Bayesian Learning Lab 3

Nazli Bilgic (nazbi056) & Simge Cinar (simci637)

2024-05-10

Question 1: Gibbs sampling for the logistic regression

Consider again the logistic regression model in problem 2 from the previous computer lab 2. Use the prior $\beta \sim N(0, \tau^2 I)$, where $\tau = 3$

```
df1 <- read.table("WomenAtWork.dat", header = TRUE)
```

```
Covs <- c(2:8) # Select which covariates/features to include
X <- as.matrix(df1[,Covs])
Xnames <- colnames(X)
y <- as.matrix(df1[,1])

nObs <- dim(df1)[1]
nPar <- dim(df1)[2] -1 # subtract y
```

```
# Setting up the prior
tau <- 3
b <- as.matrix(rep(0, nPar)) # Prior mean vector
B <- (tau^2)*diag(nPar) # Prior covariance matrix
```

Part a)

Question: Implement (code!) a Gibbs sampler that simulates from the joint posterior $p(\omega, \beta|x)$ by augmenting the data with Polya-gamma latent variables $\omega_i, i = 1, ..., n$. 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.

Answer: The joint prior for $p(\omega, \beta|y)$ is as follows:

$$\omega_i|\beta \sim PG(1, x_i'\beta), \text{where } i = 1..n$$

$$\beta|y, \omega \sim N(m_\omega, V_\omega)$$

$$V_\omega = (X^T \Omega X + B^{-1})^{-1}$$

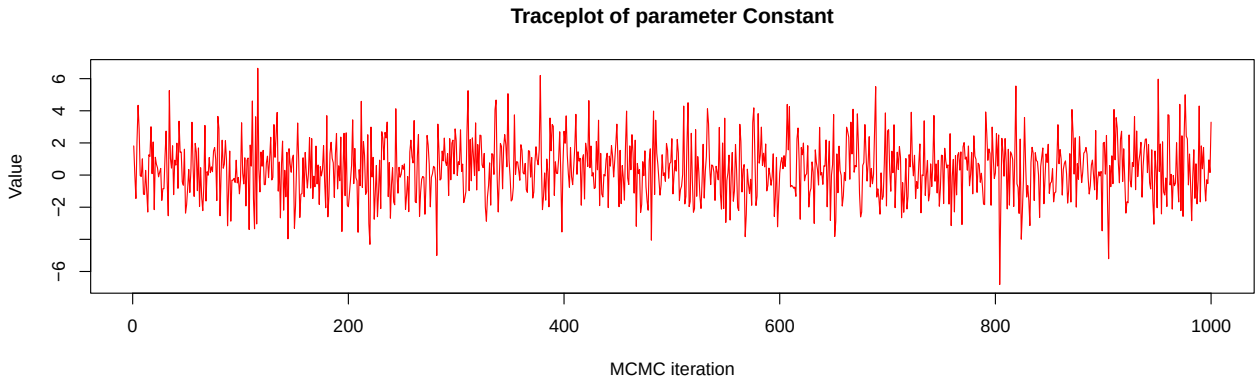
$$m_\omega = V_\omega(X^T \kappa + B^{-1} b)$$

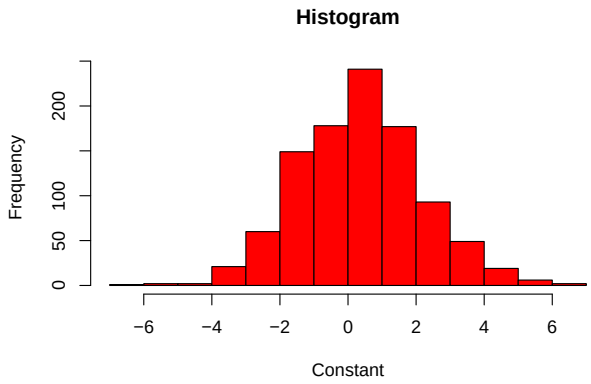
where $\kappa = (y_1 - 1/2, ..., y_n - 1/2)$ and Ω is the diagonal matrix of ω_i

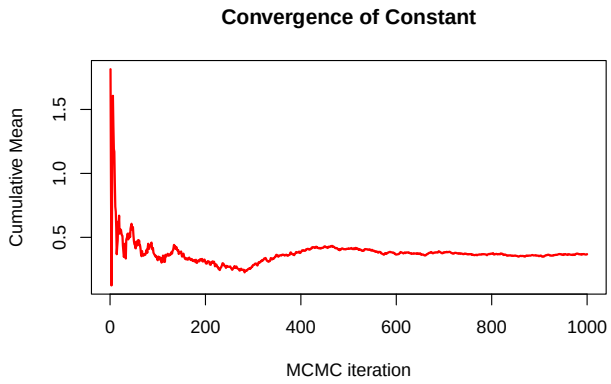
```
set.seed(12345)
# Generating beta draws
nDraws <- 1000
beta_prior <- matrix(0, nrow = nDraws, ncol = nPar)
beta_gibbs <- matrix(0, nrow = nDraws, ncol = nPar)
```

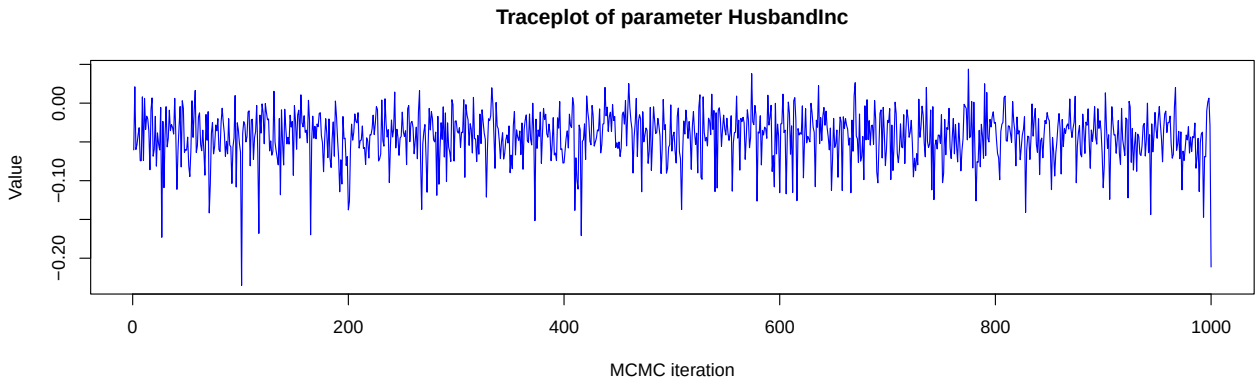
```r
calc_w <- function(row, beta_prior) {
  w <- rpg(1, h = 1, z = row %*% t(beta_prior))
  return(w)
}

for (i in 1:nDraws){
  # draws from prior beta
  beta_prior[i,] <- rmvnorm(1, b, B)

  # w | beta_prior
  w <- apply(X, 1, calc_w, beta_prior = beta_prior[i,])

  # beta | y, w
  Omega <- diag(w)
  k <- y - 0.5
  V_w <- solve((t(X) %*% Omega %*% X) + solve(B))
  m_w <- V_w %*% (t(X) %*% k + solve(B) %*% b)

  beta_gibbs[i,] <- rmvnorm(1, m_w, V_w)
}
```

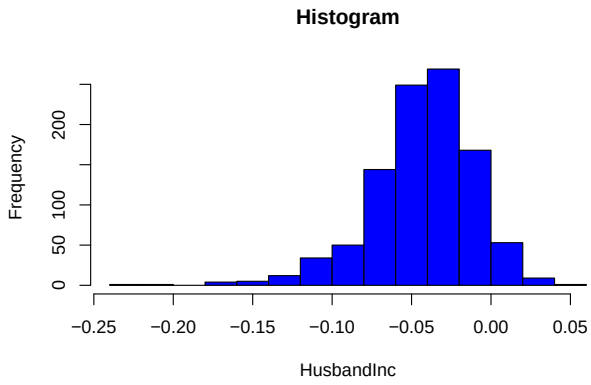
```r
colors <- c("red", "blue", "darkgreen", "brown", "magenta", "orange", "purple")
for (i in 1:nPar){
  par(mfrow=c(1,1))
  # traceplot of Gibbs draws
  plot(1:nDraws, beta_gibbs[,i],
       type = "l",
       col=colors[i],
       main = paste("Traceplot of parameter", colnames(df1)[i+1]),
       xlab = "MCMC iteration",
       ylab = "Value")

  par(mfrow=c(1,2))
  #histogram of Gibbs draws
  hist(beta_gibbs[,i],
       col=colors[i],
       main = "Histogram",
       xlab = paste(colnames(df1)[i+1]))

  # Cumulative mean value, Gibbs draws
  cusumData =  cumsum(beta_gibbs[,i])/seq(1, nDraws)
  plot(1:nDraws, cusumData, type = "l",
       col=colors[i],
       lwd = 2,
       main = paste("Convergence of", colnames(df1)[i+1]),
       ylab = "Cumulative Mean",
       xlab = "MCMC iteration")
}
```

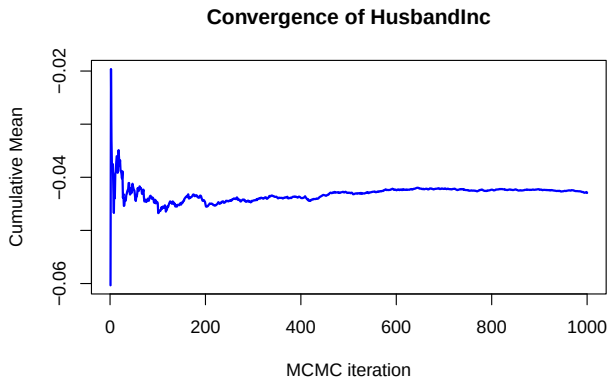

Traceplot of parameter Constant
Value
MCMC iteration

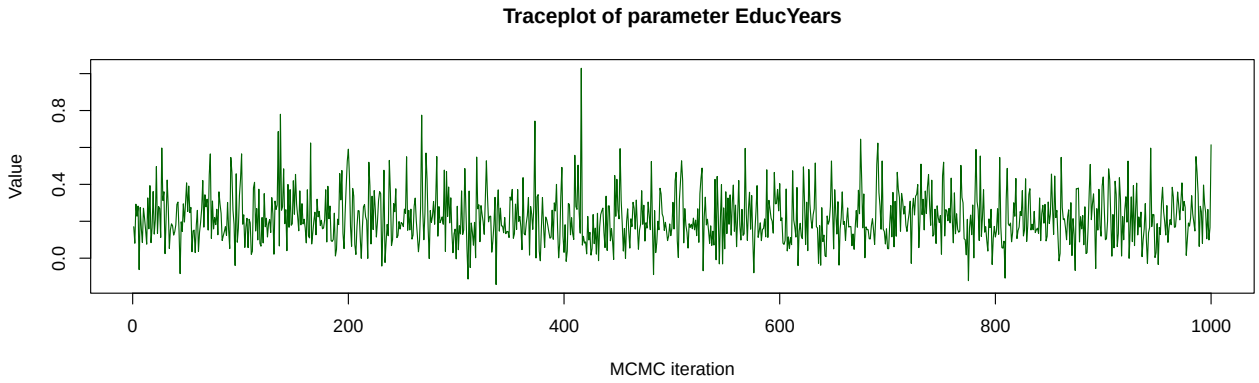

Histogram
Frequency
Constant

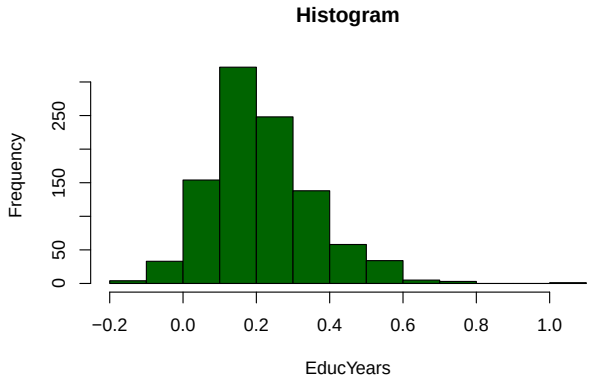

Convergence of Constant
Cumulative Mean
MCMC iteration

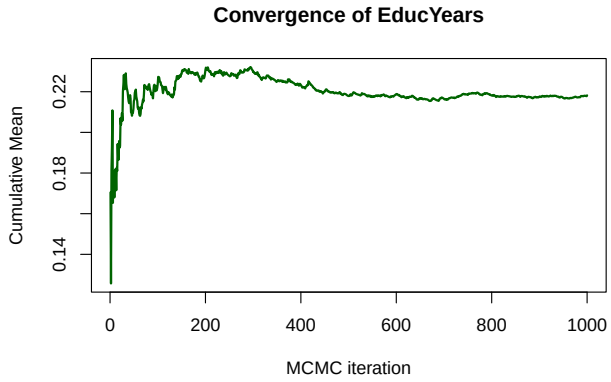

Traceplot of parameter HusbandInc
Value
MCMC iteration

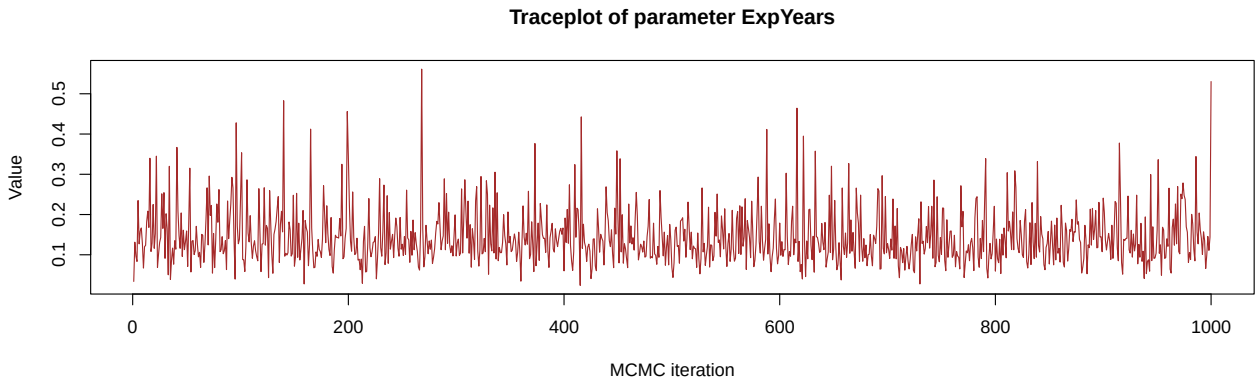

Histogram
Frequency
HusbandInc

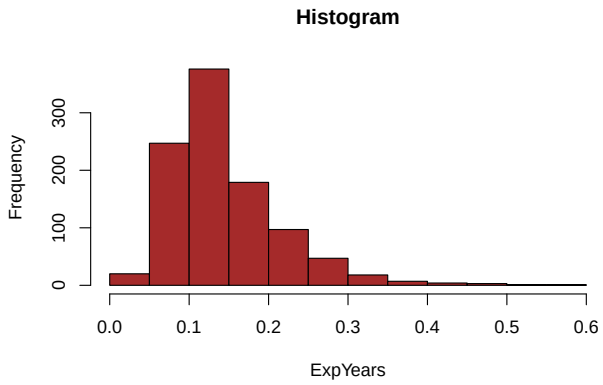

Convergence of HusbandInc
Cumulative Mean
MCMC iteration

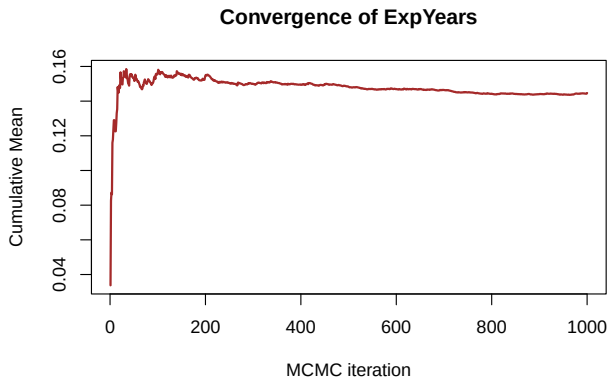

Traceplot of parameter EducYears
Value
0.0
0.4
0.8
0
200
400
600
800
1000
MCMC iteration

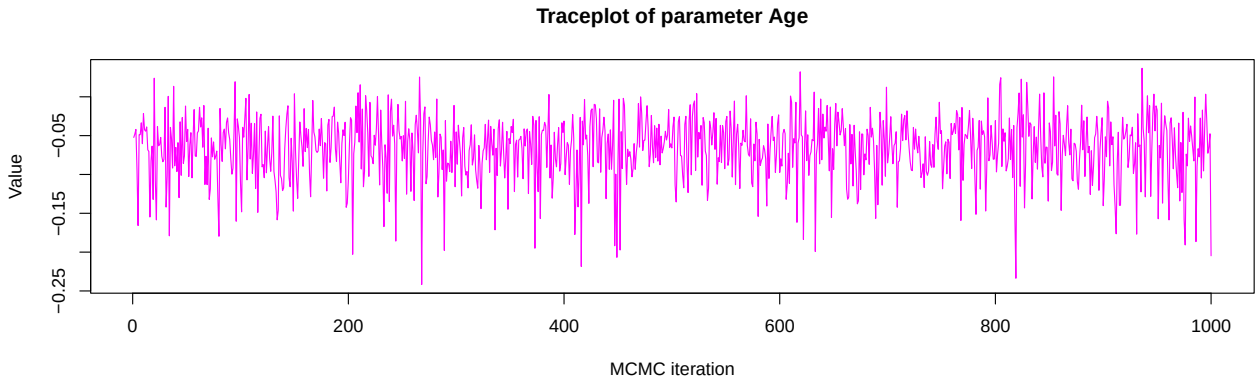

Histogram
Frequency
0
50
150
250
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
EducYears

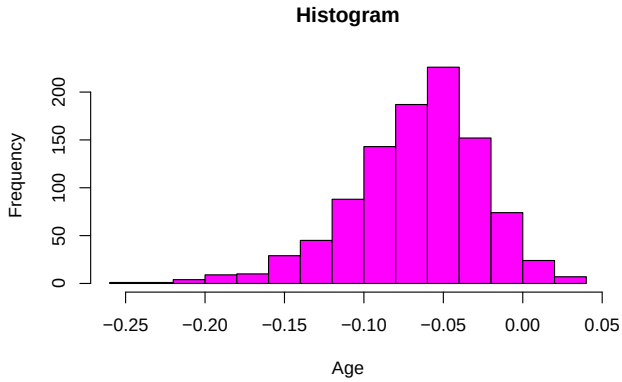

Convergence of EducYears
Cumulative Mean
0.14
0.18
0.22
0
200
400
600
800
1000
MCMC iteration

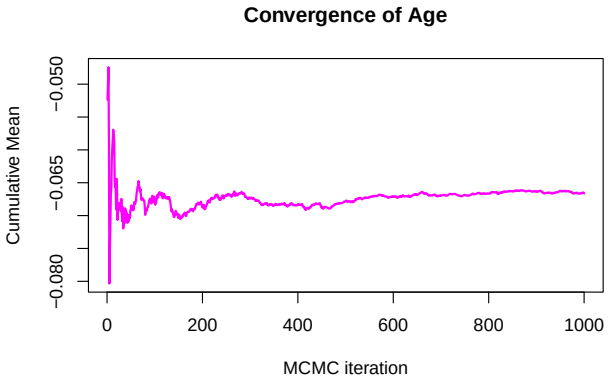

Traceplot of parameter ExpYears
Value
0.1
0.2
0.3
0.4
0.5
0
200
400
600
800
1000
MCMC iteration

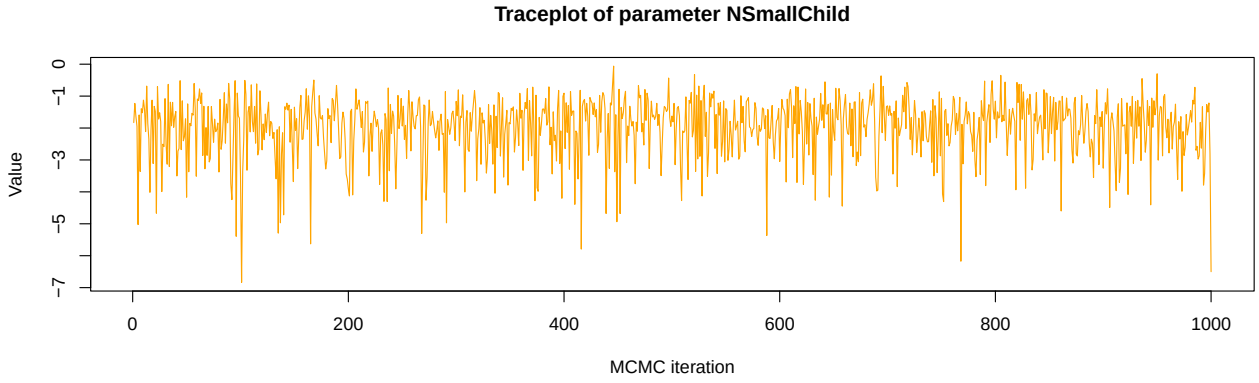

Histogram
Frequency
0
100
200
300
0.0
0.1
0.2
0.3
0.4
0.5
0.6
ExpYears

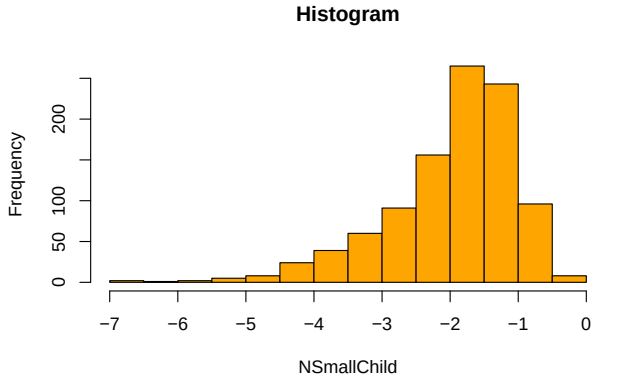

Convergence of ExpYears
Cumulative Mean
0.04
0.08
0.12
0.16
0
200
400
600
800
1000
MCMC iteration

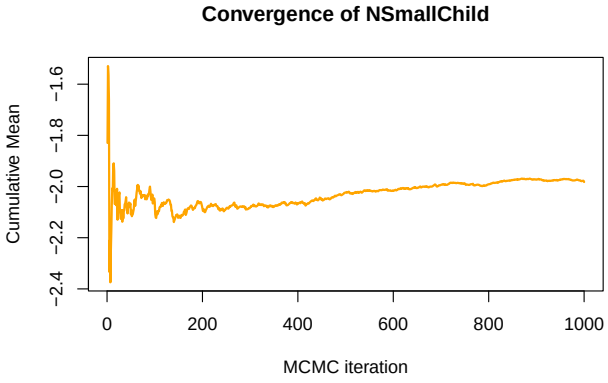

Traceplot of parameter Age
Value
−0.25
−0.15
−0.05
0
200
400
600
800
1000
MCMC iteration

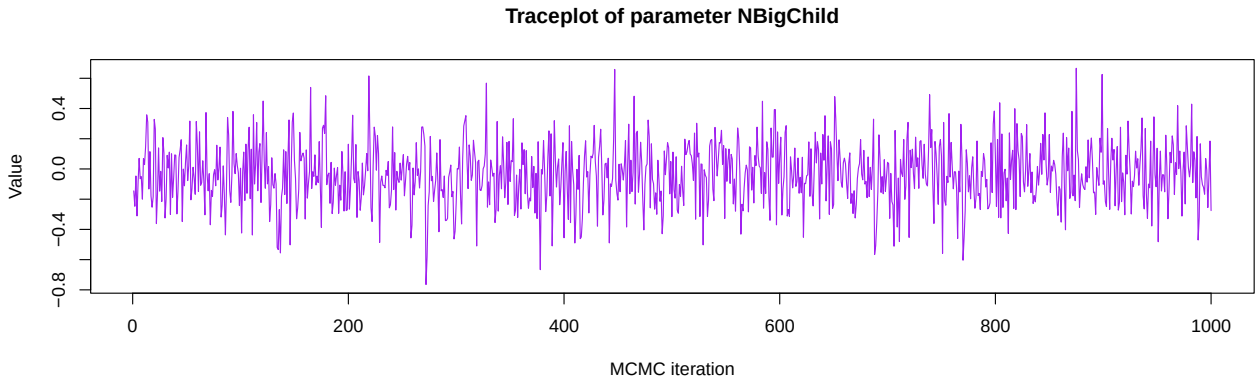

Histogram
Frequency
0
50
100
150
200
−0.25
−0.20
−0.15
−0.10
−0.05
0.00
0.05
Age

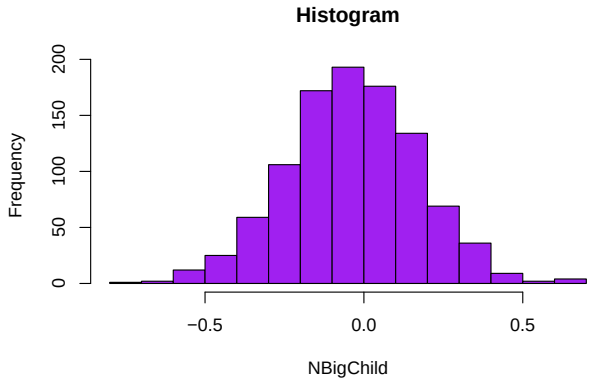

Convergence of Age
Cumulative Mean
−0.080
−0.065
−0.050
0
200
400
600
800
1000
MCMC iteration

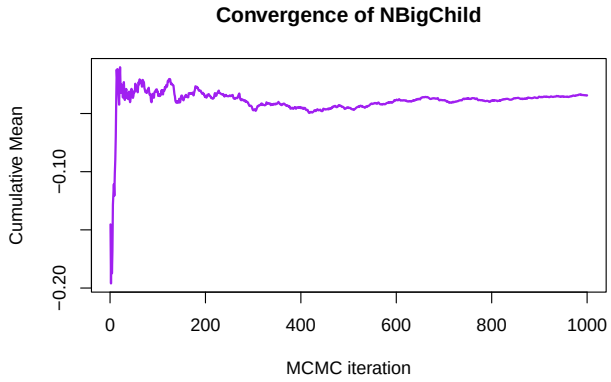

Traceplot of parameter NSmallChild
Value
−7
−5
−3
−1
0
200
400
600
800
1000
MCMC iteration


Histogram
Frequency
0
50
100
200
−7
−6
−5
−4
−3
−2
−1
0
NSmallChild


Convergence of NSmallChild
Cumulative Mean
−2.4
−2.2
−2.0
−1.8
−1.6
0
200
400
600
800
1000
MCMC iteration







The coefficient from the gibbs sampling and from the glmModel can be seen below. It can be observed that they are close to each other.

```
glmModel <- glm(Work ~ 0 + ., data = df1, family = binomial)
coefficients_df <- as.data.frame(glmModel$coefficients)
colnames(coefficients_df) <- c('glmModel')
coefficients_df$gibbs <- rep(NA, 7)

for (i in 1:nPar){
  cusumData =  cumsum(beta_gibbs[,i])/seq(1, nDraws)
  coefficients_df[i,'gibbs'] <- cusumData[nDraws]
}

print(coefficients_df)
```

```
##                glmModel       gibbs
## Constant     0.02262929  0.36857932
## HusbandInc  -0.03796308 -0.04294880
## EducYears    0.18447411  0.21819964
## ExpYears     0.12131763  0.14467091
## Age         -0.04858167 -0.06662734
## NSmallChild -1.56485140 -1.98235367
## NBigChild   -0.02526059 -0.03447693
```

```
IF_Gibbs <- rep(NA, nPar)

for (i in 1:nPar) {
  a_Gibbs <- acf(beta_gibbs[,i], lag.max = 5, plot = FALSE)
  IF_Gibbs[i] <- 1 + 2 * sum(a_Gibbs$acf[-1])
}
```

```
cat("Inefficiency Factors:\n", IF_Gibbs)
```

```
## Inefficiency Factors:
##  0.7654542 0.8671865 1.051185 0.943413 0.9147253 0.917874 1.000201
```

The inefficiency factors are calculate same way with the code in lectures but lag.max is set to 5 otherwise inefficiency factors are less than 1, it means number of effective size is less than sample size. It can be observed that inefficiency factors are so close to 1, it means generated samples are almost independent.

Part b)

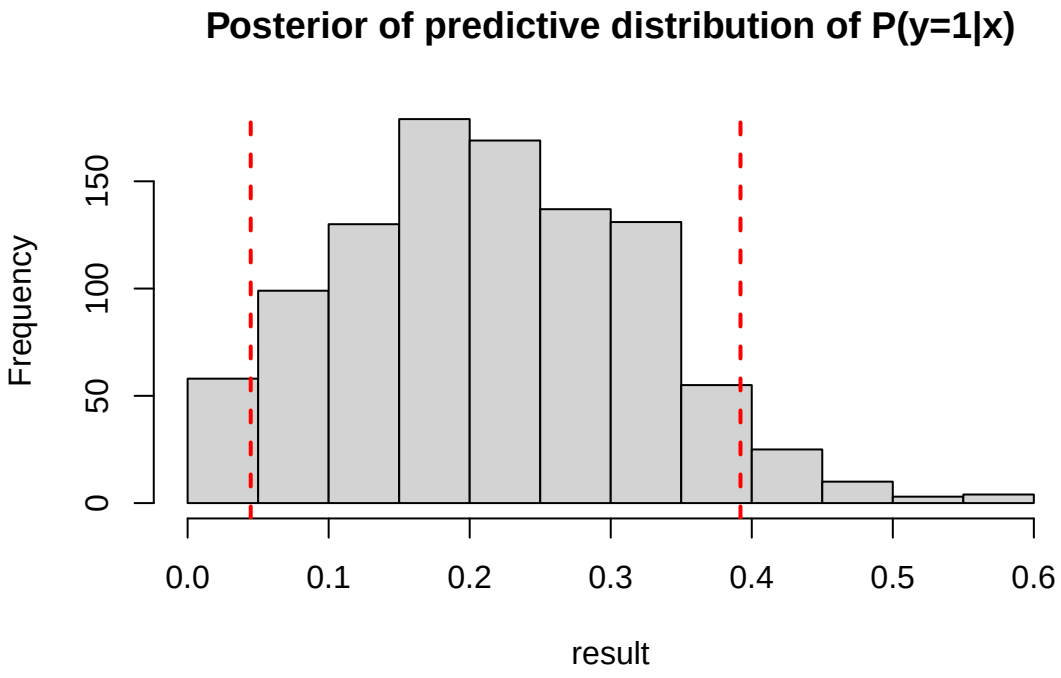
Question: Use the posterior draws from a) to compute a 90% equal tail credible interval for Pr(y = 1|x), where the values of x corresponds to a 38-year-old woman, with one child (3 years old), 12 years of education, 7 years of experience, and a husband with an income of 22. A 90% equal tail credible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

Answer:

```
input <- c(1, 22, 12, 7, 38, 1, 0)
X_input <- as.matrix(input, ncol = 1)

sigmoid_fnc <- function(linPred){
  return(exp(linPred)/(1+exp(linPred)))
}

linPred <- beta_gibbs %*% X_input
result <- apply(linPred, 1, sigmoid_fnc)
credible_interval <- quantile(result, c(0.05, 0.95))
cat("90% credible interval: (", credible_interval, ")")
```

```
## 90% credible interval: ( 0.04475653 0.3920081 )
```

```
hist(result, main = "Posterior of predictive distribution of P(y=1|x)")
abline(v = credible_interval, col = "red", lwd = 2, lty = 2)
```



Question 2: Metropolis Random Walk for Poisson regression

Consider the following Poisson regression model

$$y_i|\beta \sim \text{Poisson}[\exp(x_i^T\beta)],\ i = 1..n$$

where y_i is the count for the i^{th} observation in the sample and xi is the p-dimensional vector with covariate observations for the i^{th} observation. Use the data set eBayNumberOfBidderData_2024.dat. This dataset contains observations from 800 eBay auctions of coins. The response variable is nBids and records the number of bids in each auction. The remaining variables are features/covariates (x)

- **Const** (for the intercept)
- **PowerSeller** (equal to 1 if the seller is selling large volumes on eBay)
- **VerifyID** (equal to 1 if the seller is a verified seller by eBay)
- **Sealed** (equal to 1 if the coin was sold in an unopened envelope)
- **MinBlem** (equal to 1 if the coin has a minor defect)
- **MajBlem** (equal to 1 if the coin has a major defect)
- **LargNeg** (equal to 1 if the seller received a lot of negative feedback from customers)
- **LogBook** (logarithm of the book value of the auctioned coin according to expert sellers. Standardized)
- **MinBidShare** (ratio of the minimum selling price (starting price) to the book value. Standardized).

```
df2 <- read.table("eBayNumberOfBidderData_2024.dat", header = TRUE)
```

Part a)

Question: Obtain the maximum likelihood estimator of β in the Poisson regression model for the eBay data [Hint: glm.R, don't forget that glm() adds its own intercept so don't input the covariate Const]. Which covariates are significant?

Answer:

```
# maximum likelihood estimator of beta
glmModel <- glm(nBids ~ 0 + ., data = df2, family = "poisson")
summary(glmModel)
```

```
##
## Call:
## glm(formula = nBids ~ 0 + ., family = "poisson", data = df2)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## Const        1.07981    0.03393  31.828  < 2e-16 ***
## PowerSeller -0.03566    0.04167  -0.856 0.392109
## VerifyID    -0.45564    0.12748  -3.574 0.000351 ***
## Sealed       0.45515    0.06226   7.311 2.65e-13 ***
## Minblem     -0.06837    0.07198  -0.950 0.342228
## MajBlem     -0.22554    0.09525  -2.368 0.017894 *
## LargNeg      0.05382    0.06406   0.840 0.400787
## LogBook     -0.08499    0.03234  -2.628 0.008599 **
## MinBidShare -1.82490    0.07843 -23.269  < 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: 4833.6  on 800  degrees of freedom
## Residual deviance:  691.8  on 791  degrees of freedom
## AIC: 2879.1
##
## Number of Fisher Scoring iterations: 5
```

VerifyID, Sealed, MinBidShare and LogBook covarites are significant since their p-value in the model is less than 0.05.

```
cat("Model coefficients:\n", glmModel$coefficients)
```

```
## Model coefficients:
##  1.079805 -0.03566493 -0.4556376 0.455152 -0.06836819 -0.2255414 0.05382386 -0.08498844 -1.824901
```

Part b)

Question: Let's do a Bayesian analysis of the Poisson regression. Let the prior be $\beta \sim N[0, 100 \cdot (X^T X)^{-1}]$, where X is the n × p covariate matrix. This is a commonly used prior, which is called Zellner's g-prior. Assume first that the posterior density is approximately multivariate normal:

$$\beta|y \sim N(\tilde{\beta}, J_y^{-1}(\tilde{\beta}))$$

where $\tilde{\beta}$ is the posterior mode and $J_y(\tilde{\beta})$ is the negative Hessian at the posterior mode. $\tilde{\beta}$ and $J_y(\tilde{\beta})$ can be obtained by numerical optimization (optim.R) exactly like you already did for the logistic regression in Lab 2 (but with the log posterior function replaced by the corresponding one for the Poisson model, which you have to code up.).

Answer:

```
y <- as.matrix(df2$nBids) # y is (n x 1) matrix
X <- as.matrix(df2[2:ncol(df2)]) # X is (n x p) matrix
nObs <- nrow(df2)
nPar <- ncol(df2) - 1 # subtract y
```

$$\text{Posterior} = \text{Prior} \cdot \text{Likelihood}$$

$$\log(\text{Posterior}) = \log(\text{Prior}) + \log(\text{Likelihood})$$

$$f(y_i) = \frac{\lambda_i^{y_i} \cdot e^{-\lambda_i}}{y_i!} \text{ where } \lambda_i = \exp(x_i^T \beta)$$

$$\text{Likelihood} = \prod_{i=1}^{n} f(y_i) = \log(\prod_{i=1}^{n} \frac{\lambda_i^{y_i} \cdot e^{-\lambda_i}}{y_i!}) = \sum_{i=1}^{n} \log(\frac{\lambda_i^{y_i} \cdot e^{-\lambda_i}}{y_i!}) \text{ , where } \lambda_i = \exp(x_i^T \beta)$$

```
# prior hyperparameters
mu_prior <- rep(0, nPar)
Sigma_prior <- 100 * solve((t(X) %*% X))

# Now we will use optim. Inputs are;
# 1) log p(theta|y) function
# 2) initial values for thetas

# input 1)
logPostFunc <- function(betas, y, X, mu, Sigma){
```

```
  lambda <- exp(X %*% betas) # (n x p)(p x 1) matrix multiplication
  logLik <- sum( log( ((lambda^y) * exp(-lambda) )/ factorial(y) ) )
  logPrior <- dmvnorm(betas, mu, Sigma, log=TRUE) # densities are given as log, calculates density
  return(logLik + logPrior)
}

# input 2)
initVal <- matrix(0, nPar, 1) # (p x 1) matrix

# Now optimize
OptimRes <- optim(initVal, logPostFunc, gr=NULL, y, X, mu_prior, Sigma_prior, method=c("BFGS"),
                  control=list(fnscale=-1), hessian=TRUE)
```

```
Xnames <- colnames(df2[2:ncol(df2)])
posterior_mode <- t(OptimRes$par)
colnames(posterior_mode) <- Xnames
print('The posterior mode is:')
```

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

```
print(posterior_mode)
```

```
##         Const PowerSeller   VerifyID    Sealed     Minblem    MajBlem
## [1,] 1.077217 -0.03567963 -0.4535318 0.4548486 -0.06863401 -0.2258391
##         LargNeg     LogBook MinBidShare
## [1,] 0.05387677 -0.08454639   -1.822757
```

```
J_y_inv <- solve(-OptimRes$hessian)
approxPostStd <- sqrt(diag(J_y_inv))
names(approxPostStd) <- Xnames
print('The approximate posterior standard deviation is:')
```

```
## [1] "The approximate posterior standard deviation is:"
```

```
print(approxPostStd)
```

```
##       Const PowerSeller    VerifyID      Sealed     Minblem     MajBlem
##  0.03389556  0.04167562  0.12715595  0.06227165  0.07198300  0.09527403
##     LargNeg     LogBook MinBidShare
##  0.06408047  0.03233568  0.07826924
```

```
model_coef <- as.data.frame(glmModel$coefficients)
colnames(model_coef) <- 'glmModel'
model_coef$PosteriorMode <- t(posterior_mode)
print(model_coef)
```

```
##                glmModel PosteriorMode
## Const        1.07980512    1.07721720
## PowerSeller -0.03566493   -0.03567963
## VerifyID    -0.45563760   -0.45353183
## Sealed       0.45515199    0.45484863
## Minblem     -0.06836819   -0.06863401
## MajBlem     -0.22554138   -0.22583912
## LargNeg      0.05382386    0.05387677
## LogBook     -0.08498844   -0.08454639
## MinBidShare -1.82490142   -1.82275698
```

Note that posterior mode values are so close to the model coefficient found using glm in part (a).

Part c)

Question: Let's simulate from the actual posterior of β 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. In order to show that it is a general function for any model, we denote the vector of model parameters by θ. Let the proposal density be the multivariate normal density mentioned in Lecture 8 (random walk Metropolis):

$$\theta_p|\theta^{(i-1)} \sim N(\theta^{(i-1)}, c \cdot \Sigma)$$

where $\Sigma = J_y^{-1}(\tilde{\beta})$ was obtained in b). The value c is a tuning parameter and should be an input to your Metropolis function. The user of your Metropolis function should be able to supply her own posterior density function, not necessarily for the Poisson regression, and still be able to use your Metropolis function. This is not so straightforward, unless you have come across function objects in R. The note HowToCodeRWM.pdf in Lisam describes how you can do this in R.

Now, use your new Metropolis function to sample from the posterior of β in the Poisson regression for the eBay dataset. Assess MCMC convergence by graphical methods.

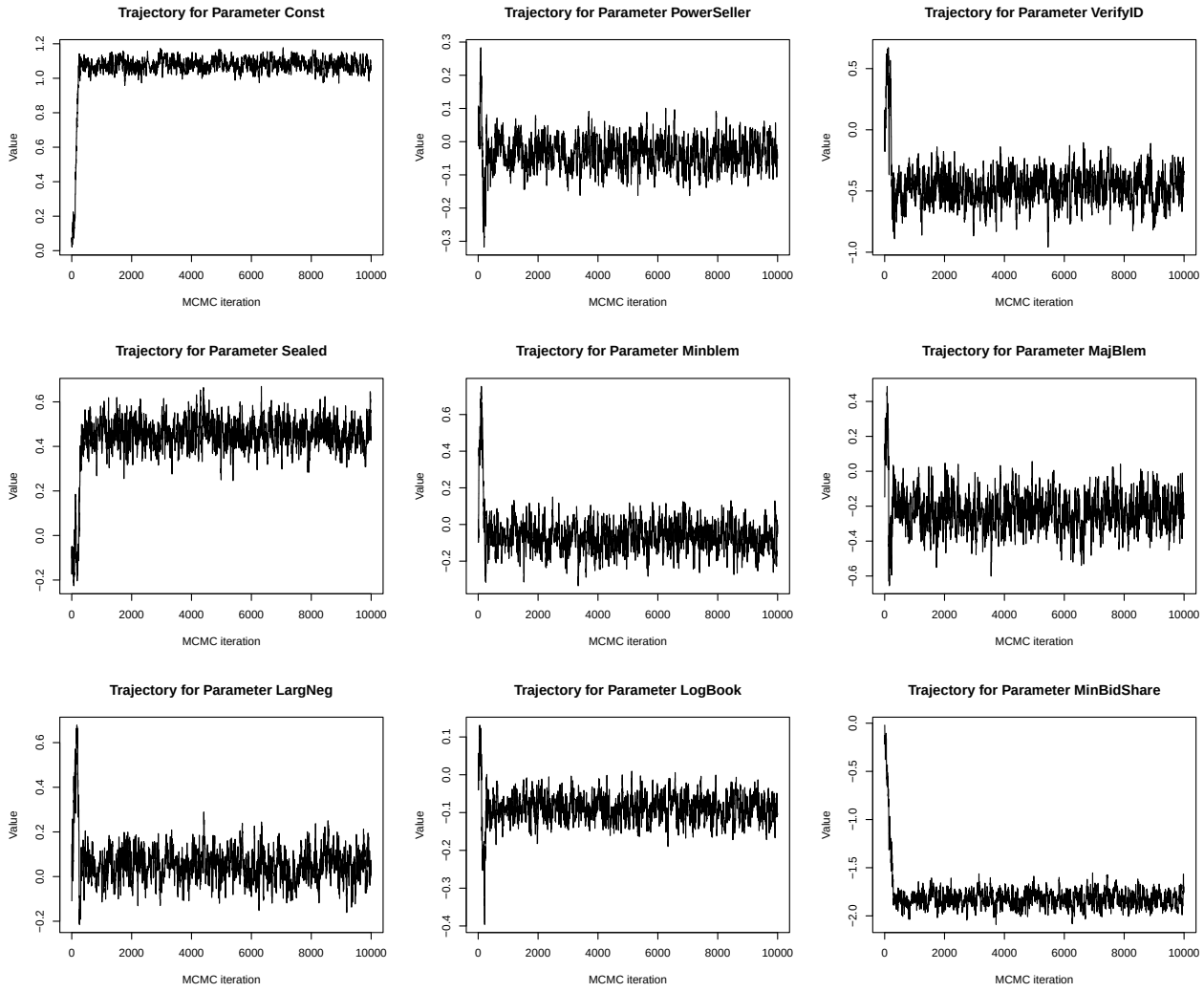
Answer:

```
set.seed(12345)
MetropolisHasting <- function(y, X, mu_prior, Sigma_prior, logPostFunc, c, nDraws = 10000){

  sum_acc_prob <- 0
  theta_sample <- matrix(0, nrow = nDraws, ncol = nPar) # initialize draws
  theta_t <- rep(0, nPar) # initialize theta

  for (s in 1:nDraws){
    # step 1: sample from proposal distribution
    theta_t1 <- as.vector(rmvnorm(1, mean = theta_t, sigma = c * J_y_inv))

     # step 2: compute acceptance probability,
    log_density_t <- logPostFunc(theta_t, y, X, mu_prior, Sigma_prior)
    log_density_t1 <- logPostFunc(theta_t1, y, X, mu_prior, Sigma_prior)

    alpha <- exp(log_density_t1 - log_density_t)
    acc_prob <- min(1, alpha)
    sum_acc_prob <- sum_acc_prob + acc_prob
    # step 3:
    if (runif(1) < acc_prob){
      # accept
      theta_sample[s,] <- theta_t1
      theta_t <- theta_t1
    } else{
      # reject
      theta_sample[s,] <- theta_t
    }
  }

  cat("Average acceptance probability:", sum_acc_prob / nDraws, "\n")
  return(theta_sample[-1,])
}
```

```
nDraws <- 10000
c <- 0.6 # set c that average acceptance probability should is 25-30%
beta_samples <- MetropolisHasting(y, X, mu_prior, Sigma_prior, logPostFunc, c, nDraws)
```

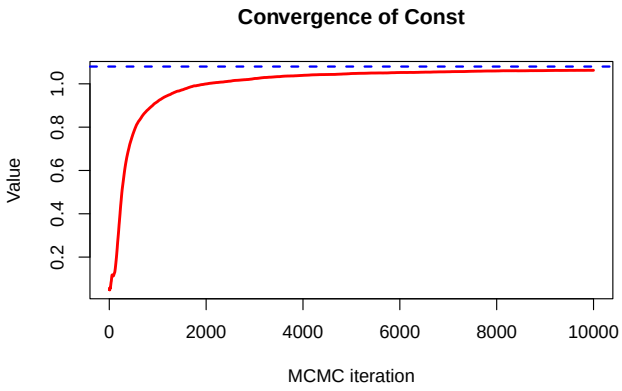
```
## Average acceptance probability: 0.2801641
```

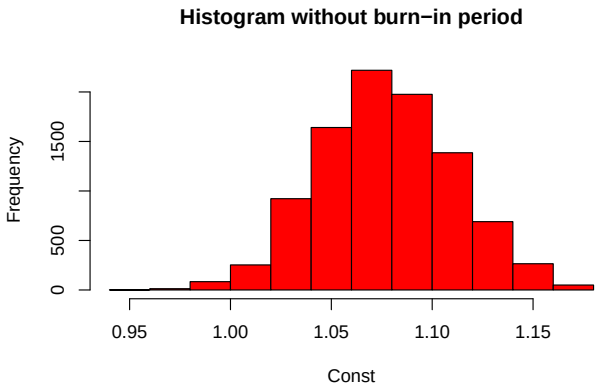
```
par(mfrow = c(3, 3))
for (p in 1:nPar){
  plot(beta_samples[, p],
       type = "l",
       main = paste("Trajectory for Parameter", colnames(df2)[p+1]),
       xlab = "MCMC iteration",
       ylab = "Value")
}
```

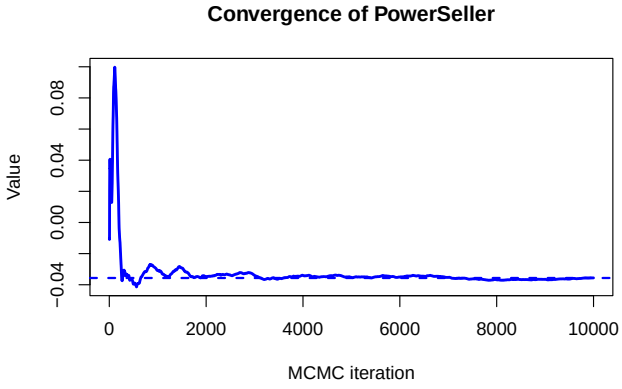


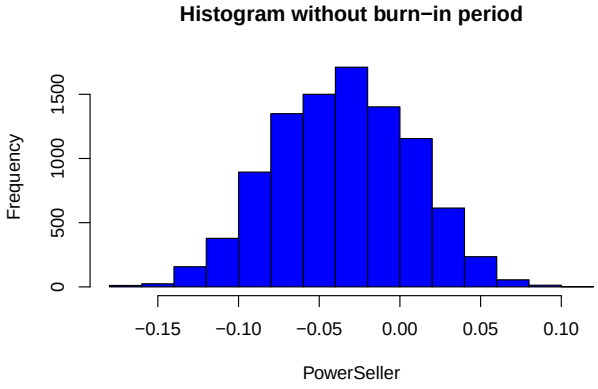

```
burn_in <- 500
beta_avg_values <- rep(0, nPar)

colors <- c("red", "blue", "darkgreen", "brown", "magenta", "orange", "purple", "yellow", "cyan")
for (i in 1:nPar){
```

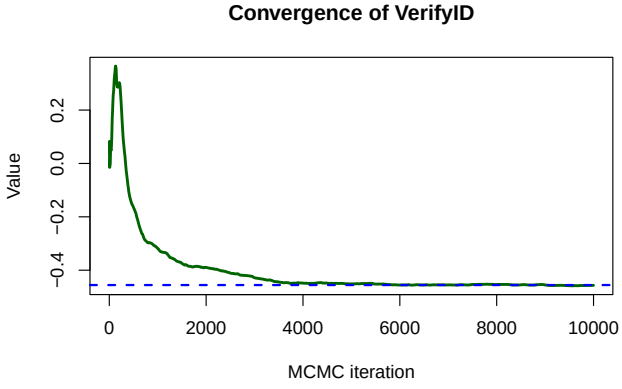
```r
  par(mfrow = c(1, 2))
  beta_avg <- cumsum(beta_samples[, i]) / (1:nrow(beta_samples))
  beta_avg_values[i] <- beta_avg[length(beta_avg)]

  plot(1:nrow(beta_samples), beta_avg,
       type = "l", col=colors[i],
       main = paste("Convergence of", colnames(df2)[i+1]),
       lwd = 2.5,
       xlab = "MCMC iteration",
       ylab = "Value")
  abline(h = glmModel$coefficients[i], col = "blue", lwd = 2, lty = 2)

  hist(beta_samples[burn_in: nrow(beta_samples), i],
       main = "Histogram without burn-in period",
       xlab =  paste(colnames(df2)[i+1]), col = colors[i])
}
```

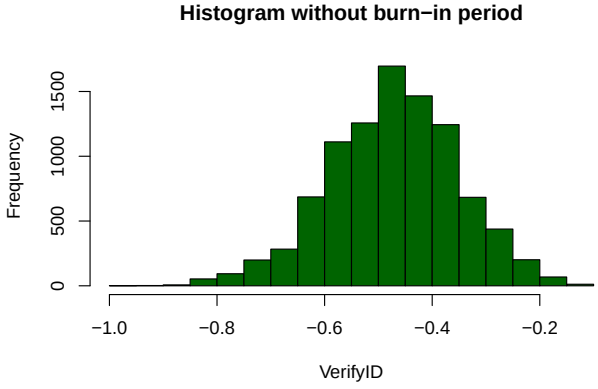


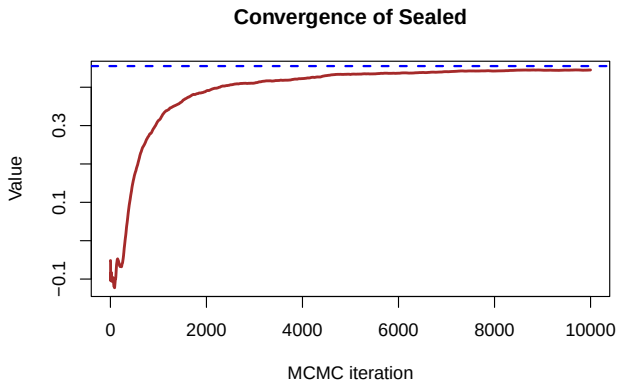


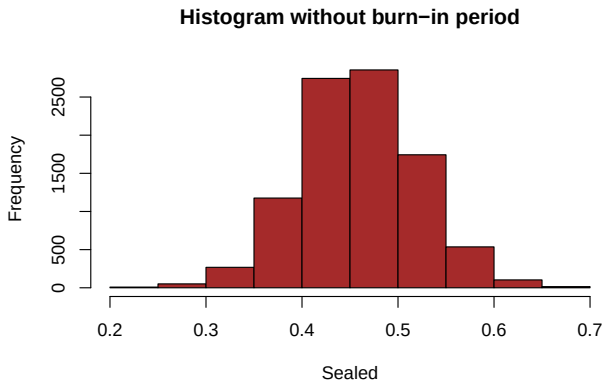


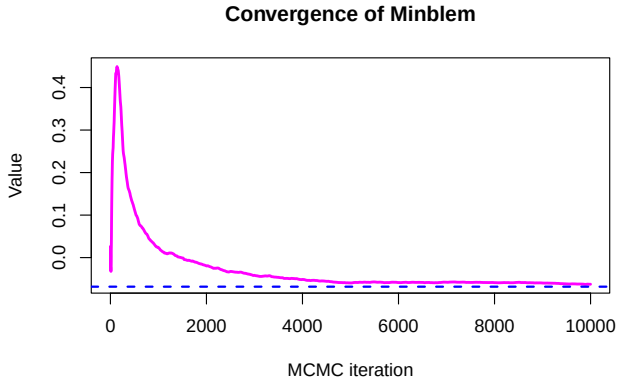


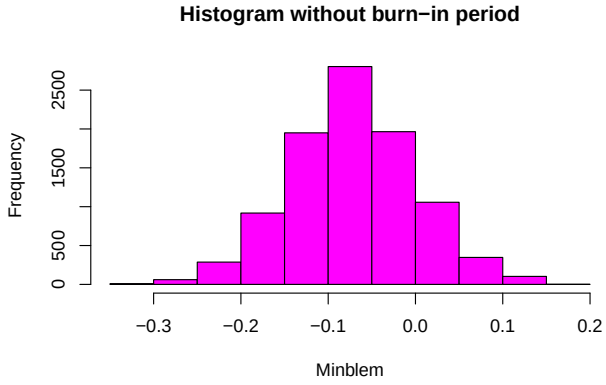


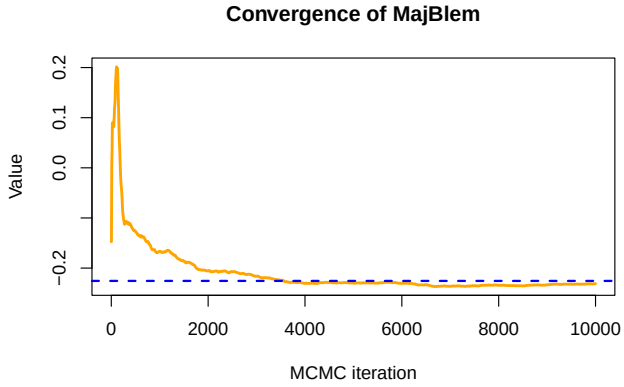


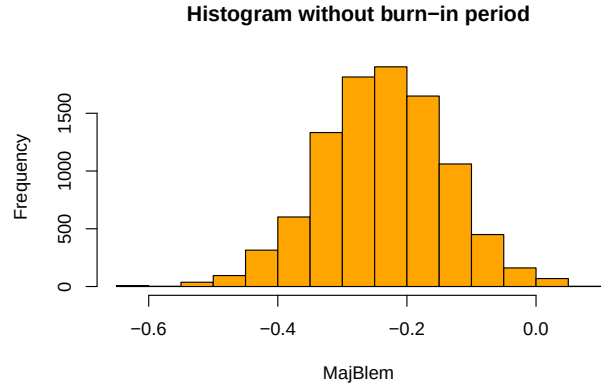

Convergence of Sealed
Value
MCMC iteration

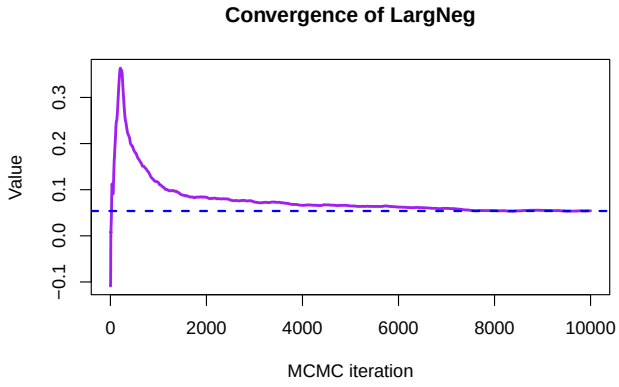

Histogram without burn−in period
Frequency
Sealed

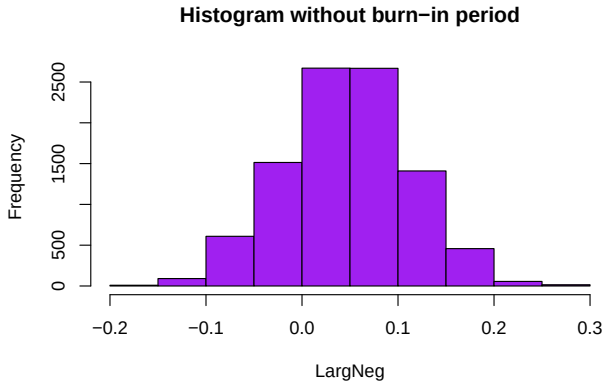

Convergence of Minblem
Value
MCMC iteration

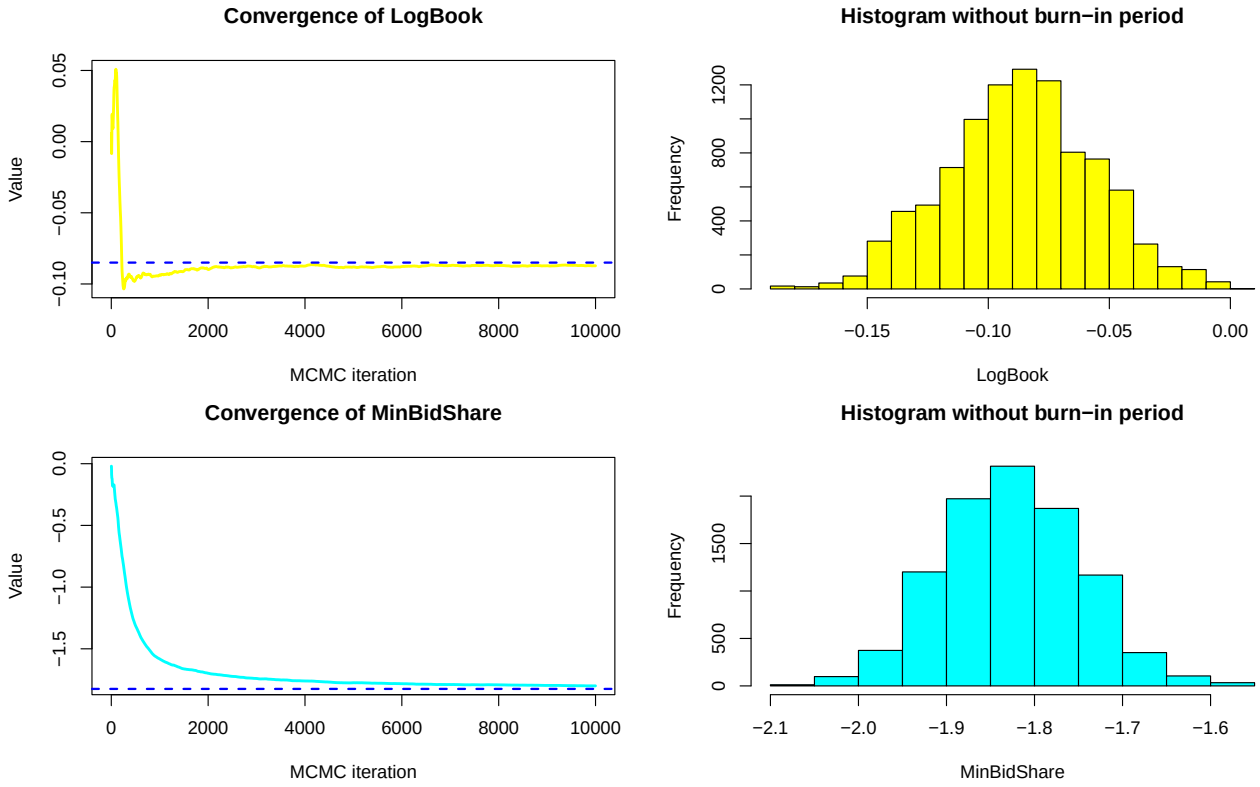

Histogram without burn−in period
Frequency
Minblem


Convergence of MajBlem
Value
MCMC iteration


Histogram without burn−in period
Frequency
MajBlem


Convergence of LargNeg
Value
MCMC iteration


Histogram without burn−in period
Frequency
LargNeg



The dashed line in the convergence graph represent the coefficients from glmModel. It can be observed that parameter converges.

Part d)

Question: 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?

- **PowerSeller** = 1
- **VerifyID** = 0
- **Sealed** = 1
- **MinBlem** = 0
- **MajBlem** = 1
- **LargNeg** = 0
- **LogBook** = 1.2
- **MinBidShare** = 0.8

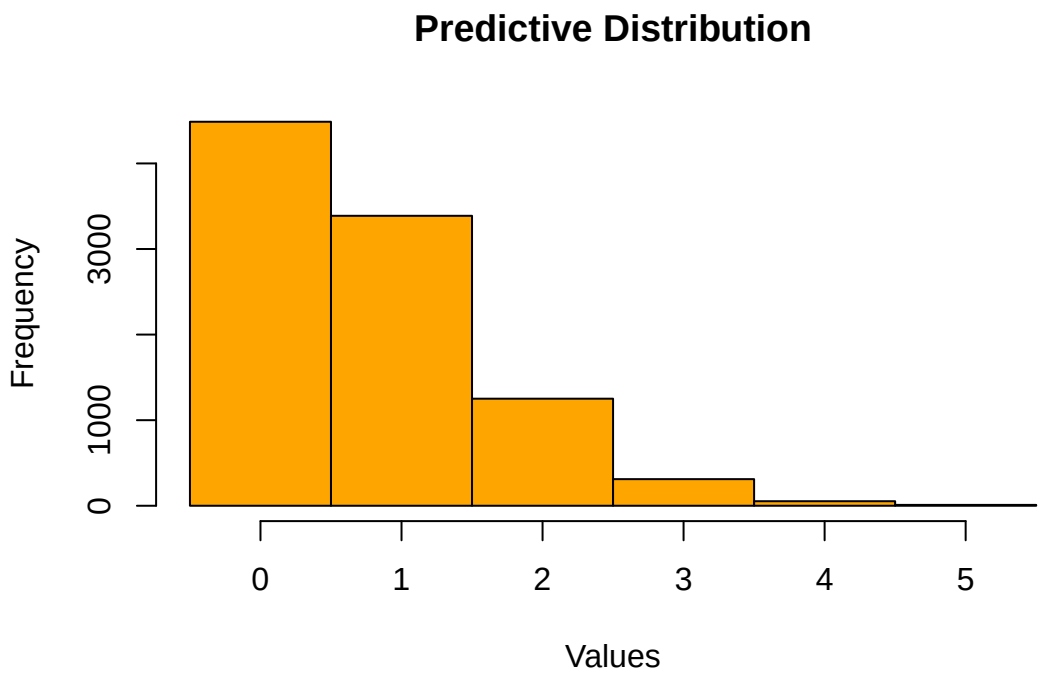
Answer:

```
X_input <- c(Const = 1,
             PowerSeller = 1,
             VerifyID = 0,
             Sealed = 1,
             Minblem = 0,
             MajBlem = 1,
             LargNeg = 0,
```

```
              LogBook = 1.2,
              MinBidShare = 0.8)

pred_beta <- beta_samples[500:nrow(beta_samples),] # burn-in period is removed

# column 1: lambda, column 2: number of bids
generated_pois <- matrix(NA, nrow = nrow(pred_beta), ncol = 2)
generated_pois[,1] <- exp(pred_beta %*% as.matrix(X_input)) # calculating lambda values

for (i in 1:nrow(pred_beta)){
  lambda <- generated_pois[i,1]
  generated_pois[i,2] <- rpois(1, lambda)
}

cat("Probability of no bidders:", sum(generated_pois[,2] == 0) / nrow(generated_pois))
```

```
## Probability of no bidders: 0.4723158
```

```
hist(generated_pois[,2],
     breaks = seq(min(generated_pois[,2])-0.5, max(generated_pois[,2])+0.5, by = 1),
     col = "orange",
     main = "Predictive Distribution",
     xlab = "Values",
     ylab = "Frequency"
)
```




```
cat("Number of expected bids from the glmModel:",
    predict(glmModel, newdata = as.data.frame(t(X_input)), type = "response"))
```

```
## Number of expected bids from the glmModel: 0.7496443
```

Question 3: Time series models in Stan

Part a)

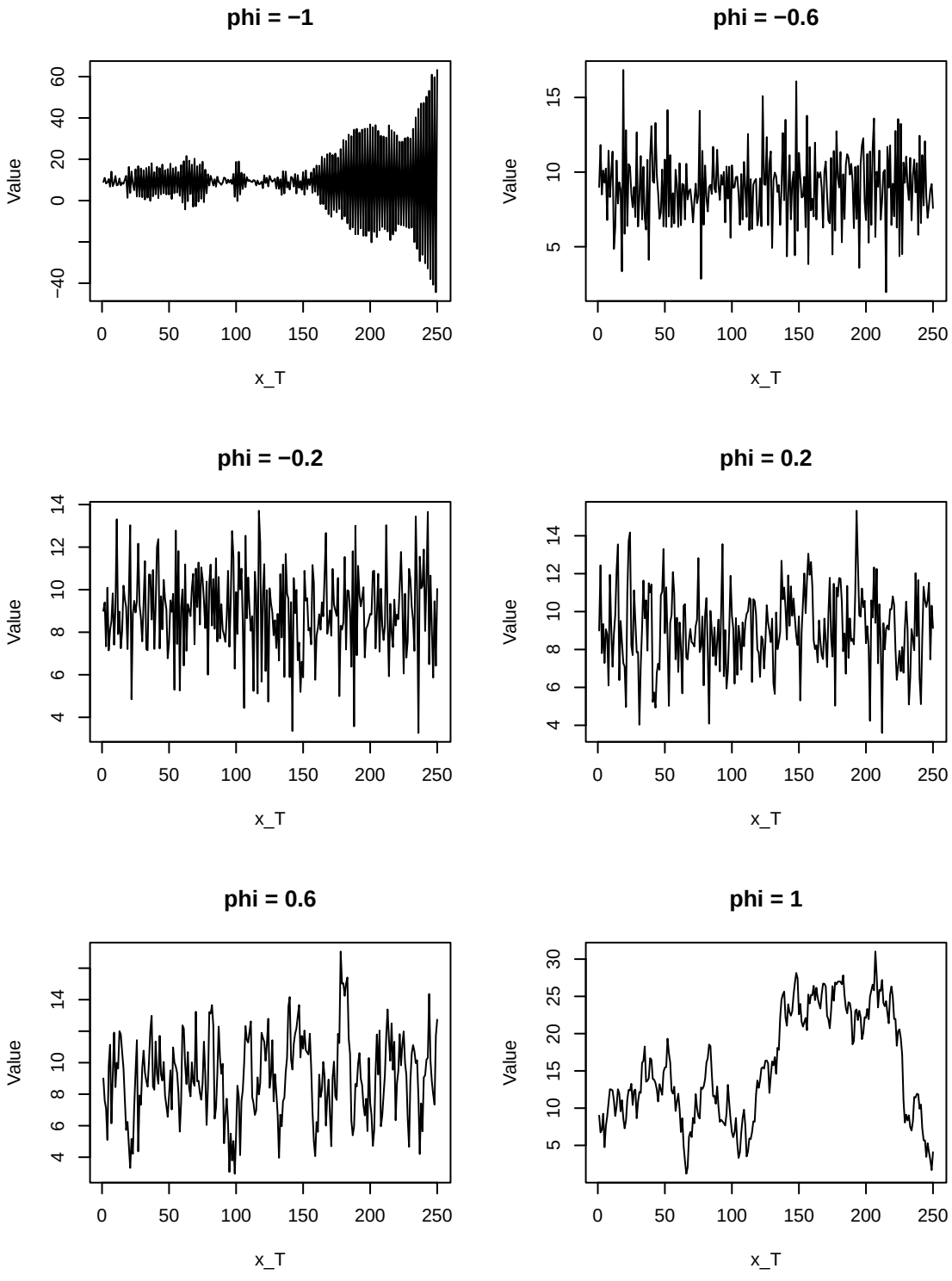
Question: Write a function in R that simulates data from the AR(1) process

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

for given values of μ, ϕ and σ^2. Start the process at $x_1 = \mu$ and then simulate values for x_t for t = 2,3,... T and return the vector $x_{1:T}$ containing all time points. Use $\mu 9$, $\sigma^2 = 4$ and $T = 250$ and look at some different realizations (simulations) of $x_{1:T}$ 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 $x_{1:T}$?

Answer: Graphs for different ϕ values can be seen below. The absolute value of ϕ parameter effects how much x_t depends on x_{t-1}. The sign of ϕ determines the direction of the relation with the previous value. When $\phi = 0$, the process is $x_t = \mu + \epsilon_t$. When $\phi = 1$, $x_t = x_{t-1} + \epsilon_t$. When $\phi = -1$, the process is $x_t = 2\mu - x_{t-1} + \epsilon_t$

```
AR1_simulate <- function(phi, mu, sigma2, T){
  x_vector <- rep(NA, T)
  x_t <- mu
  x_vector[1] <- x_t
  for (i in 2:T){
    eps <- rnorm(1, mean = 0, sd = sqrt(sigma2))
    x_t1 <- mu + phi*(x_t - mu) + eps
    x_vector[i] <- x_t1
    x_t <- x_t1
  }
  return(x_vector)
}

par(mfrow = c(3, 2))
phi_val <- seq(-1, 1, length.out = 6)
for (i in 1:length(phi_val)){
  x_simulation <- AR1_simulate(phi = phi_val[i], mu = 9, sigma2 = 4, T = 250)
  plot(x_simulation,
       type="l",
       xlab = "x_T", ylab = "Value",
       main = paste("phi =", phi_val[i]))
}
```



Part b)

Question: Use your function from a) to simulate two AR(1)-processes, $x_{1:T}$ with $\phi = 0.3$ and $y_{1:T}$ 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 Stan code that samples from the posterior of the three parameters, using suitable non-informative priors of your choice. [Hint: Look at the time-series models examples in the Stan user's guide/reference manual, and note the different parameterization used here.] 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. Are you able to estimate the true values? ii) For each of the two data sets, evaluate the convergence of the samplers and plot the joint posterior of μ and ϕ. Comments?

Answer: The selected non-informative priors are as follows:
$\mu \sim N(9, 100^2)$, high variance almost gives a flat distribution
$\phi \sim U(-1, 1)$, it is the range of ϕ and all values have equal probabilites

$\sigma^2 \sim Scale - inv - \chi^2(0.0001, 0.0001)$, density of scaled inverse chi-squared distribution is $f(\sigma^2|\nu, \tau^2) \propto \frac{exp(\frac{-\nu\tau^2}{2\sigma^2})}{(\sigma^2)^{1+\nu/2}}$. As ν and τ^2 approaches to 0, $f(\sigma^2) \propto (\frac{1}{\sigma^2})$

i) phi = 0.3 When $\phi = 0.3$ the AR(1) process becomes $x_t = 0.7 \cdot \mu + 0.3 \cdot x_{t-1} + \epsilon_t$

```
y = AR1_simulate(phi = 0.3, mu = 9, sigma2 = 4, T = 250)
N=length(y)

StanModel = '
data {
  int<lower=0> N;
  vector[N] y;
}
parameters {
  real mu;
  real<lower=-1, upper=1> phi;
  real<lower=0> sigma2;
}
model {
  mu ~ normal(9, 100); // non-informative prior
  phi ~ uniform(-1, 1);
  sigma2 ~ scaled_inv_chi_square(0.0001, 0.0001); // as nu, tau^2 -> 0, it becomes jeffreys prior

  for (n in 2:N) {
    y[n] ~ normal(mu + phi * (y[n-1] - mu), sqrt(sigma2));
  }
}'

data <- list(N=N, y=y)
warmup <- 1000
niter <- 2000
fit <- stan(model_code=StanModel, data=data, warmup=warmup, iter=niter, chains=4)
```

```
## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## using C compiler: 'Apple clang version 15.0.0 (clang-1500.3.9.4)'
## using SDK: 'MacOSX14.4.sdk'
## clang -arch x86_64 -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I"/Library/Frame
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/library/StanHeade
## In file included from /Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/library/RcppEigen
## In file included from /Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/library/RcppEigen
## /Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/library/RcppEigen/include/Eigen/src/Co
## #include <cmath>
##          ^~~~~~
## 1 error generated.
## make: *** [foo.o] Error 1
```

```
##
## 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: 0.146 seconds (Warm-up)
## Chain 1:                0.14 seconds (Sampling)
## Chain 1:                0.286 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 2.6e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.26 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: 0.142 seconds (Warm-up)
## Chain 2:                0.141 seconds (Sampling)
## Chain 2:                0.283 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 2.6e-05 seconds
```

```
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.26 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: 0.14 seconds (Warm-up)
## Chain 3:                0.151 seconds (Sampling)
## Chain 3:                0.291 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: 0.141 seconds (Warm-up)
## Chain 4:                0.141 seconds (Sampling)
## Chain 4:                0.282 seconds (Total)
## Chain 4:
```

```
# Print the fitted model
print(fit, digits_summary=3)
```

```
## Inference for Stan model: anon_model.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##            mean se_mean    sd     2.5%      25%      50%      75%    97.5%
```

```
## mu        9.005   0.003 0.202    8.602    8.873    9.005    9.140    9.401
## phi       0.338   0.001 0.060    0.219    0.297    0.338    0.377    0.458
## sigma2    4.258   0.006 0.386    3.566    3.982    4.239    4.505    5.056
## lp__   -305.534   0.029 1.256 -308.704 -306.119 -305.197 -304.627 -304.106
##        n_eff  Rhat
## mu      3843 1.001
## phi     3604 1.000
## sigma2  3817 1.000
## lp__    1886 1.003
##
## Samples were drawn using NUTS(diag_e) at Mon May 13 09:08:12 2024.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```
# Extract posterior samples
postDraws <- extract(fit)
```

The posterior mean, 95% credible interval and effective sample size can be seen below.

```
fit_summary <- summary(fit)
posterior_mean <- fit_summary$summary[, "mean"]
credible_intervals <- fit_summary$summary[, c("2.5%", "97.5%")]
effective_samples <- fit_summary$summary[, "n_eff"]

cat("posterior mean:\n")
```

```
## posterior mean:
```

```
print(posterior_mean[1:3])
```

```
##       mu      phi   sigma2
## 9.004768 0.337558 4.257748
```

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

```
## 95% credible interval:
```

```
print(credible_intervals[1:3, ])
```

```
##             2.5%    97.5%
## mu     8.6018572 9.401263
## phi    0.2191623 0.457762
## sigma2 3.5656380 5.055712
```

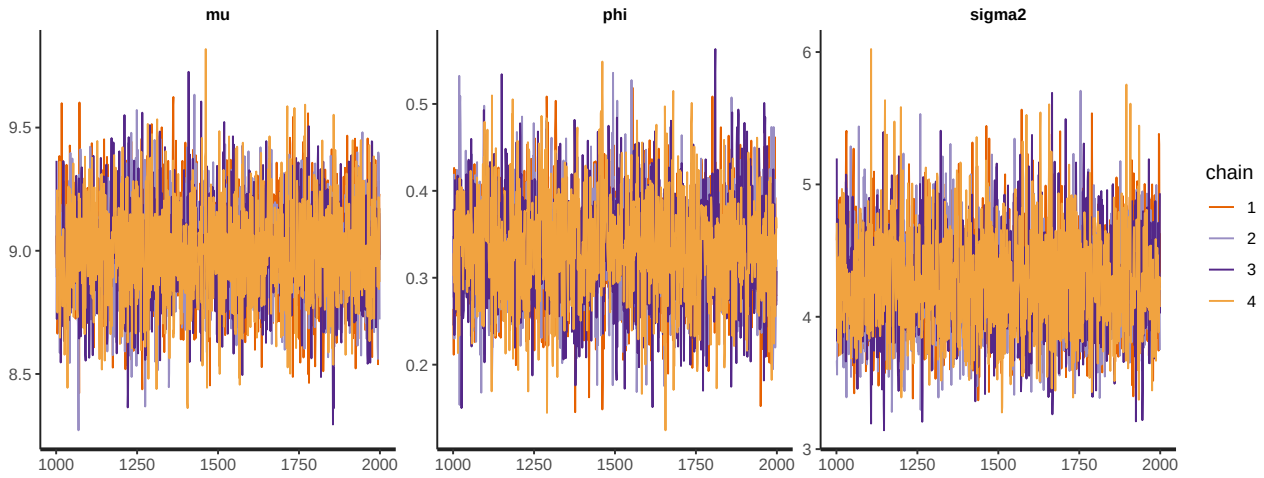
```
cat("Effective sample size:\n")
```

```
## Effective sample size:
```

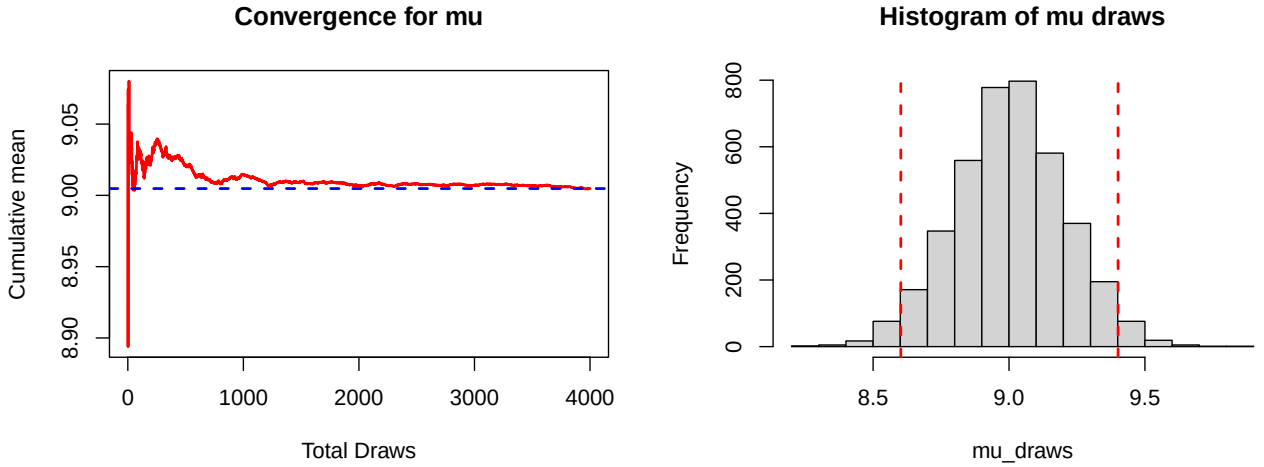
```
print(effective_samples[1:3])
```

```
##       mu      phi   sigma2
## 3843.299 3604.091 3816.714
```

```
traceplot(fit)
```

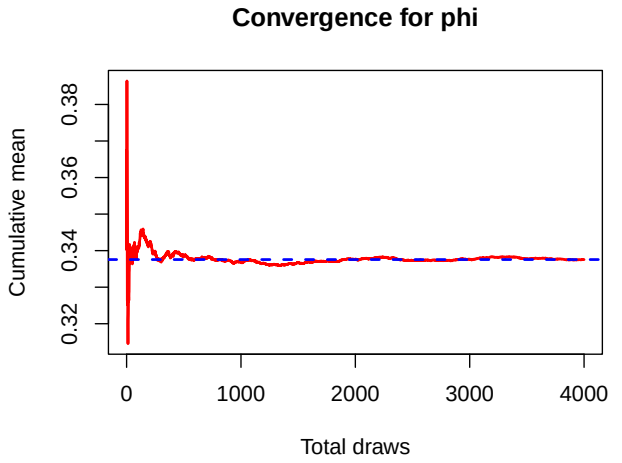


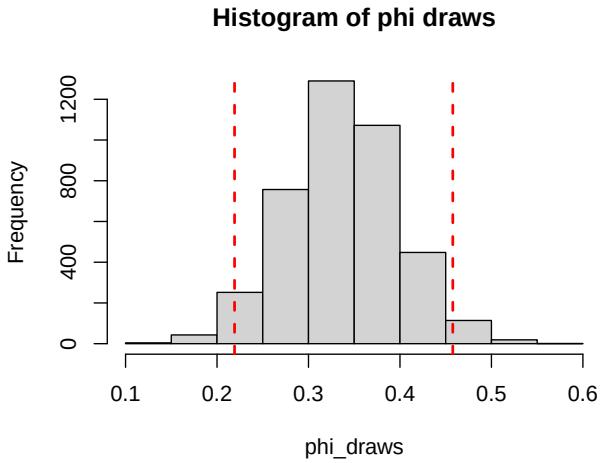

```
mu_draws <- postDraws$mu
cum_mean_mu <- cumsum(mu_draws) / (1:length(mu_draws))

par(mfrow = c(1, 2))
plot(1:length(mu_draws), cum_mean_mu, lwd = 2,
     type = "l", col = "red", main = "Convergence for mu",
     xlab = "Total Draws", ylab = "Cumulative mean")
abline(h = posterior_mean[1], lwd = 2, lty = 2, col = "blue")

hist(mu_draws, main = "Histogram of mu draws")
abline(v = credible_intervals[1,1], lty = 2, lwd = 2, col = "red")
abline(v = credible_intervals[1,2], lty = 2, lwd = 2, col = "red")
```



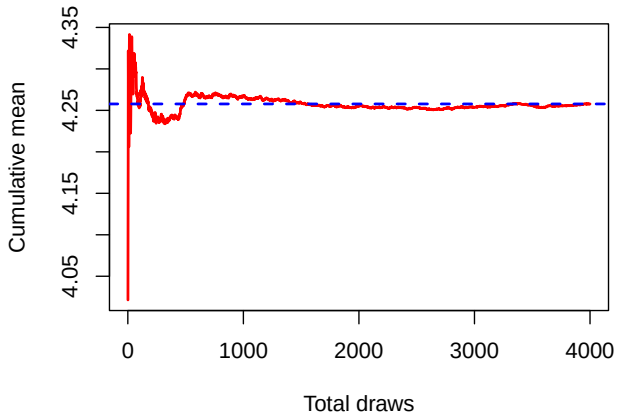

```
phi_draws <- postDraws$phi
cum_mean_phi <- cumsum(phi_draws) / (1:length(phi_draws))

par(mfrow = c(1, 2))
plot(1:length(phi_draws), cum_mean_phi, lwd = 2,
     type = "l", col = "red", main = "Convergence for phi",
     xlab = "Total draws", ylab = "Cumulative mean")
abline(h = posterior_mean[2], lwd = 2, lty = 2, col = "blue")

hist(phi_draws, main = "Histogram of phi draws")
abline(v = credible_intervals[2,1], lty = 2, lwd = 2, col = "red")
```

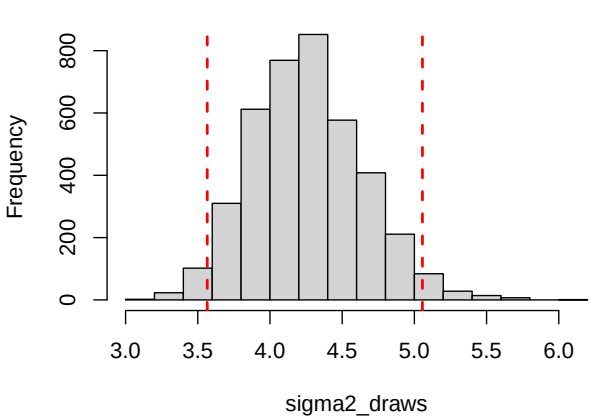
```r
abline(v = credible_intervals[2,2], lty = 2, lwd = 2, col = "red")
```






```r
sigma2_draws <- postDraws$sigma2
cum_mean_sigma2 <- cumsum(sigma2_draws) / (1:length(sigma2_draws))

par(mfrow = c(1, 2))
plot(1:length(sigma2_draws), cum_mean_sigma2, lwd = 2,
     type = "l", col = "red", main = "Convergence for sigma^2",
     xlab = "Total draws", ylab = "Cumulative mean")
abline(h = posterior_mean[3], lwd = 2, lty = 2, col = "blue")

hist(sigma2_draws, main = "Histogram of sigma^2 draws")
abline(v = credible_intervals[3,1], lty = 2, lwd = 2, col = "red")
abline(v = credible_intervals[3,2], lty = 2, lwd = 2, col = "red")
```





The converged values for μ, ϕ and σ^2 can be seen below. We can say that the values converges to values that are so close to true values

```r
cat("mu converges to", cum_mean_mu[length(cum_mean_mu)], "while true value is 9\n")
```

```
## mu converges to 9.004768 while true value is 9
```

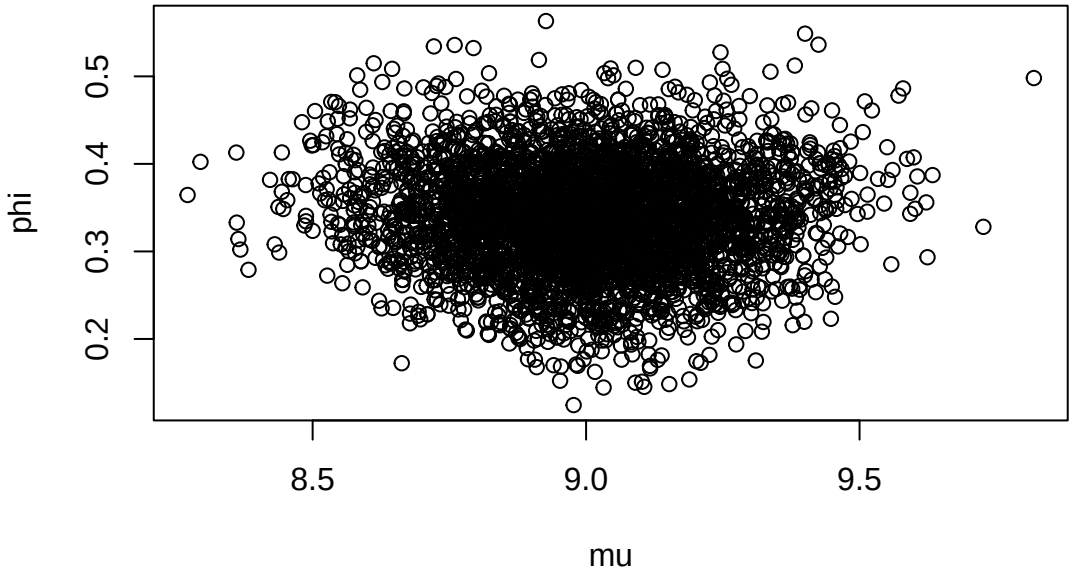
```r
cat("phi converges to", cum_mean_phi[length(cum_mean_phi)], "while true value is 0.3\n")
```

```
## phi converges to 0.337558 while true value is 0.3
```

```r
cat("sigma^2 converges to",cum_mean_sigma2[length(cum_mean_sigma2)], "while true value is 4\n")
```

```
## sigma^2 converges to 4.257748 while true value is 4
```

```r
# Joint posterior draws
plot(postDraws$mu, postDraws$phi, xlab = "mu", ylab = "phi")
```



case ii) phi = 0.97 When $\phi = 0.97$ the model becomes $x_t = 0.03 \cdot \mu + 0.97 \cdot x_{t-1} + \epsilon_t$. x_t becomes more dependent on the previous sample x_{t-1} in this case compared to $\phi = 0.3$

```r
y = AR1_simulate(phi = 0.97, mu = 9, sigma2 = 4, T = 250)
N=length(y)

StanModel = '
data {
  int<lower=0> N;
  vector[N] y;
}
parameters {
  real mu;
  real<lower=-1, upper=1> phi;
  real<lower=0> sigma2;
}
model {
  mu ~ normal(9, 100);
  phi ~ uniform(-1, 1);
  sigma2 ~ scaled_inv_chi_square(0.0001, 0.0001); // as nu, tau^2 -> 0, it becomes jeffreys prior

  for (n in 2:N) {
    y[n] ~ normal(mu + phi * (y[n-1] - mu), sqrt(sigma2));
  }
}'

data <- list(N=N, y=y)
warmup <- 1000
niter <- 2000
```

```
fit <- stan(model_code=StanModel,data=data, warmup=warmup,iter=niter,chains=4)
```

```
## Trying to compile a simple C file
```

```
## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## using C compiler: 'Apple clang version 15.0.0 (clang-1500.3.9.4)'
## using SDK: 'MacOSX14.4.sdk'
## clang -arch x86_64 -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I"/Library/Frame
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/library/StanHeade
## In file included from /Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/library/RcppEiger
## In file included from /Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/library/RcppEiger
## /Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/library/RcppEigen/include/Eigen/src/Co
## #include <cmath>
##          ^~~~~~
## 1 error generated.
## make: *** [foo.o] Error 1
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 9.6e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.96 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: 0.312 seconds (Warm-up)
## Chain 1:                0.128 seconds (Sampling)
## Chain 1:                0.44 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 2.4e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.24 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: 0.452 seconds (Warm-up)
## Chain 2:                0.363 seconds (Sampling)
## Chain 2:                0.815 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 2.4e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.24 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: 0.551 seconds (Warm-up)
## Chain 3:                0.257 seconds (Sampling)
## Chain 3:                0.808 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: 0.487 seconds (Warm-up)
## Chain 4:                0.319 seconds (Sampling)
## Chain 4:                0.806 seconds (Total)
## Chain 4:
```

```
# Print the fitted model
print(fit, digits_summary=3)
```

```
## Inference for Stan model: anon_model.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##            mean se_mean     sd     2.5%      25%      50%      75%    97.5%
## mu        9.029   1.337 24.810  -47.511    6.694   10.088   13.150   53.273
## phi       0.967   0.001  0.022    0.924    0.952    0.968    0.986    0.999
## sigma2    4.503   0.014  0.405    3.790    4.221    4.473    4.755    5.349
## lp__   -314.801   0.135  2.169 -320.239 -315.943 -314.133 -313.133 -312.340
##        n_eff  Rhat
## mu       345 1.018
## phi      336 1.013
## sigma2   893 1.001
## lp__     257 1.022
##
## Samples were drawn using NUTS(diag_e) at Mon May 13 09:09:10 2024.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```
# Extract posterior samples
postDraws <- extract(fit)
```

```
fit_summary <- summary(fit)
posterior_mean <- fit_summary$summary[, "mean"]
credible_intervals <- fit_summary$summary[, c("2.5%", "97.5%")]
effective_samples <- fit_summary$summary[, "n_eff"]

cat("posterior mean:\n")
```

```
## posterior mean:
```

```
print(posterior_mean[1:3])
```

```
##        mu       phi    sigma2
## 9.0285001 0.9673845 4.5026041
```

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

```
## 95% credible interval:
```

```
print(credible_intervals[1:3, ])
```

```
##               2.5%      97.5%
## mu     -47.5113115 53.2729149
```

```
## phi      0.9243573  0.9991802
## sigma2   3.7897965  5.3488360
```

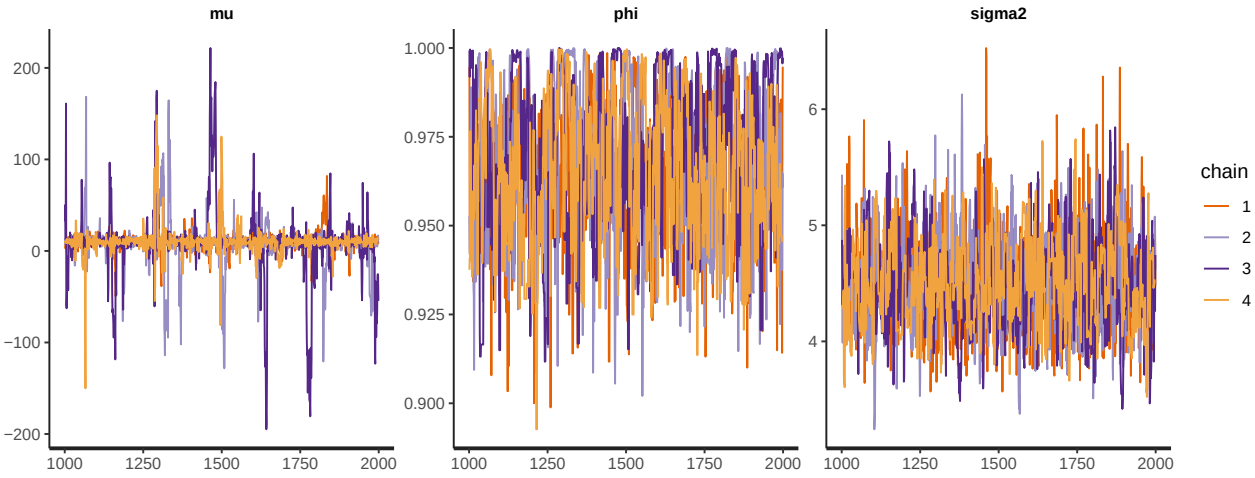
```
cat("Effective sample size:\n")
```

```
## Effective sample size:
```

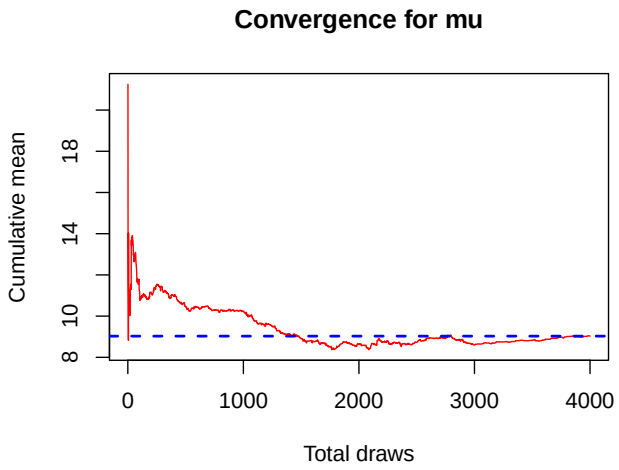
```
print(effective_samples[1:3])
```

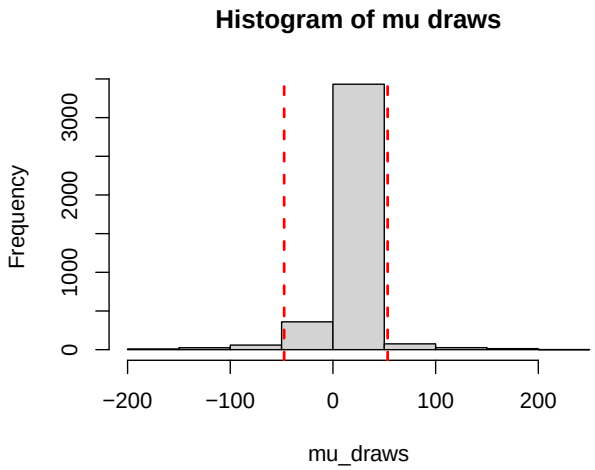
```
##       mu      phi   sigma2
## 344.5234 336.3510 893.3240
```

```
traceplot(fit)
```

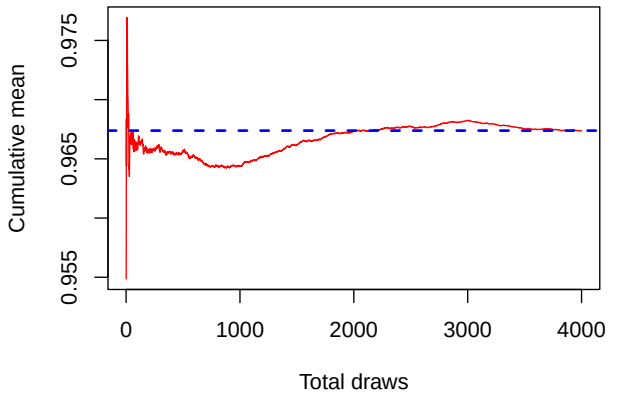


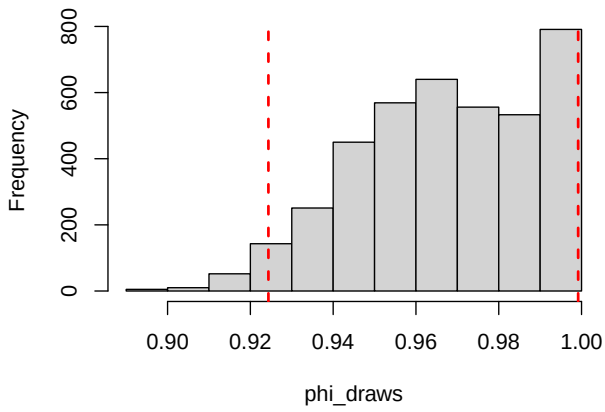

```
mu_draws <- postDraws$mu
cum_mean_mu <- cumsum(mu_draws) / (1:length(mu_draws))

par(mfrow = c(1, 2))
plot(1:length(mu_draws), cum_mean_mu,
     type = "l", col = "red", main = "Convergence for mu",
     xlab = "Total draws", ylab = "Cumulative mean")
abline(h = posterior_mean[1], lwd = 2, lty = 2, col = "blue")

hist(mu_draws, main = "Histogram of mu draws")
abline(v = credible_intervals[1,1], lty = 2, lwd = 2, col = "red")
abline(v = credible_intervals[1,2], lty = 2, lwd = 2, col = "red")
```





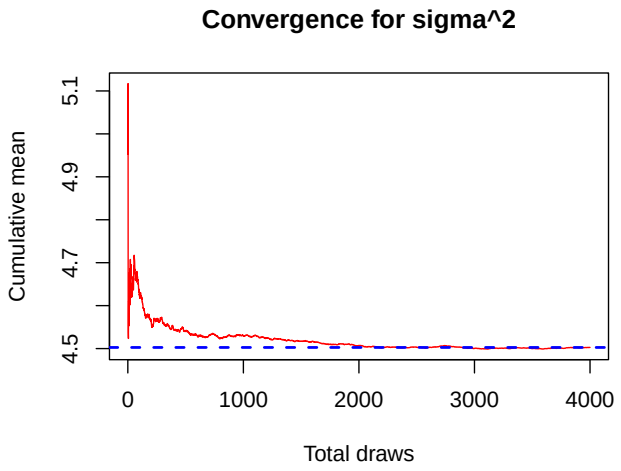

```r
phi_draws <- postDraws$phi
cum_mean_phi <- cumsum(phi_draws) / (1:length(phi_draws))

par(mfrow = c(1, 2))
plot(1:length(phi_draws), cum_mean_phi,
     type = "l", col = "red", main = "Convergence for phi",
     xlab = "Total draws", ylab = "Cumulative mean")
abline(h = posterior_mean[2], lwd = 2, lty = 2, col = "blue")

hist(phi_draws, main = "Histogram of phi draws")
abline(v = credible_intervals[2,1], lty = 2, lwd = 2, col = "red")
abline(v = credible_intervals[2,2], lty = 2, lwd = 2, col = "red")
```

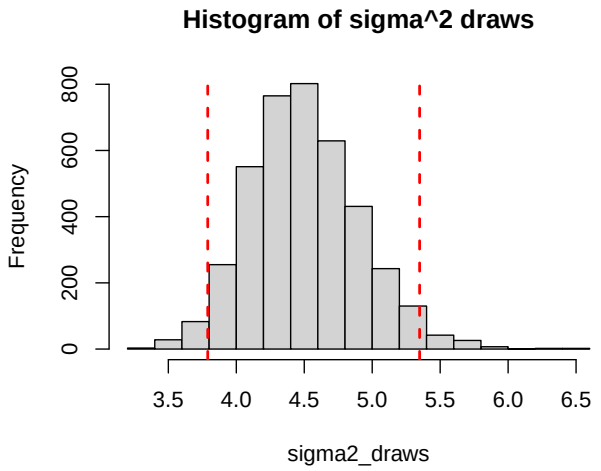





```r
sigma2_draws <- postDraws$sigma2
cum_mean_sigma2 <- cumsum(sigma2_draws) / (1:length(sigma2_draws))

par(mfrow = c(1, 2))
plot(1:length(sigma2_draws), cum_mean_sigma2,
     type = "l", col = "red", main = "Convergence for sigma^2",
     xlab = "Total draws", ylab = "Cumulative mean")
abline(h = posterior_mean[3], lwd = 2, lty = 2, col = "blue")

hist(sigma2_draws, main = "Histogram of sigma^2 draws")
```

```
abline(v = credible_intervals[3,1], lty = 2, lwd = 2, col = "red")
abline(v = credible_intervals[3,2], lty = 2, lwd = 2, col = "red")
```






```
cat("mu converges to", cum_mean_mu[length(cum_mean_mu)], "while true value is 9\n")
```

```
## mu converges to 9.0285 while true value is 9
```

```
cat("phi converges to", cum_mean_phi[length(cum_mean_phi)], "while true value is 0.97\n")
```

```
## phi converges to 0.9673845 while true value is 0.97
```

```
cat("sigma^2 converges to",cum_mean_sigma2[length(cum_mean_sigma2)], "while true value is 4\n")
```

```
## sigma^2 converges to 4.502604 while true value is 4
```

The converged values are worse with $\phi = 0.97$ compared to $\phi = 0.3$. When $\phi = 0.97$, AR(1) model puts more emphasis on the previous sample. 95% credible intervals are wider, especially for parameter μ when $\phi = 0.97$. From the joint distribution below, it can be observed that the variance of μ is high and too high or too low μ values have ϕ value close to 1. High value of ϕ yields high autocorrelation between samples

```
# Joint posterior draws
plot(postDraws$mu, postDraws$phi, xlab = "mu", ylab = "phi")
```