Bayesian Learning Lab 3

Shipeng Liu Jin Yan

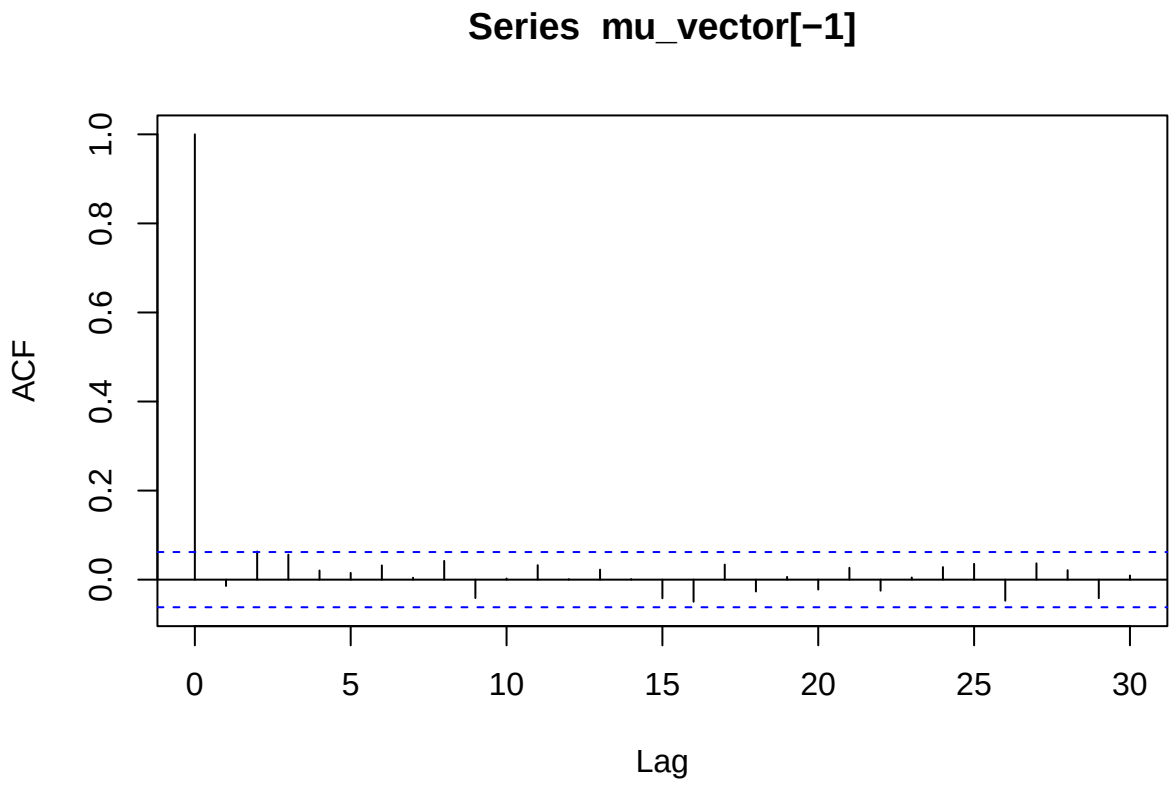
2023-05-12

Contents

Assignment 1

a) Evaluate the convergence of the Gibbs sampler

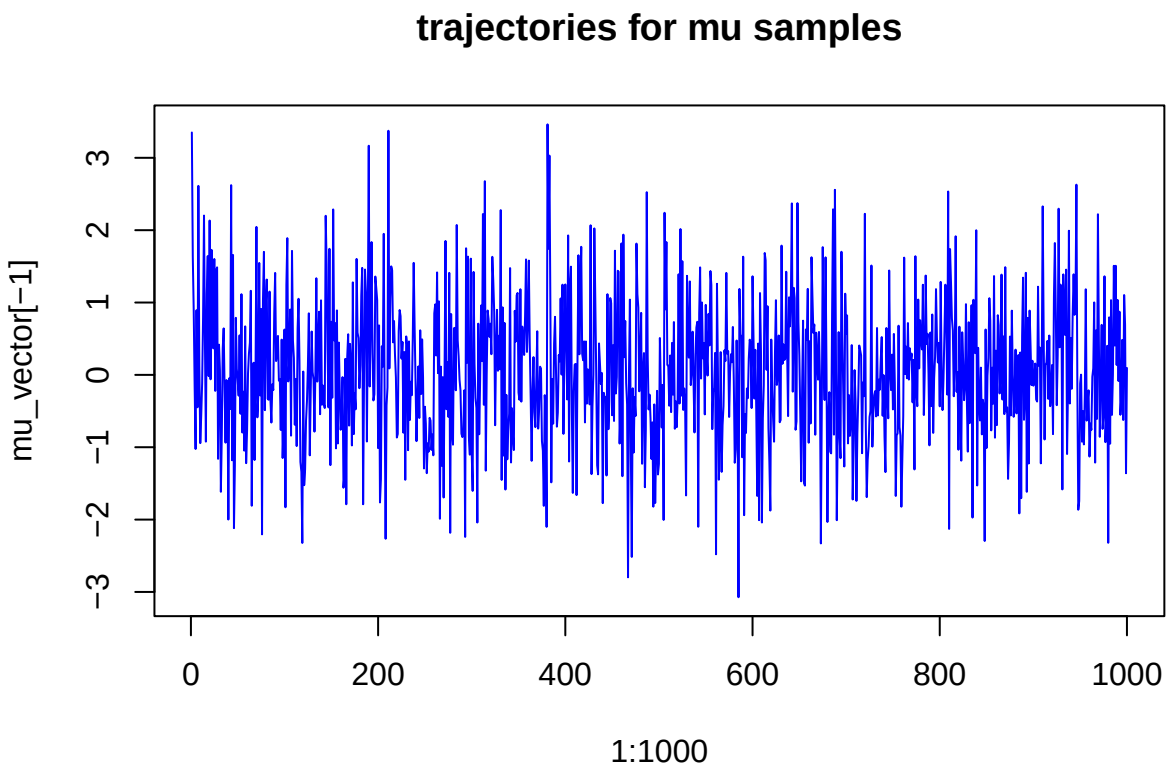
```
#1.1
data <- readRDS("Precipitation.rds")
# set initial parameters
n <- length(data)
mu_0 <- 1.5
tao_0 <- 1000
nu_0 <- 1
sigma_0_sq <- 1
nu_n <- n+nu_0
mean_lny <- mean(log(data))
sigma_sq_vector <- c(10000)
mu_vector <- NULL
# the following is used to get 1000 samples for mu and sigma.
for(i in 1:1000){
  sigma_sq <- sigma_sq_vector[i]
  #The following is find mu_n and tao_n
  tao_n <- sqrt(sigma_sq*tao_0^2/(n*tao_0^2+sigma_sq))
  w <- n/sigma_sq /(n/sigma_sq+ 1/tao_0^2)
  mu_n <- w*mean_lny+(1-w)*mu_0

  # the following is used to find mu
  mu <- rnorm(mu_n,tao_n)
  mu_vector[i+1] <- mu

  # the following is used to find new sigma_sq
  X <- rchisq(1,nu_n)
  s_sq <- nu_0*sigma_0_sq + sum((log(data)-mu)^2)/(n+nu_0)
  sigma_sq <- nu_n*s_sq/X
  sigma_sq_vector[i+1] <- sigma_sq
}


a_Gibbs <- acf(mu_vector[-1]) #not sure if it is right to omit the first element
```




```
IF_Gibbs <- 1+2*sum(a_Gibbs$acf[-1])
cat("IF_Gibbs =",IF_Gibbs)
```

```
## IF_Gibbs = 1.37057
```

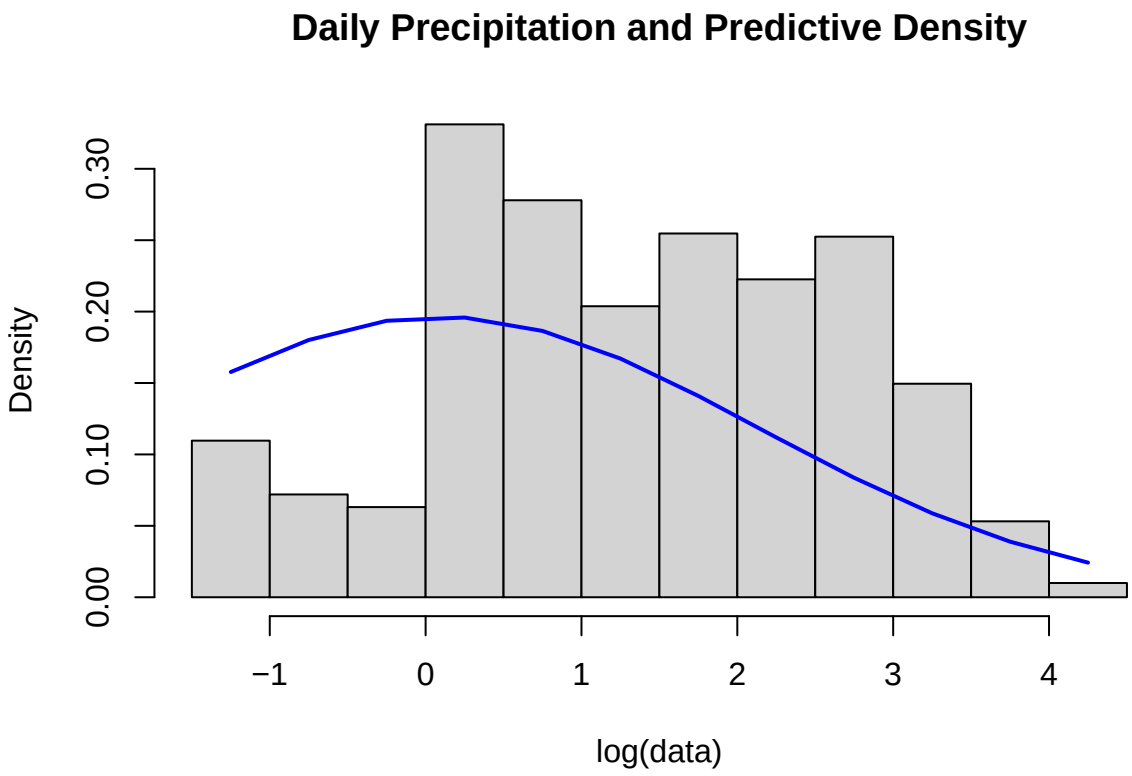
```
# plot trajectories of the samples.
plot(1:1000,mu_vector[-1],col="blue",main = "trajectories for mu samples",type = "l")
```



From above the value of IF_Gibbs and plot we can see that the mu samples generated by Gibbs sampler converge pretty well.

b) How well does the posterior predictive density agree with this data?

```
#1.2
# find the matched mu and sigma square
mu <- mu_vector[1001]
sigma <- sqrt(sigma_sq_vector[1001])

hist(log(data),freq = FALSE,main = "Daily Precipitation and Predictive Density")
hist_obj <- hist(log(data),plot = FALSE)
density_values <- dnorm(hist_obj$mids,mean = mu,sd=sigma)
#add density curve
lines(hist_obj$mid,density_values,col="blue",lwd=2)
```



From the histogram we can see that in general the tendency is pretty similar, but not exact.

Assignment 2

```
ebaydata <- read.table("eBayNumberOfBidderData.dat", header = TRUE)
```

a) Metropolis Random Walk for Poisson regression

Obtain the maximum likelihood estimator of β in the Poisson regression model for the eBay data

```
# Obtain the model
model=glm(nBids~.,family = 'poisson',data=ebaydata[,-2])

summary(model)
```

```
##
## Call:
## glm(formula = nBids ~ ., family = "poisson", data = ebaydata[,
##     -2])
##
```

```
## 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
```

It seem's the covirate MinBidShare is the most significant,Negatively correlated with outcome.The larger the ratio the minimum selling price to the book value,the smaller the number of bids.This conclusion is intuitive and reasonable.

b) $\widetilde{\beta}$ and $J_y(\widetilde{\beta})$

```
# Parameters
X=as.matrix(ebaydata[,-1])
y=ebaydata[,1]
mu=as.matrix(rep(0,dim(X)[2]))
sigma=as.matrix(100*solve(t(X)%*%X))

initBeta=as.matrix(rep(0,dim(X)[2]))

# The function to compute the posterior distribution
logPosterior=function(beta,mu,sigma,X,y){
  # The dimension of beta is 1*9(Including Intercept)

  # Prior density is multinorm distribution
  logPrior=dmvnorm(c(beta),mean=c(mu),sigma=sigma,log=TRUE)

  # Log Likelihood
  logLik=sum(y*(X%*%beta))-sum(exp(X%*%beta))#-sum(factorial(y))
  # The factorial can be omit

  logPoster=logLik+logPrior
```

```
  return(logPoster)
  }

OptimRes=optim(initBeta,logPosterior,gr=NULL,
               mu,sigma,X,y,method=c('BFGS'),
               control=list(fnscale=-1),hessian=TRUE)

# Obtain the mode of beta and it's negative Hessian matrix
betaMode=OptimRes[["par"]]
betaHessian=-OptimRes$hessian
Jbetamode=solve(betaHessian)

cat('The posterior mode is:\n')
```

```
## The posterior mode is:
```

```
print(betaMode)
```

```
##              [,1]
##  [1,]  1.06984118
##  [2,] -0.02051246
##  [3,] -0.39300599
##  [4,]  0.44355549
##  [5,] -0.05246627
##  [6,] -0.22123840
##  [7,]  0.07069683
##  [8,] -0.12021767
##  [9,] -1.89198501
```

```
cat('\nThe posterior covariance is:\n')
```

```
##
## The posterior covariance is:
```

```
print(Jbetamode)
```

```
##                [,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
```

The mode of posterior beta is close to what we get using the glm().

c) Simulate using Metropolis Algorithm

```
# A general function that uses the Metropolis algorithm to generate random draw from arbitrary posterio
RWMSampler=function(num,logPostFunc,c){
  sampleData=data.frame(matrix(0,nrow=num,ncol=9))
  colnames(sampleData)=colnames(X)
  # The first parameter should be theta
  theta=rmvnorm(1,betaMode,c*Jbetamode)
  sampleData[1,]=theta

  count=1
  while(count<num){
    theta=as.numeric(sampleData[count,])
    thetap=as.numeric(rmvnorm(1,theta,c*Jbetamode))
    acceptance_rate=min(1,
          exp(logPostFunc(thetap,mu,sigma,X,y)-logPostFunc(theta,mu,sigma,X,y)))
    if(runif(1,0,1)<acceptance_rate){
      count=count+1
      sampleData[count,]=thetap
    }
  }
  return(sampleData)
}

#Sample from the posterior of beta
sampleData=RWMSampler(1000,logPosterior,2)

print(sampleData[1:20,])
```

```
##       Const  PowerSeller   VerifyID    Sealed      Minblem     MajBlem
## 1  1.041624  0.020581850 -0.1226866 0.3523251 -0.200104054 -0.32331432
## 2  1.021902  0.035670987 -0.2456145 0.3739867 -0.213941451 -0.28247481
## 3  1.080127 -0.017073625 -0.4607540 0.4164559 -0.168048488 -0.50329435
## 4  1.032749 -0.015535474 -0.5143619 0.4992613 -0.079791793 -0.21895522
## 5  1.046959 -0.007061553 -0.3629864 0.5224744 -0.078093937 -0.19755894
## 6  1.088757  0.003537030 -0.3652534 0.4606985 -0.055800057 -0.26587951
## 7  1.054951  0.063576957 -0.5145554 0.4373271 -0.142771246  0.01938637
## 8  1.009032  0.039807354 -0.3567105 0.3999951  0.008184867 -0.16322021
## 9  1.061218 -0.007372877 -0.3569249 0.3179631 -0.071967800 -0.20395313
## 10 1.022384  0.037676797 -0.4326058 0.4032842 -0.110434206 -0.33587088
## 11 1.038646 -0.024396573 -0.3349992 0.4935568 -0.107123206 -0.07115446
## 12 1.018102 -0.039510938 -0.2969376 0.4999969  0.001560581 -0.13563517
```

```
## 13 1.025034  0.002211985 -0.2347293 0.4260271 -0.080944769 -0.24446087
## 14 1.009032  0.022055694 -0.3367180 0.4633830 -0.012057751 -0.21562257
## 15 1.059956 -0.046679882 -0.3130602 0.4319554 -0.138414881 -0.19298686
## 16 1.076481 -0.017622833 -0.2642996 0.3759995 -0.120708465 -0.11413752
## 17 1.114072 -0.032062277 -0.2600589 0.4302468 -0.069340710 -0.16218878
## 18 1.047169  0.002015524 -0.2226856 0.4679868  0.009700096 -0.02786589
## 19 1.028822 -0.007242028 -0.2027206 0.4709921 -0.031160070 -0.08883640
## 20 1.060849 -0.041612805 -0.2891343 0.4095407 -0.055290978 -0.25604759
##         LargNeg     LogBook MinBidShare
## 1   0.107280613 -0.07291369   -1.937340
## 2   0.072848288 -0.07358731   -1.993352
## 3   0.066779027 -0.06935155   -1.815053
## 4   0.128965061 -0.09203496   -1.768875
## 5  -0.050765657 -0.10643261   -1.766871
## 6   0.027002846 -0.12908606   -1.693065
## 7   0.009706625 -0.09612907   -1.837132
## 8   0.200857257 -0.04177269   -1.800302
## 9   0.105987119 -0.06927382   -1.868460
## 10  0.122254943 -0.07753497   -1.957883
## 11  0.089977352 -0.08445491   -1.895283
## 12  0.193449630 -0.10249917   -1.895240
## 13  0.030881501 -0.12242687   -1.901207
## 14  0.077903954 -0.14703765   -1.916357
## 15  0.078510935 -0.15673703   -1.876860
## 16 -0.005446383 -0.09997087   -1.862056
## 17 -0.064146292 -0.09386039   -1.793136
## 18  0.085590277 -0.12178189   -1.906720
## 19  0.147492837 -0.09097159   -1.854153
## 20  0.175902138 -0.12381561   -1.886980
```

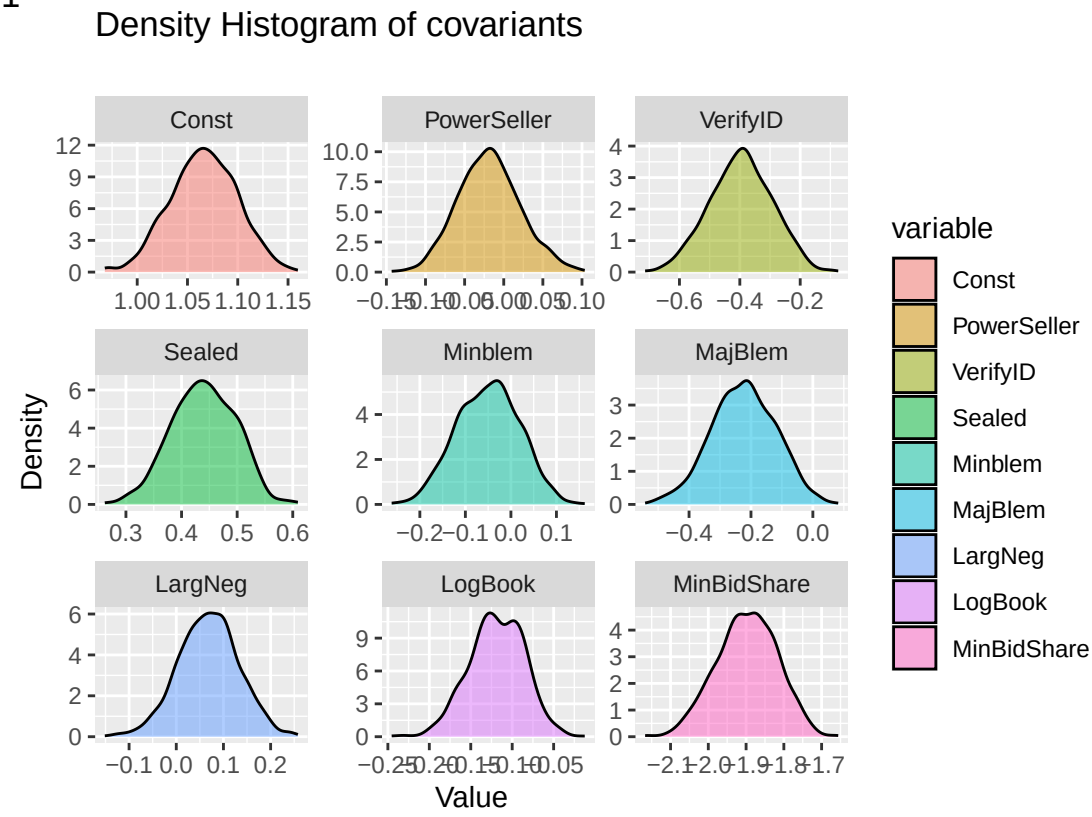
Assess MCMC convergence by graphical methods

```
sampleData_melted <- melt(sampleData)
```

```
## No id variables; using all as measure variables
```

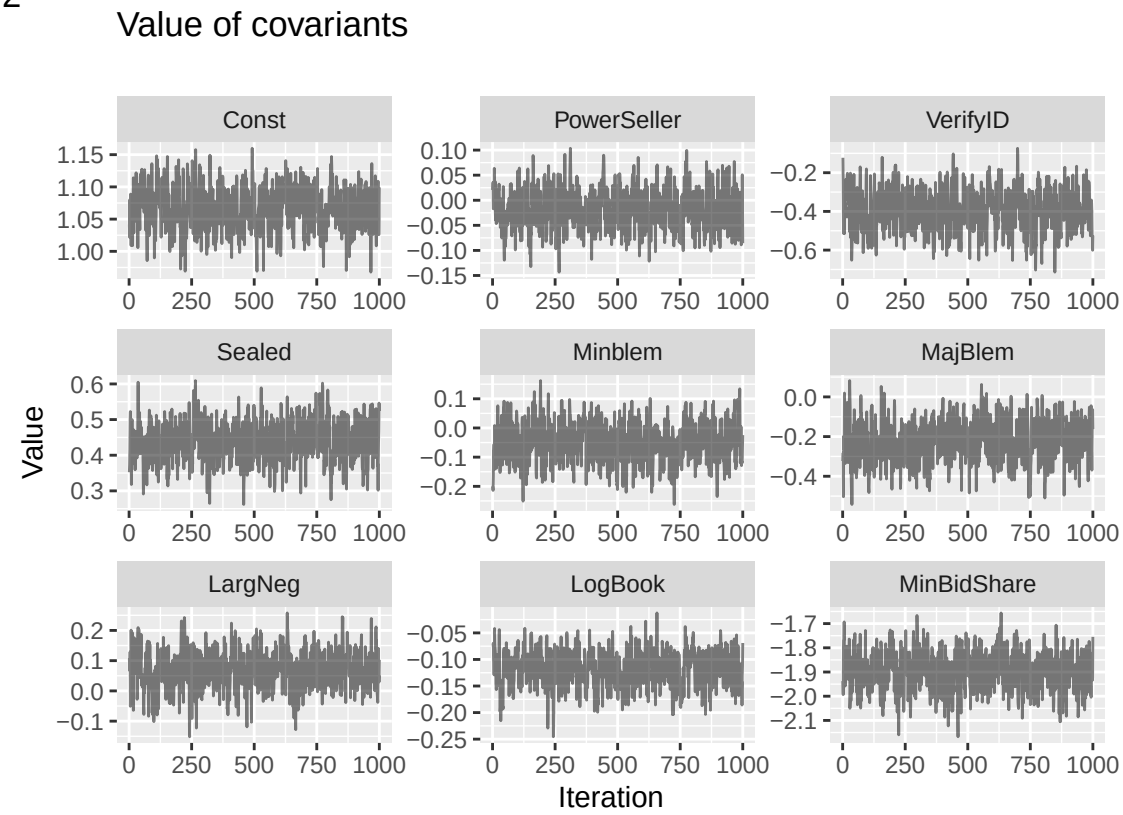
```
ggplot(sampleData_melted, aes(x = value, fill = variable)) +
  geom_density(alpha = 0.5) +
  facet_wrap(~ variable, scales = "free") +
  labs( x = "Value", y = "Density",title='Density Histogram of covariants
',tag='Fig 2.3.1')
```

Fig 2.3.1




```
sampleData_melted['index']=rep(seq(1,1000,1),9)
ggplot(sampleData_melted, aes(x = index,y=value, fill = variable)) +
  geom_line(alpha = 0.5) +
  facet_wrap(~ variable, scales = "free") +
  labs( x = "Iteration", y = "Value",title='Value of covariants
',tag='Fig 2.3.2')
```

Fig 2.3.2



It can be seen that they all converge to a certain normal distribution.

d) Simulate from the predictive distribution of the number of bidders in a new auction

```
# 1000 Sample from the RWMSampler
# PowerSeller=1,VerifyID=1,Sealed=1,MinBlem=0,MajBlem=1
# LargNeg=0,LogBook=1.2,MinBidShare=0.8
params=c(1,1,1,1,0,1,0,1.2,0.8)
result=data.frame(matrix(0,nrow=nrow(sampleData),ncol=0))
for(i in 1:nrow(sampleData)){
  lambda=exp(params%*%as.numeric(sampleData[i,]))
  result[i,1]=rpois(1,lambda)
}

colnames(result)=c('y')

#Plot
ggplot(data=result,aes(x=y))+
  geom_histogram(aes(y=..density..),binwidth=1,alpha=0.5)+
  geom_density(color='red',size=0.7,adjust=2.5)+
  labs(title='The predictive distribution of the number of bidders
',tag='Fig 2.4')+
  xlab("the number of bidders (y)")+
```

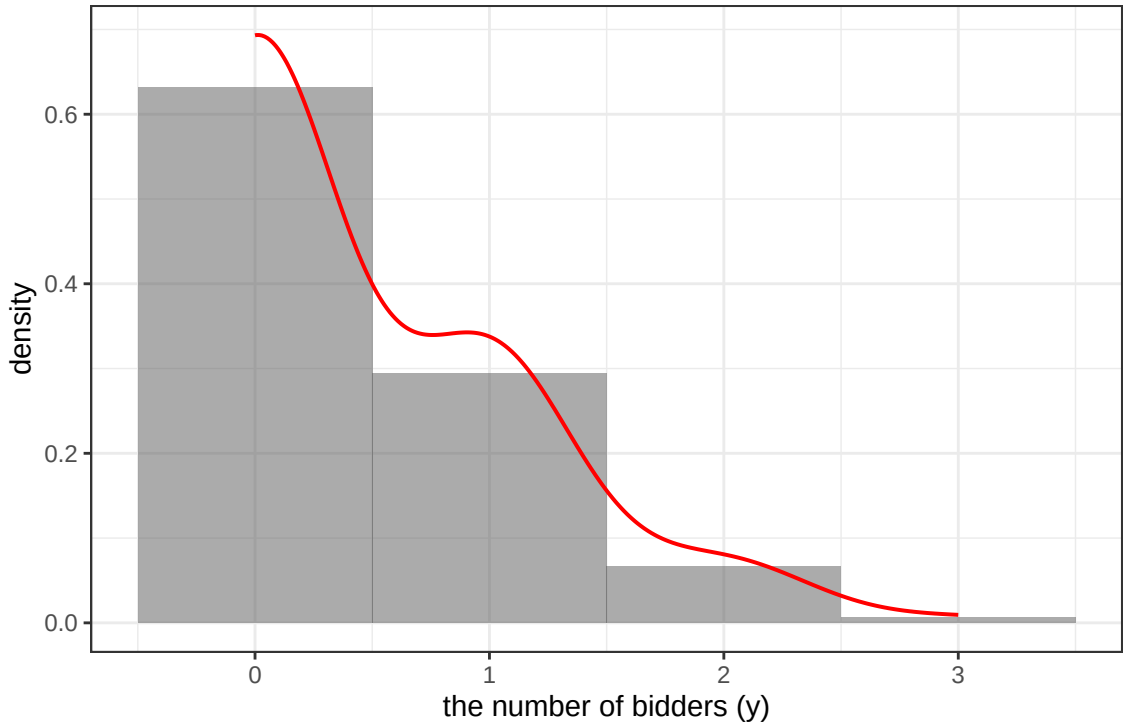
```
  theme_bw()
```

```
## Warning: Using ‘size‘ aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use ‘linewidth‘ instead.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```

```
## Warning: The dot-dot notation (‘..density..‘) was deprecated in ggplot2 3.4.0.
## i Please use ‘after_stat(density)‘ instead.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```

Fig 2.4

The predictive distribution of the number of bidders




```
cat('The probability of no bidders in this new auction:',mean(result==0))
```

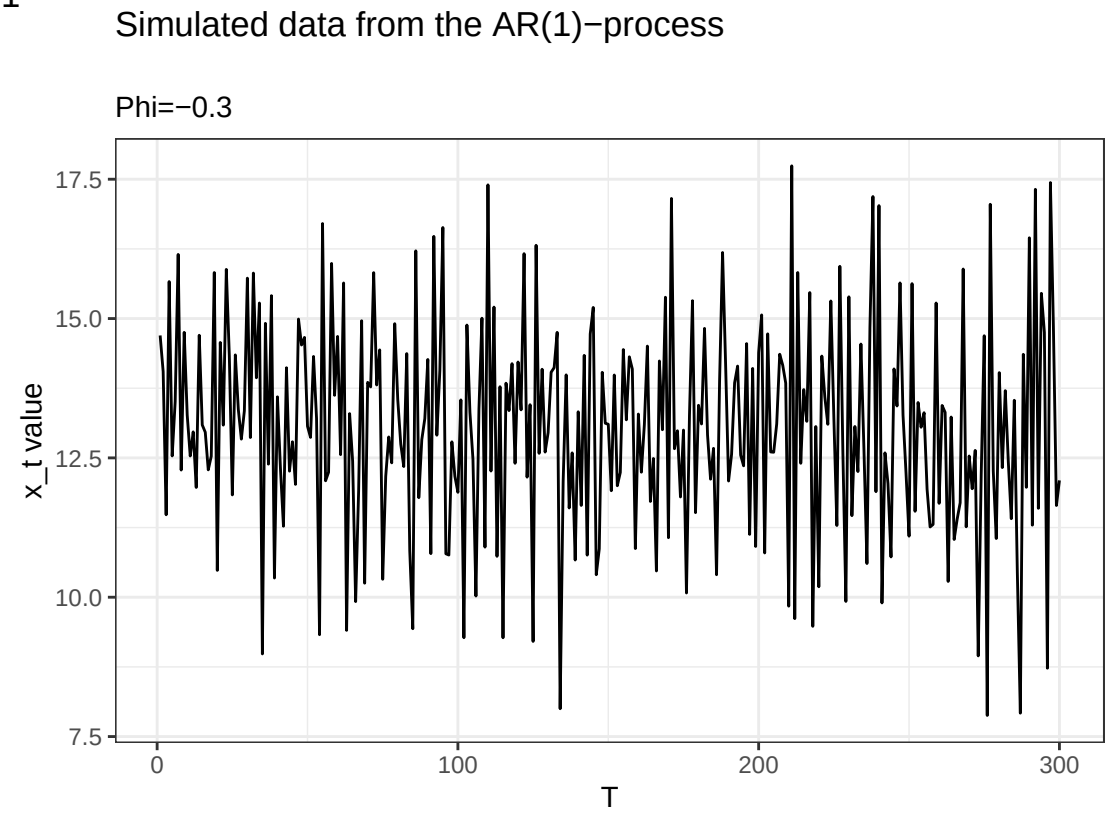
```
## The probability of no bidders in this new auction: 0.632
```

Assignment 3

a) Write a function in R that simulates data from the AR(1)-process

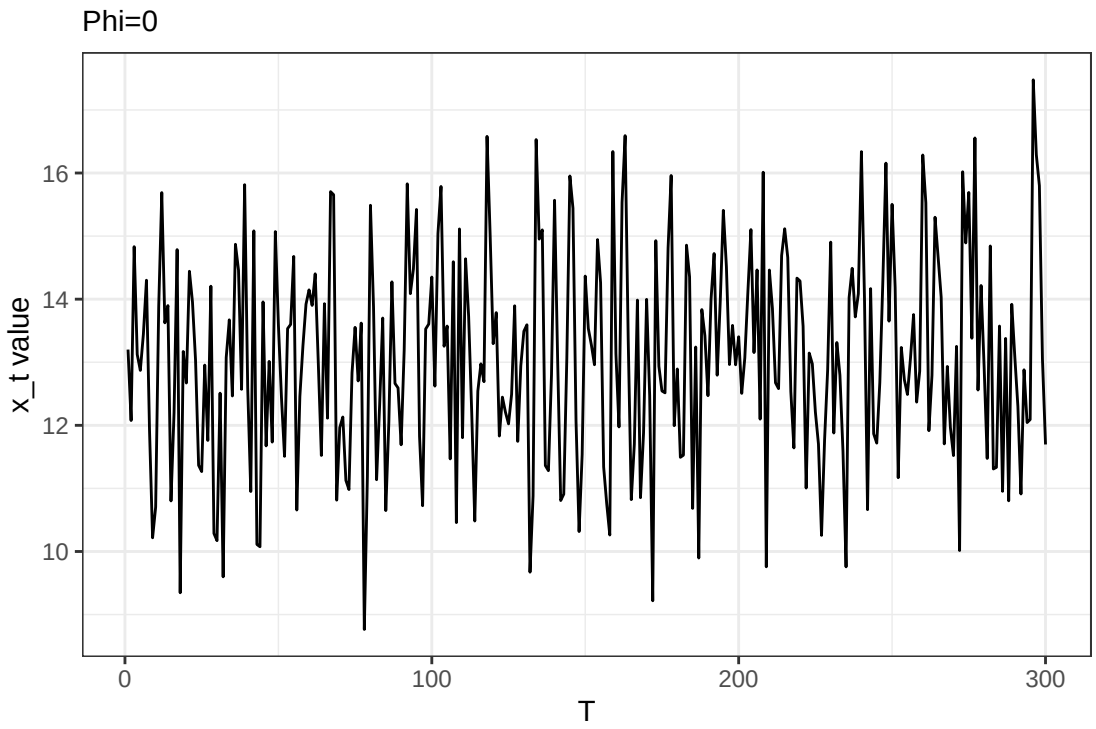
```
# Write a function in R that simulates data from the AR(1)-process
simulateAR=function(T_num,mu,phi,sigma){
  x=mu
  sampleData=c()
  for(i in 1:T_num){
    x=mu+phi*(x-mu)+rnorm(1,0,sqrt(sigma))
    sampleData=c(sampleData,x)
  }
  return(sampleData)
}

mu=13
sigma=3
T_num=300

# Simulations
# Phi=-0.3
sampleData1=simulateAR(T_num,mu,-0.3,sigma)
ggplot(data.frame("x_t"=sampleData1,"T"=1:T_num))+geom_line(aes(x=T,y=x_t))+
  labs(title='Simulated data from the AR(1)-process
',subtitle="Phi=-0.3",tag='Fig 3.1.1')+
  xlab("T")+
  ylab("x_t value")+
  theme_bw()
```

Fig 3.1.1



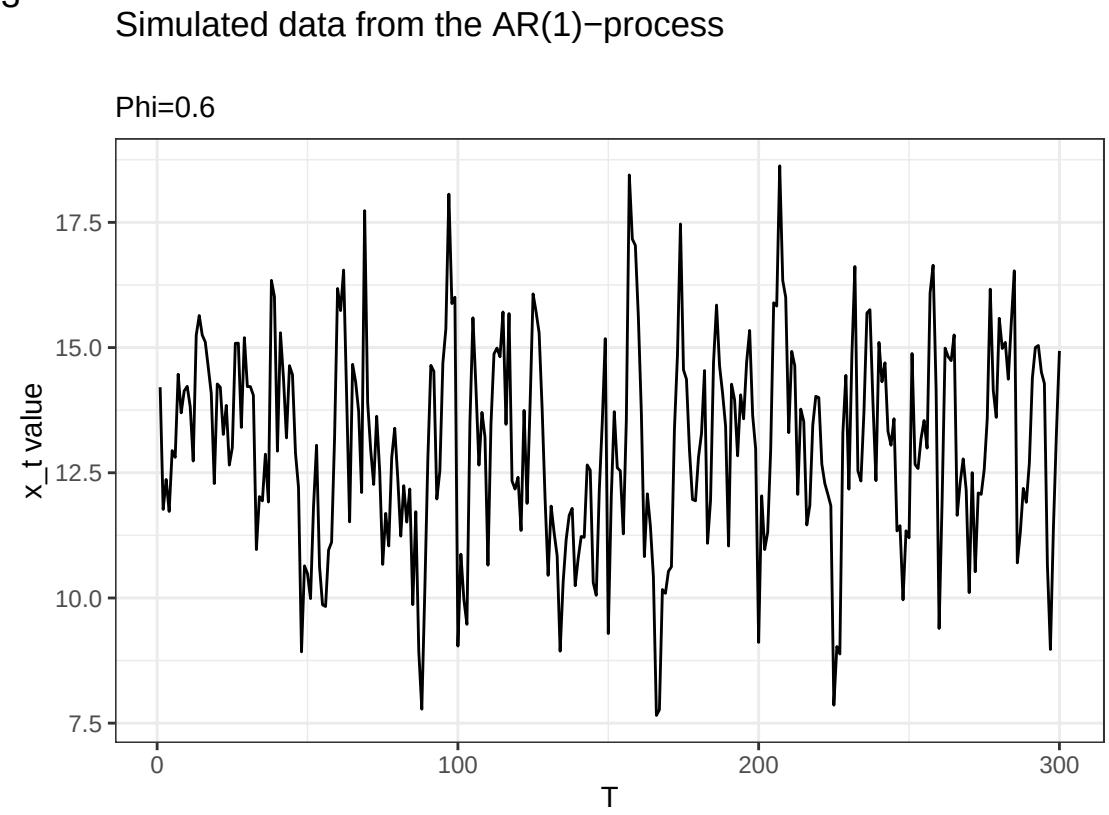

```
# Phi=0
sampleData2=simulateAR(T_num,mu,0,sigma)
ggplot(data.frame("x_t"=sampleData2,"T"=1:T_num))+geom_line(aes(x=T,y=x_t))+
  labs(title='Simulated data from the AR(1)-process
',subtitle="Phi=0",tag='Fig 3.1.2')+
  xlab("T")+
  ylab("x_t value")+
  theme_bw()
```

Fig 3.1.2




```
# Phi=0.6
sampleData3=simulateAR(T_num,mu,0.6,sigma)
ggplot(data.frame("x_t"=sampleData3,"T"=1:T_num))+geom_line(aes(x=T,y=x_t))+
  labs(title='Simulated data from the AR(1)-process
',subtitle="Phi=0.6",tag='Fig 3.1.3')+
  xlab("T")+
  ylab("x_t value")+
  theme_bw()
```

Fig 3.1.3



What effect does the value of Phi have on x1:T ?

When the value of Phi is positive, it means that there is a positive correlation between the current observation and the past observation. Therefore, the simulated numerical series may have a continuously increasing or continuously decreasing trend.

When the value of Phi is negative, it means that there is a negative correlation between the current observation and the past observation. Therefore, the simulated numerical series may have oscillatory or periodic characteristics.

In addition, the value of Phi will also affect the stability of the simulated sequence. When the absolute value of Phi is less than 1, the simulated numerical sequence is usually stable. And when the absolute value of Phi is greater than 1, the simulated numerical sequence may unable to reach a stable state.

b) Simulate two AR(1)-processes

```
mu=13
sigma=3
T_num=300

# Phi=0.2
sampleData4=simulateAR(T_num,mu,0.2,sigma)

# Phi=0.95
sampleData5=simulateAR(T_num,mu,0.95,sigma)
```

Building the stan model

```
stancode='
// Save as armodel.stan
data {
  int<lower=0> J;         //The number of observations
  vector[J] y;              //The observations
}

parameters {              //Parameters
  real mu;
  real phi;
  real<lower=0> sigma;
}

model {
  // prior distribution(Non-informative)
  mu ~ normal(0,100);
  phi ~ uniform(-1,1);
  sigma ~ scaled_inv_chi_square(1,2);

  for (n in 2:J)
    y[n] ~ normal(mu+phi*(y[n-1]-mu),sqrt(sigma));

}'
```

i) Report the posterior mean, 95% credible intervals and the number of effective posterior samples for the three inferred parameters for each of the simulated AR(1)-process.

```
# Phi=0.2
fit1 <- stan(model_code =stancode, data =list(J=300,y=sampleData4), iter = 2000, warmup = 1000, chains =
```

```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 7.7e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.77 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: 0.548 seconds (Warm-up)
## Chain 1:                0.158 seconds (Sampling)
## Chain 1:                0.706 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 3e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.3 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: 0.456 seconds (Warm-up)
## Chain 2:                0.149 seconds (Sampling)
## Chain 2:                0.605 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
## Chain 3: Rejecting initial value:
## Chain 3:   Log probability evaluates to log(0), i.e. negative infinity.
## Chain 3:   Stan can't start sampling from this initial value.
## Chain 3: Rejecting initial value:
## Chain 3:   Log probability evaluates to log(0), i.e. negative infinity.
## Chain 3:   Stan can't start sampling from this initial value.
## Chain 3:
## Chain 3: Gradient evaluation took 3.3e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.33 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: 0.236 seconds (Warm-up)
## Chain 3:                0.148 seconds (Sampling)
## Chain 3:                0.384 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 3.2e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.32 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: 0.322 seconds (Warm-up)
## Chain 4:                0.197 seconds (Sampling)
## Chain 4:                0.519 seconds (Total)
## Chain 4:
```

```
posterior_mean_1 <- summary(fit1)$summary[1:3, c(1,4,8)]
cat("Phi=0.2:\nPosterior Mean:\n")
```

```
## Phi=0.2:
## Posterior Mean:
```

```
print(posterior_mean_1[,1])
```

```
##         mu        phi      sigma
## 13.1556361  0.2099186  2.9296423
```

```
cat("95% credible intervals:\n")
```

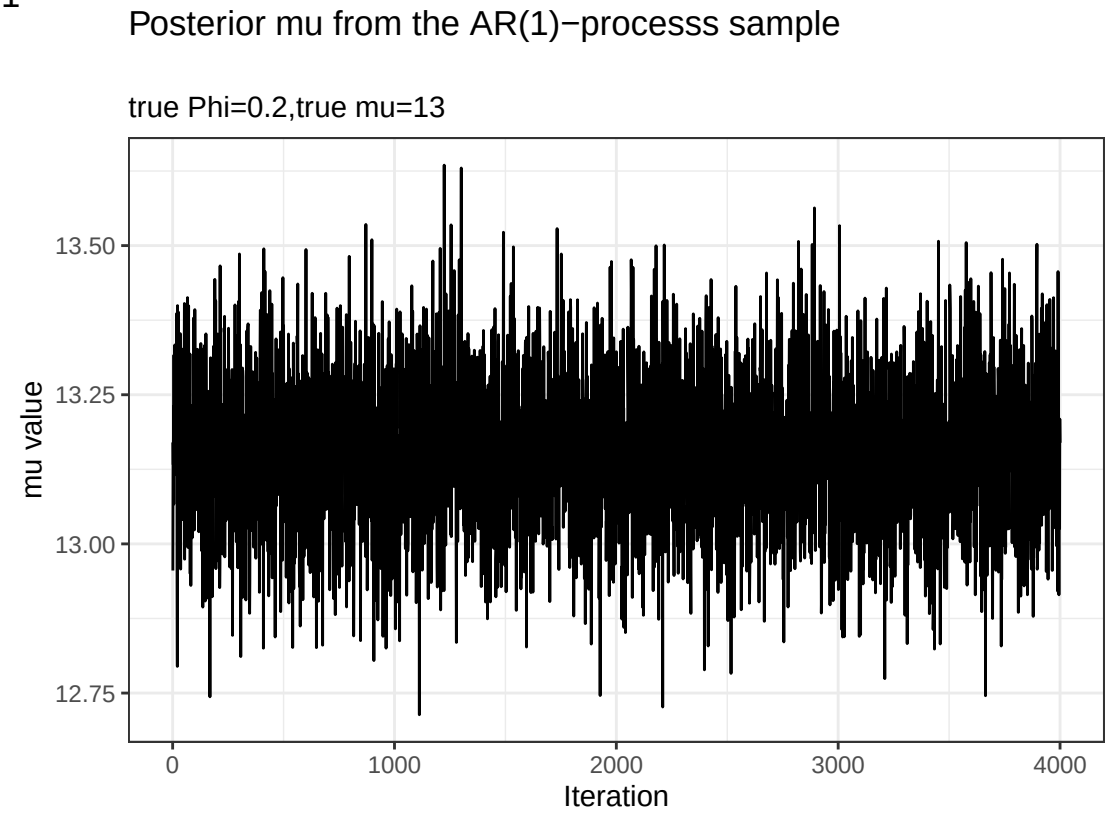
```
## 95% credible intervals:
```

```
print(posterior_mean_1[,c(2,3)])
```

```
##              2.5%      97.5%
## mu    12.91135133 13.4005572
## phi    0.09607449  0.3255905
## sigma  2.48881129  3.4513149
```

```
# Effective posterior samples
params_1=extract(fit1)
params_1_data=data.frame("iteration"=1:4000,"mu"=params_1$mu,"phi"=params_1$phi,"sigma"=params_1$sigma)

#Plot
# mu
ggplot(data=params_1_data)+geom_line(aes(x=iteration,y=mu))+
  labs(title='Posterior mu from the AR(1)-processs sample
',subtitle="true Phi=0.2,true mu=13",tag='Fig 4.1.1')+
  xlab("Iteration")+
  ylab("mu value")+
  theme_bw()
```

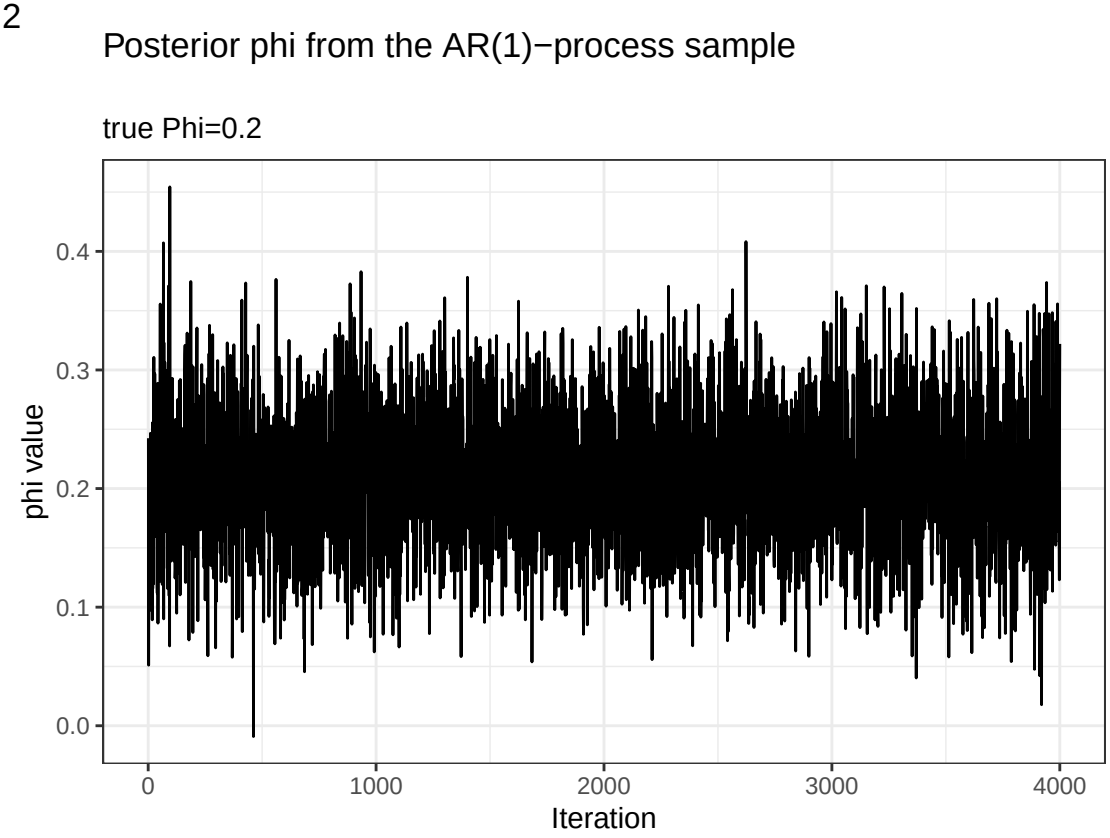
Fig 4.1.1




```
# phi
ggplot(data=params_1_data)+geom_line(aes(x=iteration,y=phi))+
  labs(title='Posterior phi from the AR(1)-process sample
',subtitle="true Phi=0.2",tag='Fig 4.1.2')+
```

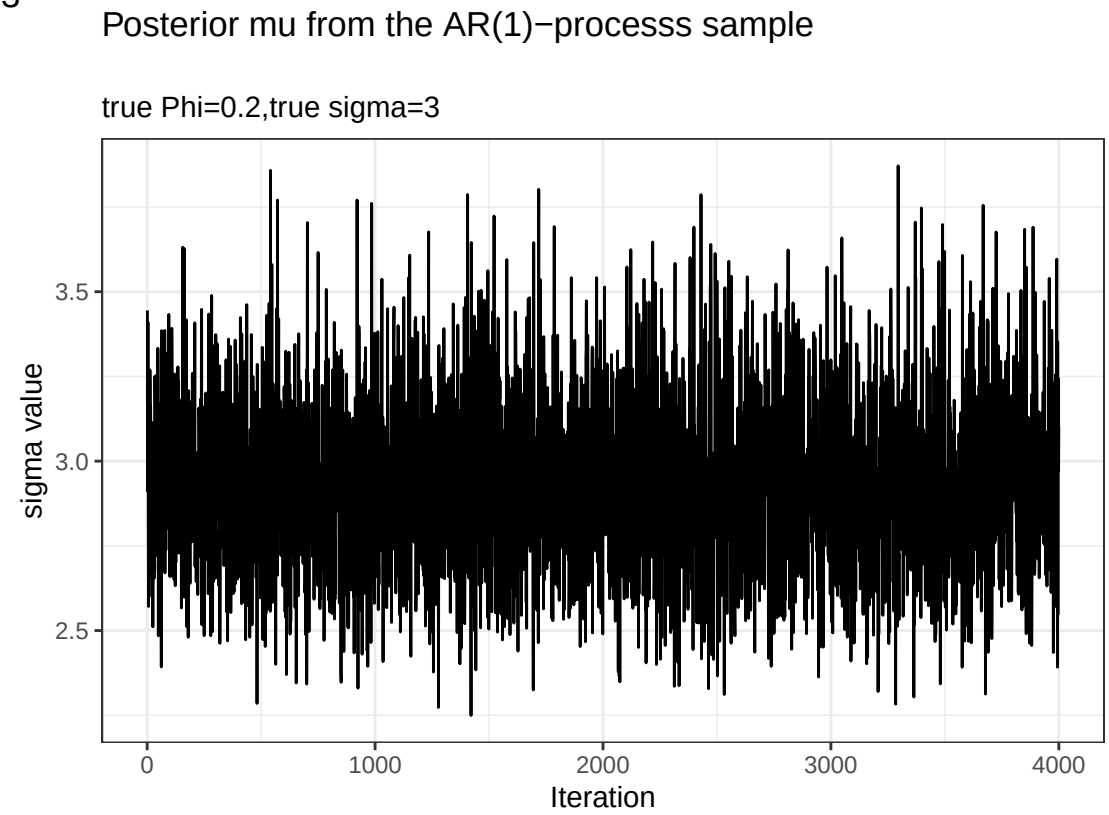
```
  xlab("Iteration")+
  ylab("phi value")+
  theme_bw()
```

Fig 4.1.2




```
#sigma
ggplot(data=params_1_data)+geom_line(aes(x=iteration,y=sigma))+
  labs(title='Posterior mu from the AR(1)-processs sample
',subtitle="true Phi=0.2,true sigma=3",tag='Fig 4.1.3')+
  xlab("Iteration")+
  ylab("sigma value")+
  theme_bw()
```

Fig 4.1.3



We obtained 1000 samples of four Markov chains after 1000 warmups, for a total of 4000. From the figure, mu, phi, and sigma all converge to the real value, and only a few samples are far from the real value. So the number of effective posterior samples is close to 4000.

```
# Phi=0.95
fit2 <- stan(model_code =stancode, data = list(J=300,y=sampleData5), iter = 2000, warmup = 1000, chains
```

```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1: Rejecting initial value:
## Chain 1:   Log probability evaluates to log(0), i.e. negative infinity.
## Chain 1:   Stan can't start sampling from this initial value.
## Chain 1:
## Chain 1: Gradient evaluation took 3.5e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.35 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: 0.796 seconds (Warm-up)
## Chain 1:                0.203 seconds (Sampling)
## Chain 1:                0.999 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 3.2e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.32 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: 0.408 seconds (Warm-up)
## Chain 2:                0.32 seconds (Sampling)
## Chain 2:                0.728 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
## Chain 3: Rejecting initial value:
## Chain 3:   Log probability evaluates to log(0), i.e. negative infinity.
## Chain 3:   Stan can't start sampling from this initial value.
## Chain 3: Rejecting initial value:
## Chain 3:   Log probability evaluates to log(0), i.e. negative infinity.
## Chain 3:   Stan can't start sampling from this initial value.
## Chain 3: Rejecting initial value:
## Chain 3:   Log probability evaluates to log(0), i.e. negative infinity.
## Chain 3:   Stan can't start sampling from this initial value.
## Chain 3:
## Chain 3: Gradient evaluation took 3.2e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.32 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: 0.731 seconds (Warm-up)
## Chain 3:                0.295 seconds (Sampling)
## Chain 3:                1.026 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 3.2e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.32 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: 0.748 seconds (Warm-up)
## Chain 4:                0.314 seconds (Sampling)
## Chain 4:                1.062 seconds (Total)
## Chain 4:
```

```
## Warning: There were 327 divergent transitions after warmup. See
## https://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
## to find out why this is a problem and how to eliminate them.
```

```
## Warning: Examine the pairs() plot to diagnose sampling problems
```

```
posterior_mean_2 <- summary(fit2)$summary[1:3, c(1,4,8)]
cat("Phi=0.95:\nPosterior Mean:\n")
```

```
## Phi=0.95:
## Posterior Mean:
```

```r
print(posterior_mean_2[,1])
```

```
##         mu        phi      sigma
## 13.9682563  0.9552897  2.7538405
```

```r
cat("95% credible intervals:\n")
```

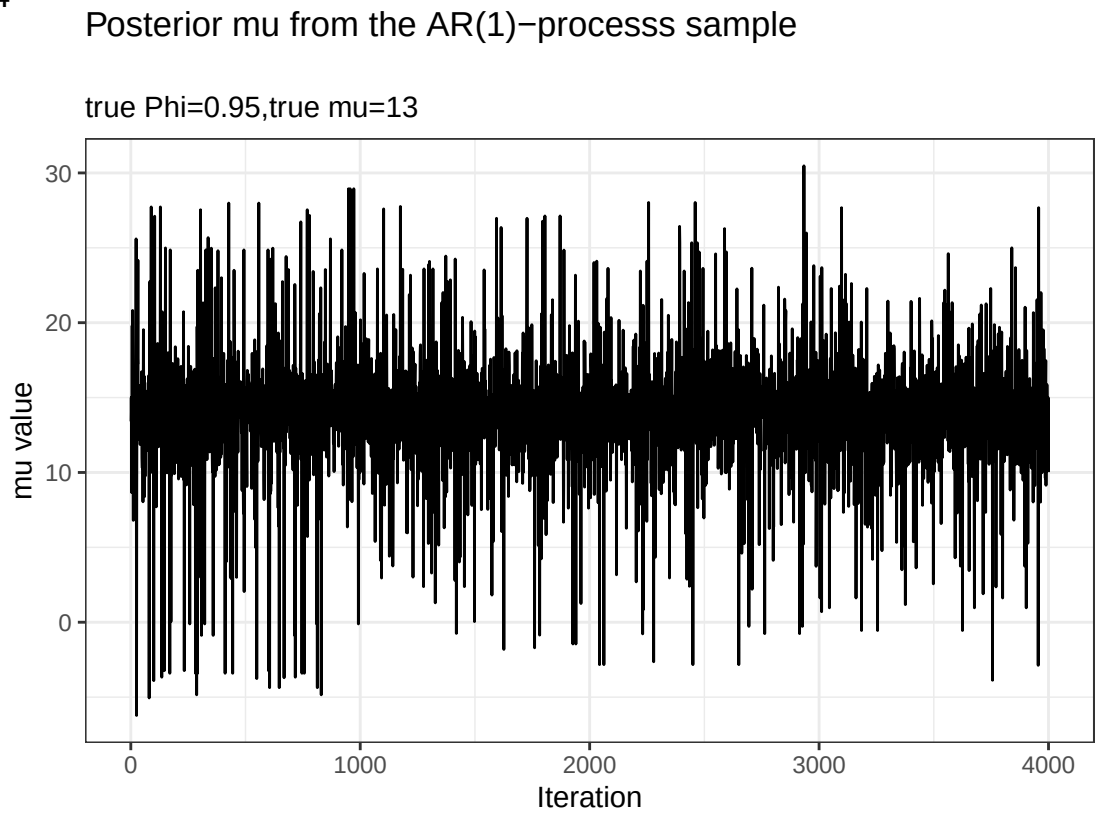
```
## 95% credible intervals:
```

```r
print(posterior_mean_2[,c(2,3)])
```

```
##            2.5%      97.5%
## mu    4.4086781 22.2573091
## phi   0.9138012  0.9950436
## sigma 2.3335060  3.2065881
```

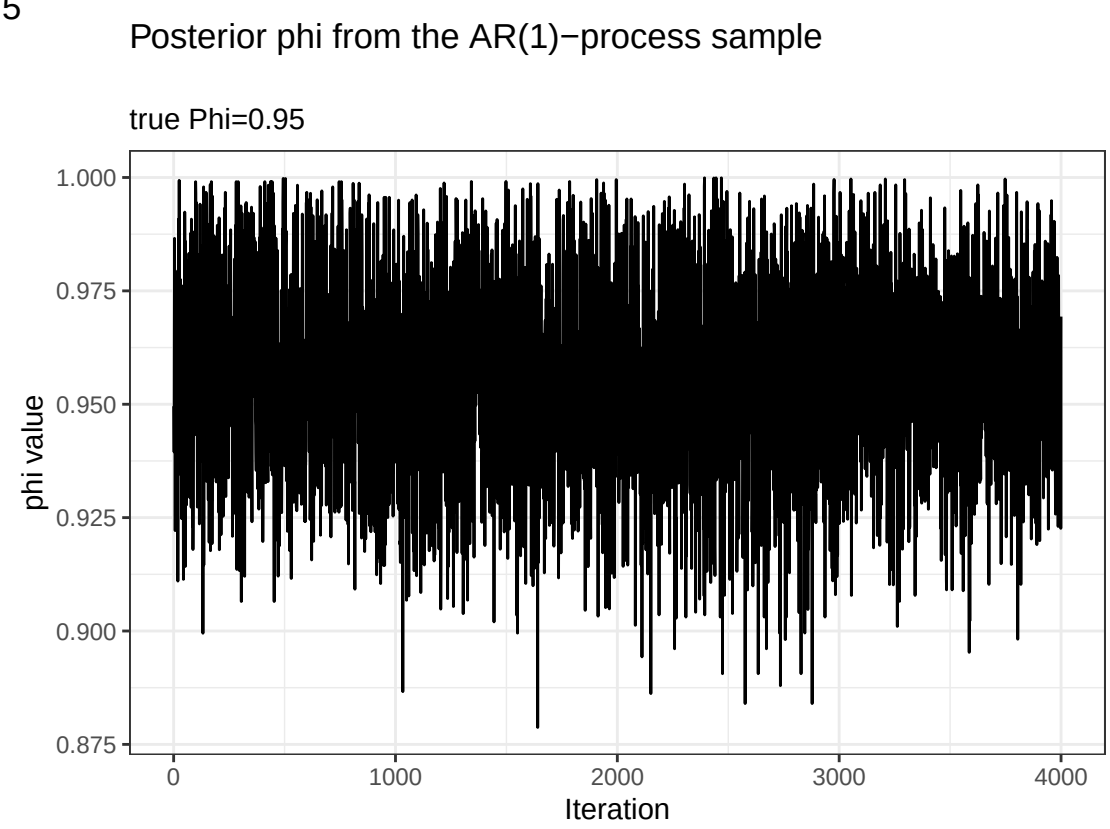
```r
# Effective posterior samples
params_2=extract(fit2)
params_2_data=data.frame("iteration"=1:4000,"mu"=params_2$mu,"phi"=params_2$phi,"sigma"=params_2$sigma)

#Plot
# mu
ggplot(data=params_2_data)+geom_line(aes(x=iteration,y=mu))+
  labs(title='Posterior mu from the AR(1)-processs sample
',subtitle="true Phi=0.95,true mu=13",tag='Fig 4.1.4')+
  xlab("Iteration")+
  ylab("mu value")+
  theme_bw()
```

Fig 4.1.4



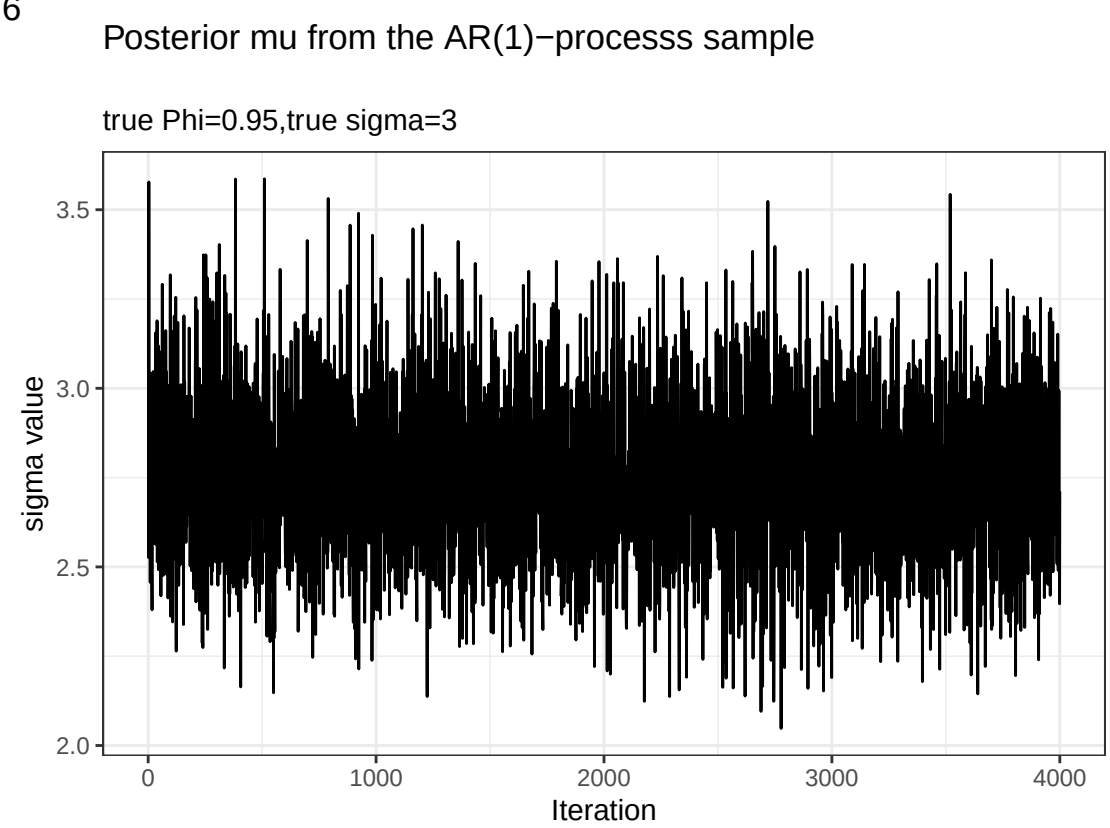

```
# phi
ggplot(data=params_2_data)+geom_line(aes(x=iteration,y=phi))+
  labs(title='Posterior phi from the AR(1)-process sample
',subtitle="true Phi=0.95",tag='Fig 4.1.5')+
  xlab("Iteration")+
  ylab("phi value")+
  theme_bw()
```

Fig 4.1.5




```
#sigma
ggplot(data=params_2_data)+geom_line(aes(x=iteration,y=sigma))+
  labs(title='Posterior mu from the AR(1)-processs sample
',subtitle="true Phi=0.95,true sigma=3",tag='Fig 4.1.6')+
  xlab("Iteration")+
  ylab("sigma value")+
  theme_bw()
```

Fig 4.1.6

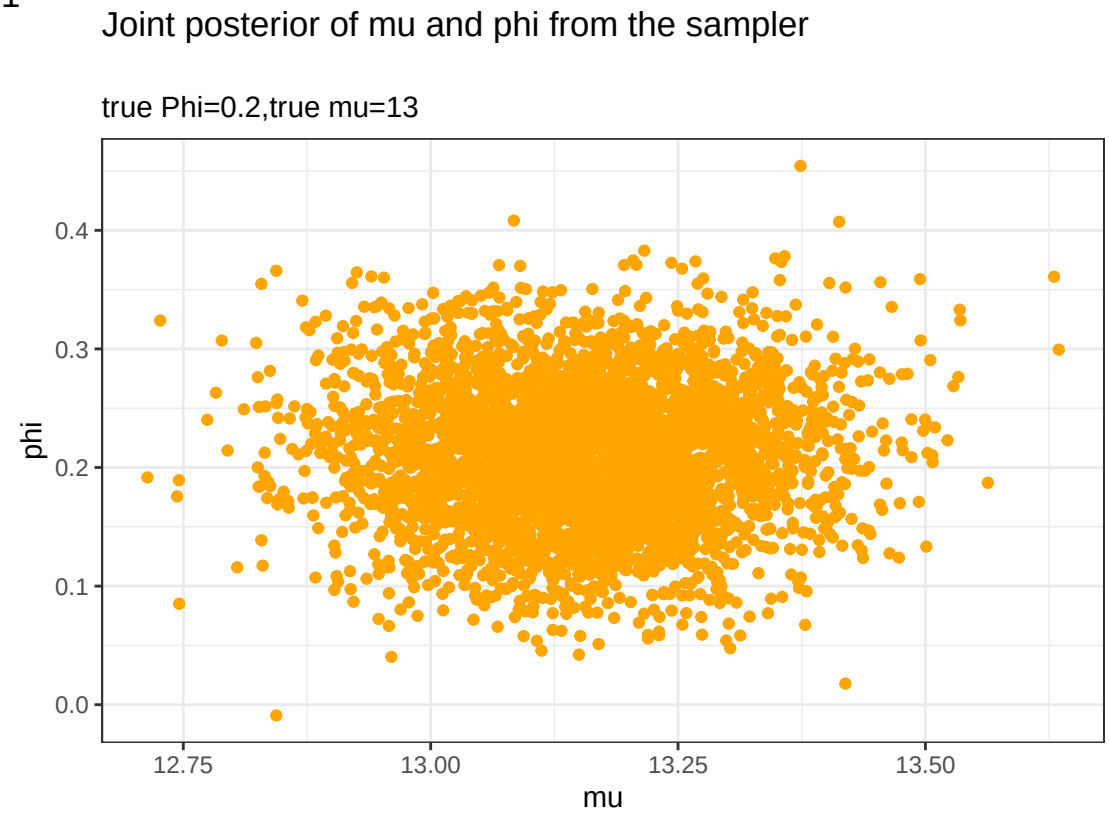


The posterior sample of phi and sigma value is close to the real value,but the mu value haven't converge and many sample a very different from the real value.So the number of effective posterior samples is much smaller than 4000.

ii) For each of the two data sets, evaluate the convergence of the samplers and plot the joint posterior of mu and phi. Comments?

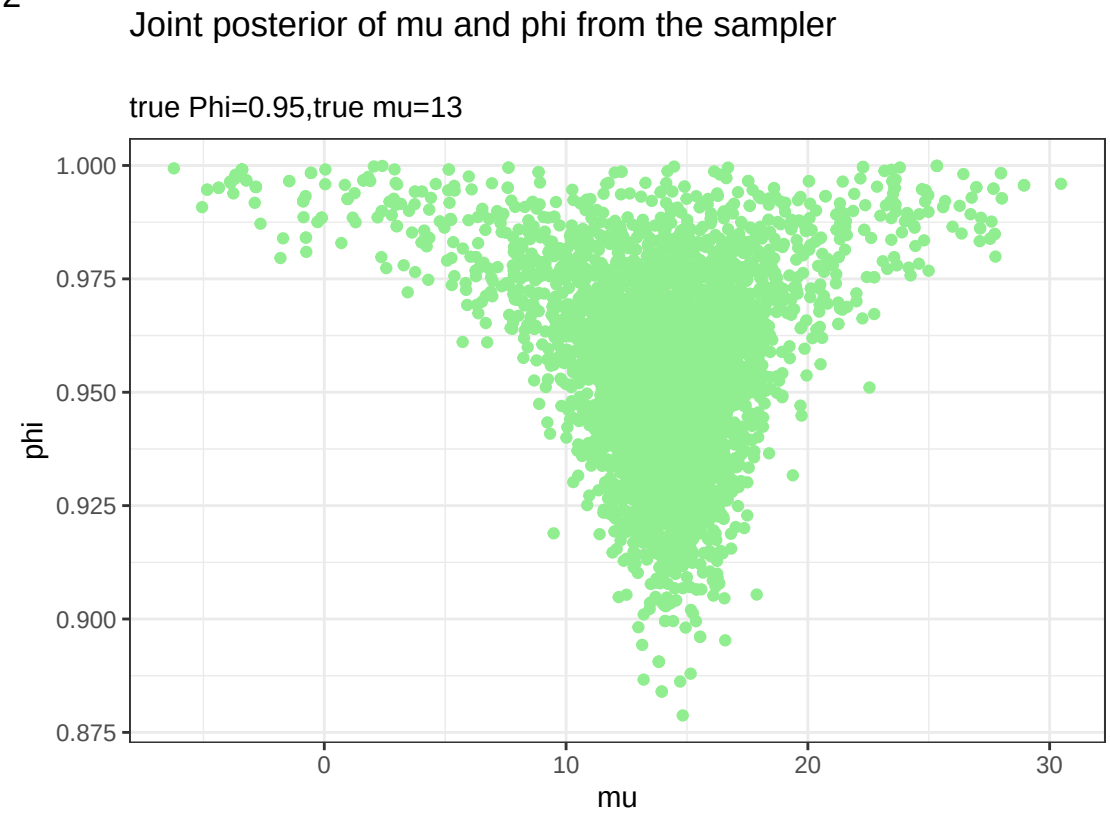
```
# Plot the joint posterior
# phi=0.2
ggplot(data=params_1_data)+geom_point(aes(x=mu,y=phi),color='orange')+
  labs(title='Joint posterior of mu and phi from the sampler
',subtitle="true Phi=0.2,true mu=13",tag='Fig 4.2.1')+
  xlab("mu")+
  ylab("phi")+
  theme_bw()
```

Fig 4.2.1




```
# phi=0.95
ggplot(data=params_2_data)+geom_point(aes(x=mu,y=phi),color='lightgreen')+
  labs(title='Joint posterior of mu and phi from the sampler
',subtitle="true Phi=0.95,true mu=13",tag='Fig 4.2.2')+
  xlab("mu")+
  ylab("phi")+
  theme_bw()
```

Fig 4.2.2



It can be seen that the sampler at phi=0.2 converges well, but the sampler at phi=0.95 does not converge, and the larger the phi, the larger the variance of mu.

The reason may be,as in the 3 a) said,when the value of Phi is positive,the simulated numerical series may have a continuously increasing or continuously decreasing trend.In addition, the value of Phi will also affect the stability of the simulated sequence.

Appendix

```
library(mvtnorm)
library(ggplot2)
library(reshape2)
library(StanHeaders)
library(rstan)
#1.1
data <- readRDS("Precipitation.rds")
# set initial parameters
n <- length(data)
mu_0 <- 1.5
tao_0 <- 1000
nu_0 <- 1
sigma_0_sq <- 1
nu_n <- n+nu_0
mean_lny <- mean(log(data))
sigma_sq_vector <- c(10000)
mu_vector <- NULL
# the following is used to get 1000 samples for mu and sigma.
for(i in 1:1000){
  sigma_sq <- sigma_sq_vector[i]
  #The following is find mu_n and tao_n
  tao_n <- sqrt(sigma_sq*tao_0^2/(n*tao_0^2+sigma_sq))
  w <- n/sigma_sq /(n/sigma_sq+ 1/tao_0^2)
  mu_n <- w*mean_lny+(1-w)*mu_0

  # the following is used to find mu
  mu <- rnorm(mu_n,tao_n)
  mu_vector[i+1] <- mu

  # the following is used to find new sigma_sq
  X <- rchisq(1,nu_n)
  s_sq <- nu_0*sigma_0_sq + sum((log(data)-mu)^2)/(n+nu_0)
  sigma_sq <- nu_n*s_sq/X
  sigma_sq_vector[i+1] <- sigma_sq
}


a_Gibbs <- acf(mu_vector[-1]) #not sure if it is right to omit the first element
IF_Gibbs <- 1+2*sum(a_Gibbs$acf[-1])
cat("IF_Gibbs =",IF_Gibbs)
# plot trajectories of the samples.
plot(1:1000,mu_vector[-1],col="blue",main = "trajectories for mu samples",type = "l")
#1.2
# find the matched mu and sigma square
mu <- mu_vector[1001]
sigma <- sqrt(sigma_sq_vector[1001])

hist(log(data),freq = FALSE,main = "Daily Precipitation and Predictive Density")
hist_obj <- hist(log(data),plot = FALSE)
density_values <- dnorm(hist_obj$mids,mean = mu,sd=sigma)
#add density curve
```

```
lines(hist_obj$mid,density_values,col="blue",lwd=2)
ebaydata <- read.table("eBayNumberOfBidderData.dat", header = TRUE)
# Obtain the model
model=glm(nBids~.,family = 'poisson',data=ebaydata[,-2])

summary(model)
# Parameters
X=as.matrix(ebaydata[,-1])
y=ebaydata[,1]
mu=as.matrix(rep(0,dim(X)[2]))
sigma=as.matrix(100*solve(t(X)%*%X))

initBeta=as.matrix(rep(0,dim(X)[2]))

# The function to compute the posterior distribution
logPosterior=function(beta,mu,sigma,X,y){
  # The dimension of beta is 1*9(Including Intercept)

  # Prior density is multinorm distribution
  logPrior=dmvnorm(c(beta),mean=c(mu),sigma=sigma,log=TRUE)

  # Log Likelihood
  logLik=sum(y*(X%*%beta))-sum(exp(X%*%beta))#-sum(factorial(y))
  # The factorial can be omit

  logPoster=logLik+logPrior

  return(logPoster)
  }

OptimRes=optim(initBeta,logPosterior,gr=NULL,
               mu,sigma,X,y,method=c('BFGS'),
               control=list(fnscale=-1),hessian=TRUE)

# Obtain the mode of beta and it's negative Hessian matrix
betaMode=OptimRes[["par"]]
betaHessian=-OptimRes$hessian
Jbetamode=solve(betaHessian)

cat('The posterior mode is:\n')
print(betaMode)
cat('\nThe posterior covariance is:\n')
print(Jbetamode)
# A general function that uses the Metropolis algorithm to generate random draw from arbitrary posterio
RWMSampler=function(num,logPostFunc,c){
  sampleData=data.frame(matrix(0,nrow=num,ncol=9))
  colnames(sampleData)=colnames(X)
  # The first parameter should be theta
  theta=rmvnorm(1,betaMode,c*Jbetamode)
  sampleData[1,]=theta

  count=1
  while(count<num){
```

```
    theta=as.numeric(sampleData[count,])
    thetap=as.numeric(rmvnorm(1,theta,c*Jbetamode))
    acceptance_rate=min(1,
          exp(logPostFunc(thetap,mu,sigma,X,y)-logPostFunc(theta,mu,sigma,X,y)))
    if(runif(1,0,1)<acceptance_rate){
      count=count+1
      sampleData[count,]=thetap
    }
  }
  return(sampleData)
}

#Sample from the posterior of beta
sampleData=RWMSampler(1000,logPosterior,2)

print(sampleData[1:20,])
sampleData_melted <- melt(sampleData)
ggplot(sampleData_melted, aes(x = value, fill = variable)) +
  geom_density(alpha = 0.5) +
  facet_wrap(~ variable, scales = "free") +
  labs( x = "Value", y = "Density",title='Density Histogram of covariants
',tag='Fig 2.3.1')

sampleData_melted['index']=rep(seq(1,1000,1),9)
ggplot(sampleData_melted, aes(x = index,y=value, fill = variable)) +
  geom_line(alpha = 0.5) +
  facet_wrap(~ variable, scales = "free") +
  labs( x = "Iteration", y = "Value",title='Value of covariants
',tag='Fig 2.3.2')
# 1000 Sample from the RWMSampler
# PowerSeller=1,VerifyID=1,Sealed=1,MinBlem=0,MajBlem=1
# LargNeg=0,LogBook=1.2,MinBidShare=0.8
params=c(1,1,1,1,0,1,0,1.2,0.8)
result=data.frame(matrix(0,nrow=nrow(sampleData),ncol=0))
for(i in 1:nrow(sampleData)){
  lambda=exp(params%*%as.numeric(sampleData[i,]))
  result[i,1]=rpois(1,lambda)
}

colnames(result)=c('y')

#Plot
ggplot(data=result,aes(x=y))+
  geom_histogram(aes(y=..density..),binwidth=1,alpha=0.5)+
  geom_density(color='red',size=0.7,adjust=2.5)+
  labs(title='The predictive distribution of the number of bidders
',tag='Fig 2.4')+
  xlab("the number of bidders (y)")+
  theme_bw()
cat('The probability of no bidders in this new auction:',mean(result==0))

# Write a function in R that simulates data from the AR(1)-process
simulateAR=function(T_num,mu,phi,sigma){
```

```
  x=mu
  sampleData=c()
  for(i in 1:T_num){
    x=mu+phi*(x-mu)+rnorm(1,0,sqrt(sigma))
    sampleData=c(sampleData,x)
  }
  return(sampleData)
}

mu=13
sigma=3
T_num=300

# Simulations
# Phi=-0.3
sampleData1=simulateAR(T_num,mu,-0.3,sigma)
ggplot(data.frame("x_t"=sampleData1,"T"=1:T_num))+geom_line(aes(x=T,y=x_t))+
  labs(title='Simulated data from the AR(1)-process
',subtitle="Phi=-0.3",tag='Fig 3.1.1')+
  xlab("T")+
  ylab("x_t value")+
  theme_bw()

# Phi=0
sampleData2=simulateAR(T_num,mu,0,sigma)
ggplot(data.frame("x_t"=sampleData2,"T"=1:T_num))+geom_line(aes(x=T,y=x_t))+
  labs(title='Simulated data from the AR(1)-process
',subtitle="Phi=0",tag='Fig 3.1.2')+
  xlab("T")+
  ylab("x_t value")+
  theme_bw()

# Phi=0.6
sampleData3=simulateAR(T_num,mu,0.6,sigma)
ggplot(data.frame("x_t"=sampleData3,"T"=1:T_num))+geom_line(aes(x=T,y=x_t))+
  labs(title='Simulated data from the AR(1)-process
',subtitle="Phi=0.6",tag='Fig 3.1.3')+
  xlab("T")+
  ylab("x_t value")+
  theme_bw()

mu=13
sigma=3
T_num=300

# Phi=0.2
sampleData4=simulateAR(T_num,mu,0.2,sigma)

# Phi=0.95
sampleData5=simulateAR(T_num,mu,0.95,sigma)

stancode='
// Save as armodel.stan
data {
```

```
  int<lower=0> J;         //The number of observations
  vector[J] y;              //The observations
}

parameters {              //Parameters
  real mu;
  real phi;
  real<lower=0> sigma;
}

model {
  // prior distribution(Non-informative)
  mu ~ normal(0,100);
  phi ~ uniform(-1,1);
  sigma ~ scaled_inv_chi_square(1,2);

  for (n in 2:J)
    y[n] ~ normal(mu+phi*(y[n-1]-mu),sqrt(sigma));

}'


# Phi=0.2
fit1 <- stan(model_code =stancode, data =list(J=300,y=sampleData4), iter = 2000, warmup = 1000, chains =

posterior_mean_1 <- summary(fit1)$summary[1:3, c(1,4,8)]
cat("Phi=0.2:\nPosterior Mean:\n")
print(posterior_mean_1[,1])
cat("95% credible intervals:\n")
print(posterior_mean_1[,c(2,3)])

# Effective posterior samples
params_1=extract(fit1)
params_1_data=data.frame("iteration"=1:4000,"mu"=params_1$mu,"phi"=params_1$phi,"sigma"=params_1$sigma)

#Plot
# mu
ggplot(data=params_1_data)+geom_line(aes(x=iteration,y=mu))+
  labs(title='Posterior mu from the AR(1)-processs sample
',subtitle="true Phi=0.2,true mu=13",tag='Fig 4.1.1')+
  xlab("Iteration")+
  ylab("mu value")+
  theme_bw()

# phi
ggplot(data=params_1_data)+geom_line(aes(x=iteration,y=phi))+
  labs(title='Posterior phi from the AR(1)-process sample
',subtitle="true Phi=0.2",tag='Fig 4.1.2')+
  xlab("Iteration")+
  ylab("phi value")+
  theme_bw()

#sigma
```

```
ggplot(data=params_1_data)+geom_line(aes(x=iteration,y=sigma))+
  labs(title='Posterior mu from the AR(1)-processs sample
',subtitle="true Phi=0.2,true sigma=3",tag='Fig 4.1.3')+
  xlab("Iteration")+
  ylab("sigma value")+
  theme_bw()
# Phi=0.95
fit2 <- stan(model_code =stancode, data = list(J=300,y=sampleData5), iter = 2000, warmup = 1000, chains

posterior_mean_2 <- summary(fit2)$summary[1:3, c(1,4,8)]
cat("Phi=0.95:\nPosterior Mean:\n")
print(posterior_mean_2[,1])
cat("95% credible intervals:\n")
print(posterior_mean_2[,c(2,3)])

# Effective posterior samples
params_2=extract(fit2)
params_2_data=data.frame("iteration"=1:4000,"mu"=params_2$mu,"phi"=params_2$phi,"sigma"=params_2$sigma)

#Plot
# mu
ggplot(data=params_2_data)+geom_line(aes(x=iteration,y=mu))+
  labs(title='Posterior mu from the AR(1)-processs sample
',subtitle="true Phi=0.95,true mu=13",tag='Fig 4.1.4')+
  xlab("Iteration")+
  ylab("mu value")+
  theme_bw()

# phi
ggplot(data=params_2_data)+geom_line(aes(x=iteration,y=phi))+
  labs(title='Posterior phi from the AR(1)-process sample
',subtitle="true Phi=0.95",tag='Fig 4.1.5')+
  xlab("Iteration")+
  ylab("phi value")+
  theme_bw()

#sigma
ggplot(data=params_2_data)+geom_line(aes(x=iteration,y=sigma))+
  labs(title='Posterior mu from the AR(1)-processs sample
',subtitle="true Phi=0.95,true sigma=3",tag='Fig 4.1.6')+
  xlab("Iteration")+
  ylab("sigma value")+
  theme_bw()

# Plot the joint posterior
# phi=0.2
ggplot(data=params_1_data)+geom_point(aes(x=mu,y=phi),color='orange')+
  labs(title='Joint posterior of mu and phi from the sampler
',subtitle="true Phi=0.2,true mu=13",tag='Fig 4.2.1')+
  xlab("mu")+
  ylab("phi")+
  theme_bw()
```

```r
# phi=0.95
ggplot(data=params_2_data)+geom_point(aes(x=mu,y=phi),color='lightgreen')+
  labs(title='Joint posterior of mu and phi from the sampler
',subtitle="true Phi=0.95,true mu=13",tag='Fig 4.2.2')+
  xlab("mu")+
  ylab("phi")+
  theme_bw()
```