Bayesian Learning Lab 3

Qinyuan Qi(qinqi464)   Satya Sai Naga Jaya Koushik Pilla (satpi345)

2024-05-27

1. Gibbs sampling for the logistic regression

According to the question, we have prior distribution as follows:

$$\beta \sim N(0, \tau^2 I) \text{ where } \tau = 3$$

and we have the following logistic regression model, if y=1 means woman works and 0 means woman not work.

$$Pr(y = 1|x, \beta) = \frac{exp(x^T \beta)}{1 + exp(x^T \beta)}$$

Using the Polya-Gamma latent variables ω_i, the likelihood can be augmented as follows:

$$Pr(y_i = 1|x_i, \beta) = \frac{exp(x_i^T \beta)}{1 + exp(x_i^T \beta)}$$

According to the L7 Slide24, we know that to simulate from the joint posterior $p(\omega, \beta|y)$ we will use the following formulas:

$$\omega_i|\beta \sim PG(1, x_i^T \beta)$$

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

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

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

$$B = \tau^2 * I$$

1.a Implement a Gibbs sampler that simulates from the joint posterior

Code as follows.

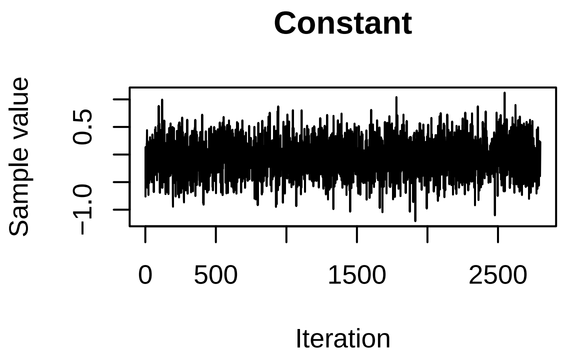
```r
#-------------------------------------------------------------------
# load the data
#-------------------------------------------------------------------
WomenAtWork = read.table("WomenAtWork.dat", header = TRUE)
y = WomenAtWork$Work
X = WomenAtWork[,2:8]
X = as.matrix(X)
Xnames <- colnames(X)

tau <- 3

#get dimensions
n <- nrow(X)
p <- ncol(X)
```

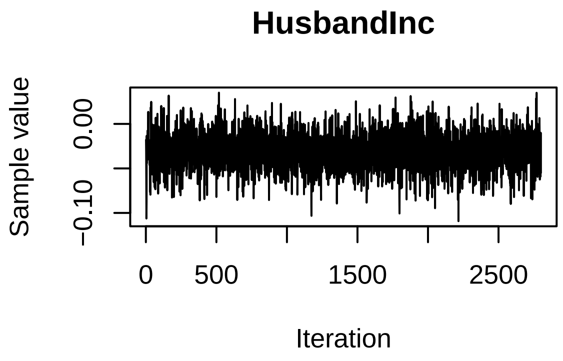
```r
set.seed(12345)

# Initialize parameters
beta <- rep(0, p)

# Number of iterations
n_iter <- 3000

burn_in = 200

beta_samples <- matrix(0, ncol = p, nrow = n_iter)
omega <- rep(1, n)

# B = tao^2 * I
B <- diag(tau^2, p)

# Gibbs Sampling
for (iter in 1:n_iter) {
    # draw samples using rpg function
    omega <- rpg(n, 1, X %*% beta)

    # Update beta according to the formula mentioned previously
    V_beta <- solve(t(X) %*% diag(omega) %*% X + B)
    # b = 0 and k = (y - 0.5)
    m_beta <- V_beta %*% t(X) %*% (y - 0.5)
    beta <- mvrnorm(1, mu = m_beta, Sigma = V_beta)

    # Store samples
    beta_samples[iter, ] <- beta
  }

# Remove burn-in samples
beta_samples <- beta_samples[-(1:burn_in), ]

# Convert samples to mcmc object
mcmc_samples <- as.mcmc(beta_samples)
summary_stats <- summary(mcmc_samples)
```

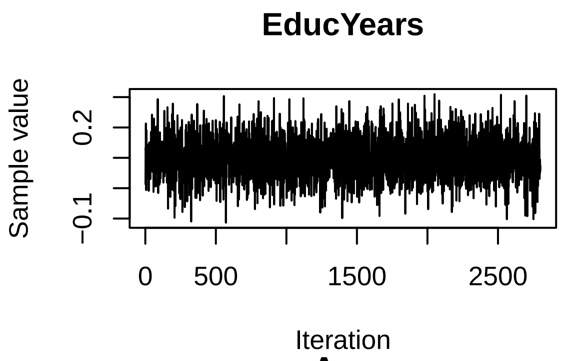
Ineciency Factors (IFs) and trajectories of the sampled Markov chains are printed and plotted below:

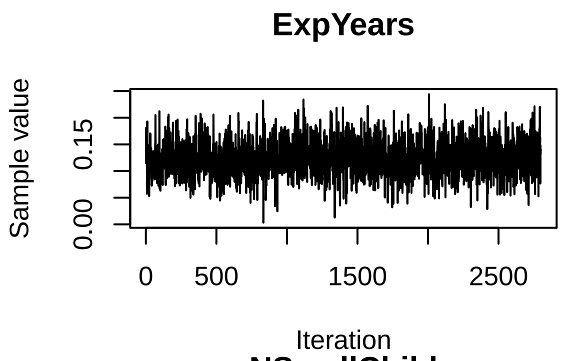
```
##
## Iterations = 1:2800
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 2800
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##            Mean      SD  Naive SE Time-series SE
## [1,] -0.0470287 0.32671 0.0061742      0.0061742
## [2,] -0.0311264 0.02086 0.0003942      0.0004495
## [3,]  0.0989328 0.06666 0.0012597      0.0012329
## [4,]  0.1242003 0.03295 0.0006227      0.0008540
## [5,] -0.0312087 0.01925 0.0003639      0.0003825
## [6,] -0.5633331 0.26523 0.0050124      0.0050124
## [7,]  0.0004081 0.13747 0.0025979      0.0028102
##
## 2. Quantiles for each variable:
##
##          2.5%      25%       50%      75%     97.5%
## var1 -0.68920 -0.27400 -0.052735  0.17967  0.580956
## var2 -0.07298 -0.04522 -0.030489 -0.01734  0.009162
## var3 -0.03051  0.05345  0.098771  0.14203  0.229556
## var4  0.06274  0.10128  0.123302  0.14557  0.190760
## var5 -0.06870 -0.04384 -0.031360 -0.01858  0.005993
## var6 -1.06285 -0.74513 -0.564934 -0.38630 -0.040067
## var7 -0.26628 -0.09179  0.001498  0.09200  0.270295
```

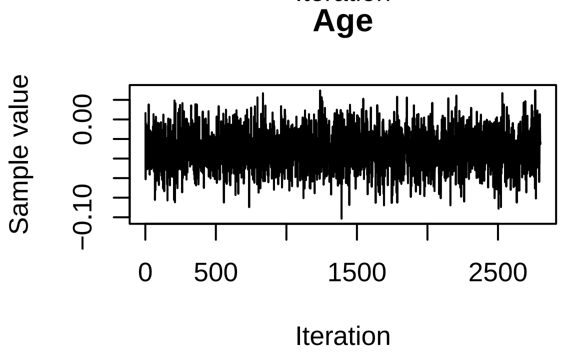
```
## [1] 0.0061742106 0.0004494684 0.0012328557 0.0008539591 0.0003824774
## [6] 0.0050124076 0.0028101753
```

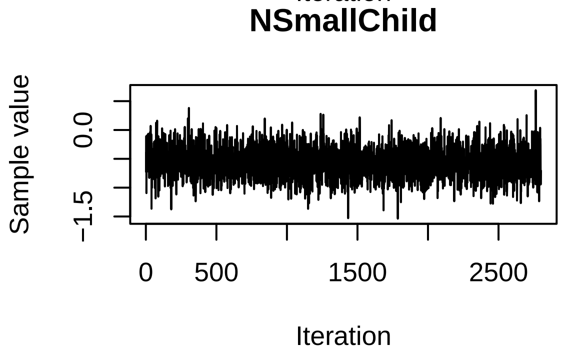


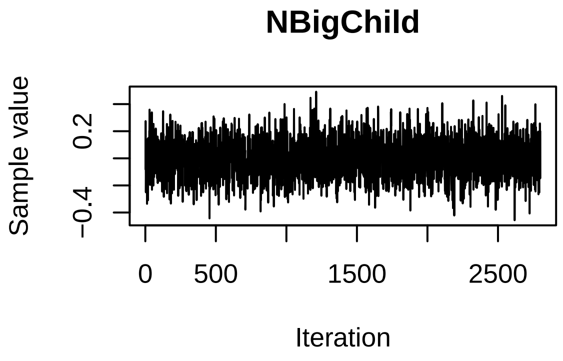












1.b compute a 90% equal tail credible interval

```
# Define the predictor vector x
# a husband with an income of 22
```

```r
# 12 years of education
# 7 years of exp erience,
# a 38-year-old woman,
# one child (3 years old)
x_new <- c(1, 22, 12, 7, 38, 1, 0)

probabilities <- apply(beta_samples, 1, function(beta) {
  exp(sum(beta * x_new)) / (1 + exp(sum(beta * x_new)))
})

# Compute the 90% equal tail credible interval for the probabilities
credible_interval <- quantile(probabilities, probs = c(0.05, 0.95))
print(credible_interval)
```

```
##        5%       95%
## 0.2741694 0.5278513
```

2 Metropolis Random Walk for Poisson regression

We have the following Poisson regression model as follows:

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

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

```r
#--------------------------------------------------------------------
# code for question 2.a
#--------------------------------------------------------------------
# remove covariate const (2nd column)
data_noconst <- ebay_data[,-2]
glm_model <- glm(nBids ~ ., family = poisson(link = "log"), data = data_noconst)
summary(glm_model)
```

```
##
## Call:
## glm(formula = nBids ~ ., family = poisson(link = "log"), data = data_noconst)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  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: 1699.6  on 799  degrees of freedom
## Residual deviance:  691.8  on 791  degrees of freedom
## AIC: 2879.1
##
## Number of Fisher Scoring iterations: 5
```

According to the Pr value in the summary of the glm model, we can find the significant covariates as follows:

- (Intercept)
- VerifyID
- Sealed
- MinBidShare

2.b Bayesian analysis of the Poisson regression

According to the question, we have prior distribution as follows:

$$\beta \sim N(0, 100 \cdot (X^T X)^{-1}]$$

Where X is the n x p matrix of covariates.

We assume the posterior density is approximately multivariate normal as follows:

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

Where $\tilde{\beta}$ is the posterior mode and $J(\tilde{\beta})$ is the negative Hessian at the posterior mode.

We know that the PMF of Poisson distribution is:

$$P(Y|X, \beta) = \frac{\lambda^Y exp(-\lambda)}{Y!}, \ where \ \lambda = exp(X^T \beta)$$

$$P(Y_i|X_i, \beta) = \prod_i \frac{exp(y_i x^T \beta) exp(-exp(x_i^T \beta)}{y!}$$

$$Likehood(\beta|X, Y) = \prod_i \frac{exp(y_i x^T \beta) exp(-exp(x_i^T \beta)}{y!}$$

We apply log on both sides.

$$LogLikehood(\beta|X, Y) = \sum_i (y_i x^T \beta - exp(x_i^T \beta) - log(y!)) \sim \sum_i (y_i x^T \beta - exp(x_i^T \beta))$$

To use optim , we need to implement the following function to calculate the optimized $\tilde{\beta}$ and $J_y^{-1}(\tilde{\beta})$.

```r
#-------------------------------------------------------------------
# code for question 2.b
#-------------------------------------------------------------------
logPost <- function(beta,X=covariates,Y=response){
  linPred <- X%*%beta;
  logLik <- sum(Y * linPred - exp(linPred));
  logPrior <- dmvnorm(t(beta),
                      mean = matrix(0,nrow=ncol(X)),
                      sigma = 100 * (solve(t(X) %*% X)),
                      log=TRUE);
  return(logLik + logPrior)
}
```

We will optimize the logPost function to get the posterior mode $\tilde{\beta}$.

```
# get response and covariates from original data
response <- as.matrix(ebay_data$nBids)
covariates <- as.matrix(ebay_data[,2:10])

# initial values
init_val <- matrix(1,nrow=9);

# optimize the log posterior
OptimRes <- optim(par = init_val,
                  fn = logPost,
                  method=c("BFGS"),
                  control=list(fnscale=-1),
                  hessian=TRUE)
# set values to print out
posterior_mode  <- OptimRes$par
beta_jacobian <- -OptimRes$hessian
beta_inverse_jacobian <- solve(beta_jacobian)
```

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

```
##         Const PowerSeller   VerifyID   Sealed     Minblem    MajBlem    LargNeg
## [1,] 1.077214 -0.03568047 -0.4535361 0.454838 -0.06862956 -0.2259093 0.05388045
##          LogBook MinBidShare
## [1,] -0.08455901   -1.822797
```

```
## [1] "The glm_model coefficients is:"
```

```
## (Intercept) PowerSeller    VerifyID      Sealed     Minblem     MajBlem
##  1.07980512 -0.03566493 -0.45563760  0.45515199 -0.06836819 -0.22554138
##     LargNeg     LogBook MinBidShare
##  0.05382386 -0.08498844 -1.82490142
```

```
## [1] "The beta_inverse_jacobian is:"
```

```
##               [,1]          [,2]          [,3]          [,4]          [,5]
##  [1,]  1.148917e-03 -8.900143e-04 -0.0003857626 -3.819286e-04 -5.338526e-04
##  [2,] -8.900143e-04  1.736863e-03 -0.0001082323 -3.044661e-04  7.428836e-05
##  [3,] -3.857626e-04 -1.082323e-04  0.0161688440 -9.387804e-04  1.663105e-04
##  [4,] -3.819286e-04 -3.044661e-04 -0.0009387804  3.877795e-03  4.467703e-04
##  [5,] -5.338526e-04  7.428836e-05  0.0001663105  4.467703e-04  5.181524e-03
##  [6,] -3.275726e-04 -2.789621e-04  0.0003412794  5.282335e-04  4.407868e-04
##  [7,] -6.095167e-04  3.646141e-04  0.0003556730  3.758844e-04  6.458170e-05
##  [8,]  4.161132e-05  1.732925e-04 -0.0003755887 -5.809023e-05 -1.413840e-06
##  [9,]  1.323991e-03 -6.727699e-04 -0.0008293591 -1.322366e-04 -1.987749e-04
##               [,6]          [,7]          [,8]          [,9]
##  [1,] -0.0003275726 -6.095167e-04  4.161132e-05  1.323991e-03
##  [2,] -0.0002789621  3.646141e-04  1.732925e-04 -6.727699e-04
##  [3,]  0.0003412794  3.556730e-04 -3.755887e-04 -8.293591e-04
##  [4,]  0.0005282335  3.758844e-04 -5.809023e-05 -1.322366e-04
## [5,]  0.0004407868  6.458170e-05 -1.413840e-06 -1.987749e-04
## [6,]  0.0090777128  5.029082e-04 -1.357652e-04  2.878182e-04
## [7,]  0.0005029082  4.106291e-03 -3.195042e-04 -5.372749e-05
## [8,] -0.0001357652 -3.195042e-04  1.045605e-03  1.247818e-03
## [9,]  0.0002878182 -5.372749e-05  1.247818e-03  6.126155e-03
```

From the output of posterior beta and glm_model's coefficients, we can find that the results are similar.

2.c Simulate from the actual posterior of β using the Metropolis algorithm and compare the results with the approximate results in b)

The proposal density is multivariate normal density which given by:

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

where $\sum = J_y^{-1}(\tilde{\beta})$ was obtained in 2.b.

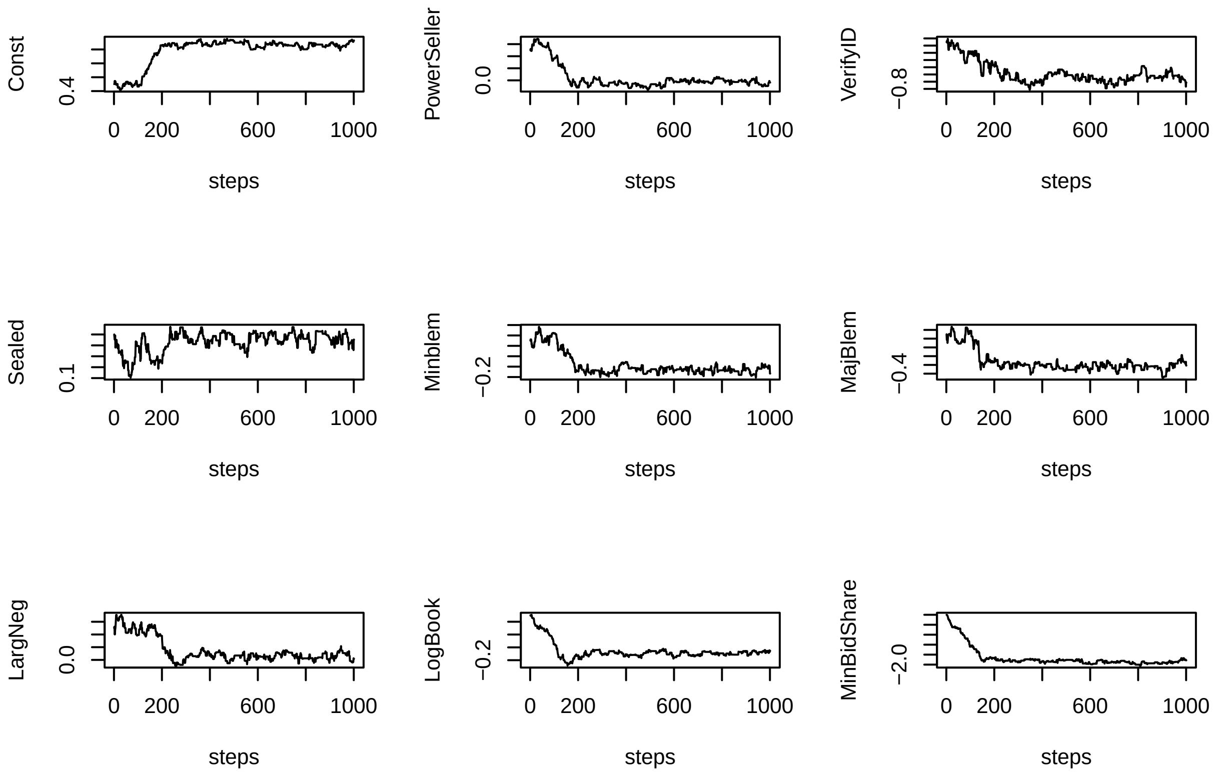
```r
#-------------------------------------------------------------------
# code for question 2.c
#-------------------------------------------------------------------
#-------------------------------------------------------------------
#  metropolis function
#-------------------------------------------------------------------
metropolis_fn <- function(tgt_density, c, theta_i, sigma_proposal, steps, X, Y){
  # init seed
  set.seed(12345)

  result <- matrix(t(theta_i), ncol=9)
  accepted_count <- 0

  for(i in 1:steps){
    # generate sample from proposal
    theta_p <- rmvnorm(n = 1, mean = as.vector(theta_i), sigma = c * sigma_proposal)

    # calculate acceptance ratio
    acceptance_ratio <- tgt_density(as.vector(theta_p), X, Y) -
                        tgt_density(as.vector(theta_i), X, Y)

    # apply exp to acceptance ratio(since original one is in log)
    acceptance_ratio <- exp(acceptance_ratio)

    # calculate alpha
    alpha <- min(1, acceptance_ratio)

    # draw from  uniform distribution
    u <- runif(1)

    # accept or reject
    if (u < alpha){
      theta_i <- theta_p
      accepted_count <- accepted_count + 1
      result <- rbind(result, theta_i)
    }else{
      result <- rbind(result, as.vector(theta_i))
    }
  }
  acceptance <- accepted_count / steps
  return(list(result = result, acceptance = acceptance))
}
```

We choose the same prarms for prior of target density.

```
#-------------------------------------------------------------------
# function call to run metropolis
#-------------------------------------------------------------------
sigma_proposal <- beta_inverse_jacobian
theta_init <- matrix(rep(0.5,9), ncol=9)

# run metropolis
metropolis_val <- metropolis_fn(tgt_density = logPost,
                                c = 0.5,
                                theta_i = theta_init,
                                sigma_proposal = sigma_proposal,
                                steps = 1000,
                                X  = covariates,
                                Y = response)

metropolis_result <- metropolis_val$result
colnames(metropolis_result) <- rownames(posterior_mode)
metropolis_result_mean <- apply(metropolis_result, 2, mean)
names(metropolis_result_mean) <- rownames(posterior_mode)
metropolis_accept <- metropolis_val$acceptance
```

```
## [1] "The metropolis_result_mean is:"

##       Const PowerSeller    VerifyID      Sealed     Minblem     MajBlem
##  0.98407853  0.03787105 -0.36383252  0.43210917  0.03046530 -0.13401017
##     LargNeg     LogBook MinBidShare
##  0.14433885 -0.06145534 -1.65778520

## [1] "The metropolis_accept is:"

## [1] 0.325

## [1] "shape of metropolis_result is:"

## [1] 1001    9
```

Then we plot the result of the metropolis coefficients as follows:



We change diffent value of c ,theta_init and steps, al the covariates are convergent very fast, but when we change the value c and theta_init, the burn-in period will be different. And when steps is large,the convergence will be more stable.

2.d Use the MCMC draws from c) to simulate from the predictive distribution and plot

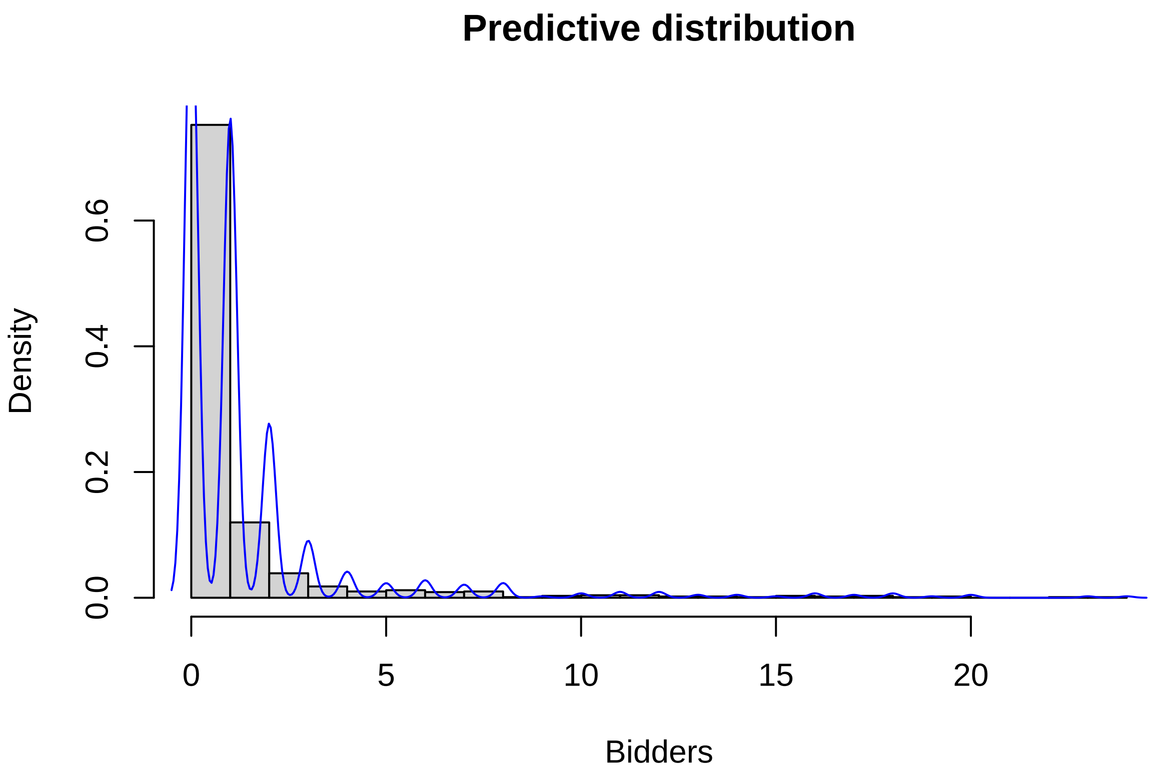
Now we use MCMC to draw coefficients from 2.c.

The probability of no bidders is around 0.0475.

```
#------------------------------------------------------------------
# code for question 2.d
#------------------------------------------------------------------
new_data <- c(1, 1, 0, 1, 0, 1, 0, 1.2, 0.8)
result2 <- data.frame(matrix(0, nrow = nrow(metropolis_result), ncol = 9))

for(i in 1:nrow(metropolis_result)){
  lambda <- exp(new_data %*% as.numeric(metropolis_result[i,]))
  result2[i,] <- rpois(1, lambda)
}
```

```
## [1] "Probability of no bidders is:"
```

```
## [1] 0.4225774
```



3 Time series models in Stan

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

According to the question, AR(1) process is defined as follows:

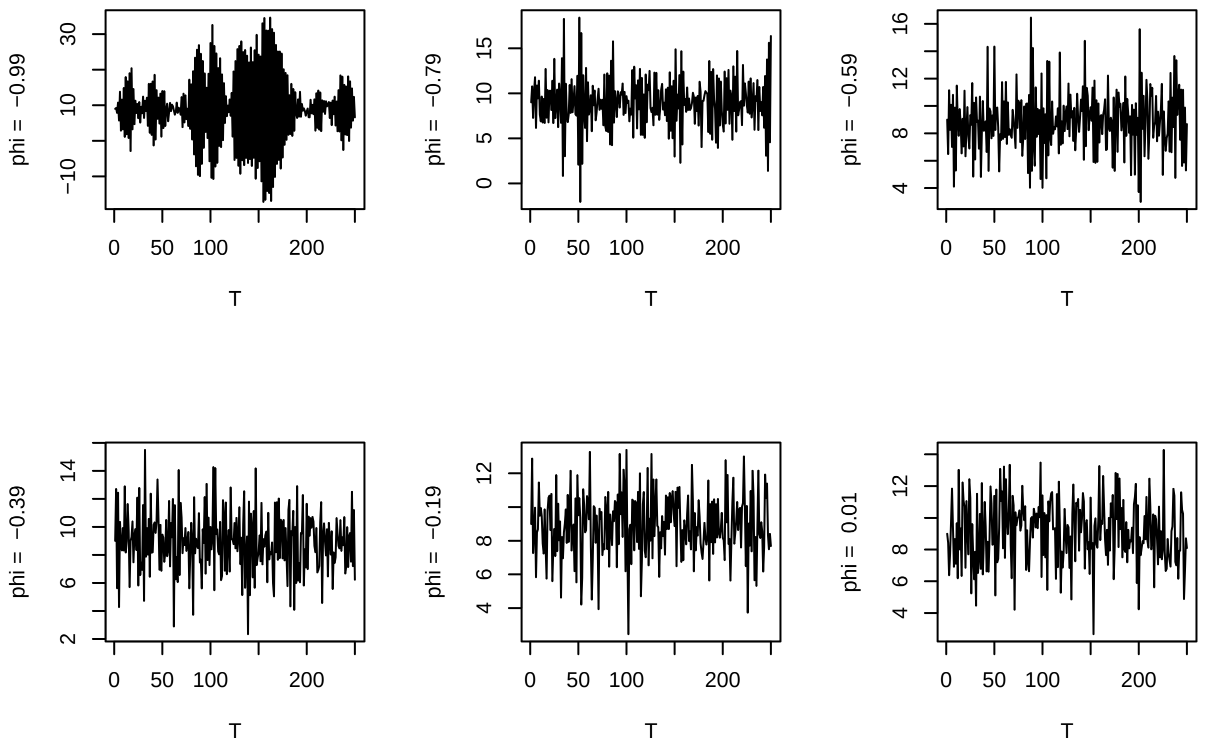
$$x_t = \mu + \phi(x_{t-1} - \mu) + \epsilon_t, \epsilon_t \overset{\text{iid}}{\sim} N(0, \sigma^2)$$

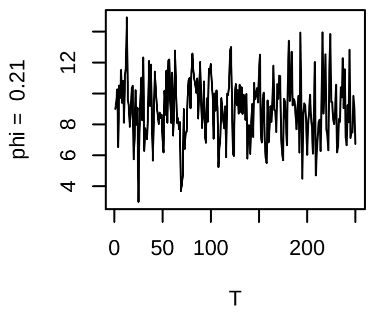
According to the question, the parameters listed below:

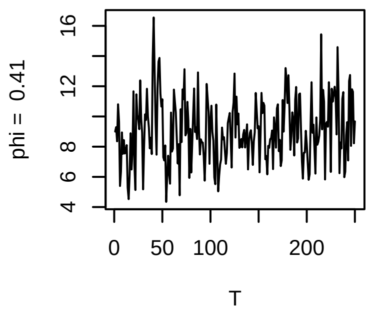
```
#---------------------------------------------------------------------
# code for question 3.a
#---------------------------------------------------------------------
#-----------------------------------------------
# Parameters for AR(1) process
#-----------------------------------------------
phis <- seq(-0.99, 0.99, by = 0.20)
sigma2 <- 4
mu <- 9
T <- 250
```

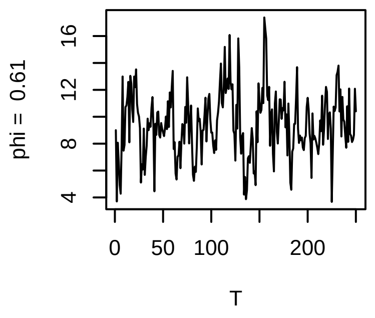
Then we define the function to simulate AR(1) process:

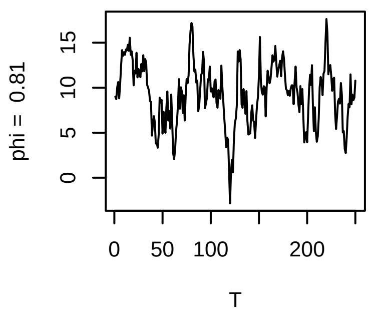
```
#-----------------------------------------------
# Function to simulate AR(1) process
#-----------------------------------------------
simulate_ar1 <- function(mu=9, phi, sigma2=4, T=250) {
  # init variables
  n <- length(phis)
  values <- data.frame(0,ncol = n)
  for(i in 1:n){
    x = mu
    values[1,i] = x

    for (j in 2:T) {
        x <- mu + phi[i] * (x - mu) + rnorm(1, 0, sqrt(sigma2))
        values[j,i] = x
    }
  }
  # set column names
  colnames(values) <- paste("phi_",phi)
  return(values)
}

# function call
ar1 <- simulate_ar1(mu = mu, phi = phis, sigma2 = sigma2,T = T)
```











From the plots above , we can found that smaller value of θ has much larger effect on the next prediction of x_t. The reason stated as below:

From the formula of the AR(1) process, we know that $\mu = 13$ is fixed in our case, when ϕ is close to -1, The first two terms of the equation cancel each other out and get close to 0, the third term of the equation, which is random variable ϵ_t will play a more important role in the prediction of x_t. As a result, the predicted value of x_t will greatly effected by random variable ϵ_t, so it has a negative correlation. But when ϕ is close to 1, the first two items will become a positive value which make ϕ positive correlation to the prediction of x_t.

3.b Use your function from a) to simulate two AR(1)-processes

According to the question, we assume that μ, ϕ, σ^2 are unknown parameters. And we choose non-informative priors of those 3 parameters are as follows:

$$\mu \sim N(0, 50)$$

$$\sigma^2 \sim Inv - \chi^2(1, 10)$$

$$\phi \sim Uniform(-1, 1)$$

$x_{1:T}$ with $\phi = 0.3$ and $y_{1:T}$ with $\phi = 0.97$.

Since model assumes that each observation is a linear combination of the previous observation and a random term ϵ_t, so AR(1)-processes will generated x will follow a normal distribution.

$$x_t | x_{t-1} \sim N(\mu + \phi(x_{t-1} - \mu), \sigma_\epsilon^2)$$

using the code in stan guide as a reference, we can write the stan code as follows:

```r
#------------------------------------------------------------------
# code for question 3.b
#------------------------------------------------------------------

model_AR <- simulate_ar1(phi = c(0.3, 0.97))

StanModel = '
data {
  int<lower=0> N; // Number of observations
  vector[N] x;    // x_t
}
parameters {
  real mu;
  real<lower = 0> sigma2;
  real<lower = -1, upper = 1> phi;
}
model {
  mu ~ normal(0,50);
  sigma2 ~ scaled_inv_chi_square(1,10);
  phi ~ uniform(-1,1);

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

3.b.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

```r
#------------------------------------------------------------------
# code for question 3.b.i
#------------------------------------------------------------------
#fit model for phi 0.3
fit_0.3 = stan(model_code = StanModel,
               data = list(x = model_AR$`phi_ 0.3`, N = 250),
               warmup = 1000,
               iter = 2000,
               chains = 4)
```

```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000103 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 1.03 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: 0.251 seconds (Warm-up)
## Chain 1:                0.22 seconds (Sampling)
## Chain 1:                0.471 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 4.3e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.43 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: 0.297 seconds (Warm-up)
## Chain 2:                0.254 seconds (Sampling)
## Chain 2:                0.551 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 4.3e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.43 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: 0.278 seconds (Warm-up)
## Chain 3:                0.231 seconds (Sampling)
## Chain 3:                0.509 seconds (Total)
## Chain 3: 
## 
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
## Chain 4: 
## Chain 4: Gradient evaluation took 4.3e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.43 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: 0.25 seconds (Warm-up)
## Chain 4:                0.242 seconds (Sampling)
## Chain 4:                0.492 seconds (Total)
## Chain 4: 
```

```
#fit model for phi 0.97
fit_0.97 = stan(model_code = StanModel,
                data = list(x = model_AR$`phi_ 0.97`, N = 250),
                warmup = 1000,
                iter = 2000,
                chains = 4)
```

```
## 
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1: 
## Chain 1: Gradient evaluation took 4.5e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.45 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: 0.853 seconds (Warm-up)
## Chain 1:                0.373 seconds (Sampling)
## Chain 1:                1.226 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 4.3e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.43 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: 0.877 seconds (Warm-up)
## Chain 2:                0.461 seconds (Sampling)
## Chain 2:                1.338 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 4.4e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.44 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: 0.785 seconds (Warm-up)
## Chain 3:                0.533 seconds (Sampling)
## Chain 3:                1.318 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 4.2e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.42 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: 0.867 seconds (Warm-up)
## Chain 4:                0.435 seconds (Sampling)
## Chain 4:                1.302 seconds (Total)
## Chain 4:
```

```
## Warning: There were 1 divergent transitions after warmup. See
## https://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
## to find out why this is a problem and how to eliminate them.
```

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

```r
post_samples_0.3 <- extract(fit_0.3)
post_mean_0.3 <- get_posterior_mean(fit_0.3)
post_samples_0.97 <- extract(fit_0.97)
post_mean_0.97 <- get_posterior_mean(fit_0.97)
```

```r
#---------------------------------------------------------------------
# code for question 3.b.ii
#---------------------------------------------------------------------
post_samples_0.3_df <- data.frame(mu = post_samples_0.3$mu,
                                  sigma2 = post_samples_0.3$sigma2,
                                  phi = post_samples_0.3$phi)

post_samples_0.97_df <- data.frame(mu = post_samples_0.97$mu,
                                   sigma2 = post_samples_0.97$sigma2,
                                   phi = post_samples_0.97$phi)

CI_0.3 <- sapply(post_samples_0.3_df, function(x) quantile(x, probs = c(0.025, 0.975)))
CI_0.97 <- sapply(post_samples_0.97_df, function(x) quantile(x, probs = c(0.025, 0.975)))
```

```
## [1] "Posterior means when phi = 0.3 :"
```

```
##        mean-chain:1 mean-chain:2 mean-chain:3 mean-chain:4 mean-all chains
## mu         8.803321    8.8001200    8.7896558     8.788493       8.7953976
## sigma2     4.235799    4.2420982    4.2295482     4.215222       4.2306668
## phi        0.306810    0.3100866    0.3123154     0.308769       0.3094953
## lp__    -305.770701 -305.8547696 -305.8717848  -305.799298    -305.8241385
```

```
## [1] "Posterior means when phi = 0.97 :"
```

```
##        mean-chain:1 mean-chain:2 mean-chain:3 mean-chain:4 mean-all chains
## mu        7.5771386   10.1043238   10.2561931   10.4835487       9.6053011
## sigma2    4.0645614    4.1377727    4.1234889    4.1510724       4.1192238
## phi       0.9874855    0.9876529    0.9876463    0.9876317       0.9876041
## lp__   -306.9167626 -307.0030590 -306.9939647 -307.1178407    -307.0079067
```

```
## [1] "Posterior 95% CI when phi=0.3 :"
```

```
##             mu   sigma2       phi
## 2.5%  8.411703 3.544072 0.1827296
## 97.5% 9.172174 5.051581 0.4391495
```

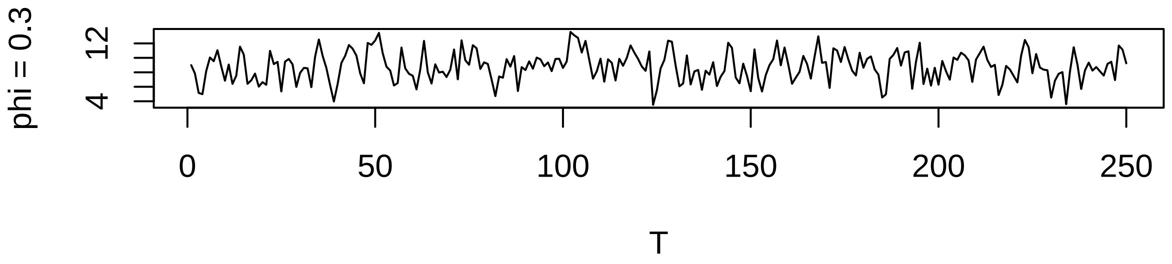
```
## [1] "Posterior 95% CI when phi=0.97 :"
```

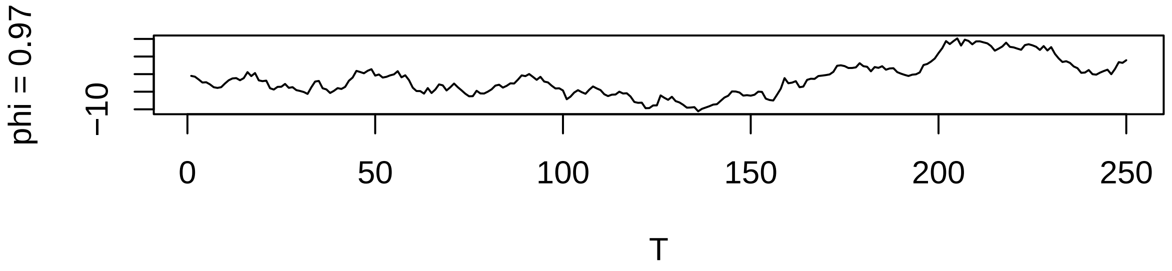
```
##              mu   sigma2       phi
## 2.5%  -42.55449 3.403069 0.9635401
## 97.5%  57.72196 4.939324 0.9994975
```

We can estimate the true values within suitable confidence interval. for example,95% confidence interval.

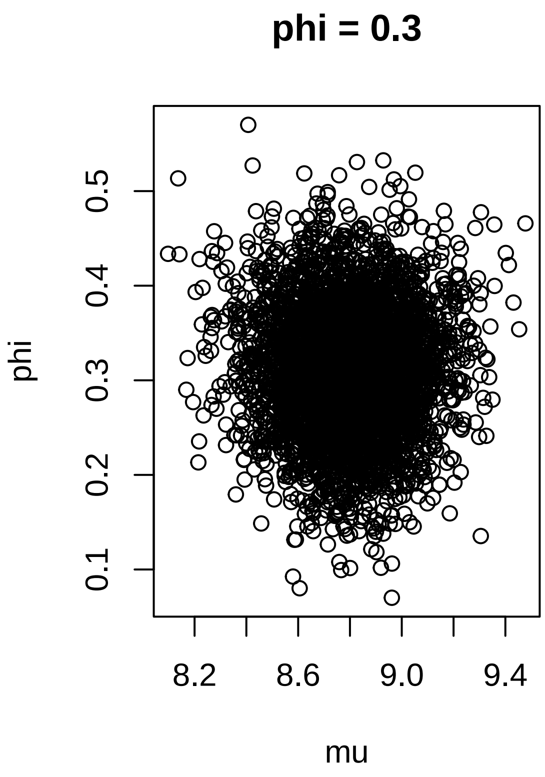
3.b.ii evaluate the convergence of the samplers and plot the joint posterior of μ and ϕ.

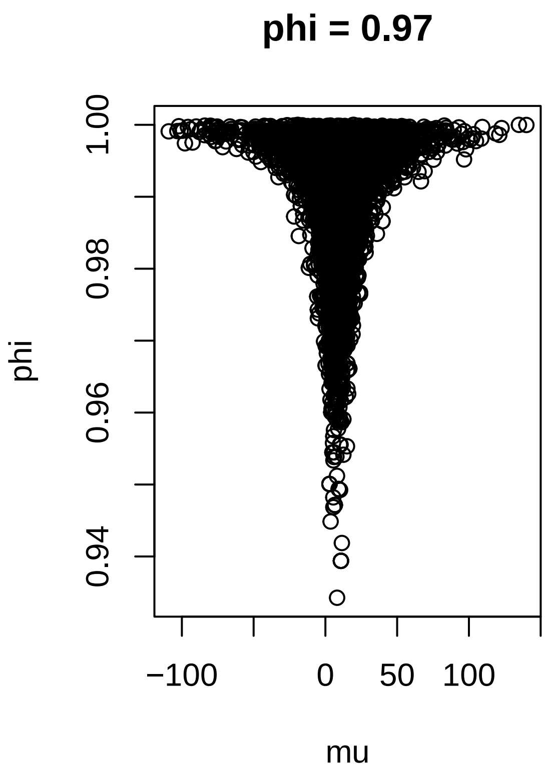
We will plot the models for $\phi = 0.3$ and $\phi = 0.97$.





then we plot the convergence of the samplers and the joint posterior





All parameters are not convergent to a stable value but fluctuate around a value.

From the joint posterior plots above, we can find that μ is more accurate when $\phi = 0.3$. and when ϕ get bigger, the distribution of μ becomes very wide.

APPENDIX: CODE

```r
# ====================================================================
# Init Library
# ====================================================================
knitr::opts_chunk$set(echo = TRUE)
library(ggplot2)
library(gridExtra)
library(readxl)
library(LaplacesDemon)
library(MASS)
library(matrixStats)
library(mvtnorm)
library(rstan)
library(loo)
library(coda)


library(BayesLogit)
rstan_options(auto_write = TRUE)
Sys.setenv(LOCAL_CPPFLAGS = '-march=native')
#--------------------------------------------------------------------
# Init code for question 1
#--------------------------------------------------------------------
rm(list = ls())
#--------------------------------------------------------------------
# load the data
#--------------------------------------------------------------------
WomenAtWork = read.table("WomenAtWork.dat", header = TRUE)
y = WomenAtWork$Work
X = WomenAtWork[,2:8]
X = as.matrix(X)
Xnames <- colnames(X)

tau <- 3

#get dimensions
n <- nrow(X)
p <- ncol(X)

set.seed(12345)

# Initialize parameters
beta <- rep(0, p)

# Number of iterations
n_iter <- 3000

burn_in = 200

beta_samples <- matrix(0, ncol = p, nrow = n_iter)
omega <- rep(1, n)

# B = tao^2 * I
```

```r
B <- diag(tau^2, p)

# Gibbs Sampling
for (iter in 1:n_iter) {
    # draw samples using rpg function
    omega <- rpg(n, 1, X %*% beta)

    # Update beta according to the formula mentioned previously
    V_beta <- solve(t(X) %*% diag(omega) %*% X + B)
    # b = 0 and k = (y - 0.5)
    m_beta <- V_beta %*% t(X) %*% (y - 0.5)
    beta <- mvrnorm(1, mu = m_beta, Sigma = V_beta)

    # Store samples
    beta_samples[iter, ] <- beta
  }

# Remove burn-in samples
beta_samples <- beta_samples[-(1:burn_in), ]

# Convert samples to mcmc object
mcmc_samples <- as.mcmc(beta_samples)
summary_stats <- summary(mcmc_samples)
# print out the summary and the time-series standard error
print(summary_stats)
print(summary_stats$statistics[,"Time-series SE"])

# Plot trajectories of the sampled Markov chains
par(mfrow = c(2, 2))
for (j in 1:ncol(beta_samples)) {
  plot(beta_samples[, j], type = "l", main = Xnames[j],
       xlab = "Iteration", ylab = "Sample value")
}
# Define the predictor vector x
# a husband with an income of 22
# 12 years of education
# 7 years of exp erience,
# a 38-year-old woman,
# one child (3 years old)
x_new <- c(1, 22, 12, 7, 38, 1, 0)

probabilities <- apply(beta_samples, 1, function(beta) {
  exp(sum(beta * x_new)) / (1 + exp(sum(beta * x_new)))
})

# Compute the 90% equal tail credible interval for the probabilities
credible_interval <- quantile(probabilities, probs = c(0.05, 0.95))
print(credible_interval)
#-------------------------------------------------------------------
# Init code for question 2
#-------------------------------------------------------------------
rm(list = ls())
ebay_data <- read.table("eBayNumberOfBidderData_2024.dat", header = T)
```

```
#-----------------------------------------------------------------------
# code for question 2.a
#-----------------------------------------------------------------------
# remove covariate const (2nd column)
data_noconst <- ebay_data[,-2]
glm_model <- glm(nBids ~ ., family = poisson(link = "log"), data = data_noconst)
summary(glm_model)
#-----------------------------------------------------------------------
# code for question 2.b
#-----------------------------------------------------------------------
logPost <- function(beta,X=covariates,Y=response){
  linPred <- X%*%beta;
  logLik <- sum(Y * linPred - exp(linPred));
  logPrior <- dmvnorm(t(beta),
                      mean = matrix(0,nrow=ncol(X)),
                      sigma = 100 * (solve(t(X) %*% X)),
                      log=TRUE);
  return(logLik + logPrior)
}
# get response and covariates from original data
response <- as.matrix(ebay_data$nBids)
covariates <- as.matrix(ebay_data[,2:10])

# initial values
init_val <- matrix(1,nrow=9);

# optimize the log posterior
OptimRes <- optim(par = init_val,
                  fn = logPost,
                  method=c("BFGS"),
                  control=list(fnscale=-1),
                  hessian=TRUE)
# set values to print out
posterior_mode  <- OptimRes$par
beta_jacobian <- -OptimRes$hessian
beta_inverse_jacobian <- solve(beta_jacobian)
#-----------------------------------------------------------------------
# print values
#-----------------------------------------------------------------------
rownames(posterior_mode) <- colnames(covariates)
print('The posterior beta is:')
print(t(posterior_mode))
print('The glm_model coefficients is:')
print(glm_model$coefficients)
print('The beta_inverse_jacobian is:')
print(beta_inverse_jacobian)
#-----------------------------------------------------------------------
# code for question 2.c
#-----------------------------------------------------------------------
#-----------------------------------------------------------------------
#  metropolis function
#-----------------------------------------------------------------------
metropolis_fn <- function(tgt_density, c, theta_i, sigma_proposal, steps, X, Y){
```

```r
  # init seed
  set.seed(12345)

  result <- matrix(t(theta_i), ncol=9)
  accepted_count <- 0

  for(i in 1:steps){
    # generate sample from proposal
    theta_p <- rmvnorm(n = 1, mean = as.vector(theta_i), sigma = c * sigma_proposal)

    # calculate acceptance ratio
    acceptance_ratio <- tgt_density(as.vector(theta_p), X, Y) -
                        tgt_density(as.vector(theta_i), X, Y)

    # apply exp to acceptance ratio(since original one is in log)
    acceptance_ratio <- exp(acceptance_ratio)

    # calculate alpha
    alpha <- min(1, acceptance_ratio)

    # draw from  uniform distribution
    u <- runif(1)

    # accept or reject
    if (u < alpha){
      theta_i <- theta_p
      accepted_count <- accepted_count + 1
      result <- rbind(result, theta_i)
    }else{
      result <- rbind(result, as.vector(theta_i))
    }
  }
  acceptance <- accepted_count / steps
  return(list(result = result, acceptance = acceptance))
}
#-----------------------------------------------------------------
# function call to run metropolis
#-----------------------------------------------------------------
sigma_proposal <- beta_inverse_jacobian
theta_init <- matrix(rep(0.5,9), ncol=9)

# run metropolis
metropolis_val <- metropolis_fn(tgt_density = logPost,
                                c = 0.5,
                                theta_i = theta_init,
                                sigma_proposal = sigma_proposal,
                                steps = 1000,
                                X  = covariates,
                                Y = response)

metropolis_result <- metropolis_val$result
colnames(metropolis_result) <- rownames(posterior_mode)
metropolis_result_mean <- apply(metropolis_result, 2, mean)
```

```r
names(metropolis_result_mean) <- rownames(posterior_mode)
metropolis_accept <- metropolis_val$acceptance
print('The metropolis_result_mean is:')
print(metropolis_result_mean)
print('The metropolis_accept is:')
print(metropolis_accept)
print('shape of metropolis_result is:')
print(dim(metropolis_result))
#-------------------------------------------------------------------
# plot
#-------------------------------------------------------------------
par(mfrow = c(3,3))
for(i in 1:9){
  plot((metropolis_result[,i]), type="l", ylab= colnames(metropolis_result)[i],xlab = "steps")
}

#-------------------------------------------------------------------
# code for question 2.d
#-------------------------------------------------------------------
new_data <- c(1, 1, 0, 1, 0, 1, 0, 1.2, 0.8)
result2 <- data.frame(matrix(0, nrow = nrow(metropolis_result), ncol = 9))

for(i in 1:nrow(metropolis_result)){
  lambda <- exp(new_data %*% as.numeric(metropolis_result[i,]))
  result2[i,] <- rpois(1, lambda)
}
#-------------------------------------------------------------------
# plot
#-------------------------------------------------------------------
print("Probability of no bidders is:")
print(mean(result2 == 0))
hist(result2[,1], main = "Predictive distribution", xlab = "Bidders",breaks = 30, freq = FALSE)
lines(density(result2[,1]),lwd=1,col="blue")
#-------------------------------------------------------------------
# Code for question 3
#-------------------------------------------------------------------
rm(list = ls())
#-------------------------------------------------------------------
# code for question 3.a
#-------------------------------------------------------------------
#---------------------------------------------
# Parameters for AR(1) process
#---------------------------------------------
phis <- seq(-0.99, 0.99, by = 0.20)
sigma2 <- 4
mu <- 9
T <- 250
#---------------------------------------------
# Function to simulate AR(1) process
#---------------------------------------------
simulate_ar1 <- function(mu=9, phi, sigma2=4, T=250) {
  # init variables
  n <- length(phis)
  values <- data.frame(0,ncol = n)

  for(i in 1:n){
    x = mu
    values[1,i] = x

    for (j in 2:T) {
        x <- mu + phi[i] * (x - mu) + rnorm(1, 0, sqrt(sigma2))
        values[j,i] = x
    }
  }
  # set column names
  colnames(values) <- paste("phi_",phi)
  return(values)
}

# function call
ar1 <- simulate_ar1(mu = mu, phi = phis, sigma2 = sigma2,T = T)
#----------------------------------------------
# Plot AR(1) process simulated data
#----------------------------------------------
# set layout to 2 x 3
par(mfrow = c(2,3))
for(i in 1:length(phis)){
  plot(ar1[,i], type = 'l', ylab = paste("phi = ", phis[i]), xlab = "T")
}

#-----------------------------------------------------------------
# code for question 3.b
#-----------------------------------------------------------------

model_AR <- simulate_ar1(phi = c(0.3, 0.97))

StanModel = '
data {
  int<lower=0> N; // Number of observations
  vector[N] x;    // x_t
}
parameters {
  real mu;
  real<lower = 0> sigma2;
  real<lower = -1, upper = 1> phi;
}
model {
  mu ~ normal(0,50);
  sigma2 ~ scaled_inv_chi_square(1,10);
  phi ~ uniform(-1,1);

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

```
#-------------------------------------------------------------------
# code for question 3.b.i
#-------------------------------------------------------------------
#fit model for phi 0.3
fit_0.3 = stan(model_code = StanModel,
               data = list(x = model_AR$`phi_ 0.3`, N = 250),
               warmup = 1000,
               iter = 2000,
               chains = 4)

#fit model for phi 0.97
fit_0.97 = stan(model_code = StanModel,
               data = list(x = model_AR$`phi_ 0.97`, N = 250),
               warmup = 1000,
               iter = 2000,
               chains = 4)

post_samples_0.3 <- extract(fit_0.3)
post_mean_0.3 <- get_posterior_mean(fit_0.3)
post_samples_0.97 <- extract(fit_0.97)
post_mean_0.97 <- get_posterior_mean(fit_0.97)
#-------------------------------------------------------------------
# code for question 3.b.ii
#-------------------------------------------------------------------
post_samples_0.3_df <- data.frame(mu = post_samples_0.3$mu,
                                  sigma2 = post_samples_0.3$sigma2,
                                  phi = post_samples_0.3$phi)

post_samples_0.97_df <- data.frame(mu = post_samples_0.97$mu,
                                  sigma2 = post_samples_0.97$sigma2,
                                  phi = post_samples_0.97$phi)

CI_0.3 <- sapply(post_samples_0.3_df, function(x) quantile(x, probs = c(0.025, 0.975)))
CI_0.97 <- sapply(post_samples_0.97_df, function(x) quantile(x, probs = c(0.025, 0.975)))

#-------------------------------------------------------------------
# print
#-------------------------------------------------------------------
print("Posterior means when phi = 0.3 :")
print(post_mean_0.3)
print("Posterior means when phi = 0.97 :")
print(post_mean_0.97)

print("Posterior 95% CI when phi=0.3 :")
print(CI_0.3)
print("Posterior 95% CI when phi=0.97 :")
print(CI_0.97)
#-------------------------------------------------------------------
# plot
#-------------------------------------------------------------------
par(mfrow = c(2,1))
plot(y=model_AR[,1],x=c(1:250),type='l',ylab='phi = 0.3',xlab='T')
plot(y=model_AR[,2],x=c(1:250),type='l',ylab='phi = 0.97',xlab='T')
```

```r
par(mfrow = c(2,2))
plot(post_samples_0.3$mu, type = 'l', ylab="posterior mu",main="phi = 0.3")
plot(post_samples_0.97$mu, type = 'l', ylab="posterior mu",main="phi = 0.97")

plot(post_samples_0.3$sigma2, type = 'l', ylab="posterior sigma2",main="phi = 0.3")
plot(post_samples_0.97$sigma2, type = 'l', ylab="posterior sigma2",main="phi = 0.97")

plot(post_samples_0.3$phi, type = 'l', ylab="posterior phi",main="phi = 0.3")
plot(post_samples_0.97$phi, type = 'l', ylab="posterior phi",main="phi = 0.97")

par(mfrow = c(1,2))

plot(post_samples_0.3$mu, post_samples_0.3$phi, type = 'p',
    xlab = "mu", ylab = "phi", main = "phi = 0.3")
plot(post_samples_0.97$mu, post_samples_0.97$phi, type = 'p',
    xlab = "mu", ylab = "phi", main = "phi = 0.97")
```