]<-c(13,0.95,3)
print(post_valuey)
```

```
##              mean      2.5%      97.5%    n_eff true_value
## mu     11.6195899 8.6196504 15.0365377 1096.469      13.00
## phi     0.9211184 0.8752915  0.9696354 1779.811       0.95
## sigma2  2.9691471 2.5075617  3.5094848 1226.889       3.00
```

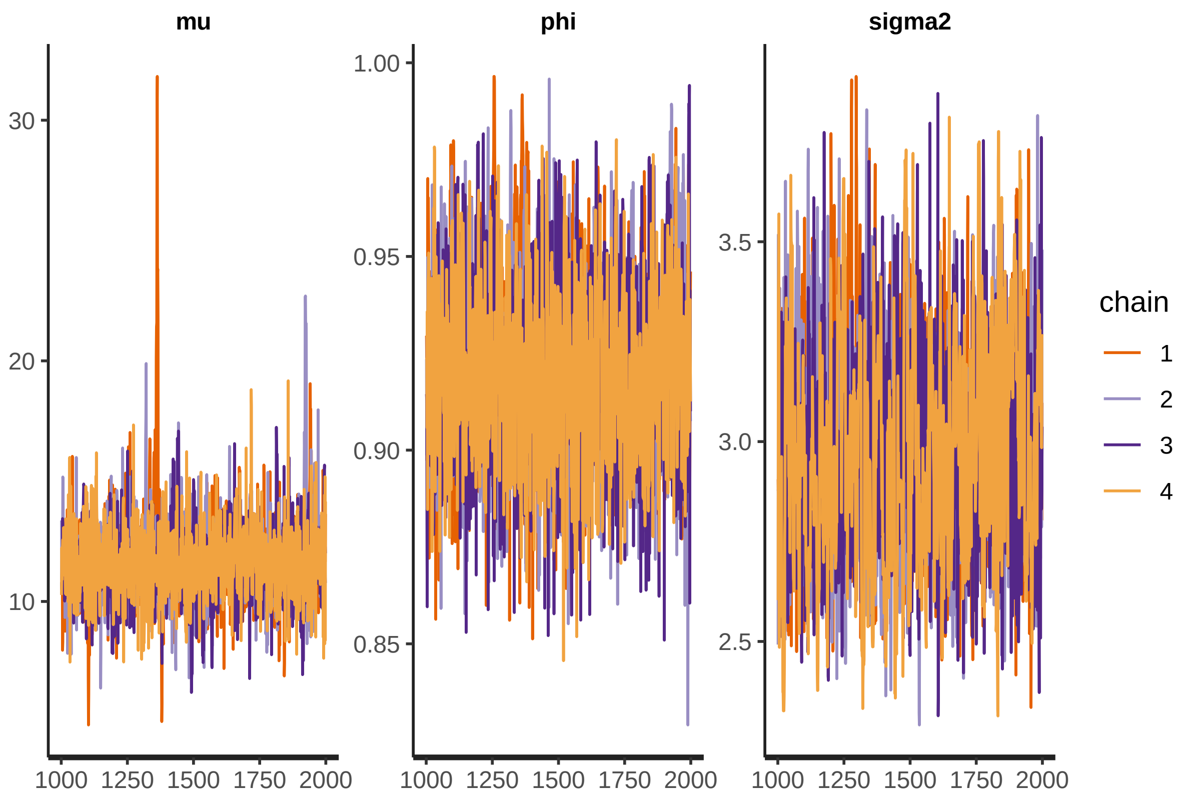
From the results above, we can see the true values of the three parameters are very close to the posterior mean values.And the true values are all in the 95% credible interval which are between 2.5% and 97.5% (two equal tails).So using this way we can estimate the true value.

ii

```r
# Do automatic traceplots of all chains of model fitx
traceplot(fitx)
```



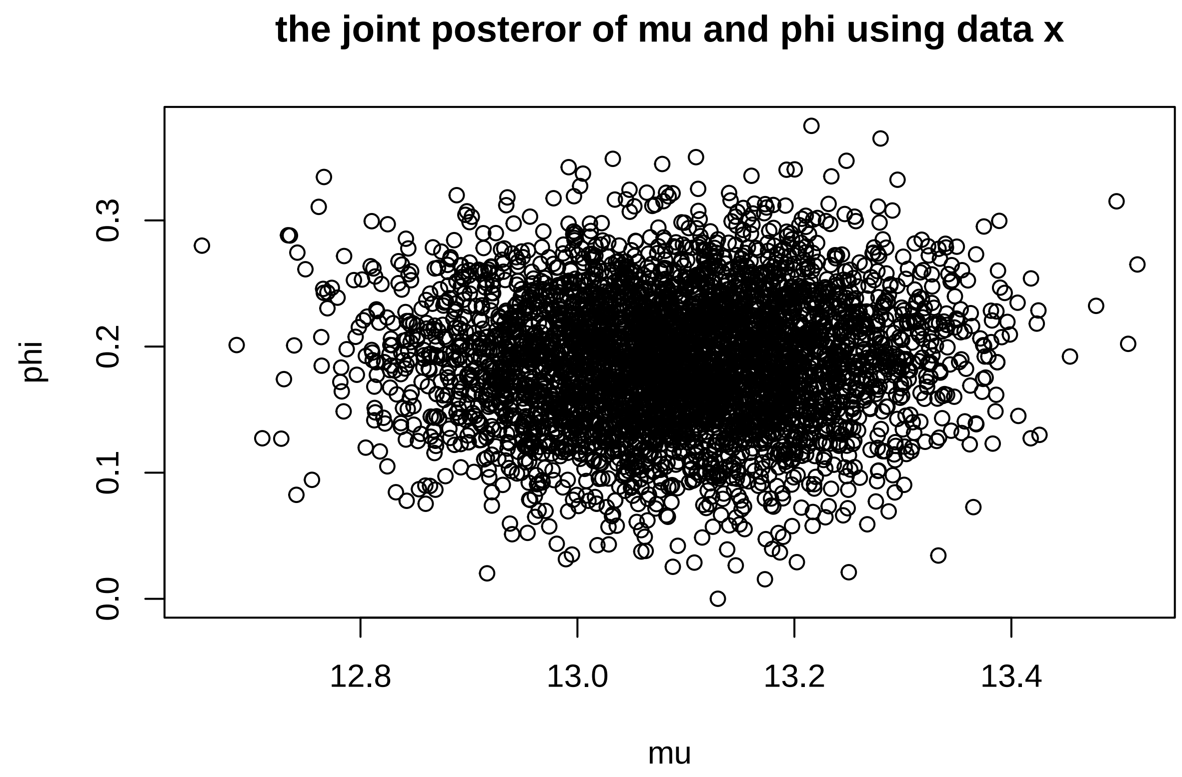
```r
# Do automatic traceplots of all chains of model fity
traceplot(fity)
```



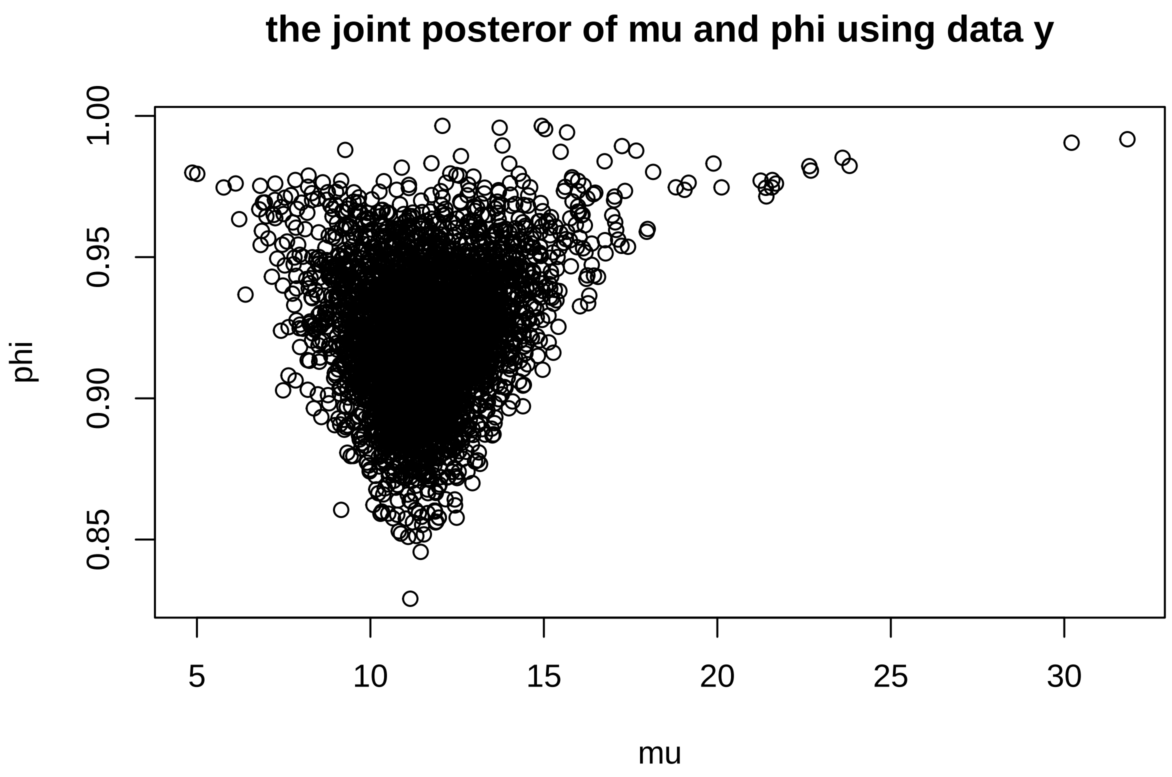
From the traceplot of all chains of the three parameters, we can see in model fitx the parameters are all covergence. In model fity, phi and sigma2,mu are also convergence.But some mu values are so big, seems not so good.

plot the joint posterior of mu and phi,we have the samples of mu from the posterior marginal density and the samples of phi from the posterior marginal density.

```
#Extract posterior samples in fitx and fity models.
x_mu<-extract(fitx)[[1]]
x_phi<-extract(fitx)[[2]]
y_mu<-extract(fity)[[1]]
y_phi<-extract(fity)[[2]]
plot(x_mu,x_phi,xlab="mu",ylab="phi",main="the joint posteror of mu and phi using data x")
```



```r
plot(y_mu,y_phi,xlab="mu",ylab="phi",main="the joint posteror of mu and phi using data y")
```



```
#check the posterior distribution
#pairs(fitx)
#pairs(fity)
```

We know the posterior distribution of mu and phi in model fitx are normal distribution, the posterior distribution of phi in model fity is also normal distribution. But mu is not. So the joint distribution in model fitx is normal but in model fity is a pecial shape.