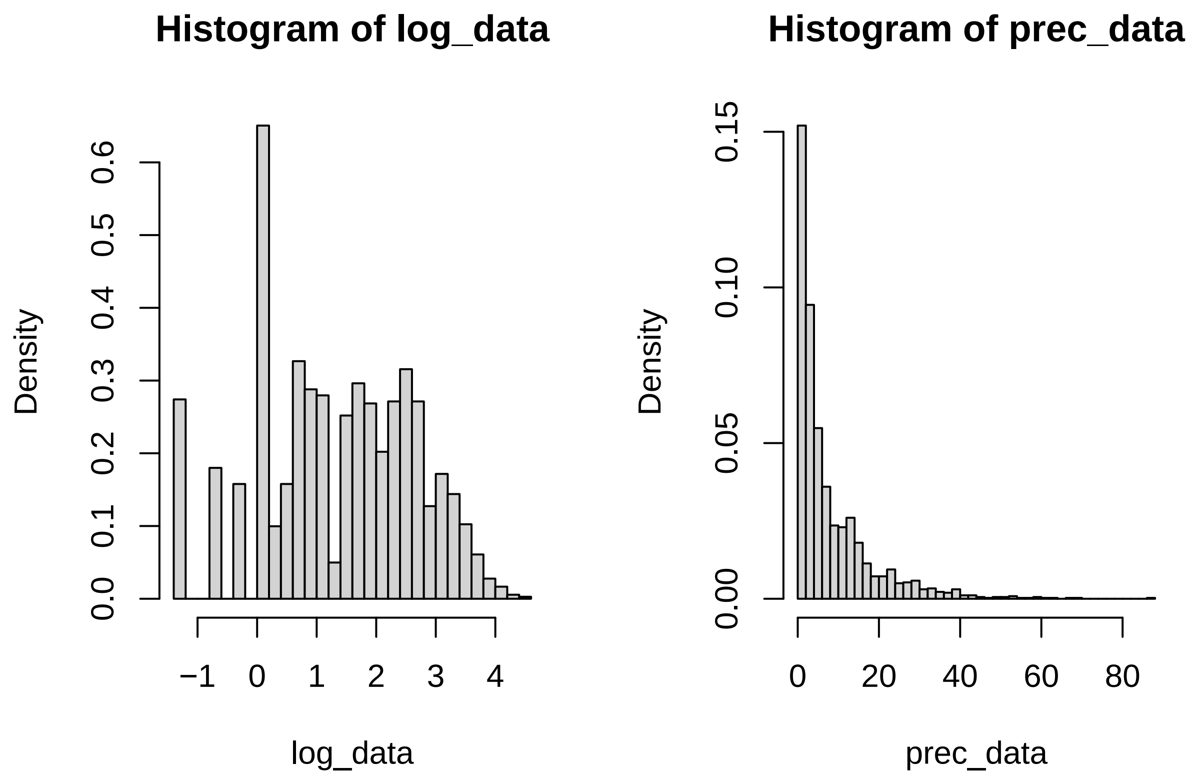
Bayesian Learning - Lab3

Jaskirat S Marar & Dinuke Jayaweera

5/3/2022

GIBBS SAMPLER

```r
# QUES 1: GIBBS SAMPLER

prec_data <- readRDS("precipitation.rds")

log_data <- log(prec_data)

par(mfrow = c(1,2))
hist(log_data, breaks = 40, prob = TRUE)
hist(prec_data, breaks = 40, prob = TRUE)
```



## PART A

We first set up the priors and posteriors as follows:

PRIORS:

$$\mu \sim N(\mu_0, \tau_0^2)$$
$$\sigma^2 \sim Inv - \chi^2(\nu_0, \sigma_0^2)$$

FULL CONDITIONAL POSTERIORS:

μ

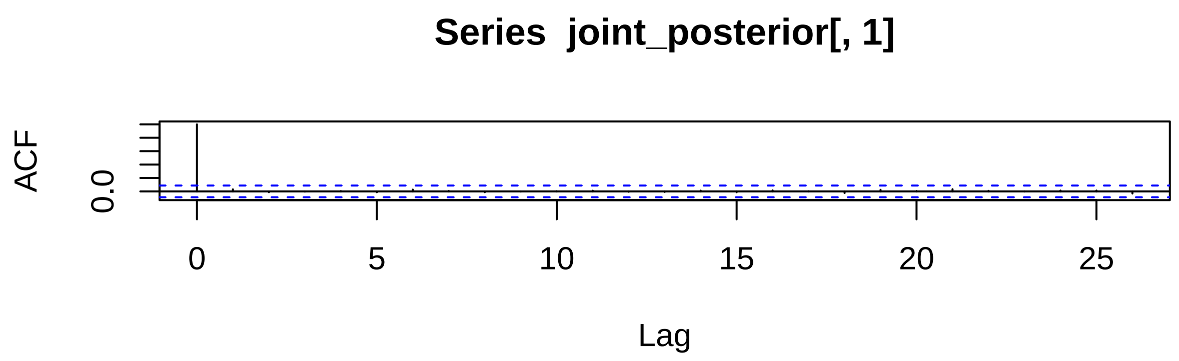
$$\mu|\sigma^2, x \sim N(\mu_n, \tau_n^2)$$
$$where,\ \mu_n = w * (\bar{x}) + (1 - w) * \mu_0,$$
$$\frac{1}{\tau_n^2} = \frac{1}{(\frac{n}{\sigma^2} + \frac{1}{\tau_0^2})},$$
$$w = \frac{\frac{n}{\sigma^2}}{(\frac{n}{\sigma^2} + \frac{1}{\tau_0^2})}$$

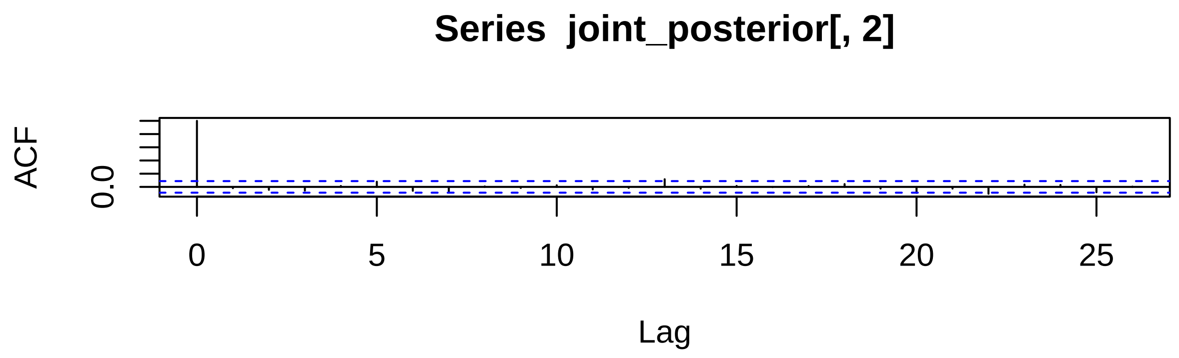
σ^2

$$\sigma^2|\mu, x \sim Inv - \chi^2\Big(\nu_n, \frac{\nu_0\sigma_0^2 + \sum_{i=1}^{1}(x_i - \mu)^2}{n + \nu_0}\Big)$$
$$where,\ \nu_n = \nu_0 + n$$

We'll have to initialize some values for the parameters of the priors. For μ_0 & σ_0^2 we choose the sample mean and sample variance. For ν_0 & τ_0^2 we choose to initialize them to 1 since the data is log normally distributed.

```r
#PART A

# setting up the initial values of the priors

n = length(log_data)

# for mu
mu0 = mean(log_data)
tau20 = 1
# for sigma
sig20 = var(log_data)
nu0 = 1

set.seed(1234)

#Gibbs Sampling function
Gibbs_sampler <- function(steps = 500,
                          mu_0 = mu0,
                          tau2_0 = tau20,
                          sig2_0 = sig20,
                          nu_0 = nu0,
                          n = length(log_data)){

  #initial value for gibbs sampler
  mu_i = rnorm(1, mean = mu_0, sd = sqrt(tau2_0))
  sigma2_i =  rinvchisq(1, df = nu_0, scale = sig2_0)

  #initialize result DF
  sample_results = data.frame("mu" = mu_i,
                              "sigma2" = sigma2_i)

  #initialize loop over specified gibbs steps
  for (i in 1:steps) {

    #gibbs step for mu
    w = n/sigma2_i / (n/sigma2_i + 1/tau2_0)
    mu_n = w * mean(log_data) + (1-w) * mu_0
    tau2_n = (n/sigma2_i + 1/tau2_0)^-1
    mu_i = rnorm(1, mean = mu_n, sd = sqrt(tau2_n))

    #gibbs step for sigma
    nu_n = nu_0 + n
    scale_term = (nu_0 * sigma2_i + sum((log_data - mu_i)^2))/nu_n
    sigma_i = rinvchisq(1, df = nu_n, scale = scale_term)

    #IF
    #IF_mu = 1 + 2*(sum(cor(sample_results[i,1], mu_i)))

    #save step values to DF
    sample_results = rbind.data.frame(sample_results, c(mu_i, sigma_i))

  }

  return(sample_results)

}

joint_posterior <- Gibbs_sampler(steps = 500)

#autocorrelation
par(mfrow = c(2,1))
mu_Gibbs <- acf(joint_posterior[,1])
sig2_Gibbs <- acf(joint_posterior[,2])
```





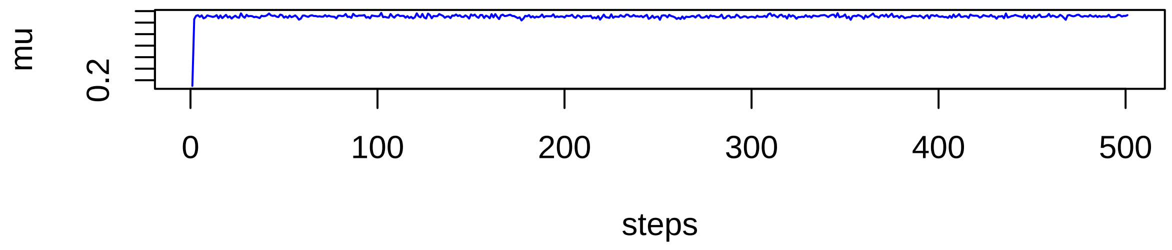
```
#inefficiency factor
IF_Gibbs_mu <- 1+2*sum(mu_Gibbs$acf[-1])
IF_Gibbs_sig2 <- 1+2*sum(sig2_Gibbs$acf[-1])

cat("\n", "IF of mu:", round(IF_Gibbs_mu,3))
```

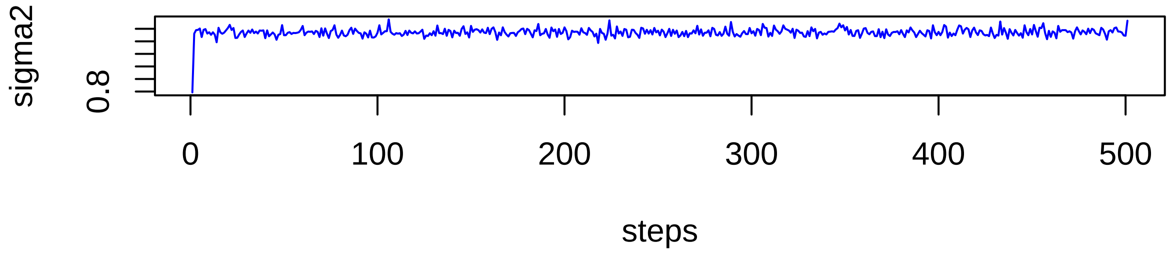
```
##
##  IF of mu: 1.147
```

```
cat("\n", "IF of sigma2:", round(IF_Gibbs_sig2,3))
```

```
##
##  IF of sigma2: 0.421
```

```
#trace plot
par(mfrow = c(2,1))
plot(1:length(joint_posterior$mu), joint_posterior[,1], type = "l",col="blue", xlab = "steps", ylab = "
plot(1:length(joint_posterior$sigma2), joint_posterior[,2], type = "l",col="blue", xlab = "steps", ylab
```



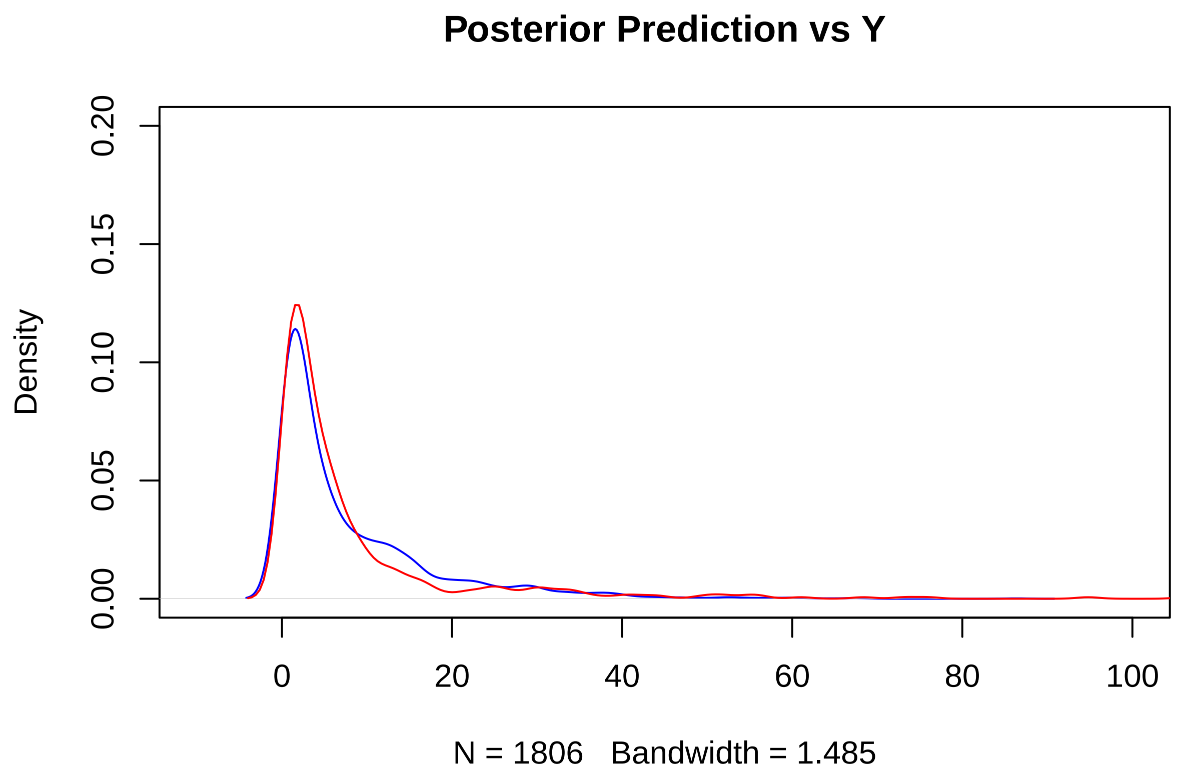


From the trace plots its very clear that the burn-in period is very short and that both chains converge very quickly.

PART B

Since the data is log normally distributed and we've modelled the posterior on the log(data), we can use the posterior draws and exponentiate them to compare to the original data.

```
# PART B

set.seed(1234)
post_pred <- c()
post_burnremoved <- joint_posterior[51:length(joint_posterior$mu),]
for (i in 1:(length(post_burnremoved$mu))){
  post_pred[i] = rnorm(1, mean = post_burnremoved[i,1], sd = sqrt(post_burnremoved[i,2]))
}

e_post_draws <- exp(post_pred)


plot(density(prec_data),
     col = "blue",
     ylim = c(0,0.2),
     xlim = c(-10,100),
     main = "Posterior Prediction vs Y")
lines(density(e_post_draws),
      col = "red")
```


Posterior Prediction vs Y
Density
0.00
0.05
0.10
0.15
0.20
0
20
40
60
80
100
N = 1806 Bandwidth = 1.485

METROPOLIS RANDOM WALK

PART A

```r
# QUES 2: METROPOLIS RW

rm(list = ls())

data <- read.table("eBayNumberOfBidderData.dat", header = T)


# PART A

#drop intercept
data_nointercept <- data[,-2]

glm_model <- glm(nBids ~ ., family = poisson, data = data_nointercept)
summary(glm_model)
```

```
##
## Call:
## glm(formula = nBids ~ ., family = poisson, data = data_nointercept)
##
## 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
```

Looking at the glm model output, we can see that the most significant features can be decided based on high abs(β) value with very small p-values. They are listed below:

1. MinBidShare
2. Intercept
3. Sealed

4. VerifyID
5. LogBook

PART B

```r
# PART B
#split response and features
response <- as.matrix(data$nBids)
covariates <- as.matrix(data[,2:10])
```

Log-Likelihood:

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

$$P(y_i|x_i,\beta) = \prod_i \frac{exp(y_i * x_i^T\beta) * exp(-e^{x_i^T\beta})}{y_i!}$$

$$L(\beta|X,Y) = \prod_i \frac{exp(y_i * x_i^T\beta) * exp(-e^{x_i^T\beta})}{y_i!}$$

$$taking\ logs\ both\ sides$$

$$logL(\beta|X,Y) = \sum_i \left(y_i * x_i^T\beta - e^{x_i^T\beta} - \cancel{log(y_i!)}\right)$$

$$logL(\beta|X,Y) = \sum_i \left(y_i * x_i^T\beta - e^{x_i^T\beta}\right)$$

Zellner's g-Prior:

$$\beta \sim N(0, 100 * (X^TX)^{-1})$$

```r
#log posterior

logPost <- function(beta,
                    X = covariates,
                    Y = response){

  loglik <- sum(Y * (X %*% beta) - exp(X %*% beta))
  logprior <- dmvnorm(t(beta), mean = matrix(0, nrow = ncol(X)), sigma = (100*(solve(t(X) %*% X))), log
  return(loglik + logprior)

}

#optimize log posterior

opt_res <- optim(par = matrix(1, nrow = 9),
                 fn = logPost,
                 method = "BFGS",
                 control = list(fnscale = -1),
                 hessian = TRUE)

beta_mode <- opt_res$par
inv_jacobian <- -solve(opt_res$hessian)

rownames(beta_mode) <- colnames(covariates)
```

```
t(beta_mode)
```

```
##        Const PowerSeller   VerifyID    Sealed     Minblem    MajBlem    LargNeg
## [1,] 1.06981 -0.02050925 -0.3928761 0.4434933 -0.05235654 -0.2212323 0.07047041
##         LogBook MinBidShare
## [1,] -0.1202723   -1.892074
```

```
glm_model$coefficients
```

```
## (Intercept) PowerSeller    VerifyID      Sealed     Minblem     MajBlem
##  1.07244206 -0.02054076 -0.39451647  0.44384257 -0.05219829 -0.22087119
##     LargNeg     LogBook MinBidShare
##  0.07067246 -0.12067761 -1.89409664
```

We can see that the approximate posterior mode is very close to the MLE from the glm model.

PART C

The metropolis algorithm will take any arbitrary posterior, which in this case that will be the log posterior & the tuning parameter c. The proposal density is given to be multivariate normal:

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

Parameter selection:

Target Density Prior: For a fair comparison with part B of this question, we choose the same values as the prior parameters for beta as in the previous part.

Proposal Density: "Σ & θ^{i-1} are initialized with the values calculated at the mode of the posterior in the previous step.

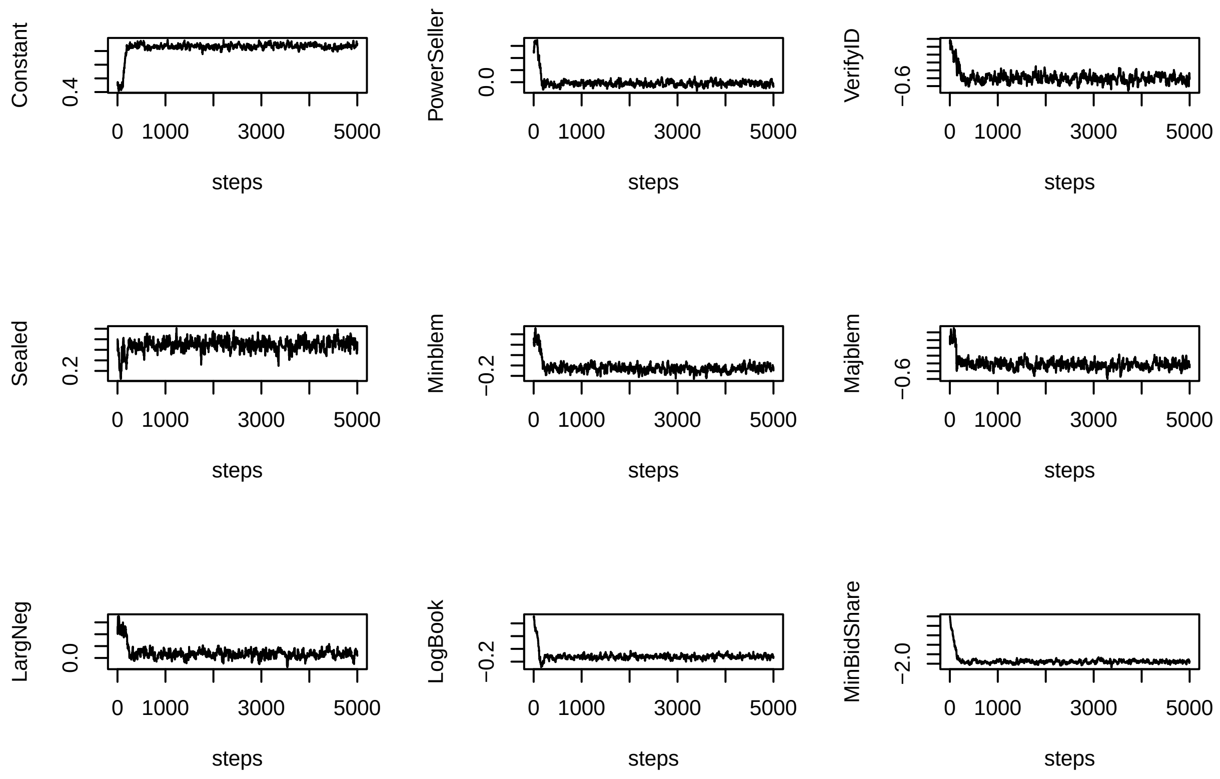
```
# PART C

# Metropolis Random Walk function
RMW_func <- function(target_density,
                     c,
                     theta_i_1,
                     sigma_proposal,
                     steps,
                     X,
                     Y){

  set.seed(12345)

  # initialize result matrix
  result <- matrix(t(theta_i_1), ncol = 9)

  accept = 0

  for (i in 1:steps) {

    #sample from the proposal distribution
    theta_p <- rmvnorm(1,
                       mean = as.vector(theta_i_1),
                       sigma = c*sigma_proposal)
    #calculate the ratio for the acceptance probability
    ratio = ((target_density(as.vector(theta_p), X, Y))
             - (target_density(as.vector(theta_i_1), X, Y)))

    #since target & proposal is defined in log form, we exponentiate to revert
    ratio = exp(ratio)

    #calculate alpha
    alpha = min(1,ratio)

    #draw from uniform
    u <- runif(1)

    #run test
    if(u < alpha){
      accept = accept + 1
      theta_i_1 = theta_p
      result <- rbind(result, theta_i_1)
    }
    else {
      result = rbind(result, as.vector(theta_i_1))
    }

  }

  return(list(result = result, acceptance = accept/steps))

}
```

```
# choose same params for the prior of target density
# choose data observed at mode for theta(i-1) & sigma for proposal density
sigma_proposal <- inv_jacobian
#theta_init <- beta_mode
theta_init <- matrix(rep(0.5,9), nrow = 9)

test_mrw <- RMW_func(target_density = logPost,
                     c = 0.6,
                     theta_i_1 = theta_init,
                     sigma_proposal = sigma_proposal,
                     steps = 5000,
                     X = covariates,
                     Y = response)

MRW_coeff <- test_mrw$result
MRW_coeff_means <- apply(MRW_coeff, 2, mean)
names(MRW_coeff_means) <- rownames(beta_mode)
MRW_coeff_means
```

```
##        Const  PowerSeller     VerifyID       Sealed      Minblem      MajBlem
##  1.053535037 -0.006066631 -0.384556715  0.446040824 -0.040732819 -0.211531073
##      LargNeg      LogBook  MinBidShare
##  0.081685121 -0.115261557 -1.855180954
```

The choice of initial values of the θ_{i-1} has an impact on how many steps we need for convergence in the random walk. When we chose the posterior mode from the previous analysis, then understandably we needed very few steps for convergence, since the random walk algorithm doesn't have to move too much farther from the initial values to get to the actual mode. But if we initiate with very randomly chosen values for eg. 0, then we have to adjust accordingly for the burn in period. So, in order to show convergence of parameters properly, we chose a starting point of 0.5 for all coefficients. We are able to see convergence pretty soon on all chains, but to get a fairly stable result we need to run the sampler for 5000 steps, we are getting coefficient values in the neighborhood of what we got earlier from MLE and Bayesian analysis. We played around with different values of the tuning parameter 'c' to get a fair acceptance rate. With a value of 0.6, we are able to hit an acceptance rate of ~28% which is within the prescribed range. We plot the convergence of the coefficients as follows:

```
colnames(MRW_coeff) <- rownames(beta_mode)

par(mfrow = c(3,3))

plot(MRW_coeff[,1], type = 'l', ylab = "Constant", xlab = "steps")
plot(MRW_coeff[,2], type = 'l', ylab = "PowerSeller", xlab = "steps")
plot(MRW_coeff[,3], type = 'l', ylab = "VerifyID", xlab = "steps")
plot(MRW_coeff[,4], type = 'l', ylab = "Sealed", xlab = "steps")
plot(MRW_coeff[,5], type = 'l', ylab = "Minblem", xlab = "steps")
plot(MRW_coeff[,6], type = 'l', ylab = "Majblem", xlab = "steps")
plot(MRW_coeff[,7], type = 'l', ylab = "LargNeg", xlab = "steps")
plot(MRW_coeff[,8], type = 'l', ylab = "LogBook", xlab = "steps")
plot(MRW_coeff[,9], type = 'l', ylab = "MinBidShare", xlab = "steps")
```



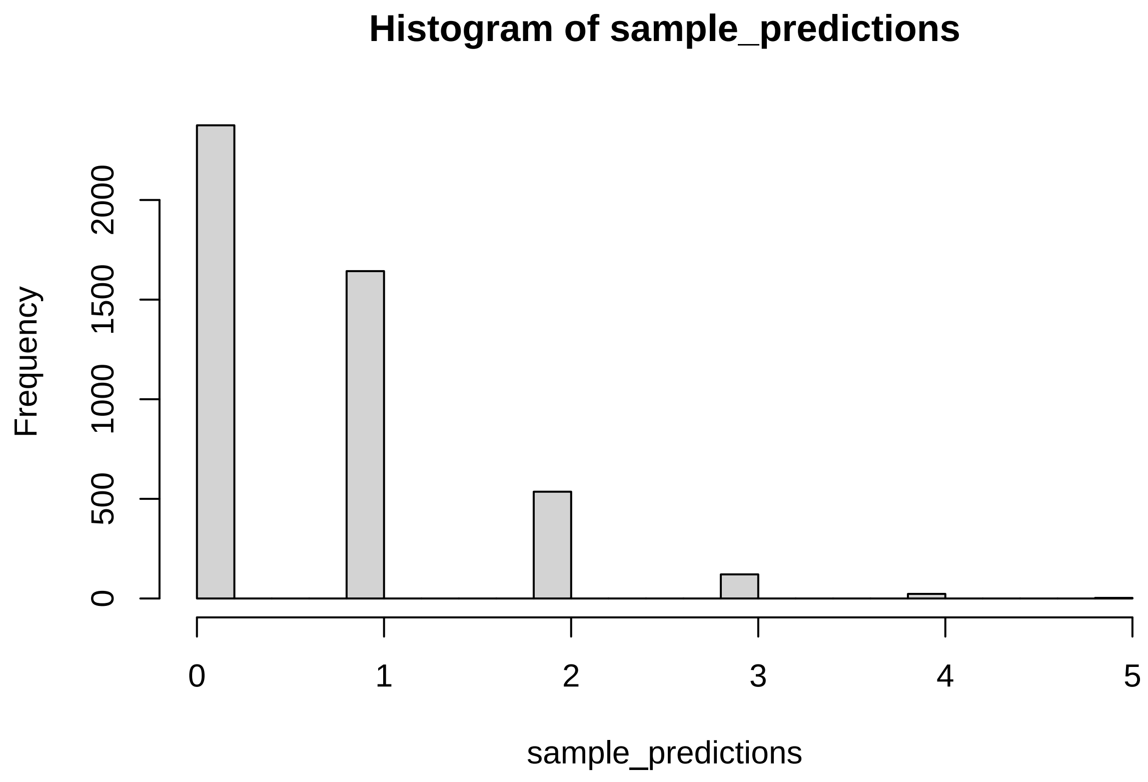
## PART D

We'll now use the MCMC draws of the coefficients from the previous step as the β parameters to the given poisson regression model:

$$y_i|\beta \ \widetilde{iid} \ Poisson[exp(x_i^T\beta)]$$

To find the distribution of λ as a linear combination, we will use the given features as X in the above model and the MCMC draws as βs. We'll then use the fitted λs to draw from the poisson distribution to get a posterior prediction for the model.

```
# PART D

sample <- as.matrix(c(1, 1, 0, 1, 0, 1, 0, 1.2, 0.8), nrow = 9)
names(sample) <- rownames(beta_mode)
posterior_samples <- MRW_coeff[301:length(MRW_coeff[,1]),]

linear_predictions <- posterior_samples %*% sample
lambda_predictons <- exp(linear_predictions)

sample_predictions <- c()

set.seed(12345)
for (i in 1:length(lambda_predictons)){
  sample_predictions[i] <- rpois(1, lambda_predictons[i])
}

hist(sample_predictions, breaks = 20)
```



```
cat("\n", "Probability of no bids on the test data:",length(which(sample_predictions == 0))/length(samp
##
##  Probability of no bids on the test data: 0.5052117
```

Based on our draws we can say that the probability that there are no bids for a listing with these features is about 50%.

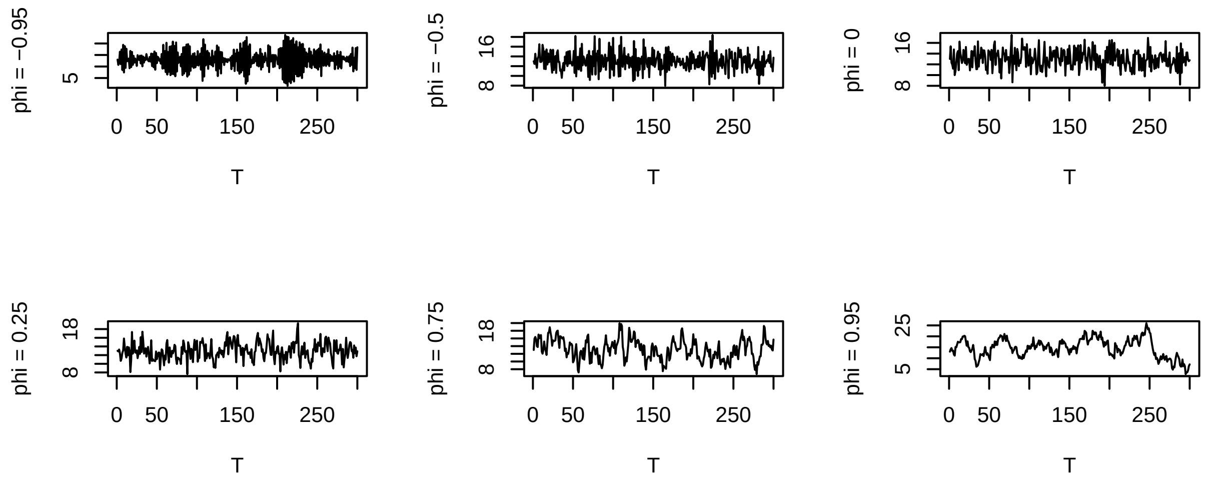
TIME SERIES MODELS

AR(1) Process:

$$x_t = \mu + \phi(x_{t-1} - \mu) + \varepsilon_t \; ; \; where \; \varepsilon_t \; \widetilde{iid} \; N(0, \sigma^2)$$

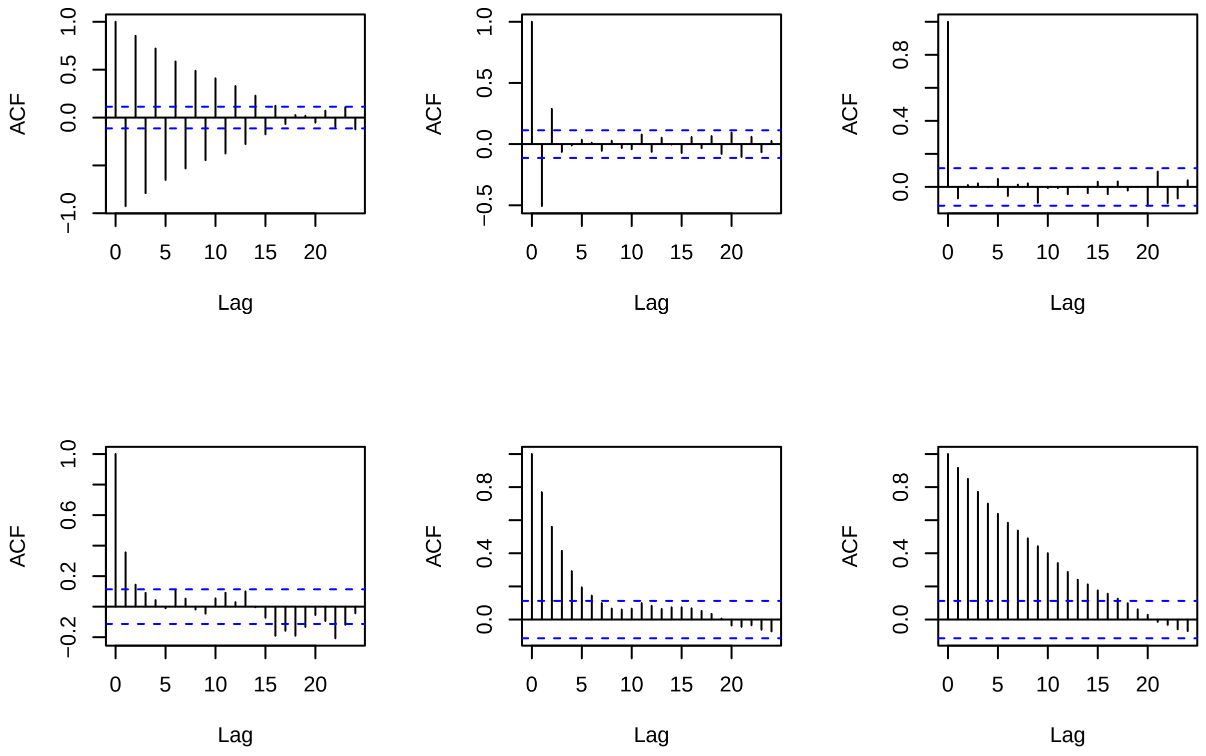
PART A

```
# QUES 3: TIME SERIES

rm(list = ls())

AR <- function(mu = 13,
               phi = c(-0.95, -0.5, 0, 0.25, 0.75, 0.95),
               sigma2 = 3,
               T = 300) {

  chains = data.frame(0, ncol = length(phi))

  for (i in 1:length(phi)){

    x_t <- mu
    chains[1,i] = x_t

    for (j in 2:T){

      x_t = mu + phi[i] * (x_t - mu) + rnorm(1, 0 , sqrt(sigma2))
      chains[j,i] = x_t

    }

  }

  colnames(chains) <- paste("phi_",phi)

  return(chains)

}

test_AR1 <- AR()

par(mfrow = c(3,3))
plot(y = test_AR1[,1], x = c(1:300), type = 'l', ylab = "phi = -0.95", xlab = "T")
plot(y = test_AR1[,2], x = c(1:300), type = 'l', ylab = "phi = -0.5", xlab = "T")
plot(y = test_AR1[,3], x = c(1:300), type = 'l', ylab = "phi = 0", xlab = "T")
plot(y = test_AR1[,4], x = c(1:300), type = 'l', ylab = "phi = 0.25", xlab = "T")
plot(y = test_AR1[,5], x = c(1:300), type = 'l', ylab = "phi = 0.75", xlab = "T")
plot(y = test_AR1[,6], x = c(1:300), type = 'l', ylab = "phi = 0.95", xlab = "T")
```



For lower values of ϕ we can see that the error term of the AR(1) process i.e. ε_t has a much larger impact on the next prediction as the μ dependent terms cancel out each others effect on the prediction, thus resulting in a negative correlation. But as we go towards a larger value of ϕ, the first 2 terms dependent on μ become additive and we see a stronger positive correlation in the chain for every subsequent prediction. To better depict whats going on with the chains, we plot the ACFs for them:

```
par(mfrow = c(2,3))
sapply(test_AR1, function(x) acf(x, main=""))
```



```
##        phi_ -0.95    phi_ -0.5     phi_ 0        phi_ 0.25     phi_ 0.75
## acf    numeric,25    numeric,25    numeric,25    numeric,25    numeric,25
## type   "correlation" "correlation" "correlation" "correlation" "correlation"
## n.used 300           300           300           300           300
## lag    numeric,25    numeric,25    numeric,25    numeric,25    numeric,25
## series "x"           "x"           "x"           "x"           "x"
## snames NULL          NULL          NULL          NULL          NULL
##        phi_ 0.95
## acf    numeric,25
## type   "correlation"
## n.used 300
## lag    numeric,25
## series "x"
## snames NULL
```

As we can infer from the ACF plots, for a positive value of the ϕ with increasing lag, the autocorrelation keeps decreasing. Whereas for a negative value of ϕ the autocorrelation keeps oscillating between consecutive lags. For $\phi = 0$, the successive values are independent of each other as the only variation is due to the normally distributed white noise.

PART B

Now we are assuming the three parameters μ, σ^2 *and* ϕ to be unknown and are going to assume non-informative priors for them as follows:

1. $\mu \sim N(0, 50)$; going with what we have used earlier for modelling μ for a normal model
2. $\sigma^2 \sim Inv - \chi^2(1, 10)$; since we used the inverse-χ^2 distribution to model σ^2 for a normal model

3. $\phi \sim Unif(-1, 1)$; since we know the interval for phi

Since, it can be shown that if the white noise ε_t is gaussian then the AR(1) process for x_t is also gaussian with the following parameters:

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

```
# PART B

model_AR <- AR(phi = c(0.2, 0.95))

StanModel = '
data {
  int<lower=0> T; // Number of observations
  vector[T] x;  // indicating the chain x over T obs
}
parameters {
  real mu;
  real<lower = 0> sigma2;
  real<lower = -1, upper = 1> phi;
}
model {
  mu ~ normal(0,50); // Normal with mean 0, st.dev. 50
  sigma2 ~ scaled_inv_chi_square(1,10); // Scaled-inv-chi2 with nu 1,sigma 10
  phi ~ uniform(-1,1);

  for(i in 2:T){
    x[i] ~ normal(mu + phi * (x[i-1] - mu), sqrt(sigma2));
  }
}'

#fit model for phi 0.2
fit_0.2 = stan(model_code = StanModel,
               data = list(x = model_AR$`phi_ 0.2`, T = 300),
               warmup = 1000,
               iter = 2000,
               chains = 4)
```

```
##
## SAMPLING FOR MODEL 'c4b76683c55ce82da98055a195c28b7c' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0 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: 0.15 seconds (Warm-up)
## Chain 1:                0.157 seconds (Sampling)
## Chain 1:                0.307 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'c4b76683c55ce82da98055a195c28b7c' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0 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: 0.132 seconds (Warm-up)
## Chain 2:                0.161 seconds (Sampling)
## Chain 2:                0.293 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'c4b76683c55ce82da98055a195c28b7c' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0 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: 0.15 seconds (Warm-up)
## Chain 3:                0.156 seconds (Sampling)
## Chain 3:                0.306 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'c4b76683c55ce82da98055a195c28b7c' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0 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: 0.162 seconds (Warm-up)
## Chain 4:                0.141 seconds (Sampling)
## Chain 4:                0.303 seconds (Total)
## Chain 4:
```

```r
#fit model for phi 0.95
fit_0.95 = stan(model_code = StanModel,
               data = list(x = model_AR$`phi_ 0.95`, T = 300),
               warmup = 1000,
               iter = 2000,
               chains = 4)
```

```
##
## SAMPLING FOR MODEL 'c4b76683c55ce82da98055a195c28b7c' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0 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: 0.218 seconds (Warm-up)
## Chain 1:                0.163 seconds (Sampling)
## Chain 1:                0.381 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'c4b76683c55ce82da98055a195c28b7c' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0 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: 0.21 seconds (Warm-up)
## Chain 2:                0.142 seconds (Sampling)
## Chain 2:                0.352 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'c4b76683c55ce82da98055a195c28b7c' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0 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: 0.242 seconds (Warm-up)
## Chain 3:                0.222 seconds (Sampling)
## Chain 3:                0.464 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'c4b76683c55ce82da98055a195c28b7c' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0 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: 0.278 seconds (Warm-up)
## Chain 4:                0.269 seconds (Sampling)
## Chain 4:                0.547 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
```

```
#PART B.i
#get posterior samples and means
post_samples_0.2 <- extract(fit_0.2)
post_mean_0.2 <- get_posterior_mean(fit_0.2)

post_samples_0.95 <- extract(fit_0.95)
post_mean_0.95 <- get_posterior_mean(fit_0.95)

cat("\n")
```

```
print("Posterior Means when phi = 0.2 :")
```

```
## [1] "Posterior Means when phi = 0.2 :"
```

```
cat("\n")
```

```
print(post_mean_0.2)
```

```
##        mean-chain:1 mean-chain:2 mean-chain:3 mean-chain:4 mean-all chains
## mu       12.9204957   12.9303981   12.9249563   12.9247568      12.9251517
## sigma2    3.3237206    3.3251022    3.3289650    3.3377813       3.3288922
## phi       0.3106549    0.3088885    0.3086545    0.3061329       0.3085827
## lp__   -330.7501262 -330.7647463 -330.7196701 -330.7139225    -330.7371163
```

```r
cat("\n")
```

```r
print("Posterior Means when phi = 0.95 :")
```

```
## [1] "Posterior Means when phi = 0.95 :"
```

```r
cat("\n")
```

```r
print(post_mean_0.95)
```

```
##        mean-chain:1 mean-chain:2 mean-chain:3 mean-chain:4 mean-all chains
## mu       11.7034128   12.5272756   12.2439620   12.1933107       12.166990
## sigma2    3.2775391    3.2854598    3.3100297    3.2613159        3.283586
## phi       0.9140362    0.9174577    0.9188501    0.9196601        0.917501
## lp__   -330.8182469 -330.9610729 -330.7634806 -331.0478086     -330.897652
```

```r
post_samples_0.2_df <- data.frame(mu = post_samples_0.2$mu,
                                  sigma2 = post_samples_0.2$sigma2,
                                  phi = post_samples_0.2$phi)

CI_0.2 <- sapply(post_samples_0.2_df, function(x) quantile(x, probs=c(0.025, 0.975)))

post_samples_0.95_df <- data.frame(mu = post_samples_0.95$mu,
                                   sigma2 = post_samples_0.95$sigma2,
                                   phi = post_samples_0.95$phi)

CI_0.95 <- sapply(post_samples_0.95_df, function(x) quantile(x, probs=c(0.025, 0.975)))

cat("\n")
```

```r
print("Posterior 95% CI when phi = 0.2 :")
```

```
## [1] "Posterior 95% CI when phi = 0.2 :"
```

```r
cat("\n")
```

```r
print(CI_0.2)
```

```
##             mu   sigma2       phi
## 2.5%  12.62712 2.826965 0.1921938
## 97.5% 13.22725 3.894716 0.4257022
```

```r
cat("\n")
```

```r
print("Posterior 95% CI when phi = 0.95 :")
```

```
## [1] "Posterior 95% CI when phi = 0.95 :"
```

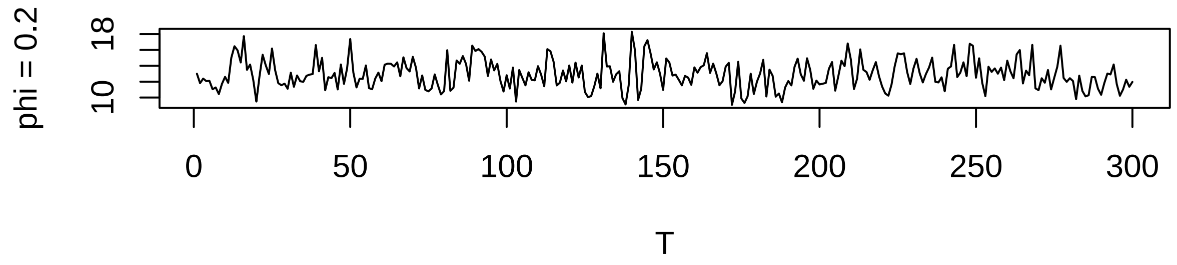
```r
cat("\n")
```

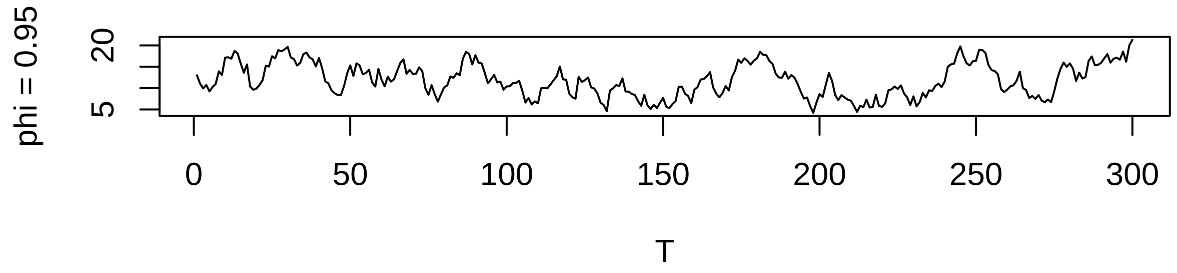
```r
print(CI_0.95)
```

```
##              mu   sigma2       phi
## 2.5%   8.707761 2.803629 0.8600886
## 97.5% 16.173745 3.862484 0.9786845
```

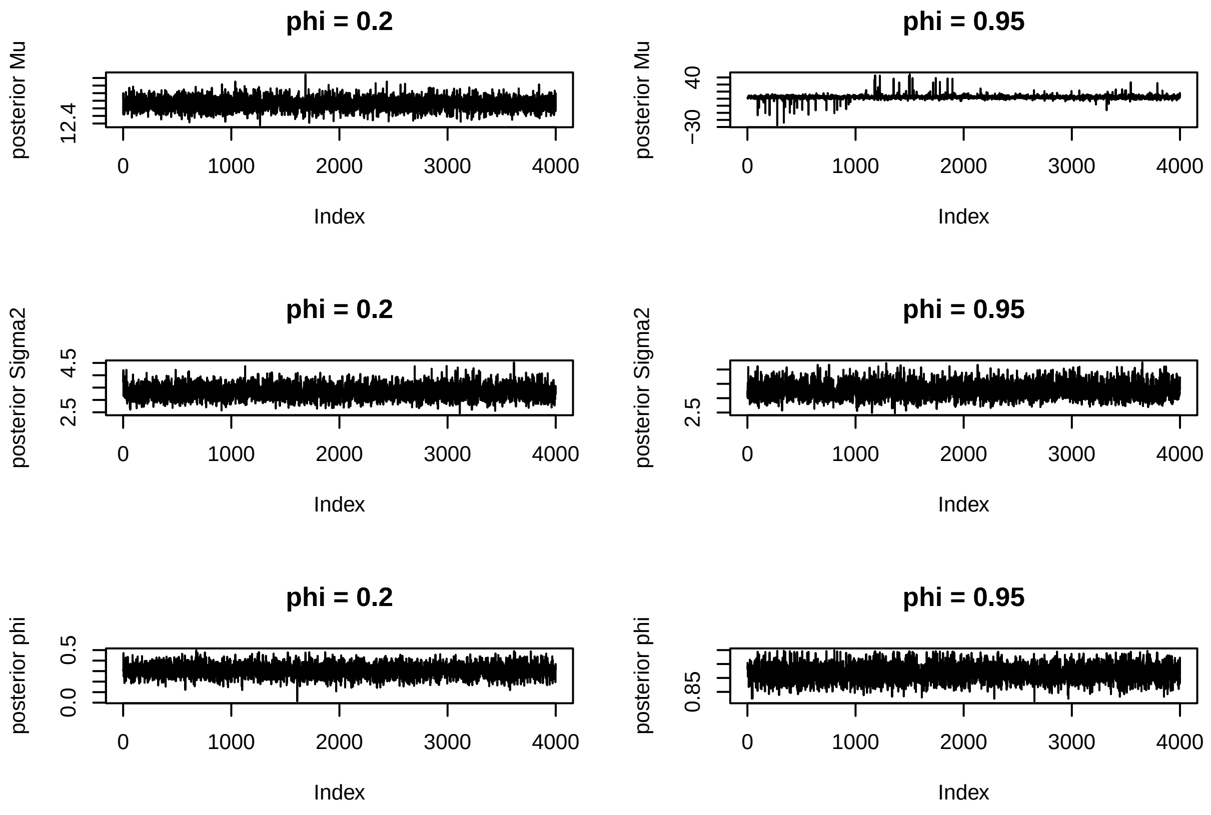
Yes, we are able to estimate the true values of the parameters with reasonable width of CIs. But in case of the model for $\phi = 0.95$, the CI for μ is very wide as compared to the other model.

```r
#PART B.ii.

#plotting the two models
par(mfrow = c(2,1))
plot(y = model_AR[,1], x = c(1:300), type = 'l', ylab = "phi = 0.2", xlab = "T")
plot(y = model_AR[,2], x = c(1:300), type = 'l', ylab = "phi = 0.95", xlab = "T")
```

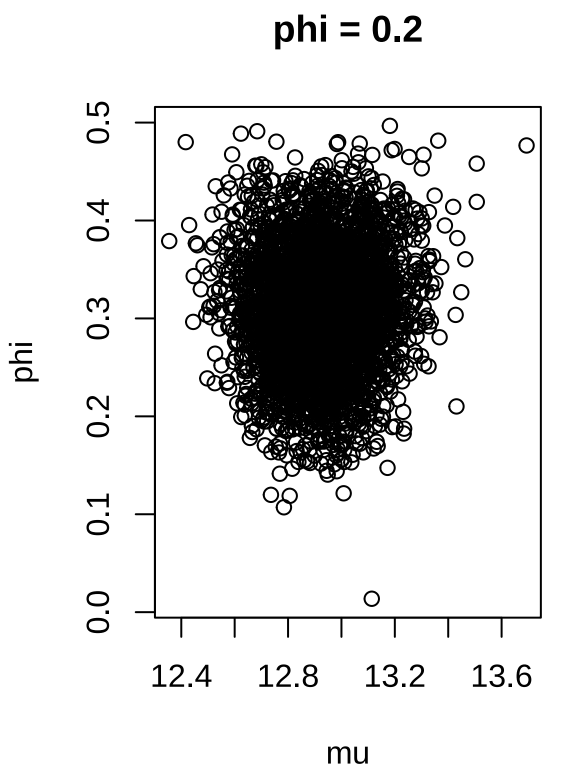


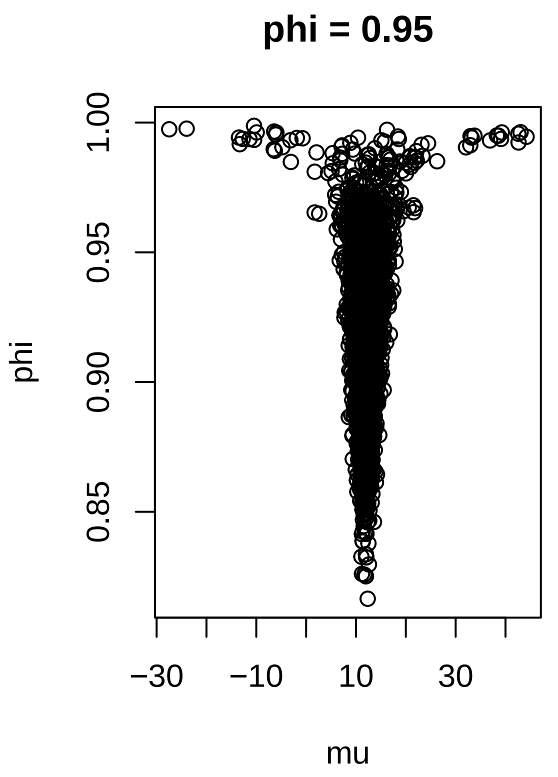


```r
#checking convergence of the samplers
par(mfrow = c(3,2))
plot(post_samples_0.2$mu, type = 'l', ylab = "posterior Mu", main = "phi = 0.2")
plot(post_samples_0.95$mu, type = 'l', ylab = "posterior Mu", main = "phi = 0.95")
plot(post_samples_0.2$sigma2, type = 'l', ylab = "posterior Sigma2", main = "phi = 0.2")
plot(post_samples_0.95$sigma2, type = 'l', ylab = "posterior Sigma2", main = "phi = 0.95")
plot(post_samples_0.2$phi, type = 'l', ylab = "posterior phi", main = "phi = 0.2")
plot(post_samples_0.95$phi, type = 'l', ylab = "posterior phi", main = "phi = 0.95")
```



```r
#joint posterior
par(mfrow = c(1,2))
plot(x = post_samples_0.2_df$mu,
     y = post_samples_0.2_df$phi,
     main = "phi = 0.2",
     xlab = "mu",
     ylab = "phi")
plot(x = post_samples_0.95_df$mu,
     y = post_samples_0.95_df$phi,
     main = "phi = 0.95",
     xlab = "mu",
     ylab = "phi")
```





All the parameter chains are converging pretty quickly. From the joint posterior it is evident that the estimate for μ is more accurate for the model with $\phi = 0.2$ than the other model. An explanation for this can be that since the sample mean in an AR(1) process with a shifted mean (as is the case here), is $\sim N(0, \sigma^2/(1-\phi)^2)$, the CI for the sample mean is then given by:

$$C.I._{.95}(\mu) = \bar{x}_n \pm \frac{1.96}{\sqrt{n}}\Big(\frac{\sigma}{|1-\phi|}\Big)$$

From this expression we can concur that the CI for the mean will be much wider when the ϕ is small. From the joint posterior plots we can draw a similar inference that as the value of *phi* starts approaching 1, the stationary properties start getting lost and the distribution of μ becomes very wide.

APPENDIX: CODE

```
knitr::opts_chunk$set(echo = TRUE)
library(mvtnorm)
library(LaplacesDemon)
library(rstan)

# QUES 1: GIBBS SAMPLER

prec_data <- readRDS("precipitation.rds")

log_data <- log(prec_data)

par(mfrow = c(1,2))
hist(log_data, breaks = 40, prob = TRUE)
hist(prec_data, breaks = 40, prob = TRUE)

#PART A

# setting up the initial values of the priors

n = length(log_data)

# for mu
mu0 = mean(log_data)
tau20 = 1
# for sigma
sig20 = var(log_data)
nu0 = 1

set.seed(1234)

#Gibbs Sampling function
Gibbs_sampler <- function(steps = 500,
                          mu_0 = mu0,
                          tau2_0 = tau20,
                          sig2_0 = sig20,
                          nu_0 = nu0,
                          n = length(log_data)){

  #initial value for gibbs sampler
  mu_i = rnorm(1, mean = mu_0, sd = sqrt(tau2_0))
  sigma2_i =  rinvchisq(1, df = nu_0, scale = sig2_0)

  #initialize result DF
  sample_results = data.frame("mu" = mu_i,
                              "sigma2" = sigma2_i)

  #initialize loop over specified gibbs steps
  for (i in 1:steps) {

    #gibbs step for mu
    w = n/sigma2_i / (n/sigma2_i + 1/tau2_0)
    mu_n = w * mean(log_data) + (1-w) * mu_0
    tau2_n = (n/sigma2_i + 1/tau2_0)^-1
    mu_i = rnorm(1, mean = mu_n, sd = sqrt(tau2_n))

    #gibbs step for sigma
    nu_n = nu_0 + n
    scale_term = (nu_0 * sigma2_i + sum((log_data - mu_i)^2))/nu_n
    sigma_i = rinvchisq(1, df = nu_n, scale = scale_term)

    #IF
    #IF_mu = 1 + 2*(sum(cor(sample_results[i,1], mu_i)))

    #save step values to DF
    sample_results = rbind.data.frame(sample_results, c(mu_i, sigma_i))

  }

  return(sample_results)

}

joint_posterior <- Gibbs_sampler(steps = 500)

#autocorrelation
par(mfrow = c(2,1))
mu_Gibbs <- acf(joint_posterior[,1])
sig2_Gibbs <- acf(joint_posterior[,2])

#inefficiency factor
IF_Gibbs_mu <- 1+2*sum(mu_Gibbs$acf[-1])
IF_Gibbs_sig2 <- 1+2*sum(sig2_Gibbs$acf[-1])

cat("\n", "IF of mu:", round(IF_Gibbs_mu,3))
cat("\n", "IF of sigma2:", round(IF_Gibbs_sig2,3))

#trace plot
par(mfrow = c(2,1))
plot(1:length(joint_posterior$mu), joint_posterior[,1], type = "l",col="blue", xlab = "steps", ylab = "
plot(1:length(joint_posterior$sigma2), joint_posterior[,2], type = "l",col="blue", xlab = "steps", ylab

# PART B

set.seed(1234)
post_pred <- c()
post_burnremoved <- joint_posterior[51:length(joint_posterior$mu),]
for (i in 1:(length(post_burnremoved$mu))){
  post_pred[i] = rnorm(1, mean = post_burnremoved[i,1], sd = sqrt(post_burnremoved[i,2]))
}

e_post_draws <- exp(post_pred)


plot(density(prec_data),
     col = "blue",
     ylim = c(0,0.2),
     xlim = c(-10,100),
     main = "Posterior Prediction vs Y")
lines(density(e_post_draws),
      col = "red")

# QUES 2: METROPOLIS RW

rm(list = ls())

data <- read.table("eBayNumberOfBidderData.dat", header = T)


# PART A

#drop intercept
data_nointercept <- data[,-2]

glm_model <- glm(nBids ~ ., family = poisson, data = data_nointercept)
summary(glm_model)

# PART B
#split response and features
response <- as.matrix(data$nBids)
covariates <- as.matrix(data[,2:10])

#log posterior

logPost <- function(beta,
                    X = covariates,
                    Y = response){

  loglik <- sum(Y * (X %*% beta) - exp(X %*% beta))
  logprior <- dmvnorm(t(beta), mean = matrix(0, nrow = ncol(X)), sigma = (100*(solve(t(X) %*% X)), log
  return(loglik + logprior)

}

#optimize log posterior

opt_res <- optim(par = matrix(1, nrow = 9),
                 fn = logPost,
                 method = "BFGS",
                 control = list(fnscale = -1),
                 hessian = TRUE)

beta_mode <- opt_res$par
inv_jacobian <- -solve(opt_res$hessian)

rownames(beta_mode) <- colnames(covariates)

t(beta_mode)
```

```r
glm_model$coefficients
# PART C

# Metropolis Random Walk function
RMW_func <- function(target_density,
                     c,
                     theta_i_1,
                     sigma_proposal,
                     steps,
                     X,
                     Y){

  set.seed(12345)

  # initialize result matrix
  result <- matrix(t(theta_i_1), ncol = 9)

  accept = 0

  for (i in 1:steps) {

    #sample from the proposal distribution
    theta_p <- rmvnorm(1,
                       mean = as.vector(theta_i_1),
                       sigma = c*sigma_proposal)

    #calculate the ratio for the acceptance probability
    ratio = ((target_density(as.vector(theta_p), X, Y))
             - (target_density(as.vector(theta_i_1), X, Y)))

    #since target & proposal is defined in log form, we exponentiate to revert
    ratio = exp(ratio)

    #calculate alpha
    alpha = min(1,ratio)

    #draw from uniform
    u <- runif(1)

    #run test
    if(u < alpha){
      accept = accept + 1
      theta_i_1 = theta_p
      result <- rbind(result, theta_i_1)
    }
    else {
      result = rbind(result, as.vector(theta_i_1))
    }

  }

  return(list(result = result, acceptance = accept/steps))
```

```
}
# choose same params for the prior of target density
# choose data observed at mode for theta(i-1) & sigma for proposal density
sigma_proposal <- inv_jacobian
#theta_init <- beta_mode
theta_init <- matrix(rep(0.5,9), nrow = 9)

test_mrw <- RMW_func(target_density = logPost,
                     c = 0.6,
                     theta_i_1 = theta_init,
                     sigma_proposal = sigma_proposal,
                     steps = 5000,
                     X = covariates,
                     Y = response)

MRW_coeff <- test_mrw$result
MRW_coeff_means <- apply(MRW_coeff, 2, mean)
names(MRW_coeff_means) <- rownames(beta_mode)
MRW_coeff_means

colnames(MRW_coeff) <- rownames(beta_mode)

par(mfrow = c(3,3))

plot(MRW_coeff[,1], type = 'l', ylab = "Constant", xlab = "steps")
plot(MRW_coeff[,2], type = 'l', ylab = "PowerSeller", xlab = "steps")
plot(MRW_coeff[,3], type = 'l', ylab = "VerifyID", xlab = "steps")
plot(MRW_coeff[,4], type = 'l', ylab = "Sealed", xlab = "steps")
plot(MRW_coeff[,5], type = 'l', ylab = "Minblem", xlab = "steps")
plot(MRW_coeff[,6], type = 'l', ylab = "Majblem", xlab = "steps")
plot(MRW_coeff[,7], type = 'l', ylab = "LargNeg", xlab = "steps")
plot(MRW_coeff[,8], type = 'l', ylab = "LogBook", xlab = "steps")
plot(MRW_coeff[,9], type = 'l', ylab = "MinBidShare", xlab = "steps")

# PART D

sample <- as.matrix(c(1, 1, 0, 1, 0, 1, 0, 1.2, 0.8), nrow = 9)
names(sample) <- rownames(beta_mode)
posterior_samples <- MRW_coeff[301:length(MRW_coeff[,1]),]

linear_predictions <- posterior_samples %*% sample
lambda_predictons <- exp(linear_predictions)

sample_predictions <- c()

set.seed(12345)
for (i in 1:length(lambda_predictons)){
  sample_predictions[i] <- rpois(1, lambda_predictons[i])
}

hist(sample_predictions, breaks = 20)

cat("\n", "Probability of no bids on the test data:",length(which(sample_predictions == 0))/length(sampl
```

```r
# QUES 3: TIME SERIES

rm(list = ls())

AR <- function(mu = 13,
               phi = c(-0.95, -0.5, 0, 0.25, 0.75, 0.95),
               sigma2 = 3,
               T = 300) {

  chains = data.frame(0, ncol = length(phi))

  for (i in 1:length(phi)){

    x_t <- mu
    chains[1,i] = x_t

    for (j in 2:T){

      x_t = mu + phi[i] * (x_t - mu) + rnorm(1, 0 , sqrt(sigma2))
      chains[j,i] = x_t

    }

  }

  colnames(chains) <- paste("phi_",phi)

  return(chains)

}

test_AR1 <- AR()

par(mfrow = c(3,3))
plot(y = test_AR1[,1], x = c(1:300), type = 'l', ylab = "phi = -0.95", xlab = "T")
plot(y = test_AR1[,2], x = c(1:300), type = 'l', ylab = "phi = -0.5", xlab = "T")
plot(y = test_AR1[,3], x = c(1:300), type = 'l', ylab = "phi = 0", xlab = "T")
plot(y = test_AR1[,4], x = c(1:300), type = 'l', ylab = "phi = 0.25", xlab = "T")
plot(y = test_AR1[,5], x = c(1:300), type = 'l', ylab = "phi = 0.75", xlab = "T")
plot(y = test_AR1[,6], x = c(1:300), type = 'l', ylab = "phi = 0.95", xlab = "T")

par(mfrow = c(2,3))
sapply(test_AR1, function(x) acf(x, main=""))

# PART B

model_AR <- AR(phi = c(0.2, 0.95))

StanModel = '
data {
  int<lower=0> T; // Number of observations
  vector[T] x;  // indicating the chain x over T obs
```

```
}
parameters {
  real mu;
  real<lower = 0> sigma2;
  real<lower = -1, upper = 1> phi;
}
model {
  mu ~ normal(0,50); // Normal with mean 0, st.dev. 50
  sigma2 ~ scaled_inv_chi_square(1,10); // Scaled-inv-chi2 with nu 1,sigma 10
  phi ~ uniform(-1,1);

  for(i in 2:T){
    x[i] ~ normal(mu + phi * (x[i-1] - mu), sqrt(sigma2));
  }
}'

#fit model for phi 0.2
fit_0.2 = stan(model_code = StanModel,
               data = list(x = model_AR$`phi_ 0.2`, T = 300),
               warmup = 1000,
               iter = 2000,
               chains = 4)

#fit model for phi 0.95
fit_0.95 = stan(model_code = StanModel,
               data = list(x = model_AR$`phi_ 0.95`, T = 300),
               warmup = 1000,
               iter = 2000,
               chains = 4)


#PART B.i
#get posterior samples and means
post_samples_0.2 <- extract(fit_0.2)
post_mean_0.2 <- get_posterior_mean(fit_0.2)

post_samples_0.95 <- extract(fit_0.95)
post_mean_0.95 <- get_posterior_mean(fit_0.95)

cat("\n")
print("Posterior Means when phi = 0.2 :")
cat("\n")
print(post_mean_0.2)


cat("\n")
print("Posterior Means when phi = 0.95 :")
cat("\n")
print(post_mean_0.95)




post_samples_0.2_df <- data.frame(mu = post_samples_0.2$mu,
                                  sigma2 = post_samples_0.2$sigma2,
                                  phi = post_samples_0.2$phi)

CI_0.2 <- sapply(post_samples_0.2_df, function(x) quantile(x, probs=c(0.025, 0.975)))

post_samples_0.95_df <- data.frame(mu = post_samples_0.95$mu,
                                  sigma2 = post_samples_0.95$sigma2,
                                  phi = post_samples_0.95$phi)

CI_0.95 <- sapply(post_samples_0.95_df, function(x) quantile(x, probs=c(0.025, 0.975)))

cat("\n")
print("Posterior 95% CI when phi = 0.2 :")
cat("\n")
print(CI_0.2)

cat("\n")
print("Posterior 95% CI when phi = 0.95 :")
cat("\n")
print(CI_0.95)

#PART B.ii.

#plotting the two models
par(mfrow = c(2,1))
plot(y = model_AR[,1], x = c(1:300), type = 'l', ylab = "phi = 0.2", xlab = "T")
plot(y = model_AR[,2], x = c(1:300), type = 'l', ylab = "phi = 0.95", xlab = "T")

#checking convergence of the samplers
par(mfrow = c(3,2))
plot(post_samples_0.2$mu, type = 'l', ylab = "posterior Mu", main = "phi = 0.2")
plot(post_samples_0.95$mu, type = 'l', ylab = "posterior Mu", main = "phi = 0.95")
plot(post_samples_0.2$sigma2, type = 'l', ylab = "posterior Sigma2", main = "phi = 0.2")
plot(post_samples_0.95$sigma2, type = 'l', ylab = "posterior Sigma2", main = "phi = 0.95")
plot(post_samples_0.2$phi, type = 'l', ylab = "posterior phi", main = "phi = 0.2")
plot(post_samples_0.95$phi, type = 'l', ylab = "posterior phi", main = "phi = 0.95")

#joint posterior
par(mfrow = c(1,2))
plot(x = post_samples_0.2_df$mu,
     y = post_samples_0.2_df$phi,
     main = "phi = 0.2",
     xlab = "mu",
     ylab = "phi")
plot(x = post_samples_0.95_df$mu,
     y = post_samples_0.95_df$phi,
     main = "phi = 0.95",
     xlab = "mu",
     ylab = "phi")
```