Lab3 report

Yi Hung Chen, Jonathan Dorairaj

2023-05-14

Q1 Gibbs sampler for a normal model

(a Implement (code!) a Gibbs sampler that simulates from the joint posterior

```
#####Q1
library(ggplot2)
library(mvtnorm)
#library(extraDistr)
precipitation <- readRDS("Precipitation.rds")
log_prec <- log(precipitation)

mu_post <- function(mu_0,tau_0_sqr,sigma_square_current,y,n){

  tau_n_sqr = 1 / (1/tau_0_sqr + n/sigma_square_current)
  mu_n <- tau_n_sqr * (mu_0/tau_0_sqr + sum(y)/sigma_square_current)
  mupost <- rnorm(1, mu_n, sqrt(tau_n_sqr))
  return(mupost)

}

sigma_square_post <- function(mu_current, v_0, sigma_square_0, y,n, use_myinvchi=TRUE) {

  v_n <- v_0 + n
  elem1 <- v_0*sigma_square_0
  elem2 <- sum((y - mu_current)^2)
  elem3 <- n+v_0
  elem_comb <- (elem1+elem2)/elem3
  if (use_myinvchi){
    sigma_square_post <- my_inv_chi(v_n,elem_comb)
  }
  else {
    sigma_square_post <- rinvchisq(1, v_n, elem_comb) #Requires package extraDistr
  }



  return(sigma_square_post)
}

my_inv_chi<- function(df,tau_sqr) {
  X <- rchisq(1,df)
```

```
  inv_chi <- (df*tau_sqr)/X
  return(inv_chi)
}

#init
mu_0 <- 0
tau_0_sqr <- 1
sigma_square_0 <- 1
v_0 <- 1 #degree of freedom for chi square

gibbs_sampler <- function(nstep, data, mu_0, tau_0_sqr, v_0, sigma_square_0) {
  # Init parameters
  mu_current <- 0
  sigma_square_current <- 1

  mu_samples <- rep(0,nstep)
  sigma_square_samples <- rep(0,nstep)

  for (i in 1:nstep) {

    mu_current <- mu_post(mu_0, tau_0_sqr, sigma_square_current, y=data, length(data))
    #print(mu_current)
    sigma_square_current <- sigma_square_post(mu_current, v_0, sigma_square_0, y=data,
                                              length(data),use_myinvchi=TRUE)
    #print(sigma_square_current)

    mu_samples[i] <- mu_current
    sigma_square_samples[i] <- sigma_square_current
  }

  output_df <- data.frame(mu_sample = mu_samples, sigma_sample = sigma_square_samples)
  return(output_df)
}

sample_gibbs <- gibbs_sampler(nstep=10000, data=log_prec, mu_0, tau_0_sqr, v_0, sigma_square_0)
```

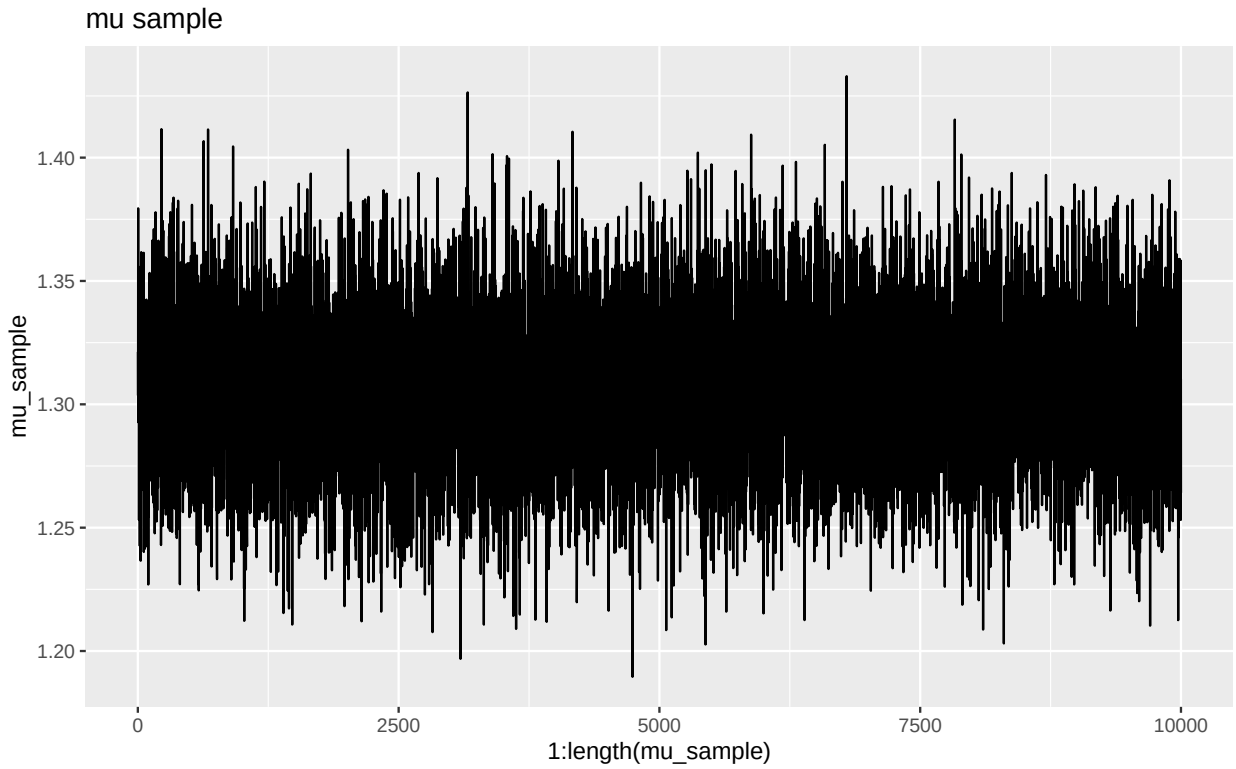
In lecture slide$IF = 1 + 2\sum_{k=1}^{\infty} \rho_k$ where ρ_k is autocorrelation at lag k

```
my_acf <- acf(sample_gibbs$mu_sample,plot = F) # getting the autocorrelation
if_mu <- 1 + 2 * sum(my_acf$acf[-1])
my_acf <- acf(sample_gibbs$sigma_sample,plot = F)
if_sigma <- 1 + 2 * sum(my_acf$acf[-1])
```

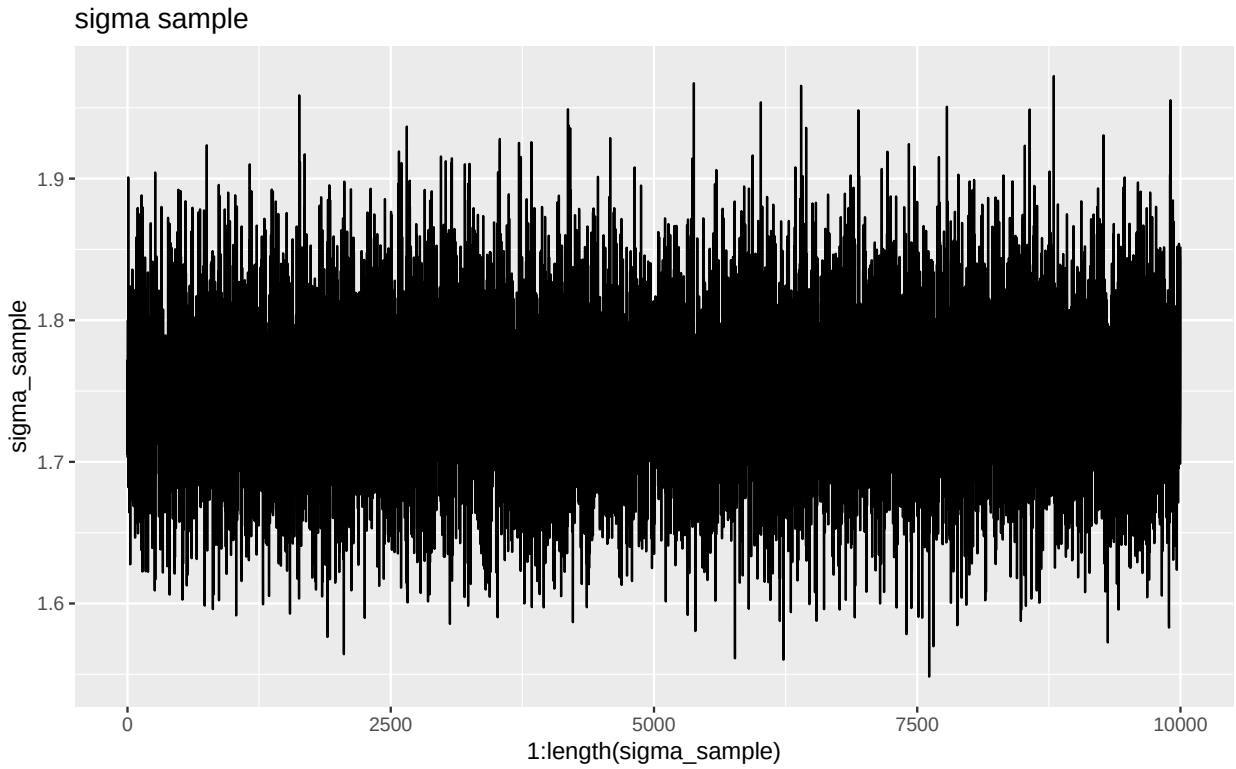
The the Inefficiency Factors (IF) of $\mu = 0.9982076$ and IF of $\sigma = 0.9522093$. The value of IF is around 1 indicates that the chain has converged to the stationary distribution.

Below, we present two trace plot that also confirm the samples' convergence.

```
ggplot(data=sample_gibbs, aes(x = 1:length(mu_sample), y = mu_sample)) +
  geom_line()+labs(title = "mu sample")
```



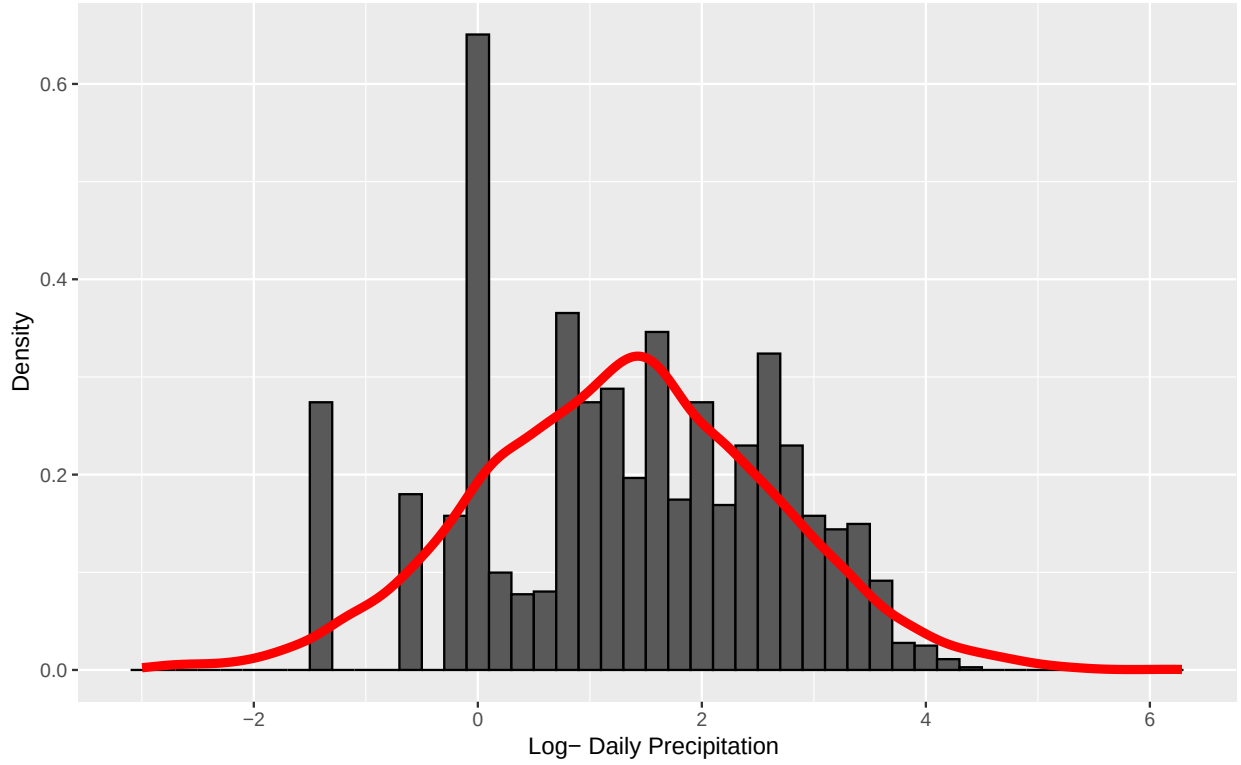

```
ggplot(data=sample_gibbs, aes(x = 1:length(sigma_sample), y = sigma_sample)) +
  geom_line() +labs(title = "sigma sample")
```



(b)

Plot the following in one figure:
1) a histogram or kernel density estimate of the daily precipitation
2) The resulting posterior predictive density using the simulated posterior draws from How well does the posterior predictive density agree with this data?

```
# Using Gibbs sample's mu and sigma to sample posterior prediction
post_pred_samples <- rnorm(n = 10000, mean = sample_gibbs$mu_sample,
                           sd = sqrt(sample_gibbs$sigma_sample))

ggplot(data = data.frame(y = log_prec), aes(x = y)) +
  geom_histogram(aes(y =after_stat(density)), color = "black", binwidth = 0.2) +
  geom_density(data = data.frame(y = post_pred_samples), aes(x = y, y = after_stat(density)),
               color = "red",linewidth = 2) +
  labs(x = "Log- Daily Precipitation ", y = "Density")
```



Above is the plot of the histogram of the daily precipitation, and the resulting posterior predictive density(red line). We can say that the posterior prediction does not totally agree with the existing data, especially towards to left-hand side of the mode.

Q2 Metropolis Random Walk for Poisson regression

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 signifcant?

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

model <- glm(formula = nBids ~ .,data = ebay_data[,-2],family = 'poisson')

summary(model)
```

```
##
## Call:
## glm(formula = nBids ~ ., family = "poisson", data = ebay_data[,
##     -2])
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.5800  -0.7222  -0.0441   0.5269   2.4605
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.07244    0.03077  34.848  < 2e-16 ***
## PowerSeller -0.02054    0.03678  -0.558   0.5765
## VerifyID    -0.39452    0.09243  -4.268 1.97e-05 ***
## Sealed       0.44384    0.05056   8.778  < 2e-16 ***
## Minblem     -0.05220    0.06020  -0.867   0.3859
## MajBlem     -0.22087    0.09144  -2.416   0.0157 *
## LargNeg      0.07067    0.05633   1.255   0.2096
## LogBook     -0.12068    0.02896  -4.166 3.09e-05 ***
## MinBidShare -1.89410    0.07124 -26.588  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 2151.28  on 999  degrees of freedom
## Residual deviance:  867.47  on 991  degrees of freedom
## AIC: 3610.3
##
## Number of Fisher Scoring iterations: 5
```

The estimates of beta are 1.0724421, -0.0205408, -0.3945165, 0.4438426, -0.0521983, -0.2208712, 0.0706725, -0.1206776, -1.8940966

The significant covariates are **'VerfiyID','Sealed','LogBook'** and **'MinBidShare'** and **'Intercept'?'MajBlem'** is also significant but to a less degree compared to the ones mentioned before.

b) Let's do a Bayesian analysis of the Poisson regression. Let the prior be $\beta \sim \mathcal{N}[0, 100 \cdot (X^T X)^{-1}]$, 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:

$$\beta \mid y \sim \mathcal{N}\left(\tilde{\beta}, J_{\mathbf{y}}^{-1}(\tilde{\beta})\right)$$

where $\tilde{\beta}$ is the posterior mode and $J_{\mathbf{y}}(\tilde{\beta})$ is the negative Hessian at the posterior mode. $\tilde{\beta}$ and $J_{\mathbf{y}}(\tilde{\beta})$ 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.).

The Poisson distribution density is given by :

$$p(y \mid x; \beta) = \frac{e^{y\beta' x} e^{-e^{\beta'}}}{y!}$$

The log likelihood of the poisson distribution is :

$$\ell(\beta \mid X, Y) = \log L(\beta \mid X, Y) = \sum_{i=1}^{n} \left(y\beta' x - e^{\beta' x} - \log y!\right)$$

```
LogPosteriorFunc <- function(betas, X, y, mu, Sigma){
  log_prior <- dmvnorm(betas, mu, Sigma, log=TRUE)
  log_likelihood <- sum(X%*%betas * y - exp(X%*%betas) -log(factorial(y)))
  return(log_prior + log_likelihood)
}

# Initialize values
#remove 1st column since that is target variable and convert to matrix
n_cols <- ncol(ebay_data[,-1])
covariates <- as.matrix(ebay_data[,-1])
labels <- as.matrix(ebay_data[,1])
mu <- rep(0, n_cols)
initVal <- matrix(0, n_cols, 1)
Sigma <- as.matrix(100 * solve(t(covariates)%*%covariates))

# Optimizer
OptimRes <- optim(initVal, LogPosteriorFunc, gr = NULL, y = labels, X = covariates,
                  mu = mu, Sigma = Sigma, method=c("BFGS"),
                  control=list(fnscale=-1), hessian=TRUE)

beta_mode <- OptimRes$par
jacobian <- OptimRes$hessian
inv_jacobian <- -solve(jacobian)

beta_draws <- as.matrix(rmvnorm(10000,mean = beta_mode,sigma = inv_jacobian))
beta_estimates <- colMeans(beta_draws)
```

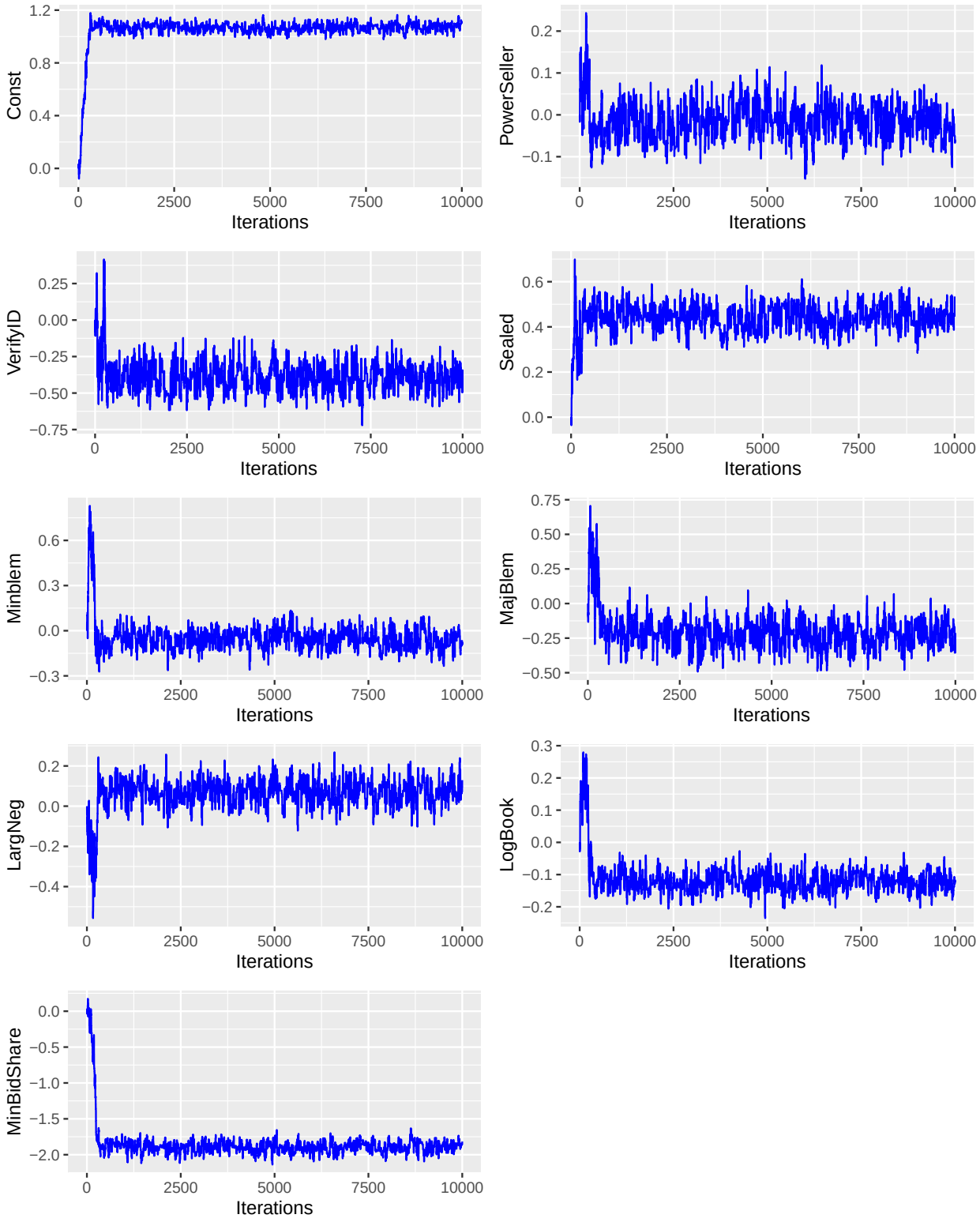
c) Let's simulate from the actual posterior of β using the Metropolis algorithm and compare the results with the approximate results in b). Program a general function that uses the Metropolis algorithm to generate random draws from an arbitrary posterior density. In order to show that it is a general 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):

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

where $\Sigma = J_{\mathbf{y}}^{-1}(\tilde{\beta})$ 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 HowToCo deRWM.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.

```
MetHas_RandomWalk <- function(nDraws,fun,mu,Sigma,c){

  #initialize matrix
  draw_matrix <- matrix(0,nrow = nDraws,ncol = n_cols)
  #initialize first row to mu
  draw_matrix[1,] <- mu

  for(i in 2:nDraws){
    # sample from multivariate normal distribution
    proposed_sample <- as.vector(rmvnorm(n = 1,mean = draw_matrix[i-1,],
                                         sigma = c*as.matrix(Sigma)))
    #print(proposed_sample)
    # IMPORTANT : the log is inside the posterior function
    log_acceptance_prob <- exp(fun(proposed_sample)- fun(draw_matrix[i-1,]))

    #random sample
    u <- runif(1)
    # calculate acceptance probability
    a <- min(1,log_acceptance_prob)


    if(u <= a){
      #accept sample
      draw_matrix[i,] <- proposed_sample
    }
    else{
      # stay at same values from previous draw
      draw_matrix[i,] <- draw_matrix[i-1,]
    }

  }
  return(draw_matrix)
}

logPostFunc <- function(theta){
  res <- dmvnorm(theta,mean = beta_estimates,sigma = inv_jacobian,log = TRUE)
  if(is.na(res)){
```

```
    print(theta)
  }
  return(res)
}

df <- MetHas_RandomWalk(nDraws = 10000,fun = logPostFunc,mu = rep(0,n_cols),
                        Sigma = inv_jacobian,c = 1)
# assign colnames
colnames(df) <- colnames(ebay_data)[2:10]

## plotting
plot_list <- list()
for (col in colnames(df)) {
  # Plot iterations vs every column
  p <- ggplot(data = as.data.frame(df), aes_string(x = 1:nrow(df), y = col)) +
    geom_line(col = 'blue') +
    labs(x = "Iterations", y = col)

  #show plot
  #print(p)
  plot_list[[col]] <- p
}

# Arranging in 1 fig
grid.arrange(grobs = plot_list, ncol = 2)
```



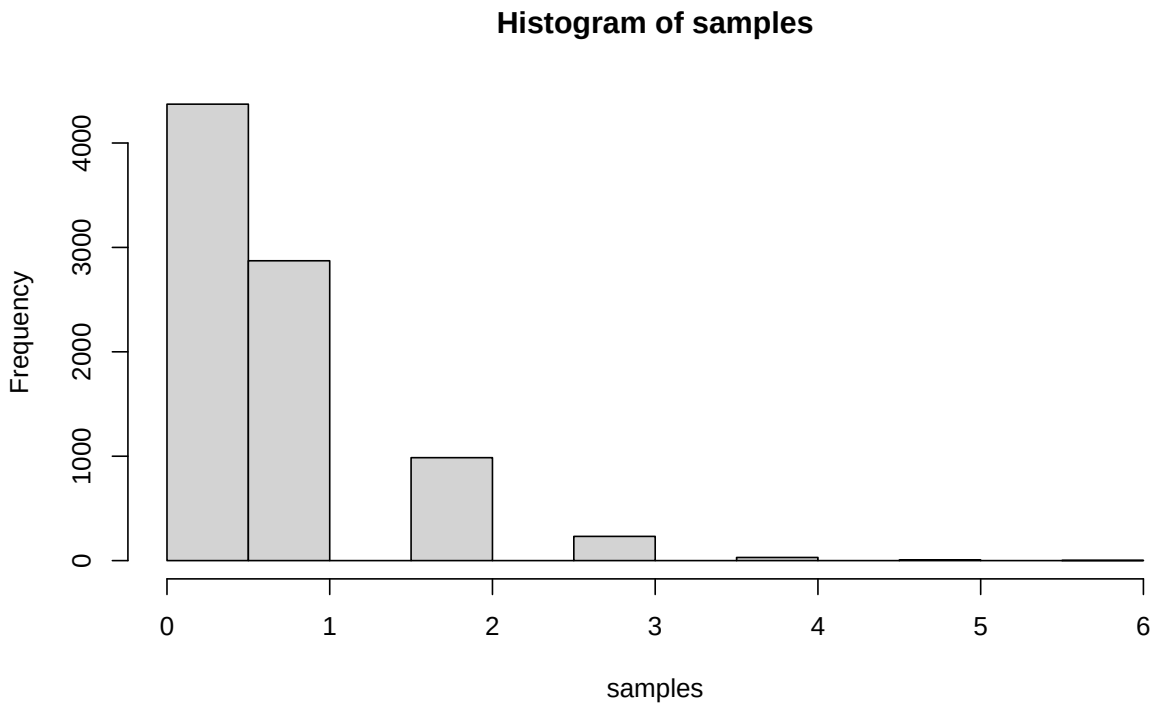
Assessing the plots of the 9 column variables vs Iterations, we see that convergence usually occurs after 1500 iterations. (burn-in period)

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?

- Power Seller = 1
- VerifyID = 0
- Sealed = 1
- MinBlem = 0
- MajBlem = 1
- LargNeg = 0
- **Log** Book = 1.2
- MinBidShare = 0.8

$$\text{Mean} = \lambda = e^{\beta' \cdot x}$$

```
# extra 1 at the start for the intercept - Const
new_data <- c(1,1,0,1,0,1,0,1.2,0.8)

# lambda =  e^Beta*x
# discarding first 1500 samples as burn-in period
lambda <- exp(df[-c(1:1500),] %*% new_data)

samples <- c()

for (i in 1:nrow(df[-c(1:1500),])) {
  #sample from each row of df to get the predictive distribution based on the
  #posterior betas
  samples[i] <- rpois(1,lambda = lambda[i])

}

hist(samples)
```




```
res <- length(samples[samples == 0])/length(samples)
```

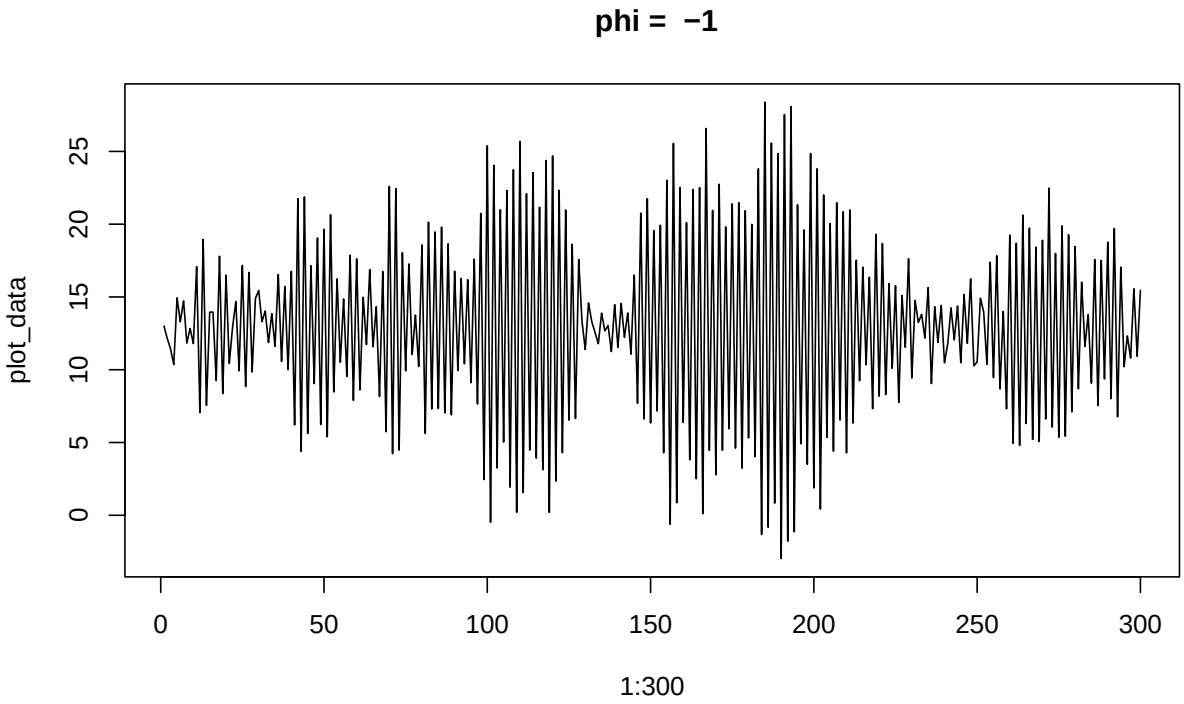
The probability of have zero bidders in the new auction is 51.4352941%.

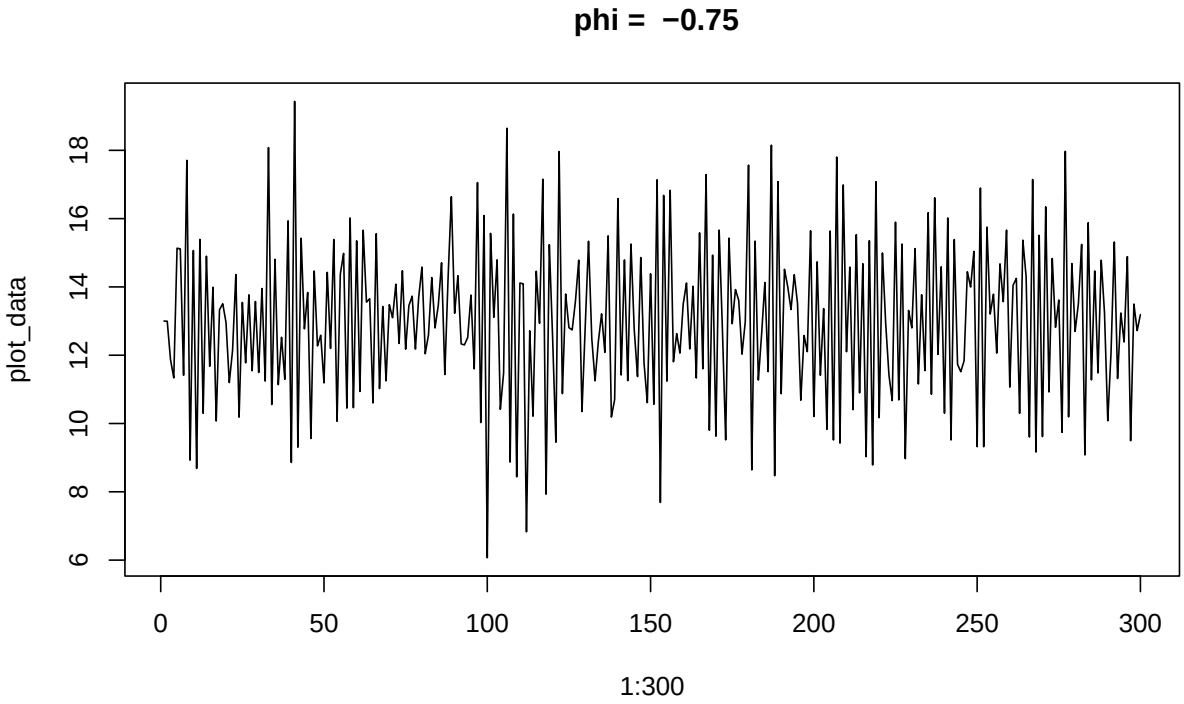
Q3 Time series models in Stan

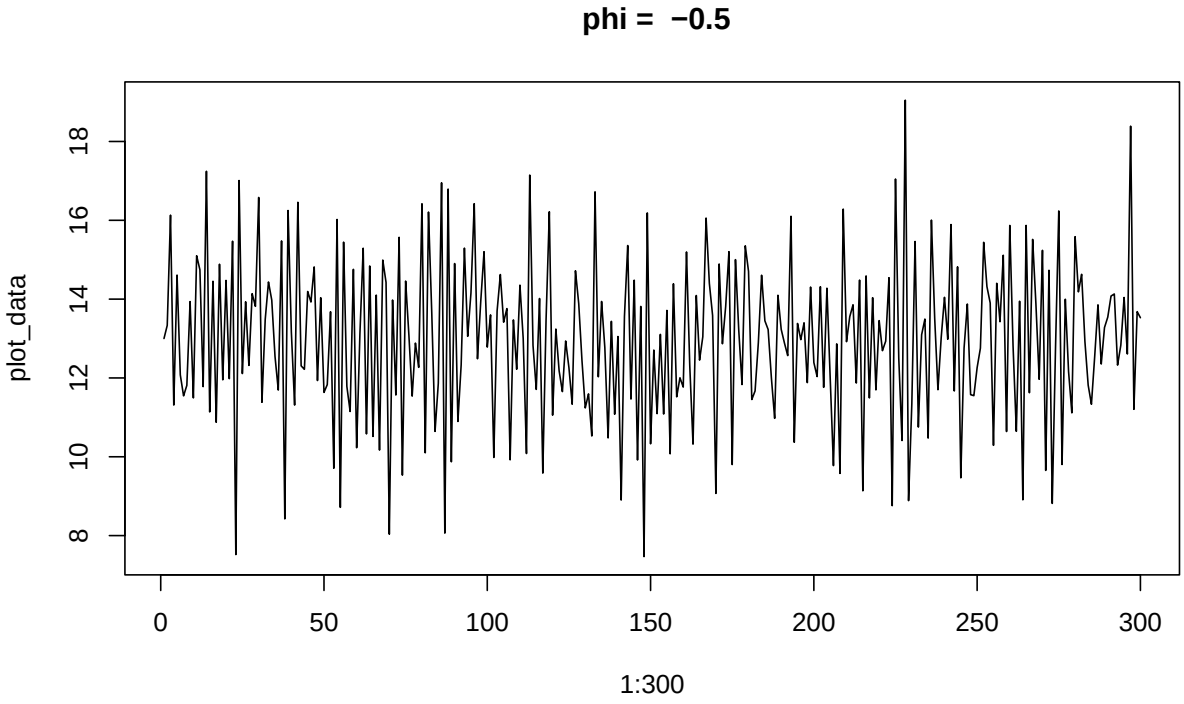
(a)

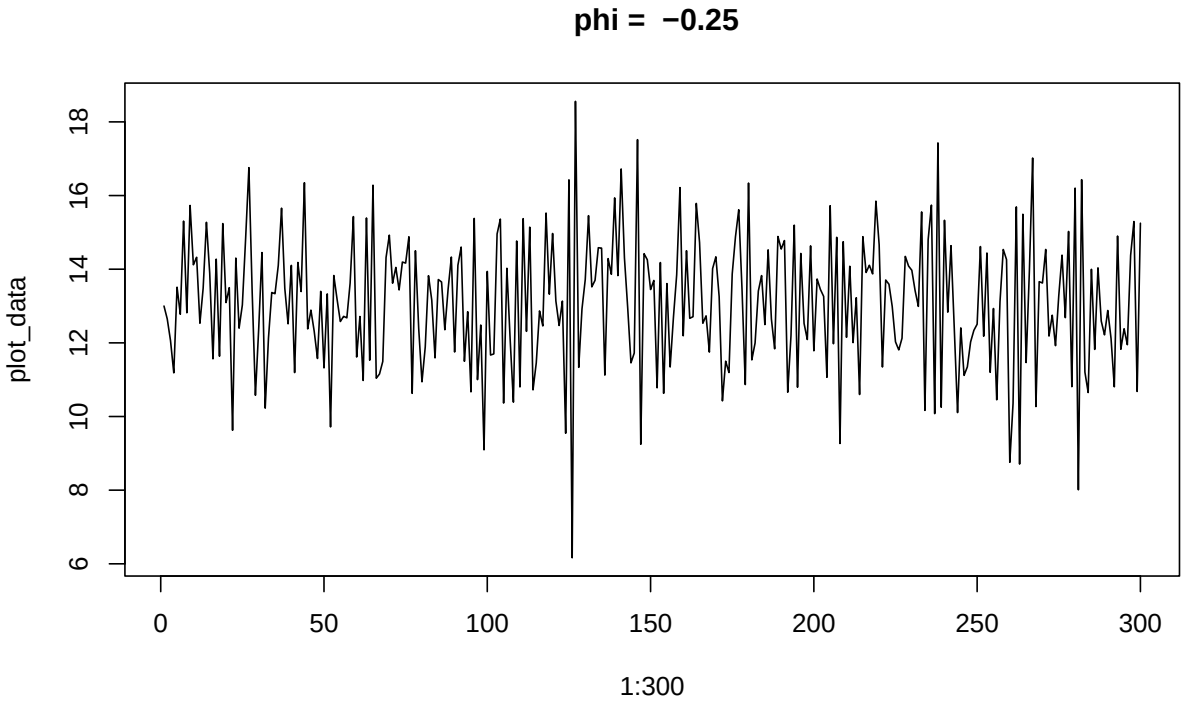
Write a function in R that simulates data from the AR(1)-process

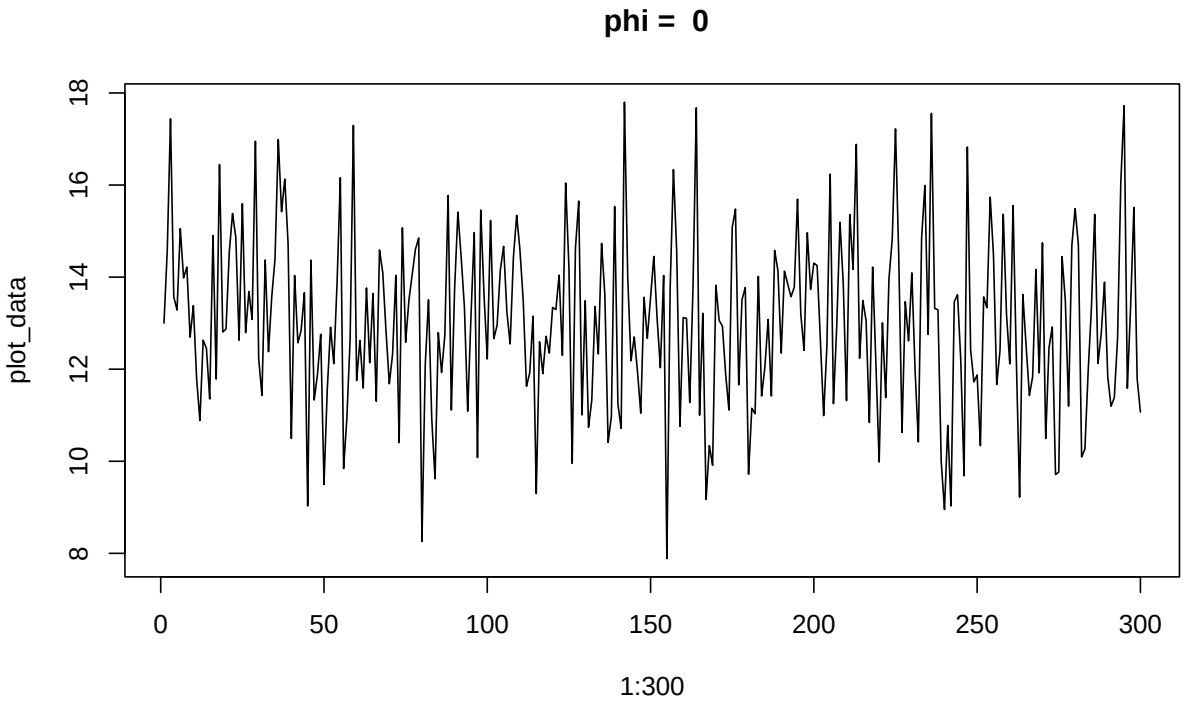
```r
#####Q3
mu <- 13
sigma_square <- 3
t <- 300
phi <- seq(from = -1 ,to = 1, by =0.25 )

ar_func <- function(phi,mu,sigma_square,t){
  counter <- 0
  result_vector <- rep(0,t)
  result_vector[1] <- mu
  for(i in 2:t){
    epislon <- rnorm(1,0,sqrt(sigma_square))
    x_i <- mu+phi*(result_vector[i-1]-mu)+ epislon
    result_vector[i] <- x_i
  }

  return(result_vector)
}


test_phi_func <- function(phi,mu,sigma_square,t){

  phi_test_df <- data.frame(matrix(0, nrow = t, ncol = length(phi)))
  colnames(phi_test_df) <- phi

  for (j in 1:length(phi)) {
    phi_test <- ar_func(phi[j], mu, sigma_square, t)
    phi_test_df[, j] <- phi_test
  }
  return(phi_test_df)
}
phi_df <- test_phi_func(phi,mu,sigma_square,t)


for(k in 1:length(phi)){
  plot_data <- phi_df[,k]
  plot(x=1:300, plot_data, type = "l", main = paste(" phi = ", phi[k]))
  #Sys.sleep(1)
}
```

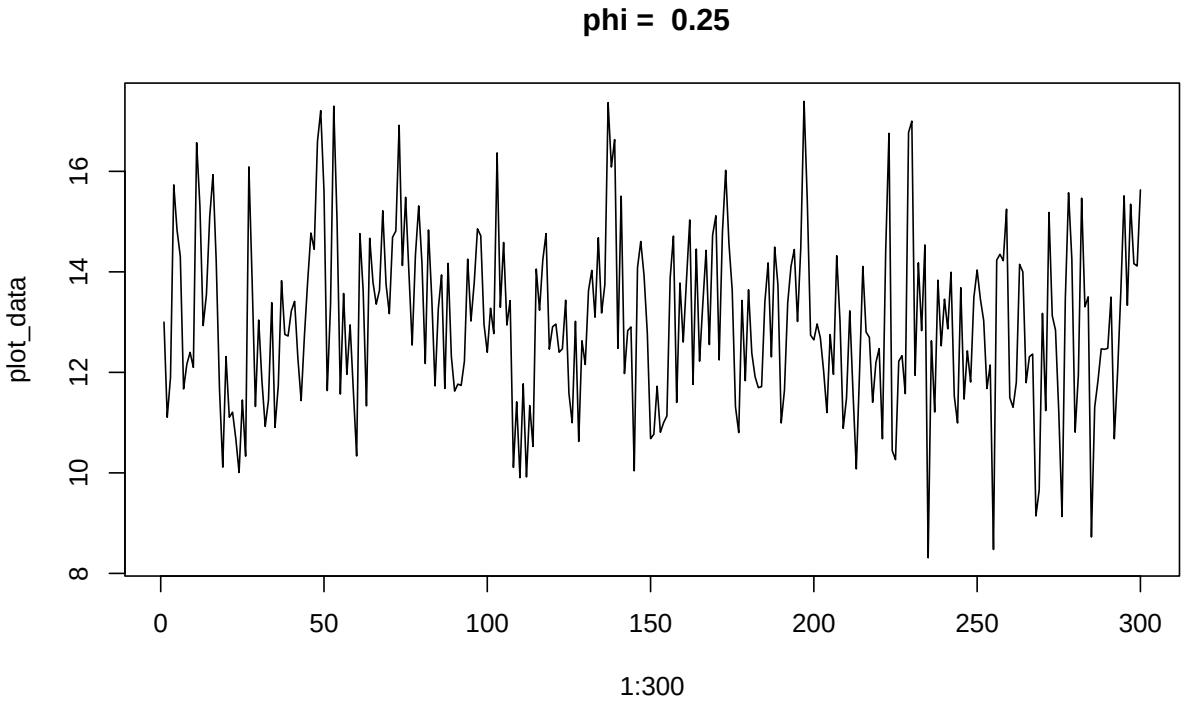

phi = −1
plot_data
0
5
10
15
20
25
0
50
100
150
200
250
300
1:300

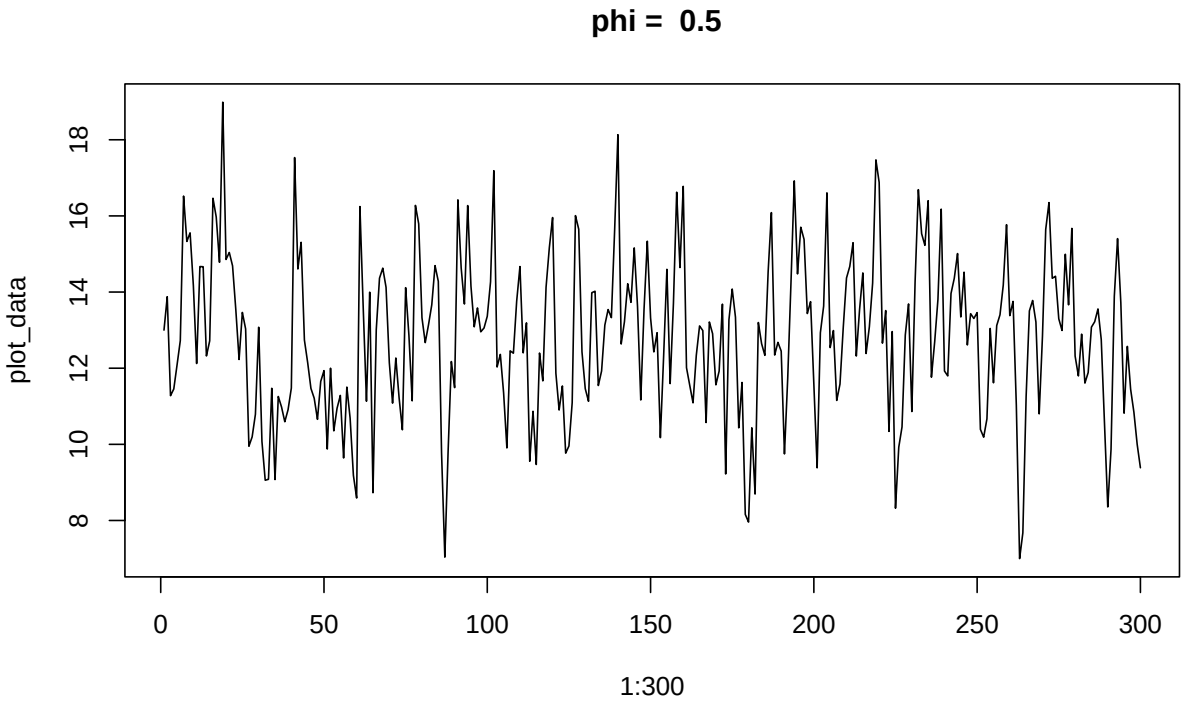

phi = −0.75
plot_data
6
8
10
12
14
16
18
0
50
100
150
200
250
300
1:300

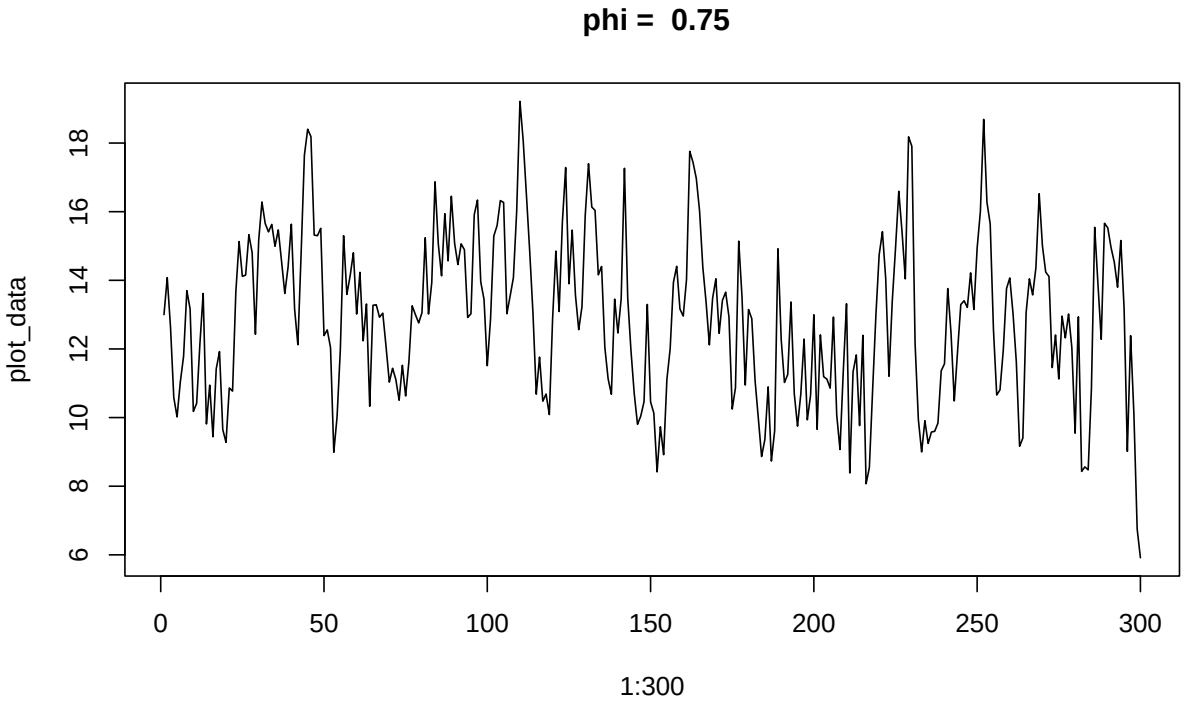

phi = −0.5
plot_data
8
10
12
14
16
18
0
50
100
150
200
250
300
1:300

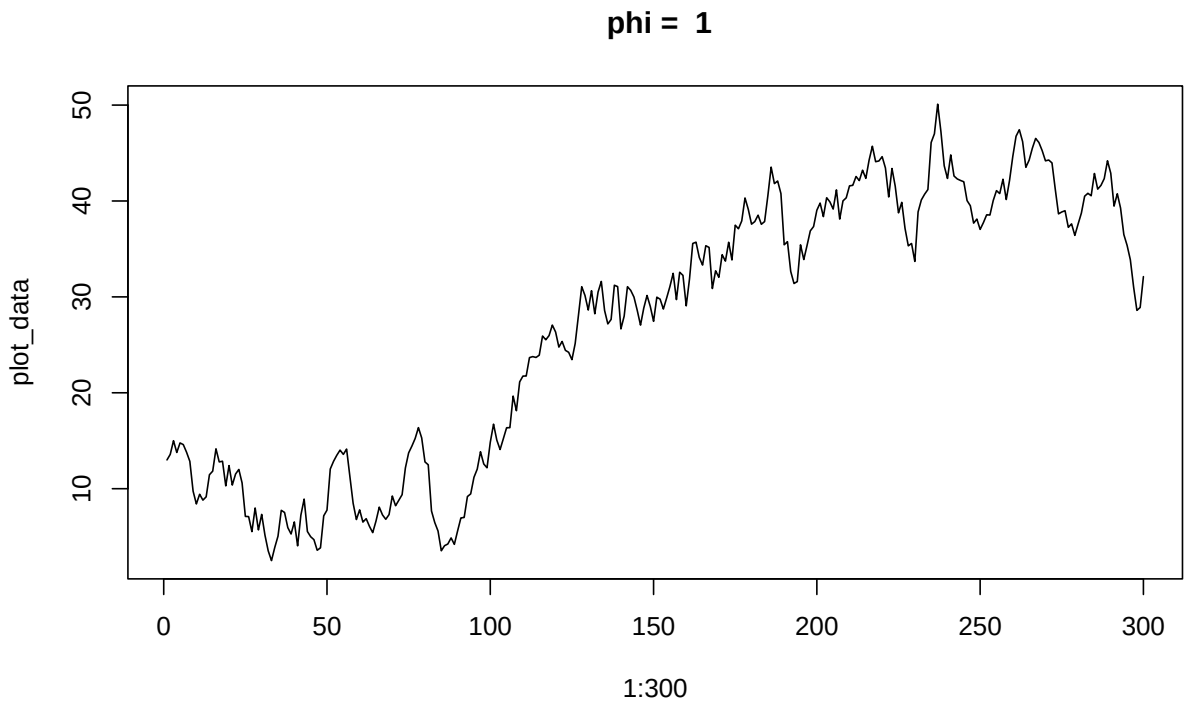

phi = −0.25
plot_data
6
8
10
12
14
16
18
0
50
100
150
200
250
300
1:300


phi = 0
plot_data
8
10
12
14
16
18
0
50
100
150
200
250
300
1:300


phi = 0.25
plot_data
8
10
12
14
16
0
50
100
150
200
250
300
1:300


phi = 0.5
plot_data
8
10
12
14
16
18
0
50
100
150
200
250
300
1:300


phi = 0.75
plot_data
6
8
10
12
14
16
18
0
50
100
150
200
250
300
1:300



As we can observed with different ϕ, as ϕ increase, the gap between peaks are getting larger (not as fluctuate as lower ϕ). This indicate that the data tends to have higher autocorrelation, which means each data point is more correlated with preceding data.

(b

Rstan

```
library(rstan)
set.seed(12345)
x <- ar_func(phi=0.2, mu, sigma_square, t)
y <- ar_func(phi=0.95, mu, sigma_square, t)

stan_code <- "
data {
  int<lower=0> T;         // Number of time points
  vector[T] x;
  vector[T] y;
}
parameters {
  real mu_x;
  real mu_y;
  real phi_x;
  real phi_y;
  real sigma_x;
  real sigma_y;
}
```

```
model {
  // After some research, it is common to use a flat prior or a vague prior as
  // non-informative prior
  // Therefore, we pick normal distribution with higher variance as prior.
  mu_x ~ normal(0, 50);
  mu_y ~ normal(0, 50);
  phi_x ~ normal(0, 10);
  phi_y ~ normal(0, 10);

  sigma_x  ~ normal(0, 50);
  sigma_y  ~ normal(0, 50);


  x[2:T] ~ normal(mu_x + phi_x * (x[1:(T - 1)] - mu_x), sigma_x);
  y[2:T] ~ normal(mu_y + phi_y * (y[1:(T - 1)] - mu_y), sigma_y);
}
"



data_list <- list(
  T = t,
  x = x,
  y = y
)

# Set the MCMC settings
niter <- 5000
warmup <- 500

# Compile the Stan model
model <- stan_model(model_code = stan_code)

# Fit the Stan model to the data
# Set the control to avoid too many divergent after warmup
control <- list(adapt_delta = 0.90, stepsize = 0.0001)
fit<- sampling(model, data = data_list, warmup = warmup, iter = niter, chains = 5,control=control,refres
#refresh = 0 to mute the printout
summary(fit)$summary
```

```
##                mean      se_mean         sd          2.5%          25%
## mu_x      13.1736454 0.0009810772 0.12894646   12.92120195   13.0866009
## mu_y      14.7662220 0.4216001425 9.66646731  -12.59440997   12.9360194
## phi_x      0.1886121 0.0004666932 0.05768479    0.07426121    0.1492954
## phi_y      0.9713154 0.0003255917 0.01843371    0.93472438    0.9583243
## sigma_x    1.7892259 0.0005562130 0.07331174    1.65097259    1.7390022
## sigma_y    1.7228587 0.0006095451 0.07117800    1.58966456    1.6741621
## lp__    -634.2138672 0.0264929349 1.83419194 -638.47628106 -635.3056310
##                 50%          75%        97.5%      n_eff     Rhat
## mu_x      13.1734372   13.2605943   13.4247425 17274.7783 1.000210
## mu_y      15.5310107   18.0162790   34.1507642   525.6953 1.006105
## phi_x      0.1889787    0.2277933    0.2991747 15277.7597 1.000192
## phi_y      0.9713980    0.9854067    1.0021845  3205.3759 1.003303
```

```
## sigma_x    1.7867845    1.8374897    1.9404140 17372.5969 1.000122
## sigma_y    1.7208153    1.7689548    1.8698400 13635.7782 1.000495
## lp__    -633.9305759 -632.8203154 -631.5496630  4793.2427 1.001257
```

```
post_mean <- summary(fit)$summary[, "mean"]
interval_025 <- summary(fit)$summary[, "25%"]
interval_975 <- summary(fit)$summary[, "97.5%"]
n_eff <- summary(fit)$summary[, "n_eff"]
Rhat <- summary(fit)$summary[, "Rhat"]
```

(i) Based on the summary, we obtain:
Posterior mean:

```
##         mu_x         mu_y        phi_x        phi_y      sigma_x      sigma_y
##   13.1736454   14.7662220    0.1886121    0.9713154    1.7892259    1.7228587
##         lp__
## -634.2138672
```

95% credible interval are
2.5%:

```
##         mu_x         mu_y        phi_x        phi_y      sigma_x      sigma_y
##   13.0866009   12.9360194    0.1492954    0.9583243    1.7390022    1.6741621
##         lp__
## -635.3056310
```

97.5%:

```
##         mu_x         mu_y        phi_x        phi_y      sigma_x      sigma_y
##   13.4247425   34.1507642    0.2991747    1.0021845    1.9404140    1.8698400
##         lp__
## -631.5496630
```

number of effective posterior sample:

```
##       mu_x       mu_y      phi_x      phi_y    sigma_x    sigma_y       lp__
## 17274.7783   525.6953 15277.7597  3205.3759 17372.5969 13635.7782  4793.2427
```

The true values are $\mu_x = \mu_y = 13$, $\phi_y = 0.95$, $\phi_x = 0.2$, $\sigma_x = \sigma_y = \sqrt{3} \approx 1.73$

For posterior mean, all the inferred parameter are very close to their true value. Hence we can say that we are able to estimate the true value. In terms of 95% credible interval and number of effective posterior sample, we would say that for estimated μ_x, ϕ_y, σ_x and σ_y has narrower credible intervals, while estimated μ_x, ϕ_x, σ_x and σ_y has larger number of effective posterior samples. This indicates the estimate of these inferred parameters are more precise.

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

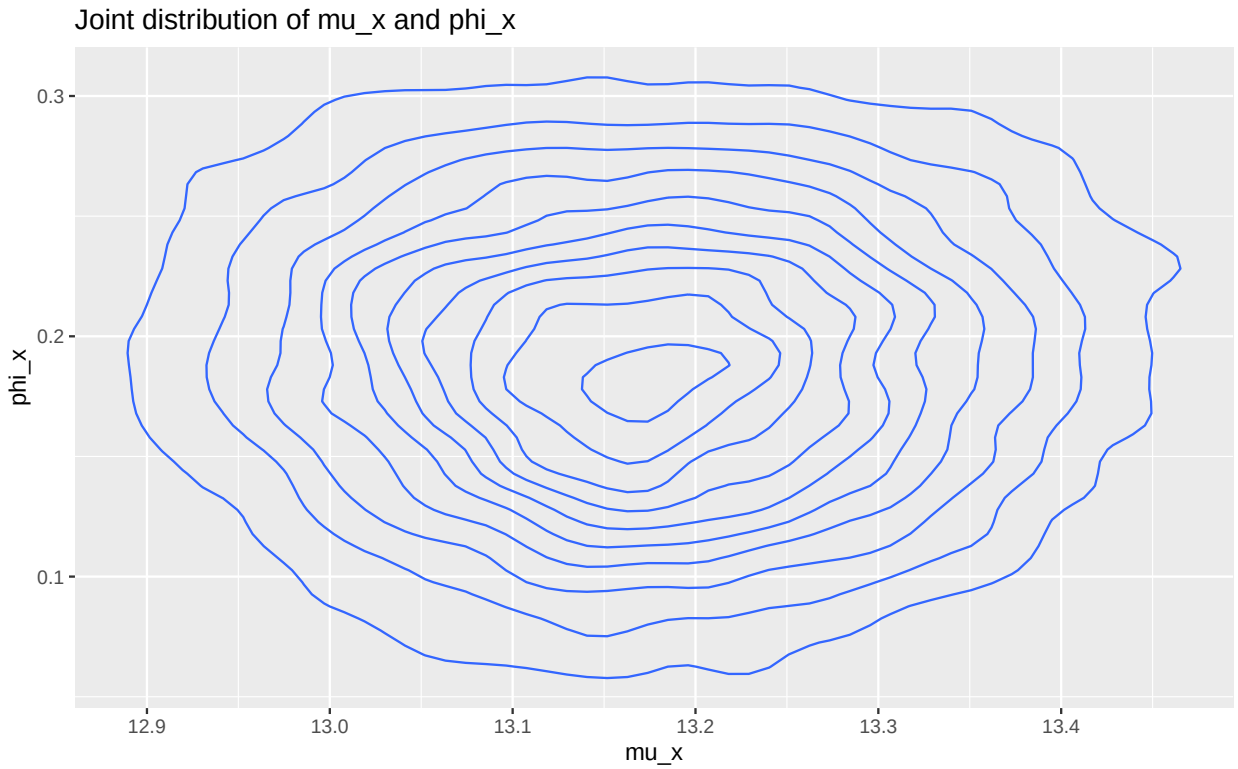
We can use Rhat to evaluate the convergence. The Rhat are as below:

```
##     mu_x     mu_y    phi_x    phi_y  sigma_x  sigma_y     lp__
## 1.000210 1.006105 1.000192 1.003303 1.000122 1.000495 1.001257
```

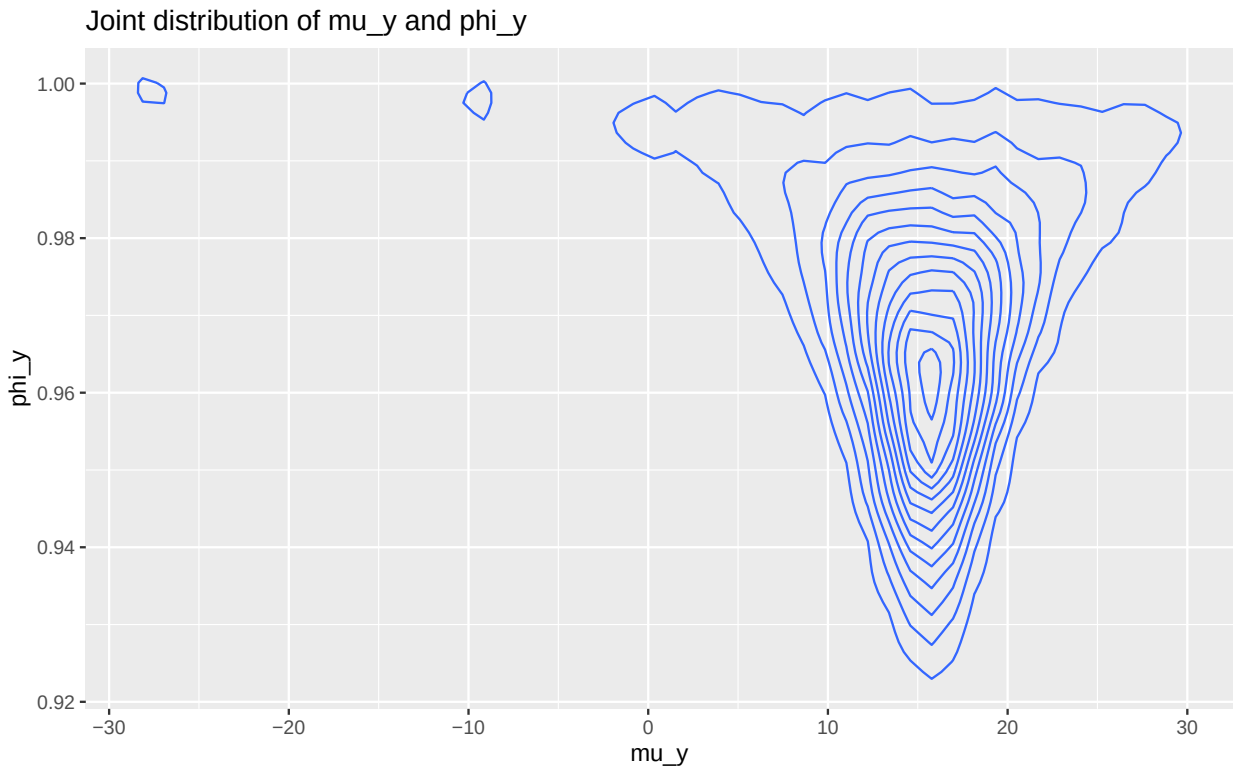
Since all inferred parameters has Rhat close to 1, this imply our sampler converged well.

Below are the plot of the joint posterior of μ and ϕ
For x, the shape of joint distribution are rather symmetric along two axis, this means μ and ϕ has higher correlation. For y, the different shape means, mu and phi has lower correlation hence make it harder to understand the relationship.

```
posterior_x <- extract(fit, pars = c("mu_x", "phi_x"))
posterior_df_x <- data.frame(posterior_x)
ggplot(data = posterior_df_x, aes(x = mu_x, y = phi_x)) +
  stat_density_2d() +
  xlab("mu_x") +
  ylab("phi_x") +
  ggtitle("Joint distribution of mu_x and phi_x")
```




```
posterior_y <- extract(fit, pars = c("mu_y", "phi_y"))
posterior_df_y <- data.frame(posterior_y)
ggplot(data = posterior_df_y, aes(x = mu_y, y = phi_y)) +
  stat_density_2d()+
  xlab("mu_y") +
  ylab("phi_y") +
  ggtitle("Joint distribution of mu_y and phi_y")
```


Joint distribution of mu_y and phi_y
1.00
0.98
0.96
0.94
0.92
phi_y
−30
−20
−10
0
10
20
30
mu_y

Appendix

```
rm(list=ls())
knitr::opts_chunk$set(echo = TRUE)
knitr::opts_chunk$set(fig.align = "center")
knitr::opts_chunk$set(fig.width=8, fig.height=5)
knitr::opts_chunk$set(warning=FALSE)
library(mvtnorm)
library(ggplot2)
library(gridExtra)
set.seed(12345)
#####Q1
library(ggplot2)
library(mvtnorm)
#library(extraDistr)
precipitation <- readRDS("Precipitation.rds")
log_prec <- log(precipitation)

mu_post <- function(mu_0,tau_0_sqr,sigma_square_current,y,n){

  tau_n_sqr = 1 / (1/tau_0_sqr + n/sigma_square_current)
  mu_n <- tau_n_sqr * (mu_0/tau_0_sqr + sum(y)/sigma_square_current)
  mupost <- rnorm(1, mu_n, sqrt(tau_n_sqr))
  return(mupost)

}

sigma_square_post <- function(mu_current, v_0, sigma_square_0, y,n, use_myinvchi=TRUE) {

  v_n <- v_0 + n
  elem1 <- v_0*sigma_square_0
  elem2 <- sum((y - mu_current)^2)
  elem3 <- n+v_0
  elem_comb <- (elem1+elem2)/elem3
  if (use_myinvchi){
    sigma_square_post <- my_inv_chi(v_n,elem_comb)
  }
  else {
    sigma_square_post <- rinvchisq(1, v_n, elem_comb) #Requires package extraDistr
  }



  return(sigma_square_post)
}

my_inv_chi<- function(df,tau_sqr) {
  X <- rchisq(1,df)
  inv_chi <- (df*tau_sqr)/X
  return(inv_chi)
}

#init
```

```r
mu_0 <- 0
tau_0_sqr <- 1
sigma_square_0 <- 1
v_0 <- 1 #degree of freedom for chi square

gibbs_sampler <- function(nstep, data, mu_0, tau_0_sqr, v_0, sigma_square_0) {
  # Init parameters
  mu_current <- 0
  sigma_square_current <- 1

  mu_samples <- rep(0,nstep)
  sigma_square_samples <- rep(0,nstep)

  for (i in 1:nstep) {

    mu_current <- mu_post(mu_0, tau_0_sqr, sigma_square_current, y=data, length(data))
    #print(mu_current)
    sigma_square_current <- sigma_square_post(mu_current, v_0, sigma_square_0, y=data,
                                        length(data),use_myinvchi=TRUE)
    #print(sigma_square_current)

    mu_samples[i] <- mu_current
    sigma_square_samples[i] <- sigma_square_current
  }

  output_df <- data.frame(mu_sample = mu_samples, sigma_sample = sigma_square_samples)
  return(output_df)
}

sample_gibbs <- gibbs_sampler(nstep=10000, data=log_prec, mu_0, tau_0_sqr, v_0, sigma_square_0)
my_acf <- acf(sample_gibbs$mu_sample,plot = F) # getting the autocorrelation
if_mu <- 1 + 2 * sum(my_acf$acf[-1])
my_acf <- acf(sample_gibbs$sigma_sample,plot = F)
if_sigma <- 1 + 2 * sum(my_acf$acf[-1])
ggplot(data=sample_gibbs, aes(x = 1:length(mu_sample), y = mu_sample)) +
  geom_line()+labs(title = "mu sample")

ggplot(data=sample_gibbs, aes(x = 1:length(sigma_sample), y = sigma_sample)) +
  geom_line() +labs(title = "sigma sample")
# Using Gibbs sample's mu and sigma to sample posterior prediction
post_pred_samples <- rnorm(n = 10000, mean = sample_gibbs$mu_sample,
                           sd = sqrt(sample_gibbs$sigma_sample))

ggplot(data = data.frame(y = log_prec), aes(x = y)) +
  geom_histogram(aes(y =after_stat(density)), color = "black", binwidth = 0.2) +
  geom_density(data = data.frame(y = post_pred_samples), aes(x = y, y = after_stat(density)),
               color = "red",linewidth = 2) +
  labs(x = "Log- Daily Precipitation ", y = "Density")

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

model <- glm(formula = nBids ~ .,data = ebay_data[,-2],family = 'poisson')
```

```
summary(model)
LogPosteriorFunc <- function(betas, X, y, mu, Sigma){
  log_prior <- dmvnorm(betas, mu, Sigma, log=TRUE)
  log_likelihood <- sum(X%*%betas * y - exp(X%*%betas) -log(factorial(y)))
  return(log_prior + log_likelihood)
}

# Initialize values
#remove 1st column since that is target variable and convert to matrix
n_cols <- ncol(ebay_data[,-1])
covariates <- as.matrix(ebay_data[,-1])
labels <- as.matrix(ebay_data[,1])
mu <- rep(0, n_cols)
initVal <- matrix(0, n_cols, 1)
Sigma <- as.matrix(100 * solve(t(covariates)%*%covariates))

# Optimizer
OptimRes <- optim(initVal, LogPosteriorFunc, gr = NULL, y = labels, X = covariates,
                  mu = mu, Sigma = Sigma, method=c("BFGS"),
                  control=list(fnscale=-1), hessian=TRUE)

beta_mode <- OptimRes$par
jacobian <- OptimRes$hessian
inv_jacobian <- -solve(jacobian)

beta_draws <- as.matrix(rmvnorm(10000,mean = beta_mode,sigma = inv_jacobian))
beta_estimates <- colMeans(beta_draws)


MetHas_RandomWalk <- function(nDraws,fun,mu,Sigma,c){

  #initialize matrix
  draw_matrix <- matrix(0,nrow = nDraws,ncol = n_cols)
  #initialize first row to mu
  draw_matrix[1,] <- mu

  for(i in 2:nDraws){
    # sample from multivariate normal distribution
    proposed_sample <- as.vector(rmvnorm(n = 1,mean = draw_matrix[i-1,],
                                         sigma = c*as.matrix(Sigma)))
    #print(proposed_sample)
    # IMPORTANT : the log is inside the posterior function
    log_acceptance_prob <- exp(fun(proposed_sample)- fun(draw_matrix[i-1,]))

    #random sample
    u <- runif(1)
    # calculate acceptance probability
    a <- min(1,log_acceptance_prob)


    if(u <= a){
      #accept sample
      draw_matrix[i,] <- proposed_sample
```

```
    }
    else{
      # stay at same values from previous draw
      draw_matrix[i,] <- draw_matrix[i-1,]
    }

  }
  return(draw_matrix)
}

logPostFunc <- function(theta){
  res <- dmvnorm(theta,mean = beta_estimates,sigma = inv_jacobian,log = TRUE)
  if(is.na(res)){
    print(theta)
  }
  return(res)
}

df <- MetHas_RandomWalk(nDraws = 10000,fun = logPostFunc,mu = rep(0,n_cols),
                        Sigma = inv_jacobian,c = 1)
# assign colnames
colnames(df) <- colnames(ebay_data)[2:10]

## plotting
plot_list <- list()
for (col in colnames(df)) {
  # Plot iterations vs every column
  p <- ggplot(data = as.data.frame(df), aes_string(x = 1:nrow(df), y = col)) +
    geom_line(col = 'blue') +
    labs(x = "Iterations", y = col)

  #show plot
  #print(p)
  plot_list[[col]] <- p
}

# Arranging in 1 fig
grid.arrange(grobs = plot_list, ncol = 2)

# extra 1 at the start for the intercept - Const
new_data <- c(1,1,0,1,0,1,0,1.2,0.8)

# lambda =  e^Beta*x
# discarding first 1500 samples as burn-in period
lambda <- exp(df[-c(1:1500),] %*% new_data)

samples <- c()

for (i in 1:nrow(df[-c(1:1500),])) {
  #sample from each row of df to get the predictive distribution based on the
  #posterior betas
  samples[i] <- rpois(1,lambda = lambda[i])
```

```
}

hist(samples)

res <- length(samples[samples == 0])/length(samples)
#####Q3
mu <- 13
sigma_square <- 3
t <- 300
phi <- seq(from = -1 ,to = 1, by =0.25 )

ar_func <- function(phi,mu,sigma_square,t){
  counter <- 0
  result_vector <- rep(0,t)
  result_vector[1] <- mu
  for(i in 2:t){
    epislon <- rnorm(1,0,sqrt(sigma_square))
    x_i <- mu+phi*(result_vector[i-1]-mu)+ epislon
    result_vector[i] <- x_i
  }

  return(result_vector)
}


test_phi_func <- function(phi,mu,sigma_square,t){

  phi_test_df <- data.frame(matrix(0, nrow = t, ncol = length(phi)))
  colnames(phi_test_df) <- phi

  for (j in 1:length(phi)) {
    phi_test <- ar_func(phi[j], mu, sigma_square, t)
    phi_test_df[, j] <- phi_test
  }
  return(phi_test_df)
}
phi_df <- test_phi_func(phi,mu,sigma_square,t)


for(k in 1:length(phi)){
  plot_data <- phi_df[,k]
  plot(x=1:300, plot_data, type = "l", main = paste(" phi = ", phi[k]))
  #Sys.sleep(1)
}
library(rstan)
set.seed(12345)
x <- ar_func(phi=0.2, mu, sigma_square, t)
y <- ar_func(phi=0.95, mu, sigma_square, t)

stan_code <- "
data {
  int<lower=0> T;         // Number of time points
  vector[T] x;
```

```
  vector[T] y;
}
parameters {
  real mu_x;
  real mu_y;
  real phi_x;
  real phi_y;
  real sigma_x;
  real sigma_y;
}
model {
  // After some research, it is common to use a flat prior or a vague prior as
  // non-informative prior
  // Therefore, we pick normal distribution with higher variance as prior.
  mu_x ~ normal(0, 50);
  mu_y ~ normal(0, 50);
  phi_x ~ normal(0, 10);
  phi_y ~ normal(0, 10);

  sigma_x  ~ normal(0, 50);
  sigma_y  ~ normal(0, 50);


  x[2:T] ~ normal(mu_x + phi_x * (x[1:(T - 1)] - mu_x), sigma_x);
  y[2:T] ~ normal(mu_y + phi_y * (y[1:(T - 1)] - mu_y), sigma_y);
}
"



data_list <- list(
  T = t,
  x = x,
  y = y
)

# Set the MCMC settings
niter <- 5000
warmup <- 500

# Compile the Stan model
model <- stan_model(model_code = stan_code)

# Fit the Stan model to the data
# Set the control to avoid too many divergent after warmup
control <- list(adapt_delta = 0.90, stepsize = 0.0001)
fit<- sampling(model, data = data_list, warmup = warmup, iter = niter, chains = 5,control=control,refres
#refresh = 0 to mute the printout
summary(fit)$summary

post_mean <- summary(fit)$summary[, "mean"]
interval_025 <- summary(fit)$summary[, "25%"]
interval_975 <- summary(fit)$summary[, "97.5%"]
```

```
n_eff <- summary(fit)$summary[, "n_eff"]
Rhat <- summary(fit)$summary[, "Rhat"]
post_mean
interval_025
interval_975
n_eff
Rhat
posterior_x <- extract(fit, pars = c("mu_x", "phi_x"))
posterior_df_x <- data.frame(posterior_x)
ggplot(data = posterior_df_x, aes(x = mu_x, y = phi_x)) +
  stat_density_2d() +
  xlab("mu_x") +
  ylab("phi_x") +
  ggtitle("Joint distribution of mu_x and phi_x")


posterior_y <- extract(fit, pars = c("mu_y", "phi_y"))
posterior_df_y <- data.frame(posterior_y)
ggplot(data = posterior_df_y, aes(x = mu_y, y = phi_y)) +
  stat_density_2d()+
  xlab("mu_y") +
  ylab("phi_y") +
  ggtitle("Joint distribution of mu_y and phi_y")
```