Bayesian_Lab3

Olayemi Morrison(olamo208) Greeshma Jeev Koothuparambil (greko370)

2024-05-13

Assignment 1

Question 1a

Implement (code!) a Gibbs sampler that simulates from the joint posterior p(ω, β|x) $by augmenting the data with Polya-gamma latent variables1 $\omega_i = 1, ..., n$. The full conditional posteriors are given on the slides from Lecture 7. Evaluate the convergence of the Gibbs sampler by calculating the Inefficiency Factors (IFs) and by plotting the trajectories of the sampled Markov chains.

```
# Library needed
library(BayesLogit)
library(mvtnorm)
#initialisation
tau <- 3
nDraws <- 1000 # Number of draws

# Loading the women dataset
WomenData <- data.frame(read.csv("WomenAtWork.dat",sep = " ")) # read data from file
Nobs <- dim(WomenData)[1] # number of observations
y <- as.matrix(WomenData$Work)

Covs <- c(2:8) # Select which covariates/features to include
X <- as.matrix(WomenData[,Covs]);
Xnames <- colnames(X)

Npar <- dim(X)[2]

# Setting up the prior
b <- matrix(0,Npar,1) # Prior mean vector
B <- tau^2*diag(Npar) # Prior covariance matrix

# Select the initial values for beta
initVal <- rmvnorm(1, mean = b, sigma = B)

betas <- initVal

w <- rep(0,nrow(X))

gibbsDrawsBeta <- matrix(0,nDraws,7)
k= y-(1/2)
for (i in 1:nDraws){


  for( row in 1:nrow(X)){
```

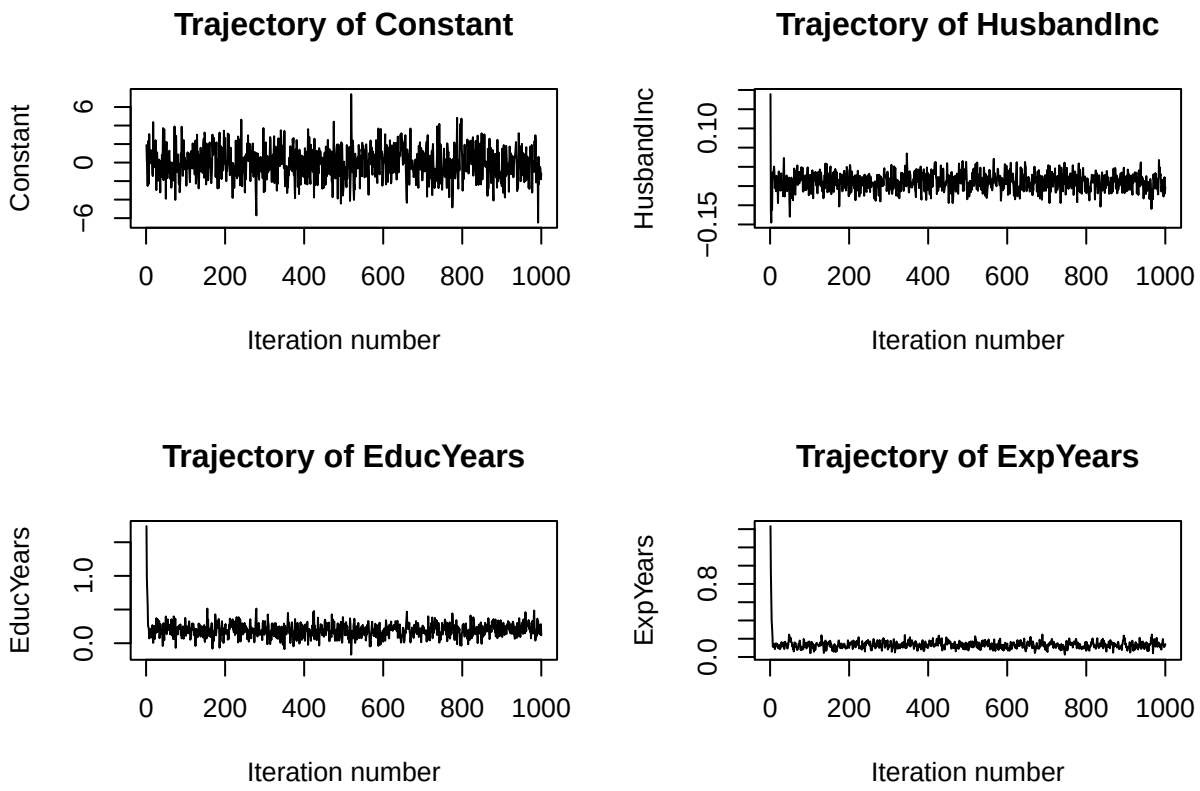
```
    # Update w given beta
    xTransposebeta <- X[row,]%*%t(betas)
    wval <- rpg( 1,  z=xTransposebeta)
    w[row] <- wval
  }
  # Update beta given w
  omega= diag(x = w)

  VOmega <- solve(t(X)%*%omega%*%X+solve(B))
  mOmega <- VOmega%*%(t(X)%*% k+solve(B)%*%b)
  betas <- rmvnorm(1, mean = mOmega, sigma = VOmega)

  gibbsDrawsBeta[i,] <- betas
}


#Inefficiency Factor Calculation
IFVals <- rep(0,ncol(X))
for (beta in 1:ncol(X)) {
  a_Gibbs <- acf(gibbsDrawsBeta[,beta],plot = FALSE)

  IFVals[beta] <- 1+2*sum(a_Gibbs$acf[-1])

}
```

The Inefficiency Factors(IF) are :

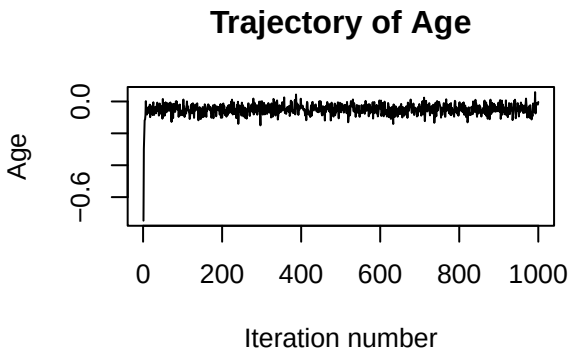
```
##    Constant  HusbandInc   EducYears    ExpYears         Age NSmallChild
##    1.518871    0.996794    2.089839    2.729482    1.801379    2.335860
##   NBigChild
##    1.801048
```

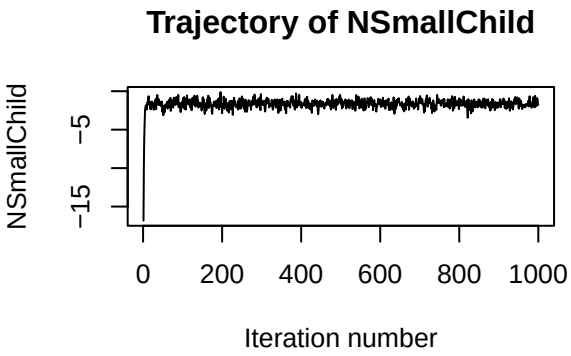
The trajectories of the sampled Markov Chains are plotted below:

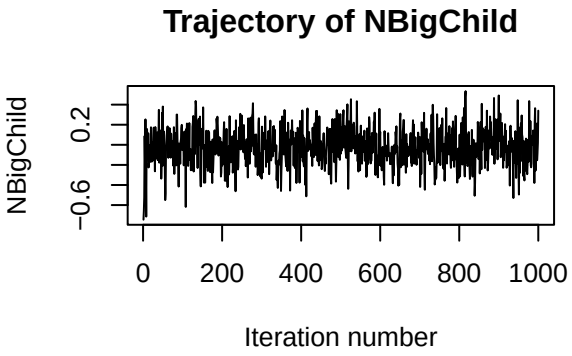
```
par(mfrow=c(2,2))

for(column in 1:4){
  main <-  paste0("Trajectory of ", colnames(X)[column])
  plot(gibbsDrawsBeta[,column],type="l", main= main, xlab='Iteration number', ylab=colnames(X)[column])

}
```




```
par(mfrow=c(2,2))

for(column in 5:ncol(X)){
  main = paste0("Trajectory of ", colnames(X)[column])
  plot(gibbsDrawsBeta[,column],type="l", main= main, xlab='Iteration number', ylab= colnames(X)[column])

}
```







Question 1b

Use the posterior draws from a) to compute a 90% equal tail credible interval for Pr(y = 1|x), where the values of x corresponds to a 38-year-old woman, with one child (3 years old), 12 years of education, 7 years of experience, and a husband with an income of 22. A 90% equal tail credible interval (a, b) cuts off 5% percent of the posterior probability mass to the left of a, and 5% to the right of b.

```
newxVals <- c( 1, 22, 12,  7, 38,  1,  0)

PredYofnewX <- newxVals %*% t(gibbsDrawsBeta)
probOfy <- exp(PredYofnewX)/1+exp(PredYofnewX)

CI90 <-  quantile(probOfy,c(0.05,0.95))
print("The credible interval of equal tail 95% for Y of the new X is :\n")
```

```
## [1] "The credible interval of equal tail 95% for Y of the new X is :\n"
```

```
print(CI90)
```

```
##        5%       95%
## 0.2654871 1.3027731
```

```
#plot(density(probOfy), type = 'l', lwd = 3, col = "blue",main = "Density distribution of Pr(Y=1|x)")
```

Question 2a

Obtain the maximum likelihood estimator of Beta in the Poisson regression model for the eBay data. Which covariates are significant?

```r
library(mvtnorm)
library(ggplot2)
library(reshape2)

bidderdf <- read.table("eBayNumberOfBidderData_2024.dat", header = TRUE)

model=glm(nBids~.,family = "poisson", data = bidderdf[,-2])

coeffs <- summary(model)$coefficients[-1,4]<0.05

sig_coeffs <- names(coeffs)[coeffs==TRUE]

cat('The significant coefficients are :',sig_coeffs)
```

```
## The significant coefficients are : VerifyID Sealed MajBlem LogBook MinBidShare
```

Question 2b

Let's do a Bayesian analysis of the Poisson regression.

```r
#Question 2b
# Parameters
X=as.matrix(bidderdf[,-1])
y=bidderdf[,1]
mu=as.matrix(rep(0,dim(X)[2]))
sigma=as.matrix(100*solve(t(X)%*%X))
Beta0=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(Beta0,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
postMode=OptimRes[["par"]]
betaHessian=-OptimRes$hessian
JpostMode=solve(betaHessian)
cat('The posterior mode is:\n')
```

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

```r
print(postMode)
```

```
##               [,1]
##  [1,]  1.07721720
##  [2,] -0.03567963
```

```
##  [3,] -0.45353183
##  [4,]  0.45484863
##  [5,] -0.06863401
##  [6,] -0.22583912
##  [7,]  0.05387677
##  [8,] -0.08454639
##  [9,] -1.82275698
```

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

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

```
print(JpostMode)
```

```
##                [,1]          [,2]          [,3]          [,4]          [,5]
##  [1,]  1.148909e-03 -8.900109e-04 -0.0003857630 -3.819281e-04 -5.338519e-04
##  [2,] -8.900109e-04  1.736857e-03 -0.0001082339 -3.044655e-04  7.428731e-05
##  [3,] -3.857630e-04 -1.082339e-04  0.0161686358 -9.387595e-04  1.663127e-04
##  [4,] -3.819281e-04 -3.044655e-04 -0.0009387595  3.877758e-03  4.467713e-04
##  [5,] -5.338519e-04  7.428731e-05  0.0001663127  4.467713e-04  5.181552e-03
##  [6,] -3.275749e-04 -2.789588e-04  0.0003412801  5.282338e-04  4.407885e-04
##  [7,] -6.095160e-04  3.646123e-04  0.0003556730  3.758850e-04  6.458372e-05
##  [8,]  4.160506e-05  1.732911e-04 -0.0003755918 -5.808656e-05 -1.413861e-06
##  [9,]  1.323960e-03 -6.727649e-04 -0.0008293714 -1.322281e-04 -1.987691e-04
##                [,6]          [,7]          [,8]          [,9]
##  [1,] -0.0003275749 -6.095160e-04  4.160506e-05  1.323960e-03
##  [2,] -0.0002789588  3.646123e-04  1.732911e-04 -6.727649e-04
##  [3,]  0.0003412801  3.556730e-04 -3.755918e-04 -8.293714e-04
##  [4,]  0.0005282338  3.758850e-04 -5.808656e-05 -1.322281e-04
##  [5,]  0.0004407885  6.458372e-05 -1.413861e-06 -1.987691e-04
##  [6,]  0.0090771414  5.029108e-04 -1.357642e-04  2.878211e-04
##  [7,]  0.0005029108  4.106307e-03 -3.195004e-04 -5.372066e-05
##  [8,] -0.0001357642 -3.195004e-04  1.045596e-03  1.247802e-03
##  [9,]  0.0002878211 -5.372066e-05  1.247802e-03  6.126074e-03
```

Question 2c

Let's simulate from the actual posterior of Beta using the Metropolis algorithm and compare the results with the approximate results in b). Program a general function that uses the Metropolis algorithm to generate random draws from an arbitrary posterior density.

```
#Question 2c
# A general function that uses the Metropolis algorithm to generate random draw from arbitrary posterio
RWMSampler=function(num,logPostFunc,c){
sampledf=data.frame(matrix(0,nrow=num,ncol=9))
colnames(sampledf)=colnames(X)
# The first parameter should be theta
theta=rmvnorm(1,postMode,c*JpostMode)
sampledf[1,]=theta
count=1
while(count<num){
  theta=as.numeric(sampledf[count,])
  thetap=as.numeric(rmvnorm(1,theta,c*JpostMode))
  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
    sampledf[count,]=thetap
  }
}
return(sampledf)
}
#Sample from the posterior of beta
sampledf=RWMSampler(1000,logPosterior,2)
print(sampledf[1:20,])
```

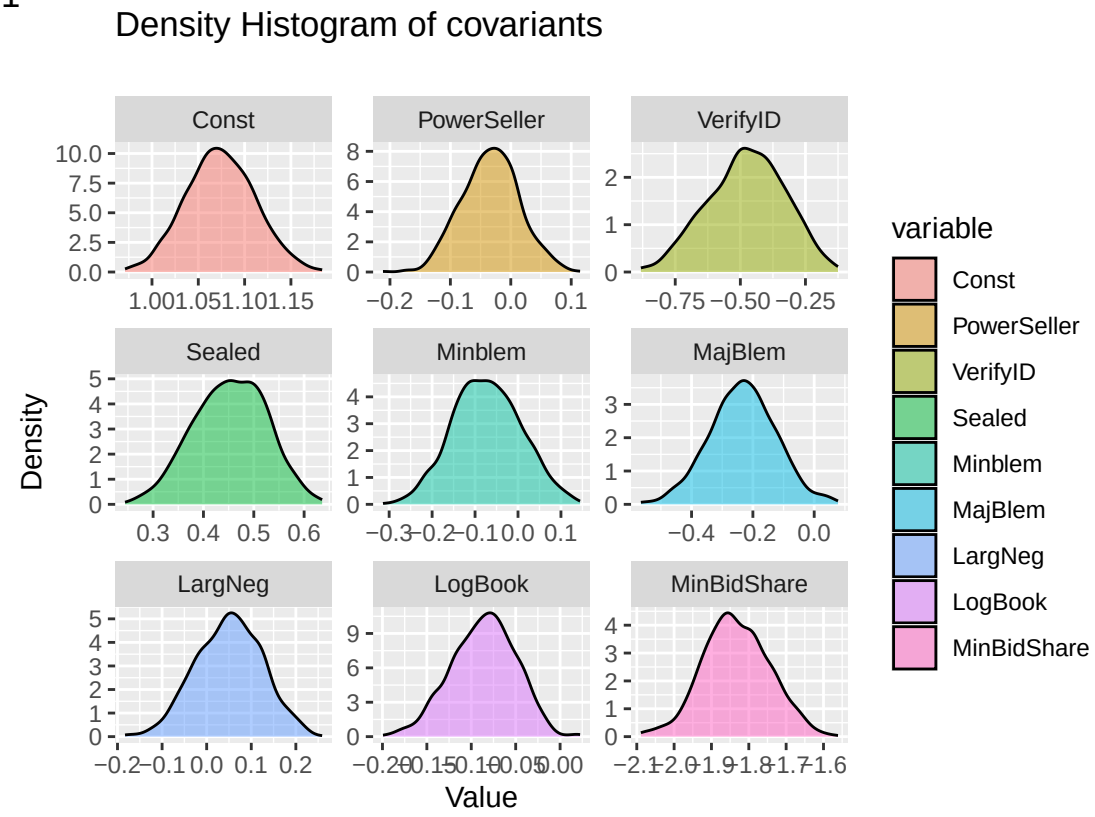
```
##       Const   PowerSeller   VerifyID    Sealed       Minblem     MajBlem
## 1  1.026587 -0.0727403852 -0.3639639 0.5126137  1.300839e-05 -0.13957457
## 2  1.055011 -0.1123012414 -0.2481763 0.4575327 -4.296562e-02 -0.17876728
## 3  1.024211 -0.0200865640 -0.3240748 0.3974649 -1.275033e-02 -0.08337553
## 4  1.070356  0.0049654837 -0.4341366 0.4231363 -1.225379e-01 -0.15722383
## 5  1.062352  0.0638089335 -0.5383879 0.3469747 -1.756131e-01 -0.22974010
## 6  1.061159  0.0478524679 -0.4420148 0.4676695 -1.422545e-01 -0.12393826
## 7  1.086053 -0.0491153603 -0.6911923 0.4491297 -5.987286e-02 -0.03798955
## 8  1.070361  0.0133783315 -0.5865714 0.4346951 -4.984131e-02 -0.27892294
## 9  1.076829  0.0006915458 -0.3716994 0.3650008 -8.159337e-02 -0.24720278
## 10 1.128723  0.0069870804 -0.4248364 0.2594463 -1.794874e-01 -0.35018648
## 11 1.139831 -0.0810014718 -0.4479535 0.3849318 -1.555056e-01 -0.18720637
## 12 1.100104 -0.0814638495 -0.5006339 0.4949088 -1.624989e-01 -0.19717640
## 13 1.090646 -0.0376524975 -0.4495437 0.5216852 -2.898301e-02 -0.29809536
## 14 1.138449 -0.0817306999 -0.4195038 0.3769454 -7.355752e-02 -0.28001622
## 15 1.108520 -0.0748403057 -0.5631412 0.4169133 -1.196988e-01 -0.16160554
## 16 1.109073 -0.0783768083 -0.6059269 0.4350438 -5.729718e-02 -0.06587613
## 17 1.141618 -0.1062840902 -0.4828497 0.3540739 -9.810661e-02 -0.31666785
## 18 1.090914 -0.1138320619 -0.3588884 0.3746487  5.220922e-02 -0.12252924
## 19 1.089762 -0.0636673523 -0.3373482 0.3842489  3.507298e-02 -0.21963280
## 20 1.074766 -0.0142418453 -0.6594162 0.4460480  2.405905e-02 -0.30942674
##         LargNeg     LogBook MinBidShare
## 1   0.043362193 -0.09912674   -1.890263
## 2   0.071596288 -0.08885583   -1.831482
## 3   0.041991079 -0.11513837   -2.007434
## 4  -0.092662632 -0.08110095   -1.758339
## 5  -0.036820632 -0.08168122   -1.804603
## 6  -0.026920062 -0.12313669   -1.870417
## 7  -0.008897534 -0.14551055   -1.893143
## 8   0.099239502 -0.11695900   -1.826682
## 9  -0.025222692 -0.12754770   -1.885787
## 10 -0.041798951 -0.10965658   -1.813216
## 11 -0.032256912 -0.07293518   -1.841198
## 12  0.047314216 -0.08919497   -1.887034
## 13  0.060175596 -0.04018544   -1.825767
## 14  0.064910478 -0.06687176   -1.777424
## 15  0.118738082 -0.12930217   -1.883492
## 16 -0.074030904 -0.13624922   -1.779506
## 17 -0.183136919 -0.12633789   -1.792013
## 18 -0.054919935 -0.09032902   -1.741707
## 19 -0.004812024 -0.01899263   -1.655775
## 20 -0.012168323 -0.02153207   -1.655321
```

```
sampledf_melted <- melt(sampledf)
```

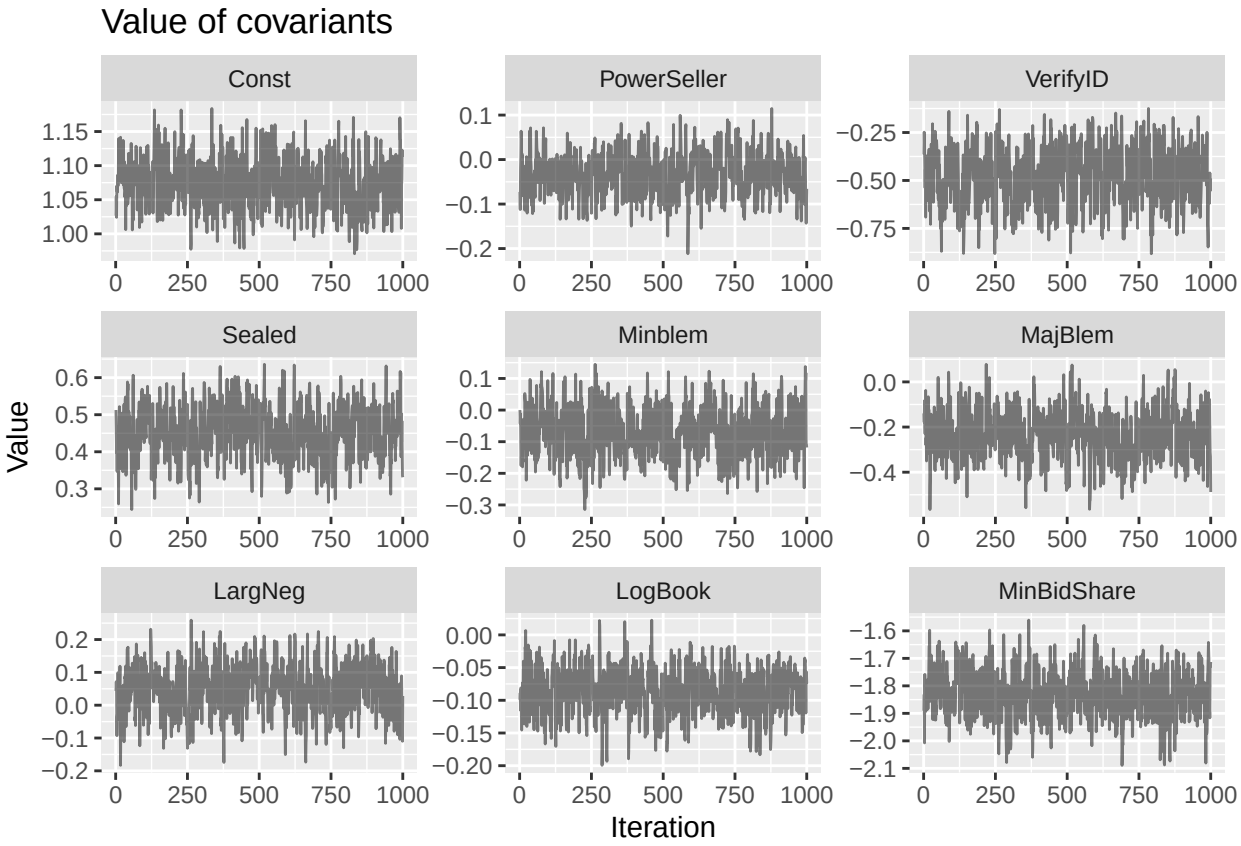
```
## No id variables; using all as measure variables
```

```
## No id variables; using all as measure variables
ggplot(sampledf_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



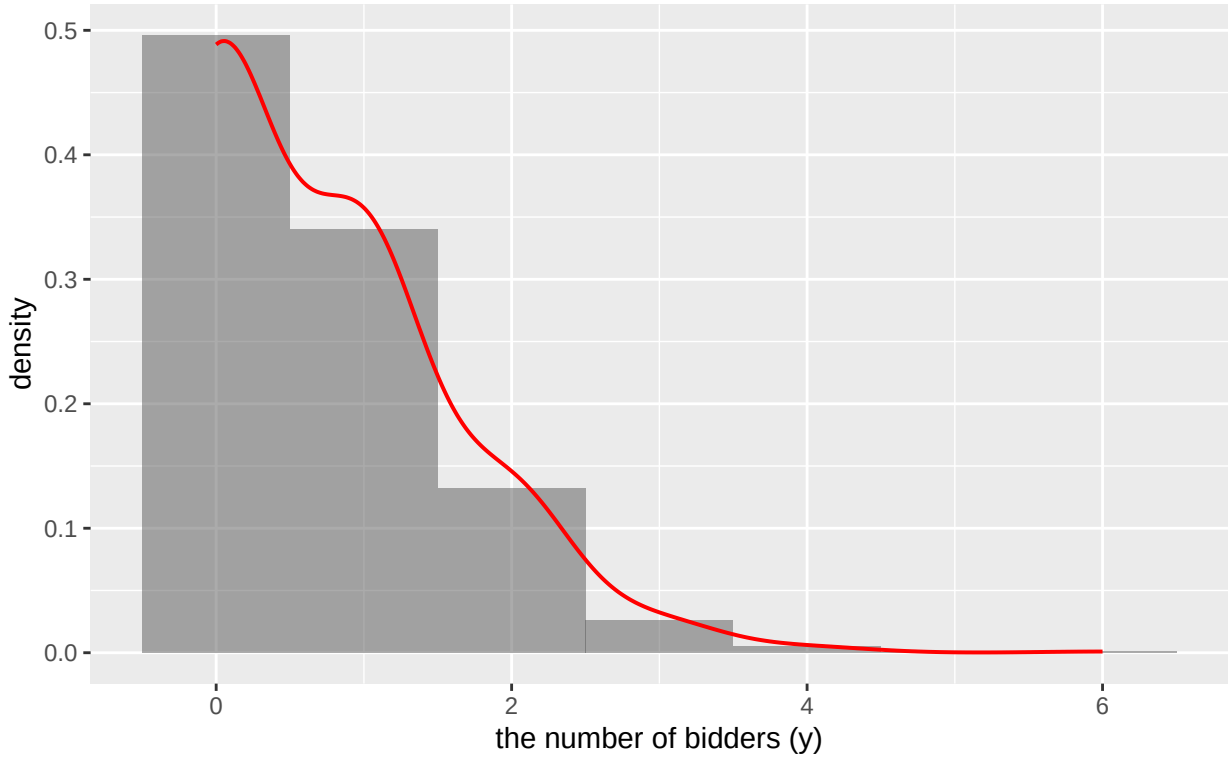

```
sampledf_melted['index']=rep(seq(1,1000,1),9)
ggplot(sampledf_melted, aes(x = index,y=value, fill = variable)) +
  geom_line(alpha = 0.5) +
  facet_wrap(~ variable, scales = "free") +
  labs( x = "Iteration", y = "Value",title='Value of covariants')
```



Question 2d

Use the MCMC draws from c) to simulate from the predictive distribution of the number of bidders in a new auction with the characteristics below. Plot the predictive distribution. What is the probability of no bidders in this new auction?

```
#Question 2d
# 1000 Sample from the RWMSampler
# PowerSeller=1,VerifyID=0,Sealed=1,MinBlem=0,MajBlem=1
# LargNeg=0,LogBook=1.2,MinBidShare=0.8

params=c(1,1,0,1,0,1,0,1.2,0.8)
result=data.frame(matrix(0,nrow=nrow(sampledf),ncol=0))
for(i in 1:nrow(sampledf)){
  lambda=exp(params%*%as.numeric(sampledf[i,]))
  result[i,1]=rpois(1,lambda)
}
colnames(result)=c('y')
#Plot
ggplot(data=result,aes(x=y))+
  geom_histogram(aes(y=after_stat(density)),binwidth=1,alpha=0.5)+
  geom_density(color='red',linewidth=0.7,adjust=2.5)+
  labs(title='The predictive distribution of the number of bidders
')+ xlab("the number of bidders (y)")
```




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

```
## The probability of no bidders in this new auction: 0.496
```

3a

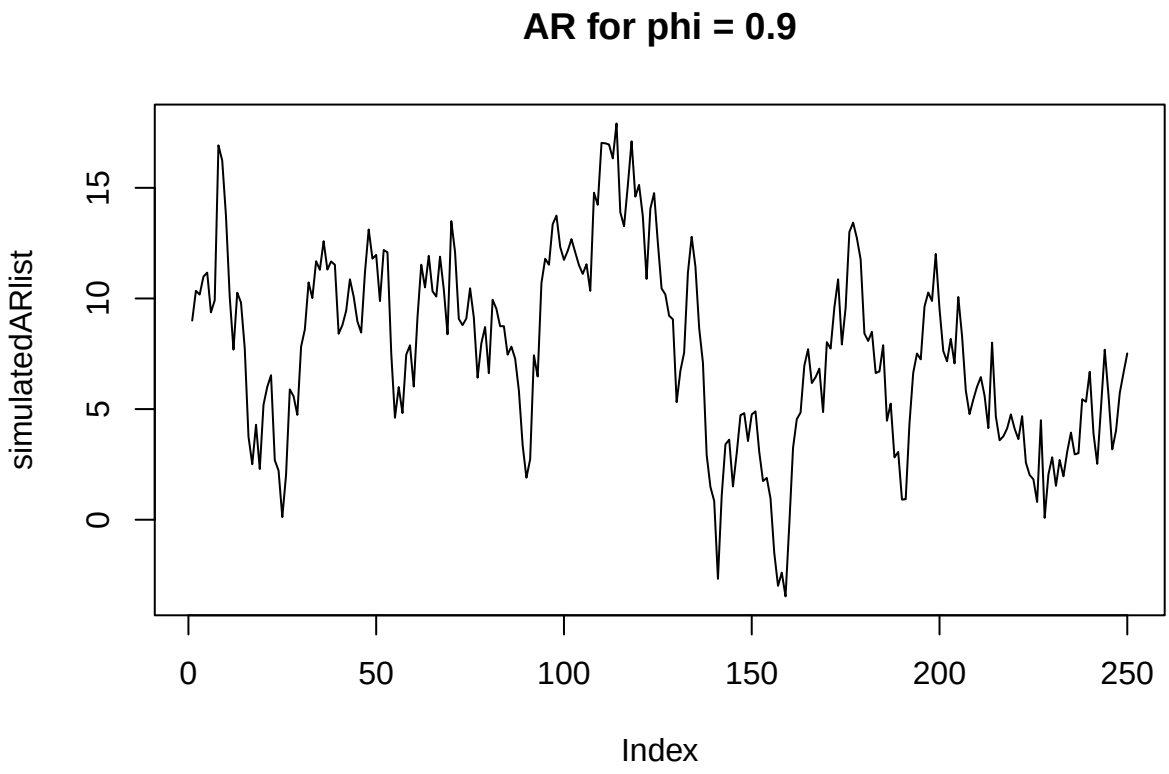
Write a function in R that simulates data from the AR(1)-process

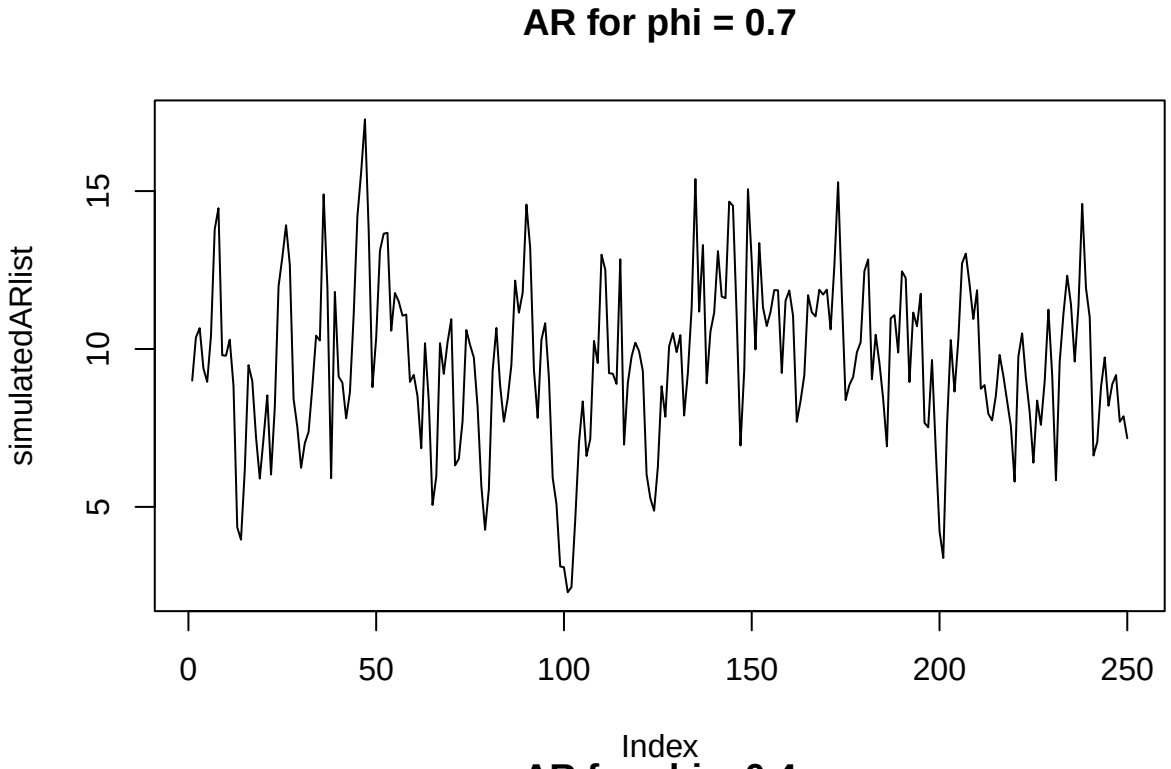
$$x_t = \mu + \phi(x_{t-1} - \mu) + \epsilon_t, \epsilon_t iid \sim \mathsf{N}(0, \sigma^2),$$

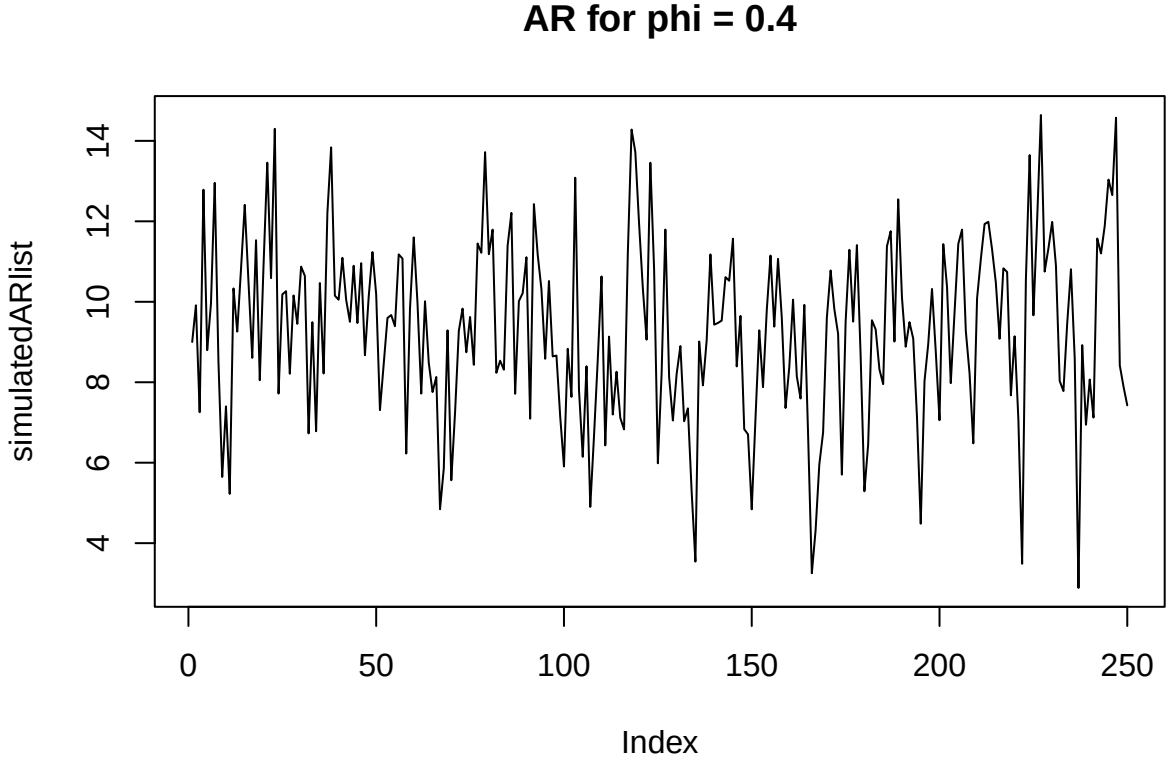
for given values of μ, ϕ and σ^2. Start the process at x1 = μ and then simulate values for xt for t = 2, 3 . . . , T and return the vector x1:T containing all time points. Use $\mu = 9$, $\sigma^2 = 4$ and T = 250 and look at some different realizations (simulations) of x1:T for values of ϕ between -1 and 1 (this is the interval of ϕ where the AR(1)-process is stationary). Include a plot of at least one realization in the report. What effect does the value of ϕ have on x1:T?

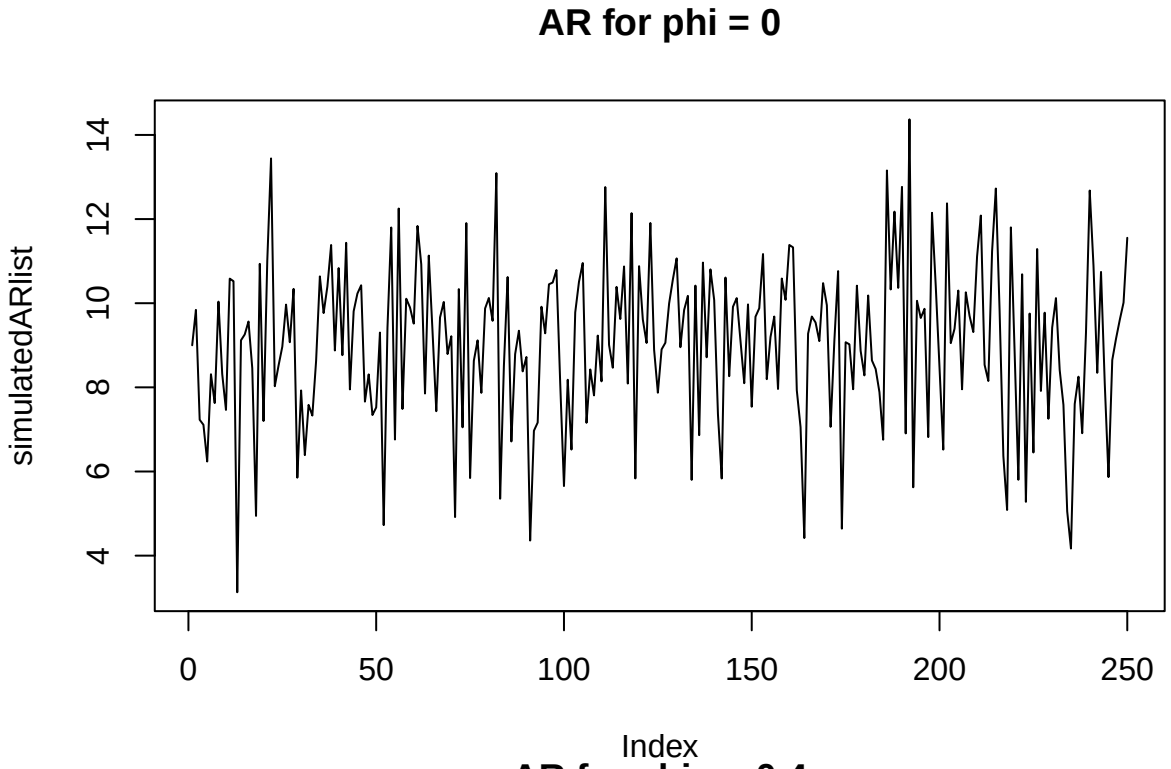
```
library(rstan)

mu <-  9
sigma2 <- 4
TVal <- 250

AR <- function(x1, mu,phi,sigma2){
  eps <- rnorm(1, 0, sqrt(sigma2))
  xnew <- mu+ (phi*(x1-mu))+eps
  return(xnew)
}
```

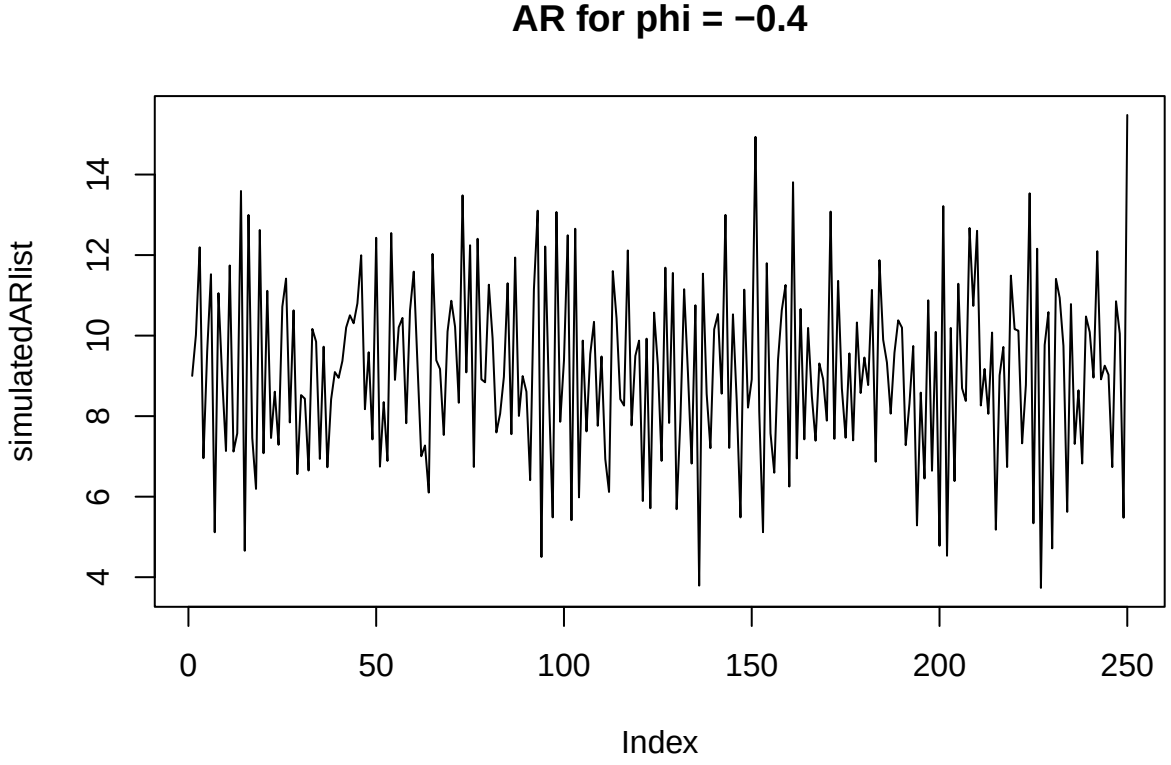
```r
phi <- 0.4
phi <- -0.4
phi <- 0.7
par(mfrow=c(1,1))

simulateAR <- function(TVals, phi, x1, mu, sigma2){
  #simulatedAR <- data.frame(matrix(nrow = TVals, ncol = length(phivals)))
  #names(simulatedAR) <- phivals
  ARTlist <-rep(0,TVal)
  ARTlist[1] <- mu
  x1 <- mu
  #phi <- phivals[i]

  for(t in 2:TVal){
    ARTlist[t] <- AR(x1,mu, phi, sigma2)
    x1 <- ARTlist[t]

  }

  return(ARTlist)
}
phiVals <- c(0.9, 0.7, 0.4, 0, -0.4, -0.7, -0.9)
for(phi in 1:length(phiVals)){
  phival <- phiVals[phi]
  simulatedARlist <-simulateAR(250, phival, x1, mu, sigma2)
  plot(simulatedARlist, type = "l",main = paste0("AR for phi = ", phival))
}
```

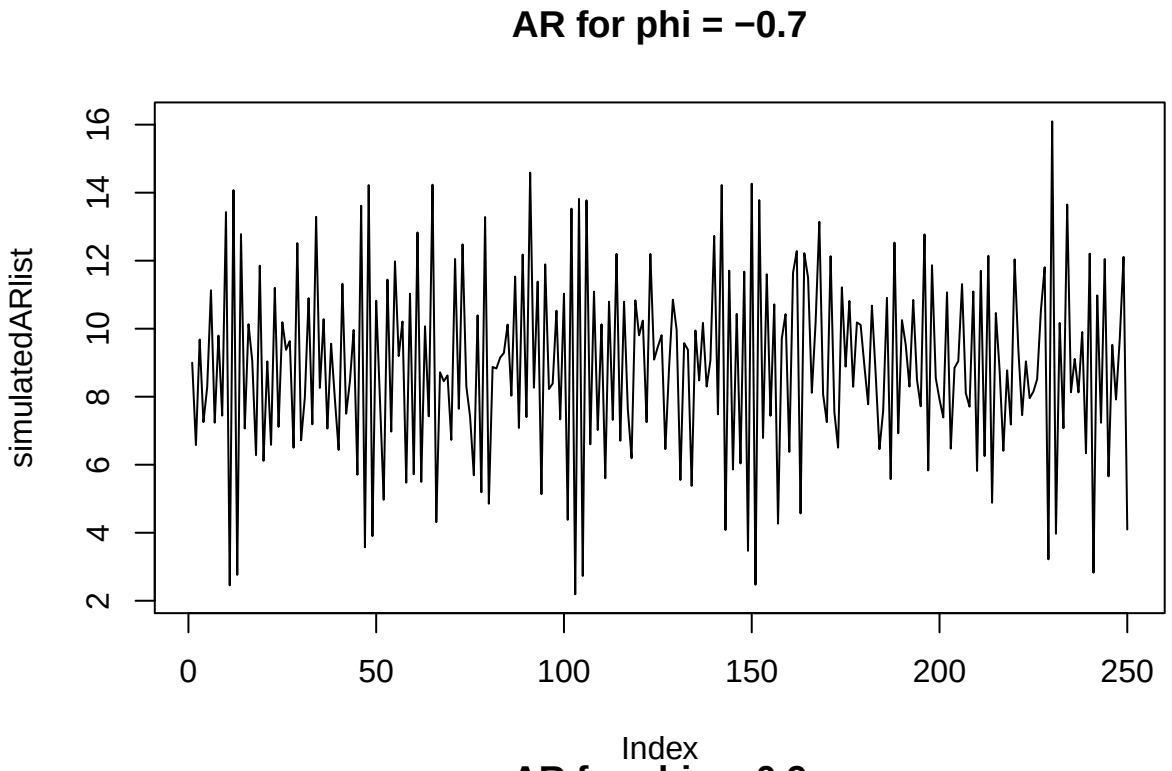


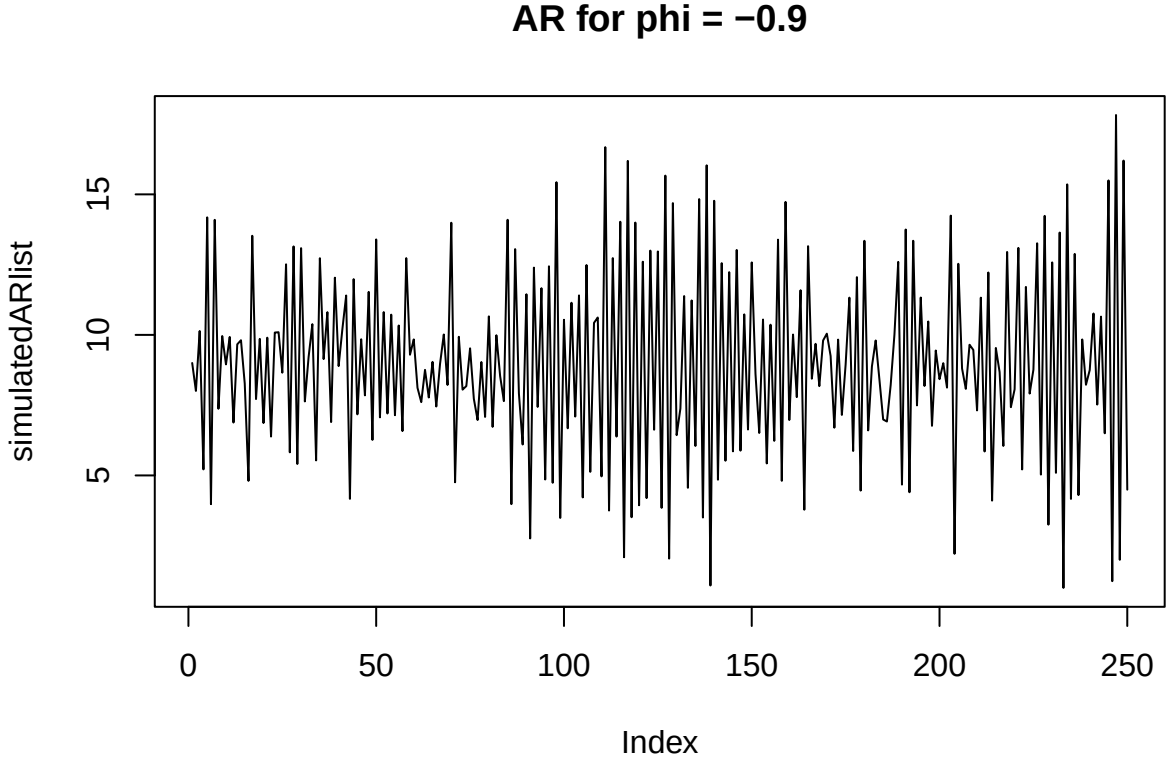

AR for phi = 0.7
simulatedARlist
5
10
15
0
50
100
150
200
250
Index


AR for phi = 0.4
simulatedARlist
4
6
8
10
12
14
0
50
100
150
200
250
Index


AR for phi = 0
simulatedARlist
4
6
8
10
12
14
0
50
100
150
200
250
Index


AR for phi = −0.4
simulatedARlist
4
6
8
10
12
14
0
50
100
150
200
250
Index





Throughout the graph series we can see that as phi values are positive they display a positive correlation between a value and its previous value. We can see a strong increasing or decreasing trend in the graph. While when the phi value falls negative we can see a negative correlation for a value with its previous value. It can be evident from the oscillatory nature of the graph for negative values.

3b

Use your function from a) to simulate two AR(1)-processes, x1:T with $\phi = 0.3$ and y1:T with $\phi = 0.97$. Now, treat your simulated vectors as synthetic data, and treat the values of μ, ϕ and σ^2 as unknown parameters. Implement Stancode that samples from the posterior of the three parameters, using suitable non-informative priors of your choice.

i

Report the posterior mean, 95% credible intervals and the number of effective posterior samples for the three inferred parameters for each of thesimulated AR(1)-process. Are you able to estimate the true values.

```
#3.bi

simulatedARPoint3 <-simulateAR(250, 0.3, x1, mu, sigma2)
simulatedARPoint97 <-simulateAR(250, 0.97, x1, mu, sigma2)

set.seed(12345)

StanModel = '
data {
  int<lower=0> TVals; // Number of observations
  vector[TVals] y ;
}
parameters {
  real mu;
  real <lower= -1, upper =1 > phi;
  real<lower=0> sigma2;
}
model {
  mu ~ normal(0,100); // Normal with mean 0, st.dev. 100
  phi ~ uniform(-1,1);
  sigma2 ~ scaled_inv_chi_square(1,2); // Scaled-inv-chi2 with nu 1,sigma 2
  for(i in 2:TVals){
    y[i] ~ normal(mu+ (phi*(y[i-1]-mu)),sqrt(sigma2));
  }
}'

warmup <- 1000
niter <- 2000
fitPoint3 <- stan(model_code=StanModel,
                  data=list(TVals = TVal,y= simulatedARPoint3),
                  warmup=warmup,iter=niter,chains=4)
```

```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000147 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 1.47 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: 0.496 seconds (Warm-up)
## Chain 1:                0.506 seconds (Sampling)
## Chain 1:                1.002 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 7.7e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.77 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: 0.493 seconds (Warm-up)
## Chain 2:                0.532 seconds (Sampling)
## Chain 2:                1.025 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 7.6e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.76 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: 0.535 seconds (Warm-up)
## Chain 3:                0.474 seconds (Sampling)
## Chain 3:                1.009 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 7.2e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.72 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: 0.71 seconds (Warm-up)
## Chain 4:                0.476 seconds (Sampling)
## Chain 4:                1.186 seconds (Total)
## Chain 4:
```

```
sumpoint3 <-summary(fitPoint3)$summary
```

The Mean, 95% CI and number of effective samples for phi = 0.3 is given in the table below :

```
knitr::kable(sumpoint3[1:3,c(1,4,8,9)],
             col.names = c("Mean", "CI 2.5%", "CI 97.5%", "Number of Effective Samples"))
```

	Mean	CI 2.5%	CI 97.5%	Number of Effective Samples
mu	8.9536819	8.5953384	9.3071211	3534.398
phi	0.3157159	0.1988456	0.4340595	3852.154
sigma2	3.8225952	3.2163111	4.5900255	3831.017

```
fitPoint97 <- stan(model_code=StanModel,
                   data=list(TVals = TVal,y= simulatedARPoint97),
                   warmup=warmup,iter=niter,chains=4)
```

```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
```

```
## Chain 1:
## Chain 1: Gradient evaluation took 9.3e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.93 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: 1.64 seconds (Warm-up)
## Chain 1:                0.743 seconds (Sampling)
## Chain 1:                2.383 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 7.6e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.76 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: 2.034 seconds (Warm-up)
## Chain 2:                1.409 seconds (Sampling)
## Chain 2:                3.443 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0.000151 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 1.51 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: 1.907 seconds (Warm-up)
## Chain 3:                1.39 seconds (Sampling)
## Chain 3:                3.297 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 7.8e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.78 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: 1.673 seconds (Warm-up)
## Chain 4:                0.65 seconds (Sampling)
## Chain 4:                2.323 seconds (Total)
## Chain 4:
```

```
sumpoint97<- summary(fitPoint97)$summary
```

The Mean, 95% CI and number of effiective samples for phi = 0.75 is given in the table below :

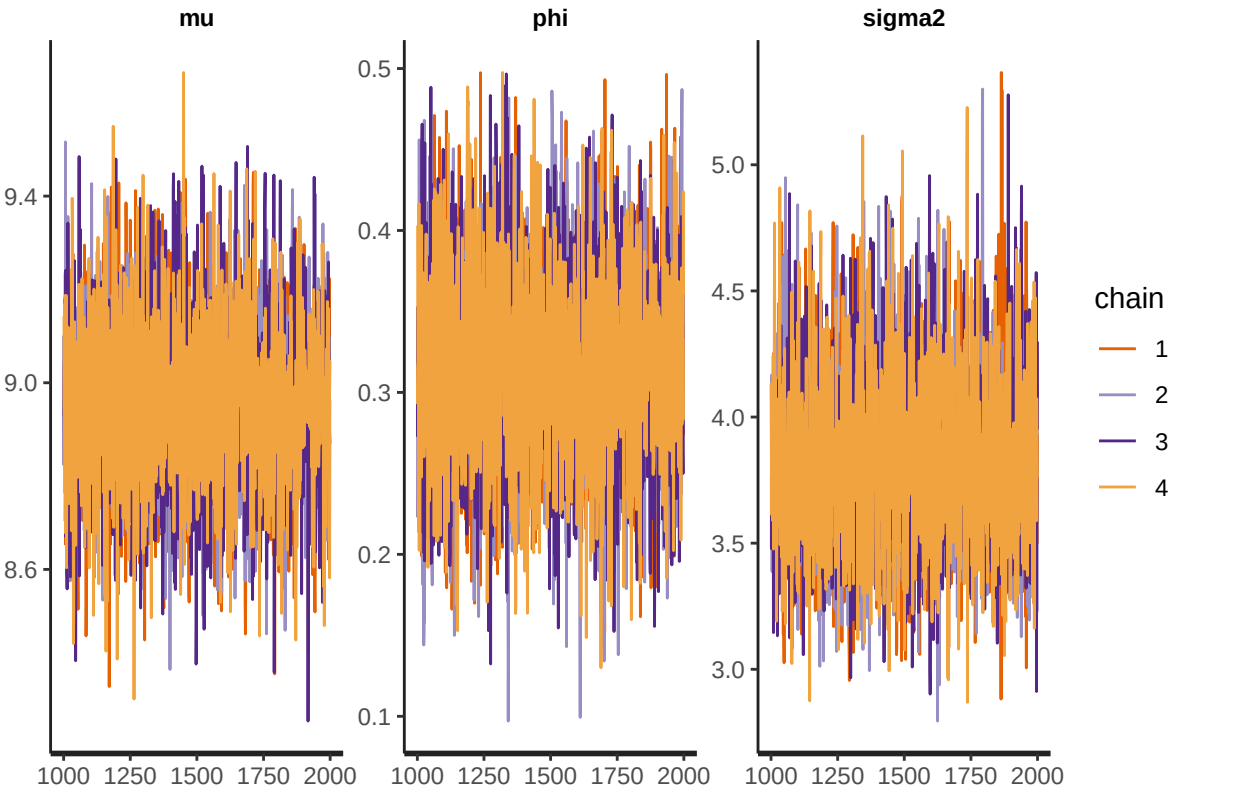
```
knitr::kable(sumpoint97[1:3,c(1,4,8,9)],
             col.names = c("Mean", "CI 2.5%", "CI 97.5%", "Number of Effective Samples"))
```

	Mean	CI 2.5%	CI 97.5%	Number of Effective Samples
mu	3.3300773	-85.0864796	57.7426418	429.7865

	Mean	CI 2.5%	CI 97.5%	Number of Effective Samples
phi	0.9810663	0.9496221	0.9996535	307.6022
sigma2	4.2134433	3.5308939	5.0135012	796.3404

The traceplot for phi =0.3:

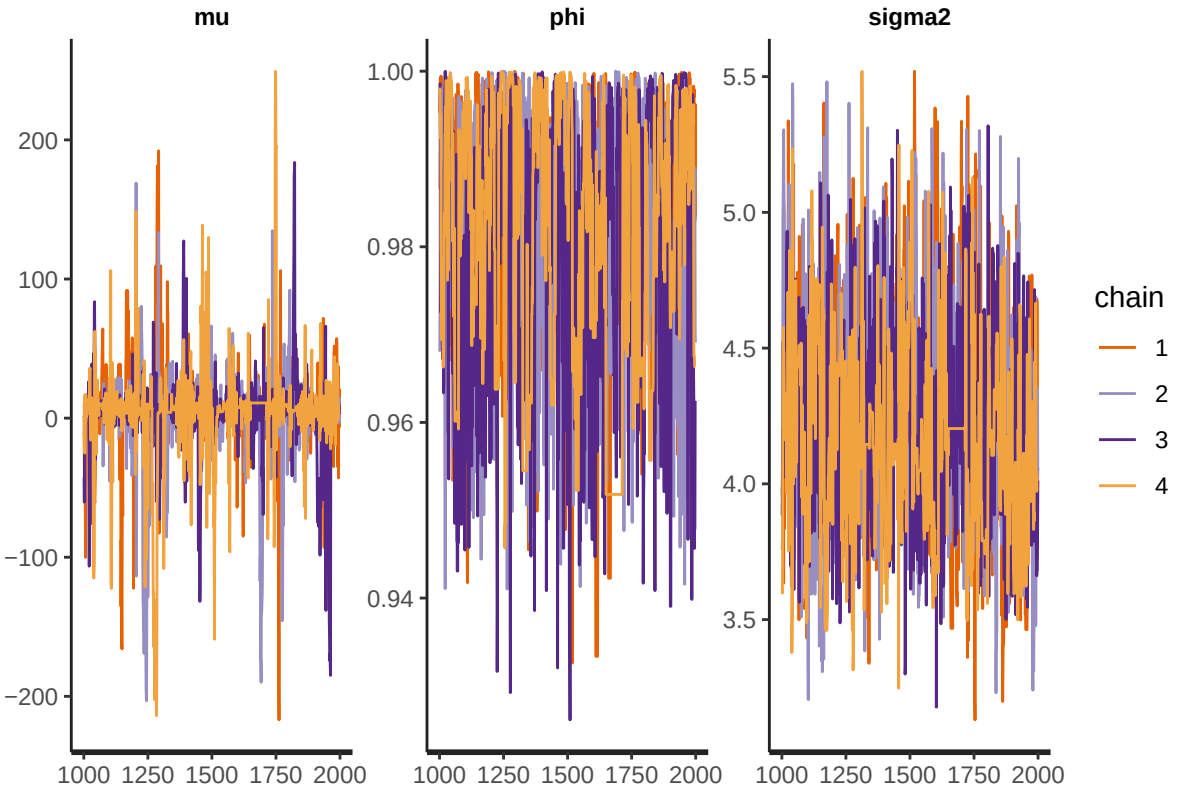
```
postDrawspoint3 <- extract(fitPoint3)
par(mfrow = c(1,1))
# Do automatic traceplots of all chains
traceplot(fitPoint3)
```



From the trace plots we can say that the mu, phi and sigma values converges to the average value and the standard deviation seems quite low which is good in terms of convergence

The traceplot for phi =0.97:

```
postDrawspoint97 <- extract(fitPoint97)
par(mfrow = c(1,1))
# Do automatic traceplots of all chains
traceplot(fitPoint97)
```

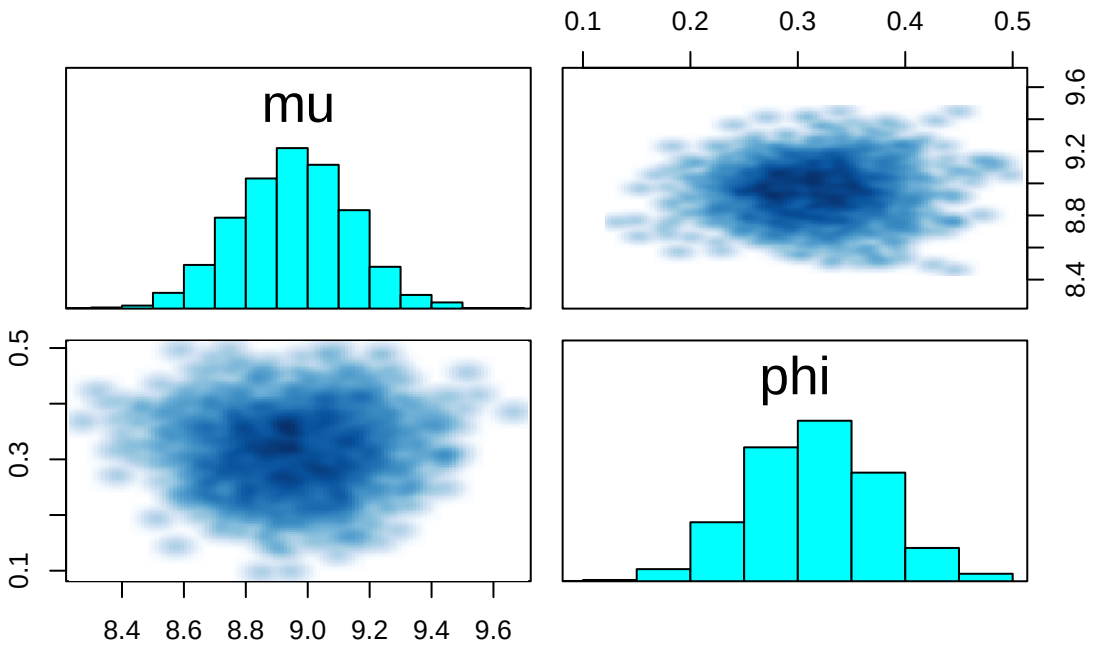


From the trace plots we can say that the mu, phi and sigma values hardly converges to the average value and the standard deviation is really high which is not good in terms of convergence

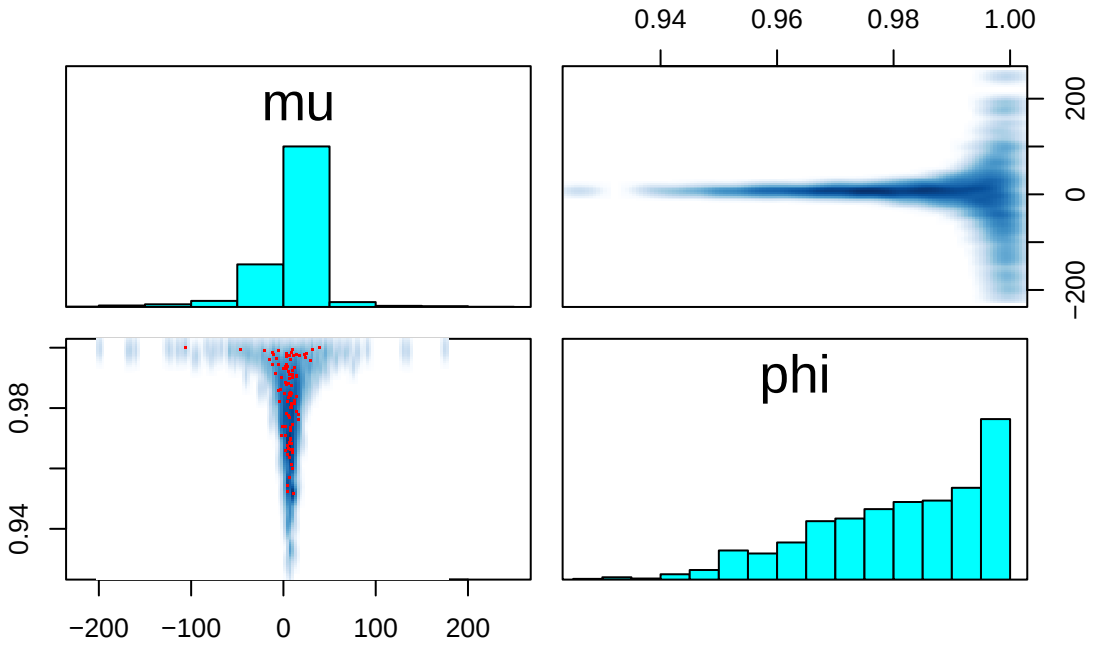
3b-ii

For each of the two data sets, evaluate the convergence of the samplers and plot the joint posterior of μ and ϕ. Comments?

```
#3.b.ii
pairs(fitPoint3, pars = c("mu", "phi"), main = "Convergence of samplers on phi=0.3")
```




```
pairs(fitPoint97, pars = c("mu", "phi"), main = "Convergence of samplers on phi=0.97")
```



For the convergence plot on phi = 0.3 the values converges neatly. It follows a symmetric ellipse pattern which shows a well distributed sample for the distribution. In the case of the convergence plot on phi = 0.97, the graph shows a skewed distribution curve instead of an ellipse. We can see that the values follows a trend in either increasing or decreasing values.