Bayesian Lab3

Dinesh Sundaramoorthy (dinsu875) and Umamaheswarababu (umama339)

Question 1: Gibbs sampler for a normal model

The dataset *Precipitation.rds* consists of daily records of weather with rain or snow (in units of mm) from the beginning of 1962 to the end of 2008 in a certain area. Assume the natural log of the daily precipitation $y_1, \ldots, y_n$ to be independent normally distributed, $lny_1, \ldots, lny_n|\mu, \sigma^2 \sim N(\mu, \sigma^2)$ where both μ and σ^2 are unknown. Let $\mu \sim N(\mu_0, \tau_0^2)$ independently of $\sigma^2 \sim Inv - \chi^2(v_0, \sigma_0^2)$.

task a)

```
set.seed(12345)
data_precipitation<- readRDS("Precipitation.rds")
log_data_precipitation<-log(readRDS("Precipitation.rds"))
log_data_precipitation<-as.data.frame(log_data_precipitation)
names(log_data_precipitation)[1] <- "weather"
head(log_data_precipitation)
```

	weather <dbl>
1	-0.67727383
2	0.01587335
3	1.40216771
4	-1.37042101
5	-1.37042101
6	2.06356619

6 rows

```r
# mu
mu_not = 0
tau_sq_not = 1
# sigma_sq
nu_not = 1
sig_sq_not = 1 # sigma is 1/nu_not

## inverse chi square function
inv_chi_sq = function(n, df, sigma_sq) {
  return((df*sigma_sq)/rchisq(n,df=df))
}

#This is the Gibbs Sampler, which takes the number of draws, a default σ (as both σ and μ depend
on each
#other we need to start somewhere) and some more parameters to calculate the posterior parameter
s.

gibbs_sampler = function(nDraws, data, default_sigma, tau_sq_not, mu_not, nu_not,
                        sig_sq_not)
{
  # Posterior Parameters (Taken from lecture 2 slide 4)
  n = length(data)
  mu_n = mean(data) + mu_not
  nu_n = nu_not + n
  default_sigma_sq = default_sigma^2

  # To store all iterative results
  val_data_frame = data.frame(matrix(NA, nrow = nDraws, ncol = 2))

  # To save current iterative results
  cur_res = list(mu = NaN, sigma_sq = default_sigma_sq)

  for (i in 1:nDraws) {
    tau_sq_n = 1 / ((n/cur_res$sigma) + (1/tau_sq_not))
    cur_res$mu = rnorm(1, mu_n, sqrt(tau_sq_n))
    cur_res$sigma_sq = inv_chi_sq(1, nu_n,(nu_not*sig_sq_not + sum((data - cur_res$mu)^2))/(n +
nu_not))
    val_data_frame[i,] = cur_res
  }
  colnames(val_data_frame) = c("MU", "SIGMA_SQUARE")
  return(val_data_frame)

}

nDraws = 500

output = gibbs_sampler(nDraws = 500, #no of draws
                   data = log_data_precipitation$weather,
                   default_sigma = 40,
                   tau_sq_not = tau_sq_not,
                   mu_not = mu_not,
                   nu_not = nu_not,
```

```
                     sig_sq_not = sig_sq_not)
head(output)
```

	MU <dbl>	SIGMA_SQUARE <dbl>
1	1.709518	1.862084
2	1.304695	1.771905
3	1.380729	1.703340
4	1.299480	1.799974
5	1.294038	1.714163
6	1.299588	1.738917

6 rows

Evaluating the convergence of the Gibbs sampler by calculating the Inefficiency Factors (IFs).

The Inefficiency Factor (IF) and Effective Sample Size (ESS) are related measures used to evaluate the convergence of Markov chain Monte Carlo (MCMC) algorithms such as Gibbs sampling.

$$InefficiencyFactor(IF) = 1 + 2\sum_{k=1}^{\infty} \rho_k$$

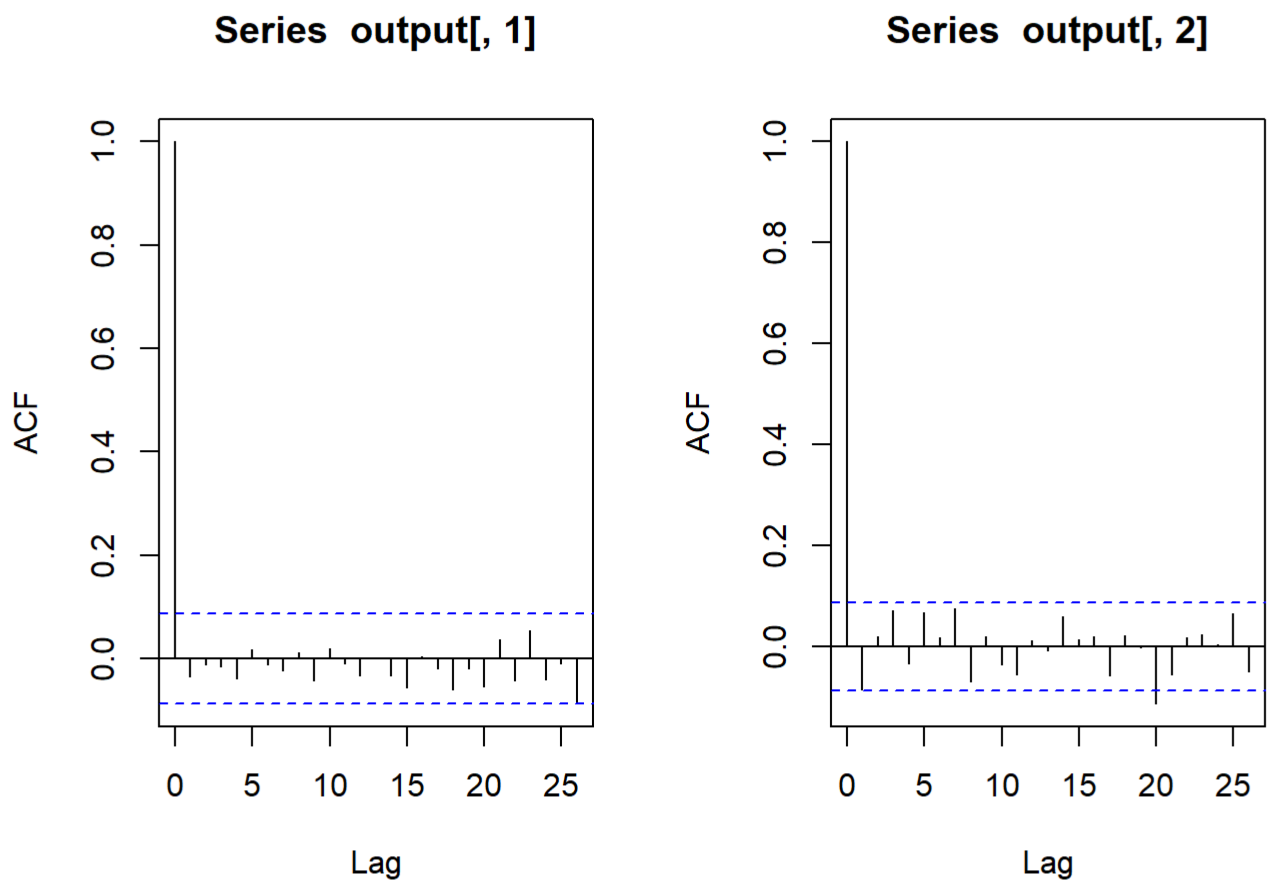
Where ρ_k is the autocorrelation at $log'k'$ and $nDraws \to \infty$

$$EffectiveSampleSize(ESS) = \frac{nDraws}{IF}$$

If the Effective sample size (ESS) is approximately equal to the number of samples drawn from the Gibbs sampler, it suggests that the Gibbs sampler has achieved good convergence.

Computing IF and ESS for ‘mu’ and ‘sigma’

```
# Computing Inefficiency Factor and Effective sample size

par(mfrow=c(1,2))

#ESS for mu
Gibbs_mu <- acf(output[,1])
IF_Gibbs_mu <- 1+2*sum(Gibbs_mu$acf[-1])
ESS_mu = nDraws/IF_Gibbs_mu

#ESS for sigma
Gibbs_sigma = acf(output[,2])
```




```
IF_Gibbs_sigma = 1 + 2*sum(Gibbs_sigma$acf[-1])
ESS_sigma = nDraws/IF_Gibbs_sigma

cat("ESS for 'mu':", ESS_mu, "\n")
```

```
## ESS for 'mu': -539429.1
```

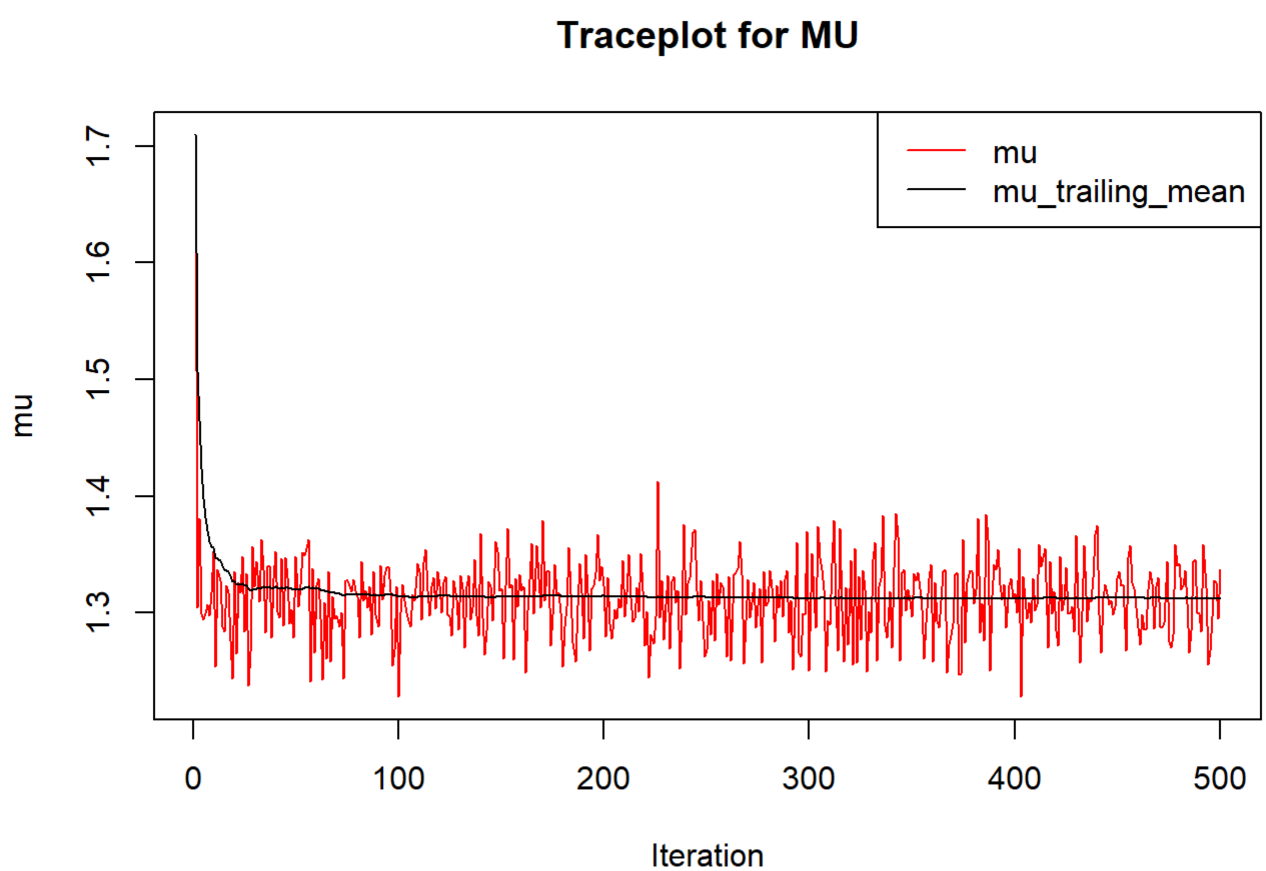
```
cat("ESS for 'sigma':", ESS_sigma)
```

```
## ESS for 'sigma': 548.7914
```

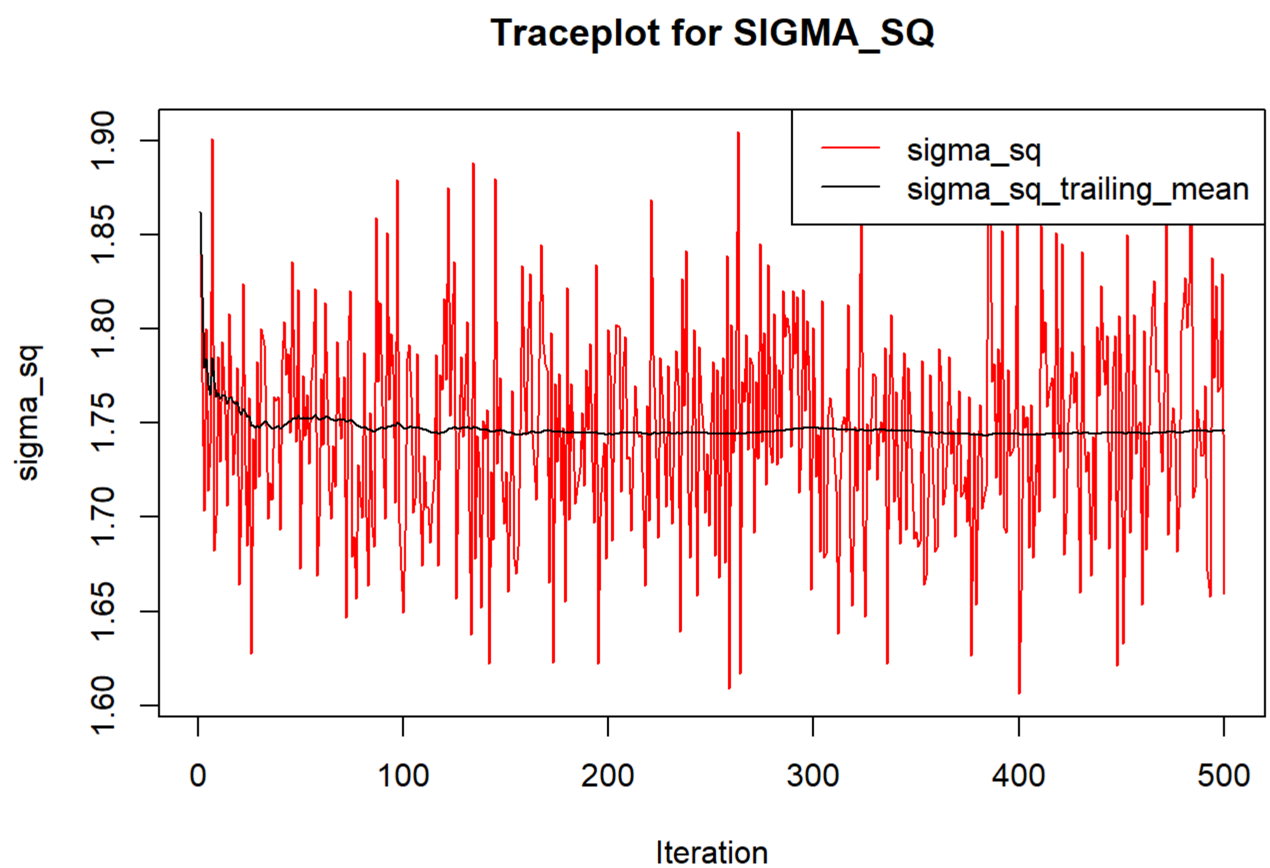
The ESS for for both ‘mu’ and ‘sigma’ roughly equal to the sample size (nDraws) so we can say that our Gibbs sampler has achieved the convergence.

Plotting the trajectories of the sampled Markov chains.

```
p1 = data.frame(1:nrow(output), output, cumsum(output$MU)/(1:nrow(output)),
                    cumsum(output$SIGMA_SQUARE)/(1:nrow(output)))
colnames(p1) = c("IDX", "MEAN", "SIGMA_SQUARE", "mu_MEAN",
                     "sigma_sq_SIGMA_SQUARE")

#plot for mu

plot(p1$IDX, p1$MEAN, type = "l", col = "red",
     main = "Traceplot for MU", ylab = "mu",
     xlab = "Iteration")
lines(p1$IDX, p1$mu_MEAN, col = "black")
legend("topright", legend = c("mu", "mu_trailing_mean"),
       col = c("red", "black"), lty = 1)
```



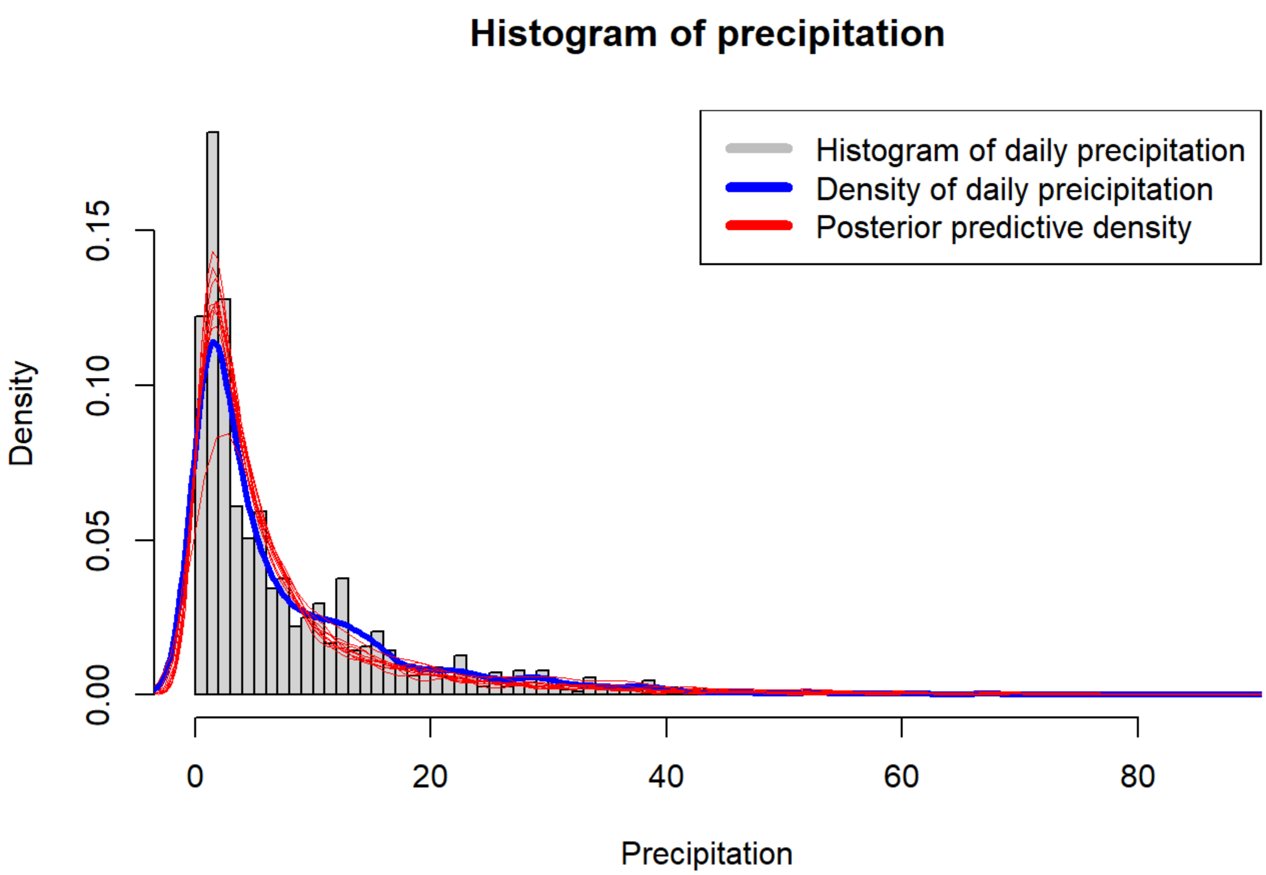

```
#plot for sigma

plot(p1$IDX, p1$SIGMA_SQUARE, type = "l", col = "red",
     main = "Traceplot for SIGMA_SQ", ylab = "sigma_sq",
     xlab = "Iteration")
lines(p1$IDX, p1$sigma_sq_SIGMA_SQUARE, col = "black")
legend("topright", legend = c("sigma_sq", "sigma_sq_trailing_mean"),
       col = c("red", "black"), lty = 1)
```



task b)

Plotting histogram of the daily precipitation and the resulting posterior predictive density

```r
# histogram of the daily precipitation
hist(data_precipitation, breaks=100, freq = FALSE, main = "Histogram of precipitation", xlab =
"Precipitation")

# plotting density of the daily precipitation
den <- density(data_precipitation)
lines(den, col="blue", lwd=3)

n=length(data_precipitation)

#posterior predictive precipitation using the simulated posterior draws from (a)
pred_den_y_hat = matrix(data = NA, nrow = length(data_precipitation), ncol = nDraws)
for(i in 1:nDraws){
  pred_den_y_hat[,i] = rlnorm(n, mean = output[i,1], sd = sqrt(output[i,2]))
}

#plotting density ofposterior predictive precipitation
for(i in seq(1,nDraws, 50)){
  den_pred_y = density(pred_den_y_hat[,i])
  lines(x = den_pred_y$x, y = den_pred_y$y, col='red', type = 'l', lwd = 0.1)
}
legend("topright", legend = c("Histogram of daily precipitation", "Density of daily preicipitati
on","Posterior predictive density"), col = c("grey", "blue","red"),lty=,lwd = 5)
```



From the plots we can observe that the shape of posterior density almost matches with the density of observed data.

Question 2. Metropolis Random Walk for Poisson regression.

Consider the following Poisson regression model $y_i|\beta \sim Poisson[exp(X_i^T\beta)]$, i = 1, …, n, where y_i is the count for the ith observation in the sample and xi is the p-dimensional vector with covariate observations for the ith observation. Use the data set eBayNumberOfBidderData.dat. This dataset contains observations from 1000 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):

task a)

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?

```
set.seed(12345)
data_ebay <- read.csv("eBayNumberOfBidderData.dat", sep="")

model <- glm(nBids~.+0, data = data_ebay, family = poisson)
summary(model)
```

```
##
## Call:
## glm(formula = nBids ~ . + 0, family = poisson, data = data_ebay)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.5800  -0.7222  -0.0441   0.5269   2.4605
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## Const        1.07244    0.03077  34.848  < 2e-16 ***
## PowerSeller -0.02054    0.03678  -0.558   0.5765
## VerifyID    -0.39452    0.09243  -4.268 1.97e-05 ***
## Sealed       0.44384    0.05056   8.778  < 2e-16 ***
## Minblem     -0.05220    0.06020  -0.867   0.3859
## MajBlem     -0.22087    0.09144  -2.416   0.0157 *
## LargNeg      0.07067    0.05633   1.255   0.2096
## LogBook     -0.12068    0.02896  -4.166 3.09e-05 ***
## MinBidShare -1.89410    0.07124 -26.588  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 6264.01  on 1000  degrees of freedom
## Residual deviance:  867.47  on  991  degrees of freedom
## AIC: 3610.3
##
## Number of Fisher Scoring iterations: 5
```

The significant terms are Const,VerifyID,Sealed,LogBook and MinBidshare.

task b)

Let’s now 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.).

```r
set.seed(12345)
library(mvtnorm)
# Parameters
Y = as.matrix(data_ebay[,1])
# We take all covariates
X = as.matrix(data_ebay[,-1])

# Feature names
col_names = colnames(data_ebay[,2:ncol(data_ebay)])
colnames(X) = col_names

# Defining the prior parameters
covariate_prior_sigma <- 100 * solve(t(X) %*% X)
prior_mu = rep(0, ncol(X))
N <- ncol(X)

posterior_log_likelihood <- function(beta,mu,sigma,Y,X){

  # loglikelihood of the poisson distribution
  llik = sum(Y * X %*% beta - exp(X %*% beta) - log(factorial(Y)))

  # if likelihood is very large or very small, stear optim away
  if (abs(llik) == Inf) llik = -100000;

  # log of the prior
  logPrior = dmvnorm(beta, mean = mu, sigma = sigma, log = TRUE)

  return(llik + logPrior)
}

# intialize
Beta_init_value <- as.vector(rep(0,N))

res = optim(par = Beta_init_value, fn = posterior_log_likelihood,Y = Y, X = X,
                  mu = prior_mu, sigma = covariate_prior_sigma, method=c("BFGS"),
                  control=list(fnscale=-1),
                  gr = NULL,
                  hessian=TRUE)


# variables with a specific names

post_mode = as.vector(res$par)
names(post_mode) = col_names
#print("The posterior mode is",post_mode)
post_covariance = - solve(res$hessian) #Because Posterior covariance matrix is -inv(Hessian)
#print("The posterior covariance is", post_covariance )
post_sd = sqrt(diag(post_covariance))
names(post_sd) = col_names
```

The posterior mode is given as:

```
post_mode
```

```
##       Const PowerSeller    VerifyID      Sealed     Minblem     MajBlem 
##  1.06984118 -0.02051246 -0.39300599  0.44355549 -0.05246627 -0.22123840 
##     LargNeg     LogBook MinBidShare 
##  0.07069683 -0.12021767 -1.89198501
```

The posterior covariance is given as:

```
post_covariance
```

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

The posterior standard deviation is given as:

```
post_sd
```

```
##       Const PowerSeller    VerifyID      Sealed     Minblem     MajBlem 
##  0.03074837  0.03678418  0.09227871  0.05057448  0.06020470  0.09146070 
##     LargNeg     LogBook MinBidShare 
##  0.05634767  0.02895635  0.07109682
```

task c)

Now, let's simulate from the actual posterior of β using the Metropolis algorithm and compare 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, I will denote the vector of model parameters by θ. Let the proposal density be the multivariate normal density mentioned in Lecture 8 (random walk Metropolis):

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

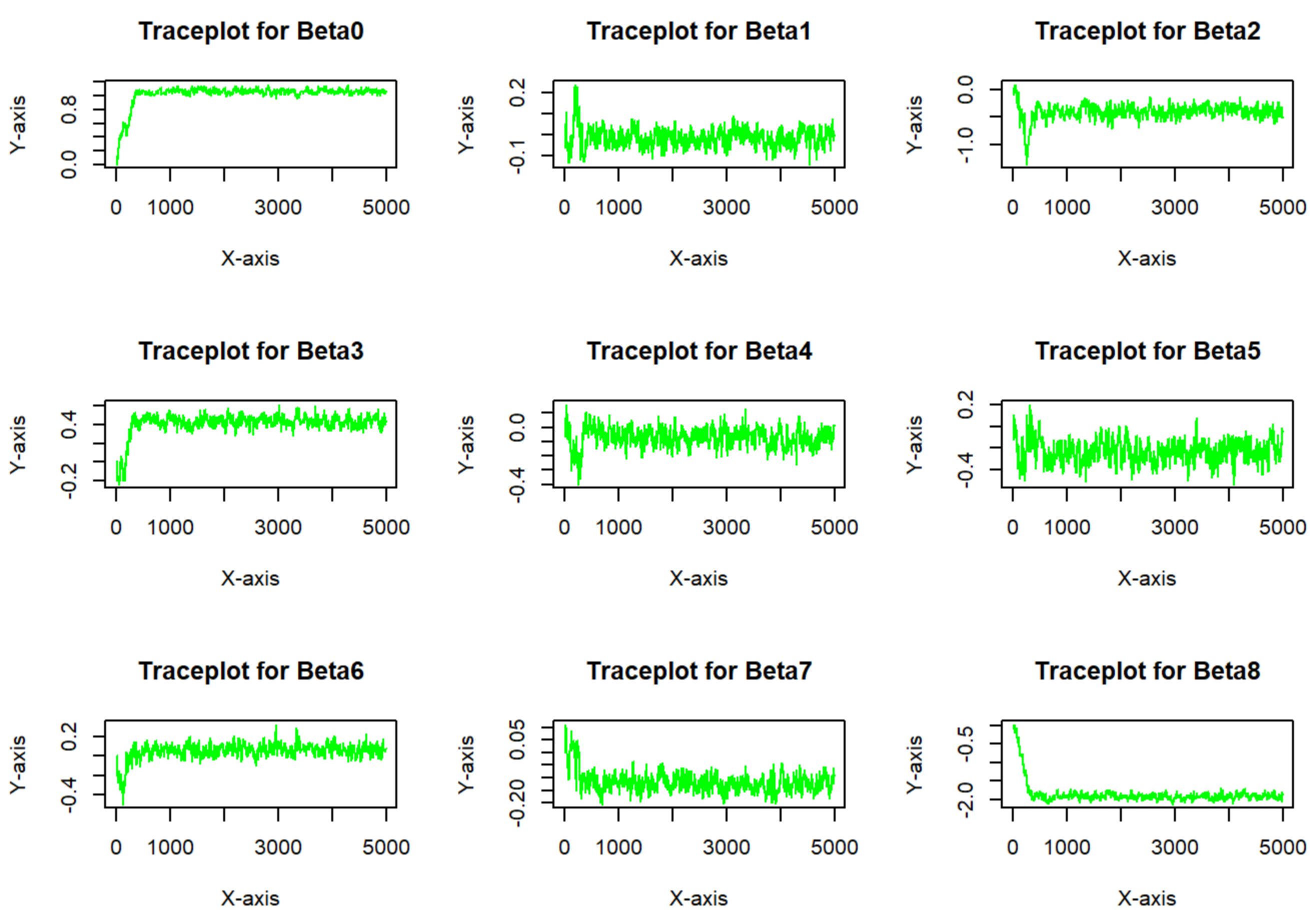
,

where $\sum = J_y^{-}1(\tilde{\beta})$ 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 and the triple dot (…) wildcard argument. I have posted a note (HowToCodeRWM.pdf) on the course web page that describes how to 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.

```r
set.seed(12345)
library(MASS) # for mvrnorm
# required input
covariate_prior_sigma <- 100 * solve(t(X) %*% X)
post_covariance <- - solve(res$hessian)
prior_mu = rep(0, ncol(X))
c = 0.5
n_draws = 5000
init_beta = rep(0, ncol(X))

RWMSampler = function(Posterior_log,theta,c,post_covariance, ... ){

  # initialize to store theta draws
  thetas = matrix(0, n_draws, length(theta))

  # start with initial value for beta
  theta_new = theta
  for(i in 1:n_draws){
    thetas[i,] = theta_new
    # Draw new theta for the proposal depending on the previous one
    theta_prop = mvrnorm(1, theta_new, c * post_covariance)
    # acceptance probability
    acceptance_prob = exp(Posterior_log(theta_prop,...)- Posterior_log(theta_new,...))
    # if the acceptance probability (i.e the ratio) < 1
    u = runif(n = 1, min = 0, max = 1)
    if( u < acceptance_prob){
      theta_new = theta_prop
    }else{
      theta_new = theta_new
    }
  }
  return(thetas)
}
drawn_res= RWMSampler(posterior_log_likelihood, init_beta,c, post_covariance,prior_mu, covariate
_prior_sigma,Y,X)

# acceptance probability
burnin_val = 1000
Acceptance = 1-mean(duplicated(drawn_res[-(1:burnin_val),]))
Acceptance
```

```
## [1] 0.327
```

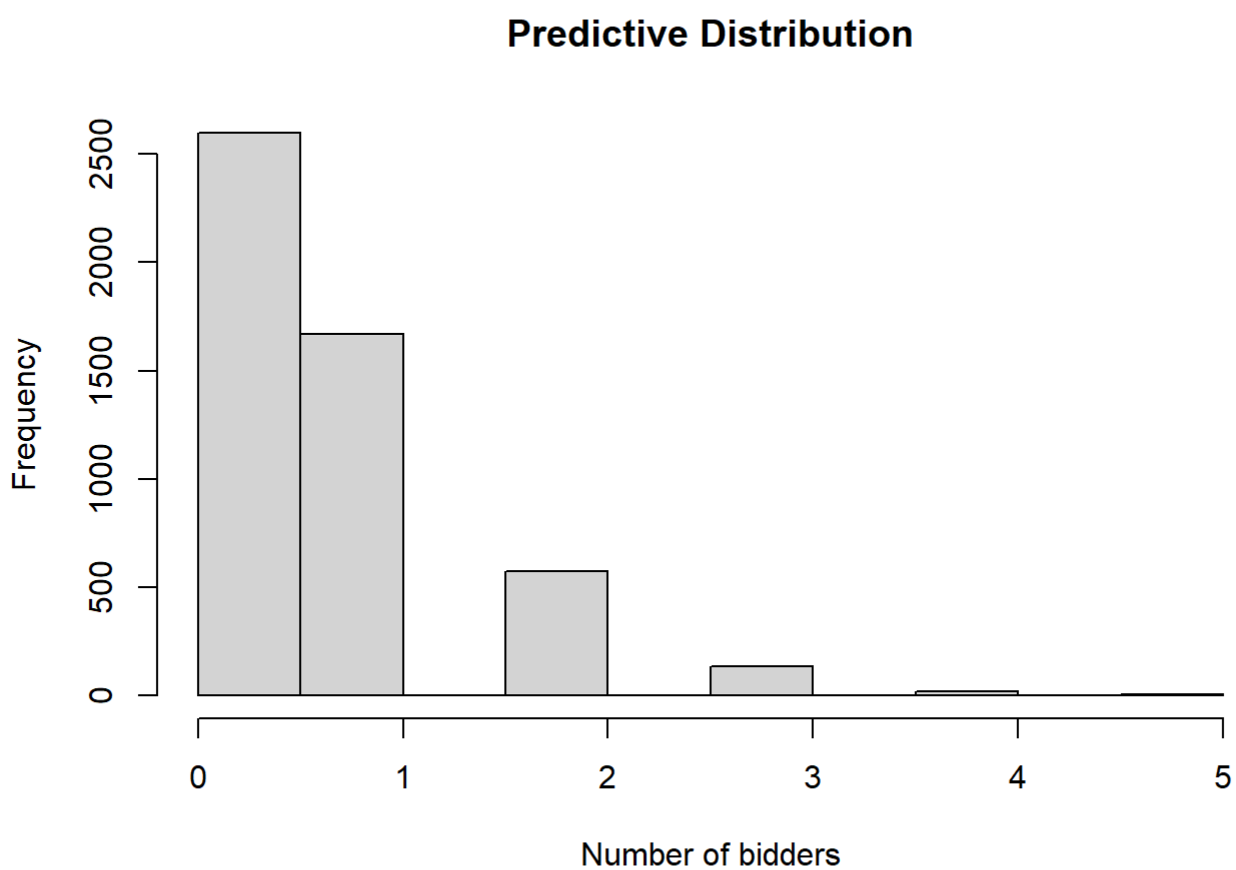
```
# Set the layout to three rows
par(mfrow = c(3,3))

# Create and display the plots
plot(1:5000, drawn_res[,1], type = "l", col = "green", main = "Traceplot for Beta0", xlab = "X-axis", ylab = "Y-axis")
plot(1:5000, drawn_res[,2], type = "l", col = "green", main = "Traceplot for Beta1", xlab = "X-axis", ylab = "Y-axis")
plot(1:5000, drawn_res[,3], type = "l", col = "green", main = "Traceplot for Beta2", xlab = "X-axis", ylab = "Y-axis")
plot(1:5000, drawn_res[,4], type = "l", col = "green", main = "Traceplot for Beta3", xlab = "X-axis", ylab = "Y-axis")
plot(1:5000, drawn_res[,5], type = "l", col = "green", main = "Traceplot for Beta4", xlab = "X-axis", ylab = "Y-axis")
plot(1:5000, drawn_res[,6], type = "l", col = "green", main = "Traceplot for Beta5", xlab = "X-axis", ylab = "Y-axis")
plot(1:5000, drawn_res[,7], type = "l", col = "green", main = "Traceplot for Beta6", xlab = "X-axis", ylab = "Y-axis")
plot(1:5000, drawn_res[,8], type = "l", col = "green", main = "Traceplot for Beta7", xlab = "X-axis", ylab = "Y-axis")
plot(1:5000, drawn_res[,9], type = "l", col = "green", main = "Traceplot for Beta8", xlab = "X-axis", ylab = "Y-axis")
```



task d)

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?

```
# New Auction data
x_new <- as.vector(c(1,1,0,1,0,1,0,1.2,0.8))
betas <- drawn_res # using draws from c)
lambda <- exp(betas%*%x_new)

#prediction
y_pred = rpois(n=length(lambda), lambda = lambda)

#plotting histogram of predictive distribution of new auction
par(mfrow=c(1,1))
hist(y_pred,xlab="Number of bidders",main="Predictive Distribution")
```




```
Pr_no_bid <-mean(y_pred==0) #Pr(nBids = 0)
```

```
cat("The probability of no bidders in the new auction: ", Pr_no_bid,"\n")
```

```
## The probability of no bidders in the new auction:  0.5192
```

3. Time series models in Stan

Task(a)

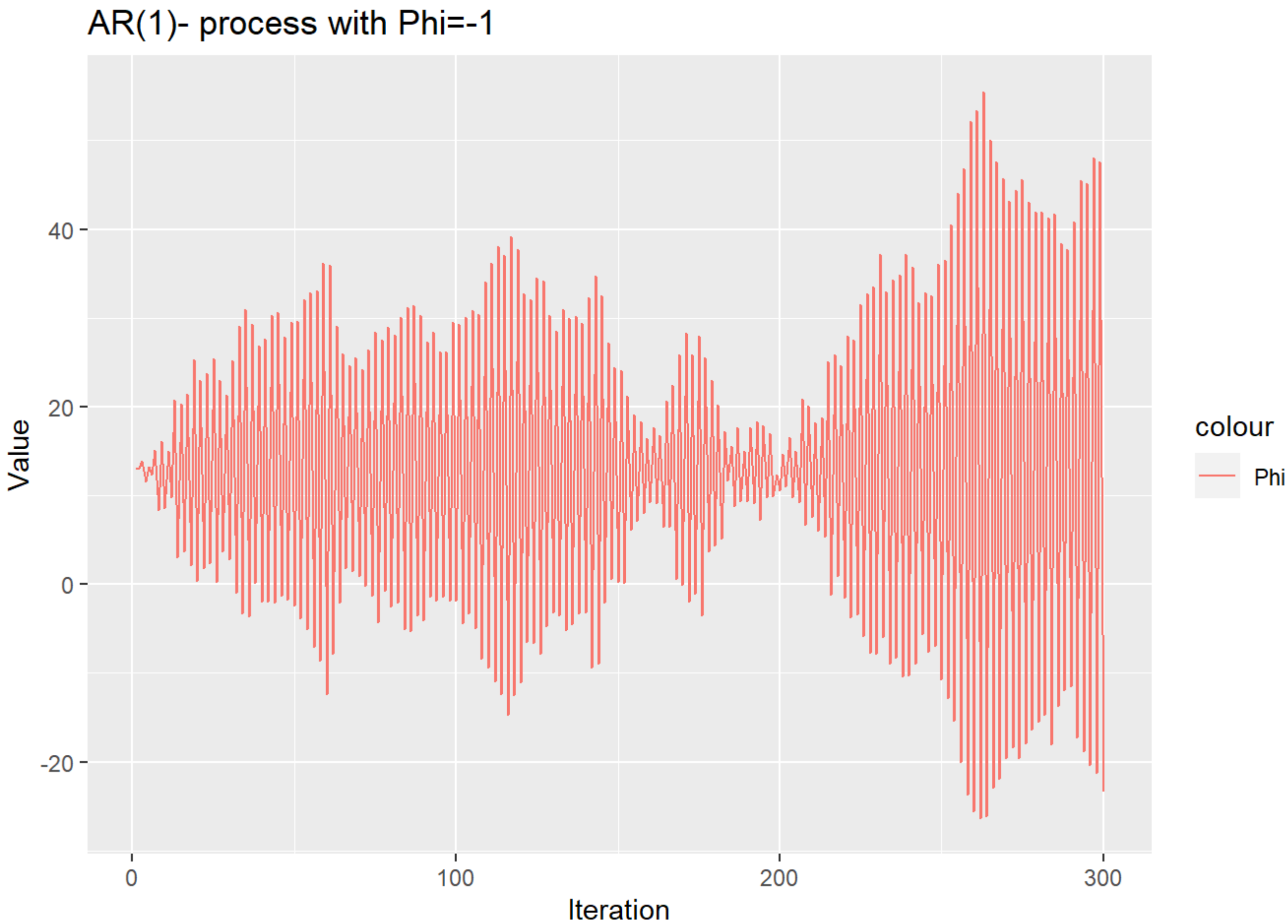
Write a function in R that simulates data from the AR(1)-process

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

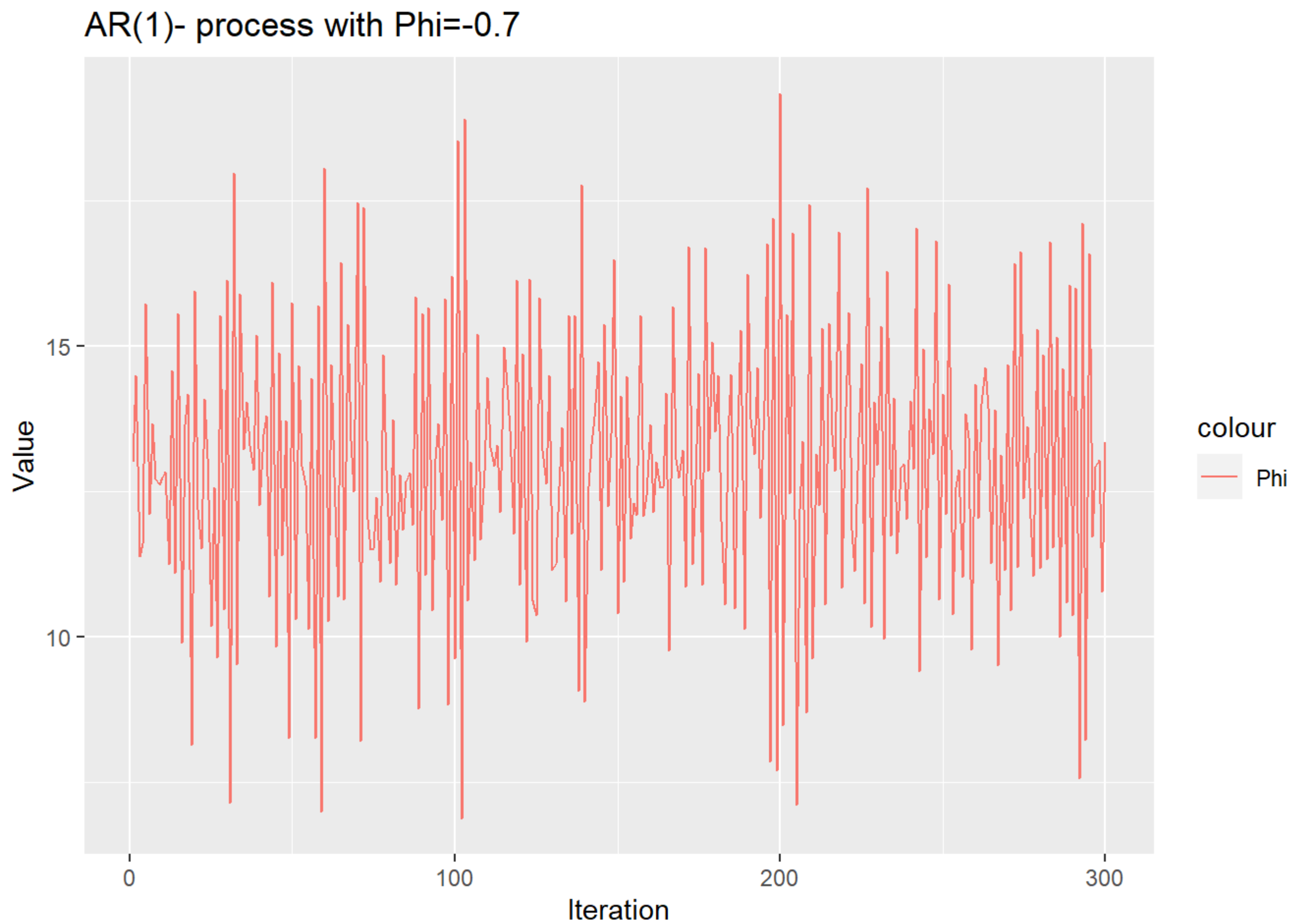
for given values of $\mu, \phi \ and \ \sigma^2$. Start the process at $x_1 = \mu$ and then simulate values for x_t for $t = 2, 3, \ldots, T$ and return the vector $x_{1:T}$ containing all time points. Use $\mu = 13, \sigma^2 = 3 \ and \ T = 300$ and look at some different realizations (simulations) of $x_{1:T}$ for values of ϕ between -1 and 1 (this is the interval of ϕ where the AR(1)-process is stable). Include a plot of at least one realization in the report. What effect does the value of ϕ have on $x_{1:T}$?

```r
set.seed(1234)
# inputs needed
#Mu = 13
#sigma2 = 3
#nDraws = 300
#phi = 1

# Gibbs sampling

AR = function(Mu, sigma2, phi, nDraws){
  x_previous = Mu
  # storing of previous time
  gibbs_nDraws = c(x_previous)

  for (i in 2:nDraws){

    # Updating the time given previous time
    epsilon = rnorm(1,0,sqrt(sigma2))
    x_new <- Mu + phi * (x_previous-Mu) +epsilon
    x_previous = x_new
    gibbs_nDraws = c(gibbs_nDraws, x_new)
  }
  return(gibbs_nDraws)

}
# phi = 1
# AR(13, 3, 1, 300)

# testing the different values of Phi -> i.e, [-1,1]
Phi_seq = seq(-1,1,0.2)

PhiDraws = matrix(0,300, length(Phi_seq))

for(j in Phi_seq){
  PhiDraws[,j]= AR(13,3,j,300)
}


# plot function for different values of phi

library(ggplot2)
# T=300
n_draws= 1:300


ggplot_func = function(phi){

  ggplot()+geom_line(aes(x = n_draws ,y= AR(13, 3, phi, 300), color = "Phi"))+
    labs(x = "Iteration", y = "Value")
}

par(mfrow = c(3, 2))
```

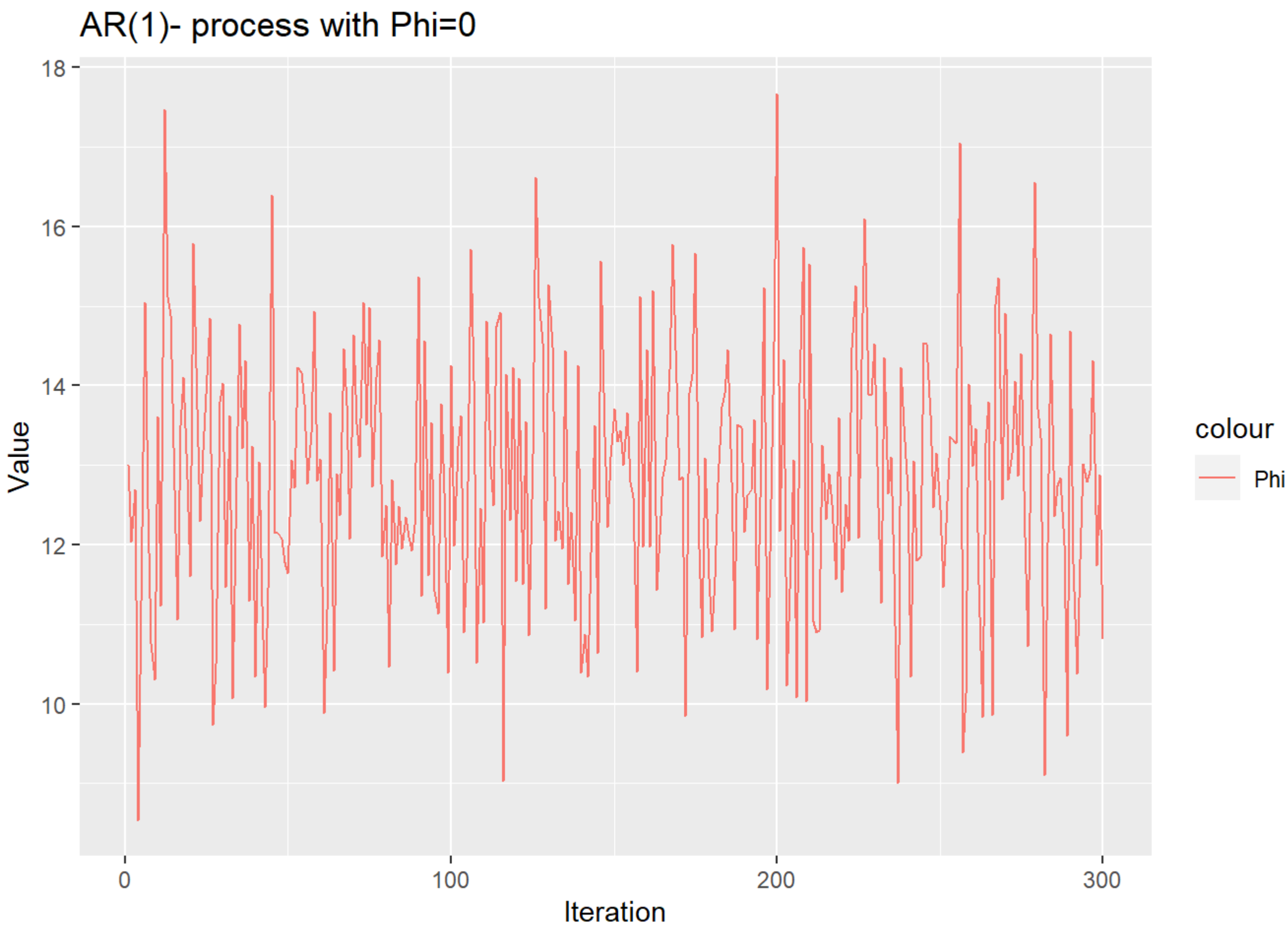
```
# Create and display the plots
p1 = ggplot_func(-1) +ggtitle("AR(1)- process with Phi=-1")
p1
```



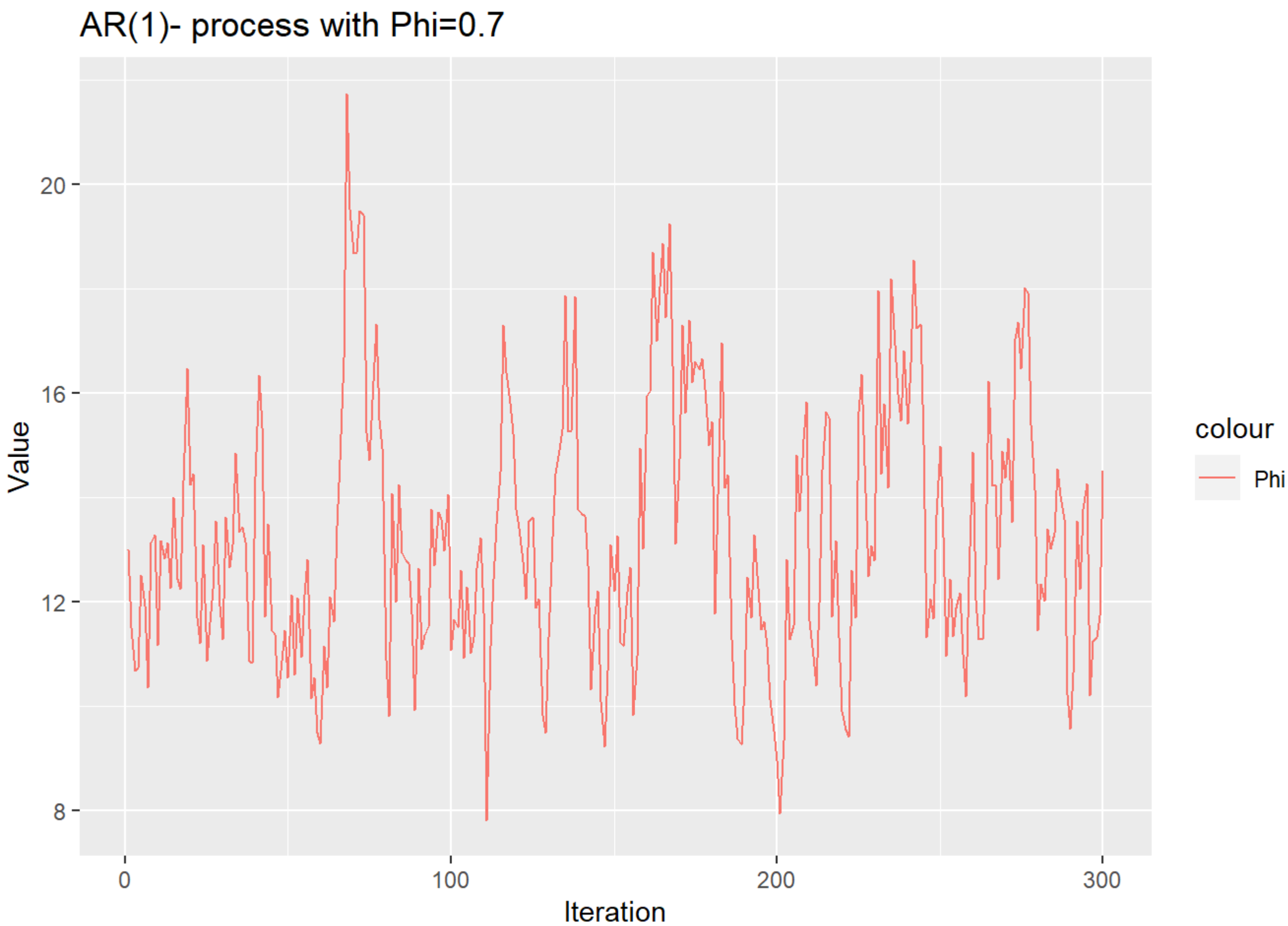

```
p2 = ggplot_func(-0.7) +ggtitle("AR(1)- process with Phi=-0.7")
p2
```



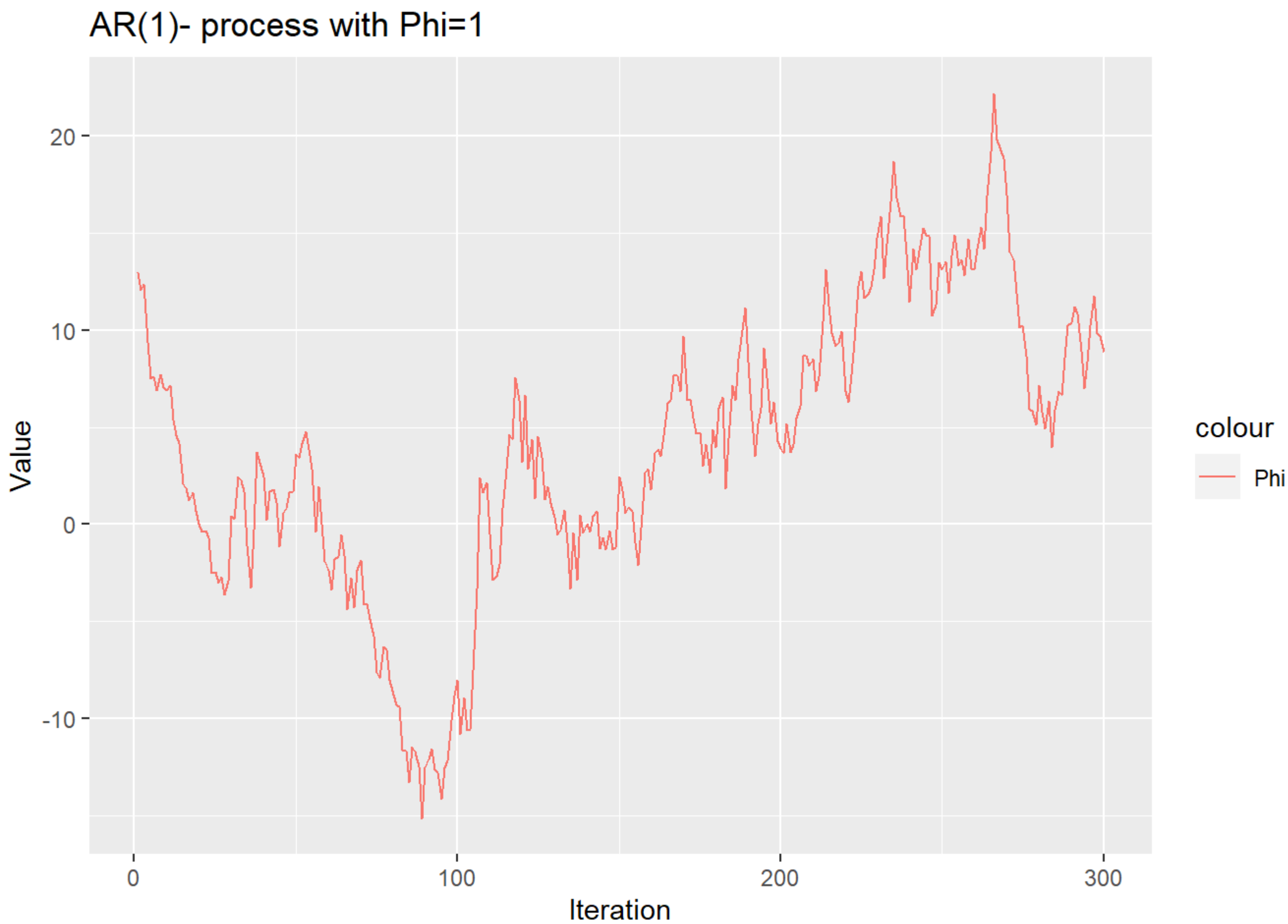

```
p3 = ggplot_func(0) +ggtitle("AR(1)- process with Phi=0")
p3
```




```
p4 = ggplot_func(0.7) +ggtitle("AR(1)- process with Phi=0.7")
p4
```




```
p5 = ggplot_func(1) +ggtitle("AR(1)- process with Phi=1")
p5
```



After testing various values for ϕ, we can observe from the previous plots that the amplitude of our draws decreases. This reduction in amplitude can be attributed to the influence of the phi values in the AR formula, $\phi(X_{t-1} - \mu)$. When ϕ is negative, it tends to pull $(X_{t-1} - \mu)$ in the opposite direction, thereby diminishing the oscillation. As ϕ increases, the impact in the opposite direction becomes less pronounced.

Source Code

```
knitr::opts_chunk$set(echo = TRUE)
set.seed(12345)
data_precipitation<- readRDS("Precipitation.rds")
log_data_precipitation<-log(readRDS("Precipitation.rds"))
log_data_precipitation<-as.data.frame(log_data_precipitation)
names(log_data_precipitation)[1] <- "weather"
head(log_data_precipitation)



# mu
mu_not = 0
tau_sq_not = 1
# sigma_sq
nu_not = 1
sig_sq_not = 1 # sigma is 1/nu_not

## inverse chi square function
inv_chi_sq = function(n, df, sigma_sq) {
  return((df*sigma_sq)/rchisq(n,df=df))
}

#This is the Gibbs Sampler, which takes the number of draws, a default σ (as both σ and μ depend
on each
#other we need to start somewhere) and some more parameters to calculate the posterior parameter
s.

gibbs_sampler = function(nDraws, data, default_sigma, tau_sq_not, mu_not, nu_not,
                        sig_sq_not)
{
  # Posterior Parameters (Taken from lecture 2 slide 4)
  n = length(data)
  mu_n = mean(data) + mu_not
  nu_n = nu_not + n
  default_sigma_sq = default_sigma^2

  # To store all iterative results
  val_data_frame = data.frame(matrix(NA, nrow = nDraws, ncol = 2))

  # To save current iterative results
  cur_res = list(mu = NaN, sigma_sq = default_sigma_sq)

  for (i in 1:nDraws) {
    tau_sq_n = 1 / ((n/cur_res$sigma) + (1/tau_sq_not))
    cur_res$mu = rnorm(1, mu_n, sqrt(tau_sq_n))
    cur_res$sigma_sq = inv_chi_sq(1, nu_n,(nu_not*sig_sq_not + sum((data - cur_res$mu)^2))/(n +
nu_not))
    val_data_frame[i,] = cur_res
  }
  colnames(val_data_frame) = c("MU", "SIGMA_SQUARE")
  return(val_data_frame)

}
```

```
nDraws = 500

output = gibbs_sampler(nDraws = 500, #no of draws
                   data = log_data_precipitation$weather,
                   default_sigma = 40,
                   tau_sq_not = tau_sq_not,
                   mu_not = mu_not,
                   nu_not = nu_not,
                   sig_sq_not = sig_sq_not)
head(output)
# Computing Inefficiency Factor and Effective sample size

par(mfrow=c(1,2))

#ESS for mu
Gibbs_mu <- acf(output[,1])
IF_Gibbs_mu <- 1+2*sum(Gibbs_mu$acf[-1])
ESS_mu = nDraws/IF_Gibbs_mu

#ESS for sigma
Gibbs_sigma = acf(output[,2])
IF_Gibbs_sigma = 1 + 2*sum(Gibbs_sigma$acf[-1])
ESS_sigma = nDraws/IF_Gibbs_sigma

cat("ESS for 'mu':", ESS_mu, "\n")
cat("ESS for 'sigma':", ESS_sigma)

p1 = data.frame(1:nrow(output), output, cumsum(output$MU)/(1:nrow(output)),
                    cumsum(output$SIGMA_SQUARE)/(1:nrow(output)))
colnames(p1) = c("IDX", "MEAN", "SIGMA_SQUARE", "mu_MEAN",
                     "sigma_sq_SIGMA_SQUARE")

#plot for mu

plot(p1$IDX, p1$MEAN, type = "l", col = "red",
     main = "Traceplot for MU", ylab = "mu",
     xlab = "Iteration")
lines(p1$IDX, p1$mu_MEAN, col = "black")
legend("topright", legend = c("mu", "mu_trailing_mean"),
       col = c("red", "black"), lty = 1)

#plot for sigma

plot(p1$IDX, p1$SIGMA_SQUARE, type = "l", col = "red",
     main = "Traceplot for SIGMA_SQ", ylab = "sigma_sq",
     xlab = "Iteration")
lines(p1$IDX, p1$sigma_sq_SIGMA_SQUARE, col = "black")
legend("topright", legend = c("sigma_sq", "sigma_sq_trailing_mean"),
       col = c("red", "black"), lty = 1)
# histogram of the daily precipitation
hist(data_precipitation, breaks=100, freq = FALSE, main = "Histogram of precipitation", xlab =
```

```r
"Precipitation")

# plotting density of the daily precipitation
den <- density(data_precipitation)
lines(den, col="blue", lwd=3)

n=length(data_precipitation)

#posterior predictive precipitation using the simulated posterior draws from (a)
pred_den_y_hat = matrix(data = NA, nrow = length(data_precipitation), ncol = nDraws)
for(i in 1:nDraws){
  pred_den_y_hat[,i] = rlnorm(n, mean = output[i,1], sd = sqrt(output[i,2]))
}

#plotting density ofposterior predictive precipitation
for(i in seq(1,nDraws, 50)){
  den_pred_y = density(pred_den_y_hat[,i])
  lines(x = den_pred_y$x, y = den_pred_y$y, col='red', type = 'l', lwd = 0.1)
}
legend("topright", legend = c("Histogram of daily precipitation", "Density of daily preicipitati
on","Posterior predictive density"), col = c("grey", "blue","red"),lty=,lwd = 5)
set.seed(12345)
data_ebay <- read.csv("eBayNumberOfBidderData.dat", sep="")

model <- glm(nBids~.+0, data = data_ebay, family = poisson)
summary(model)
set.seed(12345)
library(mvtnorm)
# Parameters
Y = as.matrix(data_ebay[,1])
# We take all covariates
X = as.matrix(data_ebay[,-1])

# Feature names
col_names = colnames(data_ebay[,2:ncol(data_ebay)])
colnames(X) = col_names

# Defining the prior parameters
covariate_prior_sigma <- 100 * solve(t(X) %*% X)
prior_mu = rep(0, ncol(X))
N <- ncol(X)

posterior_log_likelihood <- function(beta,mu,sigma,Y,X){

  # loglikelihood of the poisson distribution
  llik = sum(Y * X %*% beta - exp(X %*% beta) - log(factorial(Y)))

  # if likelihood is very large or very small, stear optim away
  if (abs(llik) == Inf) llik = -100000;

  # log of the prior
  logPrior = dmvnorm(beta, mean = mu, sigma = sigma, log = TRUE)
```

```
  return(llik + logPrior)
}

# intialize
Beta_init_value <- as.vector(rep(0,N))

res = optim(par = Beta_init_value, fn = posterior_log_likelihood,Y = Y, X = X,
                  mu = prior_mu, sigma = covariate_prior_sigma, method=c("BFGS"),
                  control=list(fnscale=-1),
                  gr = NULL,
                  hessian=TRUE)


# variables with a specific names

post_mode = as.vector(res$par)
names(post_mode) = col_names
#print("The posterior mode is",post_mode)
post_covariance = - solve(res$hessian) #Because Posterior covariance matrix is -inv(Hessian)
#print("The posterior covariance is", post_covariance )
post_sd = sqrt(diag(post_covariance))
names(post_sd) = col_names

post_mode
post_covariance
post_sd
set.seed(12345)
library(MASS) # for mvrnorm
# required input
covariate_prior_sigma <- 100 * solve(t(X) %*% X)
post_covariance <- - solve(res$hessian)
prior_mu = rep(0, ncol(X))
c = 0.5
n_draws = 5000
init_beta = rep(0, ncol(X))

RWMSampler = function(Posterior_log,theta,c,post_covariance, ... ){

  # initialize to store theta draws
  thetas = matrix(0, n_draws, length(theta))

  # start with initial value for beta
  theta_new = theta
  for(i in 1:n_draws){
    thetas[i,] = theta_new
    # Draw new theta for the proposal depending on the previous one
    theta_prop = mvrnorm(1, theta_new, c * post_covariance)
    # acceptance probability
    acceptance_prob = exp(Posterior_log(theta_prop,...)- Posterior_log(theta_new,...))
    # if the acceptance probability (i.e the ratio) < 1
    u = runif(n = 1, min = 0, max = 1)
```

```
    if( u < acceptance_prob){
      theta_new = theta_prop
    }else{
      theta_new = theta_new
    }
  }
  return(thetas)
}
drawn_res= RWMSampler(posterior_log_likelihood, init_beta,c, post_covariance,prior_mu, covariate
_prior_sigma,Y,X)

# acceptance probability
burnin_val = 1000
Acceptance = 1-mean(duplicated(drawn_res[-(1:burnin_val),]))
Acceptance

# Set the layout to three rows
par(mfrow = c(3,3))

# Create and display the plots
plot(1:5000, drawn_res[,1], type = "l", col = "green", main = "Traceplot for Beta0", xlab = "X-a
xis", ylab = "Y-axis")
plot(1:5000, drawn_res[,2], type = "l", col = "green", main = "Traceplot for Beta1", xlab = "X-a
xis", ylab = "Y-axis")
plot(1:5000, drawn_res[,3], type = "l", col = "green", main = "Traceplot for Beta2", xlab = "X-a
xis", ylab = "Y-axis")
plot(1:5000, drawn_res[,4], type = "l", col = "green", main = "Traceplot for Beta3", xlab = "X-a
xis", ylab = "Y-axis")
plot(1:5000, drawn_res[,5], type = "l", col = "green", main = "Traceplot for Beta4", xlab = "X-a
xis", ylab = "Y-axis")
plot(1:5000, drawn_res[,6], type = "l", col = "green", main = "Traceplot for Beta5", xlab = "X-a
xis", ylab = "Y-axis")
plot(1:5000, drawn_res[,7], type = "l", col = "green", main = "Traceplot for Beta6", xlab = "X-a
xis", ylab = "Y-axis")
plot(1:5000, drawn_res[,8], type = "l", col = "green", main = "Traceplot for Beta7", xlab = "X-a
xis", ylab = "Y-axis")
plot(1:5000, drawn_res[,9], type = "l", col = "green", main = "Traceplot for Beta8", xlab = "X-a
xis", ylab = "Y-axis")

# New Auction data
x_new <- as.vector(c(1,1,0,1,0,1,0,1.2,0.8))
betas <- drawn_res # using draws from c)
lambda <- exp(betas%*%x_new)

#prediction
y_pred = rpois(n=length(lambda), lambda = lambda)

#plotting histogram of predictive distribution of new auction
par(mfrow=c(1,1))
hist(y_pred,xlab="Number of bidders",main="Predictive Distribution")
Pr_no_bid <-mean(y_pred==0) #Pr(nBids = 0)
cat("The probability of no bidders in the new auction: ", Pr_no_bid,"\n")
```

```r
set.seed(1234)
# inputs needed
#Mu = 13
#sigma2 = 3
#nDraws = 300
#phi = 1

# Gibbs sampling

AR = function(Mu, sigma2, phi, nDraws){
  x_previous = Mu
  # storing of previous time
  gibbs_nDraws = c(x_previous)

  for (i in 2:nDraws){

    # Updating the time given previous time
    epsilon = rnorm(1,0,sqrt(sigma2))
    x_new <- Mu + phi * (x_previous-Mu) +epsilon
    x_previous = x_new
    gibbs_nDraws = c(gibbs_nDraws, x_new)
  }
  return(gibbs_nDraws)

}
# phi = 1
# AR(13, 3, 1, 300)

# testing the different values of Phi -> i.e, [-1,1]
Phi_seq = seq(-1,1,0.2)

PhiDraws = matrix(0,300, length(Phi_seq))

for(j in Phi_seq){
  PhiDraws[,j]= AR(13,3,j,300)
}


# plot function for different values of phi

library(ggplot2)
# T=300
n_draws= 1:300


ggplot_func = function(phi){

  ggplot()+geom_line(aes(x = n_draws ,y= AR(13, 3, phi, 300), color = "Phi"))+
    labs(x = "Iteration", y = "Value")
}

par(mfrow = c(3, 2))
```

```
# Create and display the plots
p1 = ggplot_func(-1) +ggtitle("AR(1)- process with Phi=-1")
p1

p2 = ggplot_func(-0.7) +ggtitle("AR(1)- process with Phi=-0.7")
p2

p3 = ggplot_func(0) +ggtitle("AR(1)- process with Phi=0")
p3

p4 = ggplot_func(0.7) +ggtitle("AR(1)- process with Phi=0.7")
p4

p5 = ggplot_func(1) +ggtitle("AR(1)- process with Phi=1")
p5
```