Bayesian Learning (732A91) Lab3 Report

Christoforos Spyretos (chrsp415) & Marketos Damigos (marda352)

Assignment 1 *Gibbs sampler for a normal model*

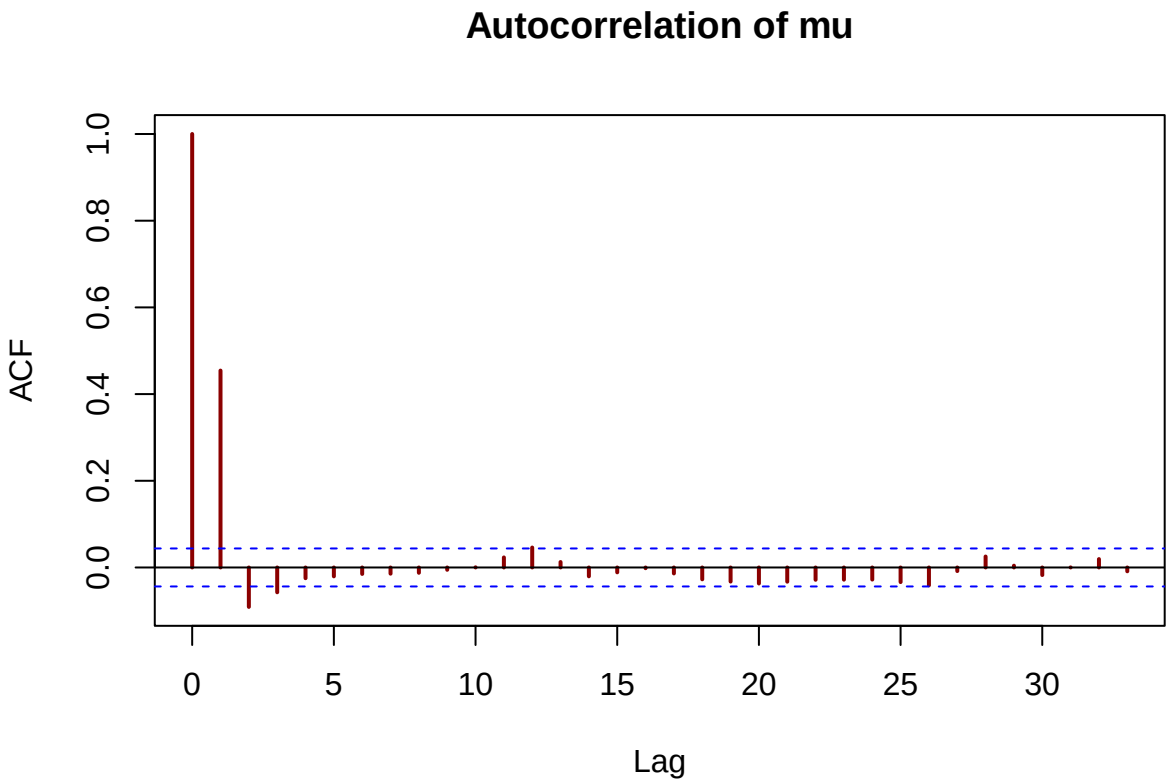
Task a)

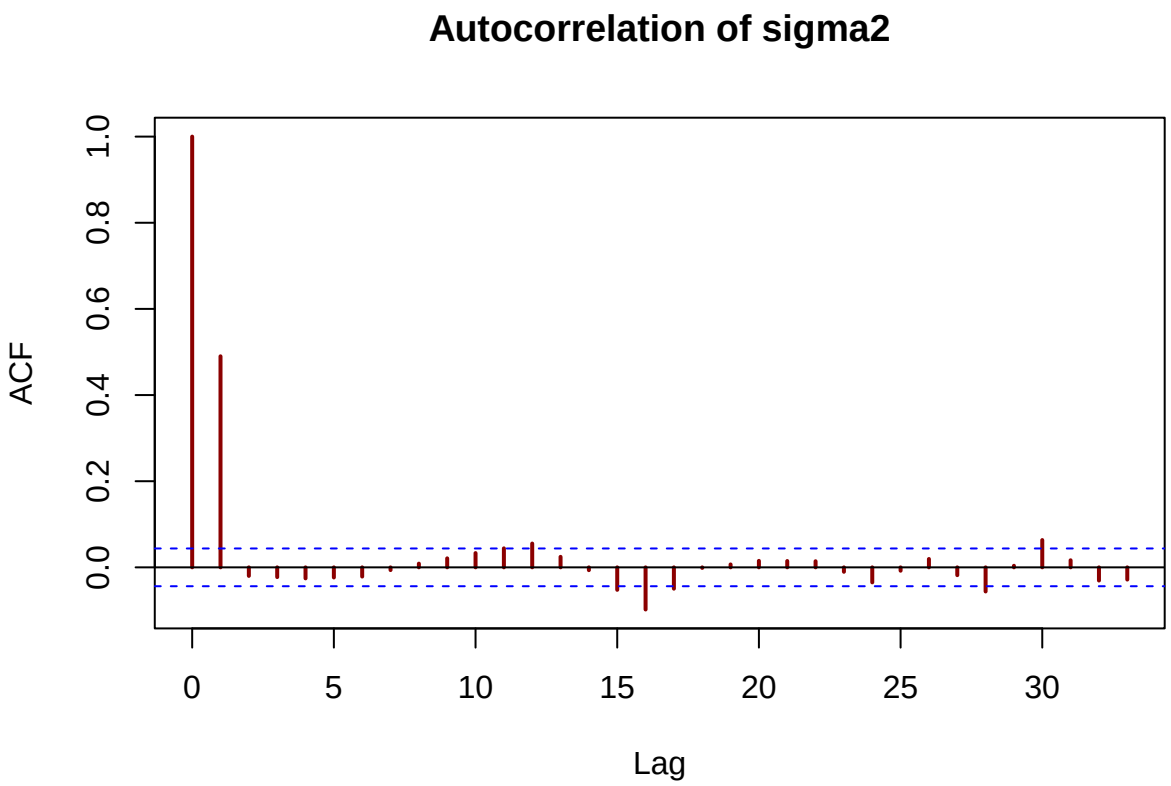
The full conditional posteriors are:

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

Where:

$$w = \frac{\frac{n}{\sigma^2}}{\frac{n}{\sigma^2} + \frac{1}{\tau_0^2}}$$
$$\mu_n = w\overline{x} + (1-w)\mu_0$$
$$\tau_n^2 = \frac{1}{\frac{n}{\sigma^2} + \frac{1}{\tau_0^2}}$$
$$\nu_n = \nu_0 + n$$





The above graphs illustrate the correlation coefficient of μ and σ^2 against the lag. The red bars illustrate the correlation coefficient of the parameters of each lag, and the blue lines represent the statistically significant boundaries. In both graphs, the ACF equals 1 for a lag 0, which is reasonable because the "first step" is always perfectly correlated with itself (the lag 0 autocorrelation is fixed at one by convention). More specifically,

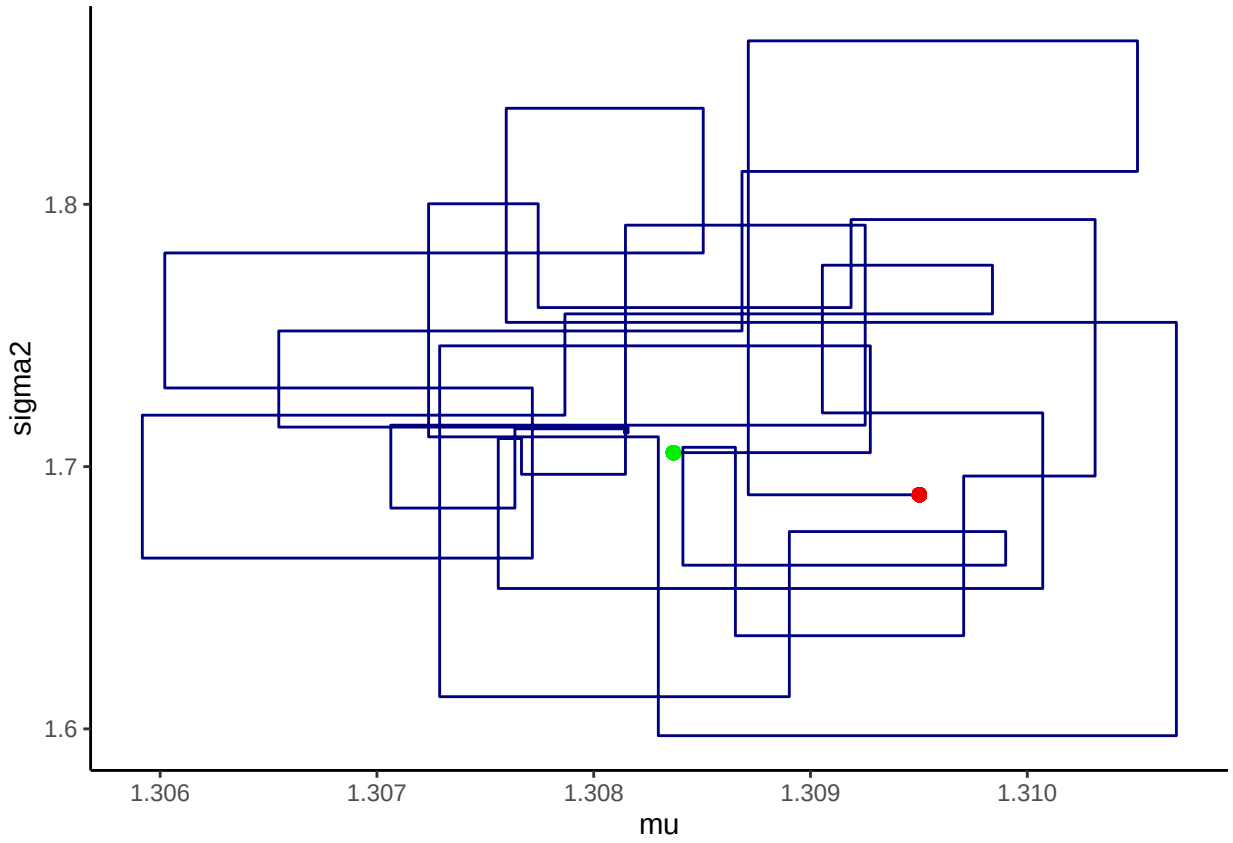
in the first graph, the correlation coefficient of μ is inside the boundaries when lag is greater than 3, which means the values are unrelated between them. Likewise, in the second graph, the correlation coefficient of σ^2 gets closer to 0 after lag is 2, which means the values are unrelated between them. Finally, however, the ACF value crosses the "boundaries" in some cases, which means that the variables are related when lag equals 12, 16 and 30.

The below table illustrates the inefficiency factors of μ and σ^2.

	mu	sigma2
Inefficiency Factors	0.9468888	1.635177

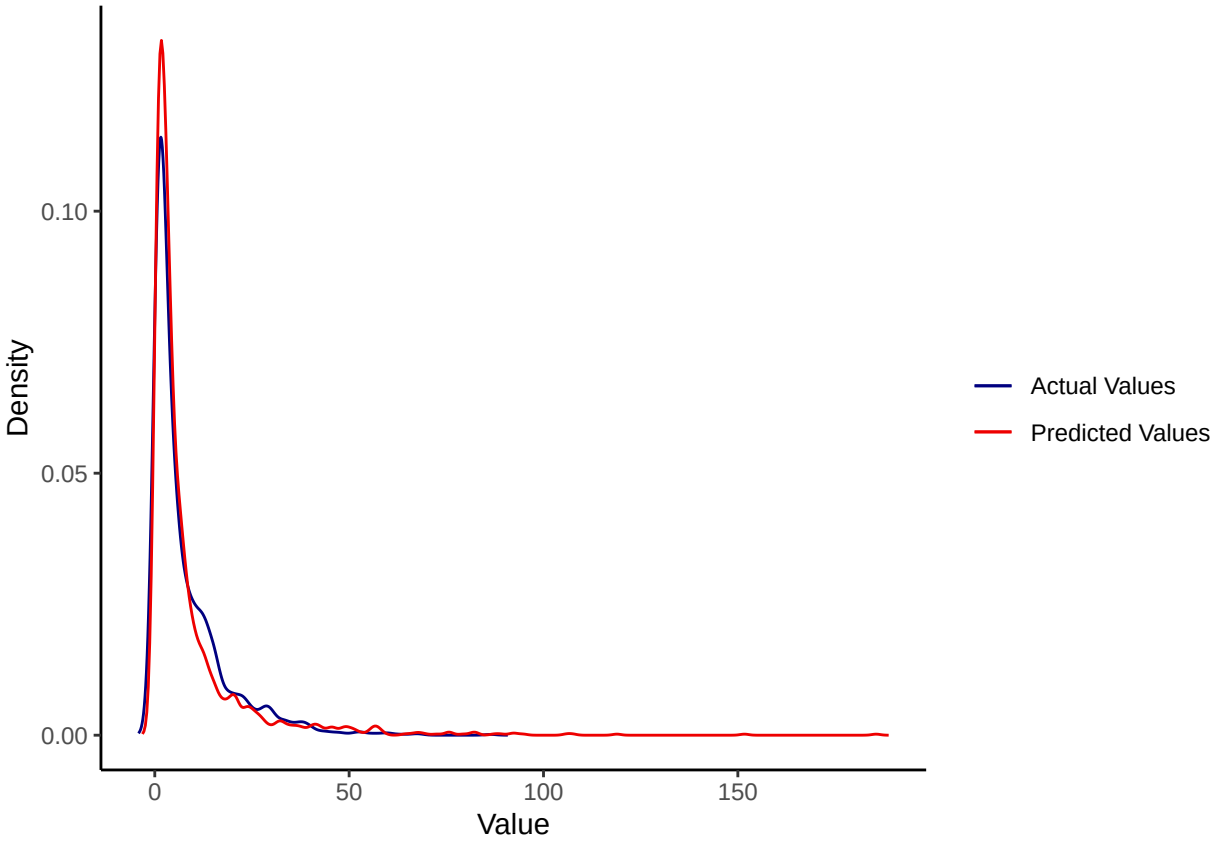
The inefficiency factor equals 1 if there is not any autocorrelation. The inefficiency factor of μ is close to 1, which means this is an efficient sample. However, the inefficiency factor of σ^2 equals approximately 1.63, which means that the sample is not efficient enough; this is because the ACF values of σ^2 cross the statistically significant boundaries when lag equals 12, 16 and 30.

https://www.baeldung.com/cs/acf-pacf-plots-arma-modeling
https://en.wikipedia.org/wiki/Autocorrelation



The above plot illustrates the trajectories of the first 70 point of the sample, the red dot is the starting point and the green dot is the ending point.

Task b



The posterior predictive density has almost the same pattern as the density of the actual values; thus the posterior's predictive values agree very well with the data.

Assignment 2 *Metropolis Random Walk for Poisson regression*

Task a

Below the summary of the glm is represented.

```
##
## Call:
## glm(formula = nBids ~ . - Const, family = poisson, 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|)
## (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 most significant covariates are the *VerifyID*, *Sealed*, *LogBook* and *MinBidShare*.

Task b

Poisson regression model: $y_i|beta \sim Poisson[exp(x_i^T\beta)]$, $i = 1, 2, ..., n$

The likelihood function of the Poisson regression model is:

$$L(\theta|X, Y) = \prod_i^n \frac{e^{Y_i\theta^T X_i} e^{-\theta^T X_i}}{Y_i!}$$

The log-likelihood of the Poisson regression model is:

$$l(\theta|X, Y) = log\Big(L(\theta|X, Y)\Big) = \sum_i^n \Big(Y_i\theta^T X_i - e^{\theta^T Xi} - log(Y_i)\Big)$$

In the above formula, θ appears in the first two terms; therefore, given that we are only interested in the finding of the best value of θ, we drop the $log(Y_i)$.

The final log-likelihood of the Poisson regression model is:

$$l(\theta|X, Y) = \sum_i^n \Big(Y_i\theta^T X_i - e^{\theta^T Xi}\Big)$$

https://en.wikipedia.org/wiki/Poisson_regression

The below table illustrates the $J_y^{-1}(\widetilde{\beta})$.

Constant	PowerSeller	VerifyID	Sealed	MinBlem	MajBlem	LargNeg	LogBook	MinBidShare
0.00095	-0.00071	-0.00027	-0.00027	-0.00045	-0.00028	-0.00051	0.00006	0.00111
-0.00071	0.00135	0.00004	-0.00029	0.00011	-0.00021	0.00028	0.00012	-0.00057
-0.00027	0.00004	0.00852	-0.00078	-0.00010	0.00023	0.00033	-0.00032	-0.00043
-0.00027	-0.00029	-0.00078	0.00256	0.00036	0.00045	0.00034	-0.00013	-0.00006
-0.00045	0.00011	-0.00010	0.00036	0.00362	0.00035	0.00006	0.00006	-0.00006
-0.00028	-0.00021	0.00023	0.00045	0.00035	0.00837	0.00040	-0.00009	0.00026
-0.00051	0.00028	0.00033	0.00034	0.00006	0.00040	0.00318	-0.00025	-0.00011
0.00006	0.00012	-0.00032	-0.00013	0.00006	-0.00009	-0.00025	0.00084	0.00104
0.00111	-0.00057	-0.00043	-0.00006	-0.00006	0.00026	-0.00011	0.00104	0.00505

The below table illustrates the posterior mode values of the features of the dataset, which are almost similar to the actual mode of the true posterior (see the summary of the glm model).

	Value
Constant	1.0698412
PowerSeller	-0.0205125
VerifyID	-0.3930060
Sealed	0.4435555
MinBlem	-0.0524663
MajBlem	-0.2212384
LargNeg	0.0706968
LogBook	-0.1202177
MinBidShare	-1.8919850

The below table illustrates the approximate posterior standard deviation values of every feature of the dataset.

	Value
Constant	0.0307484
PowerSeller	0.0367842
VerifyID	0.0922787
Sealed	0.0505745
MinBlem	0.0602047
MajBlem	0.0914607
LargNeg	0.0563477
LogBook	0.0289564
MinBidShare	0.0710968

Task c

For the Random walk Metropolis algorithm:

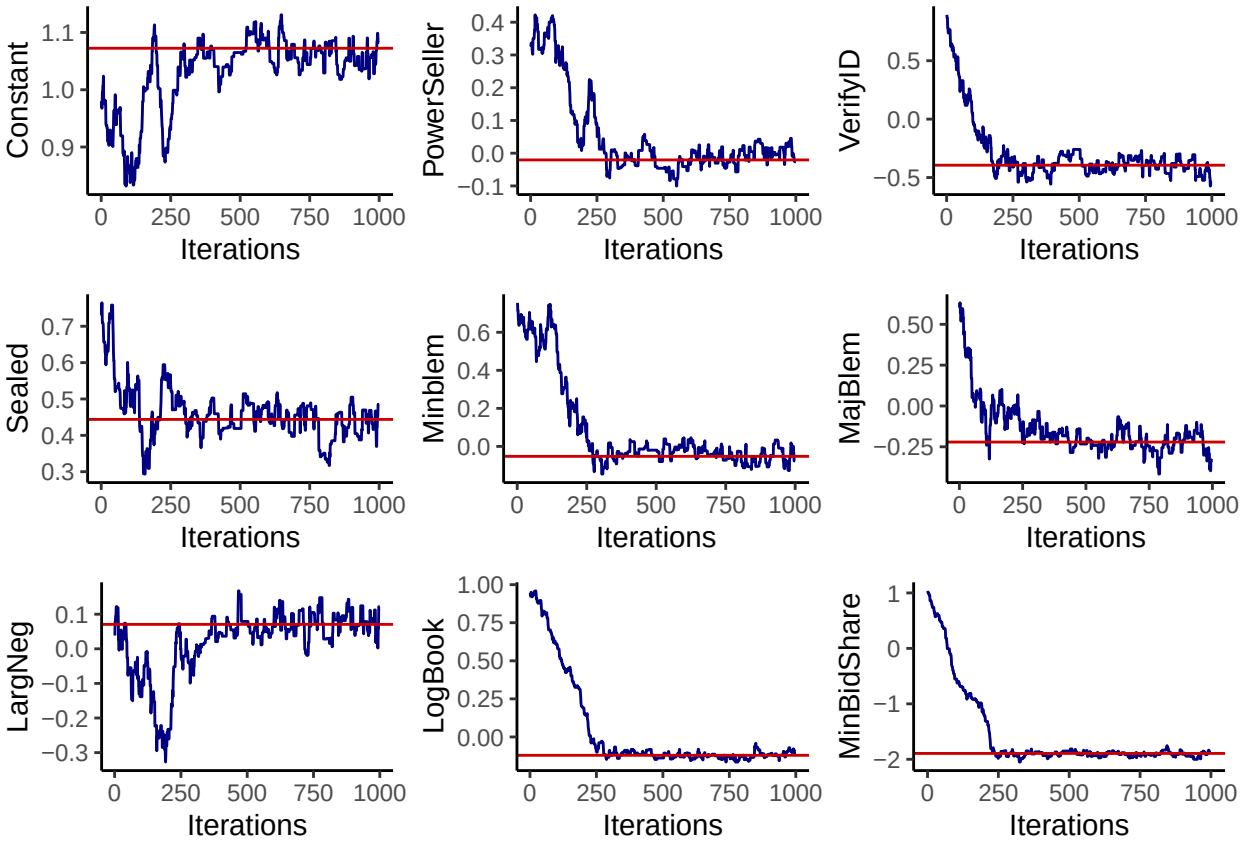
1) Initialize θ^0 and iterate for i = 1, 2, ...

2) Sample proposal: $\beta_p|\beta^{(i-1)} \sim N(\beta^{(i-1)}, c \cdot \Sigma)$, where $\Sigma = J^{-1}_{\hat{\beta},y}$

3) Compute the acceptance probability: $\alpha = min\Big(1, \frac{p(\beta p|y)}{p(\beta^{(i-1)}|y)}$, where :

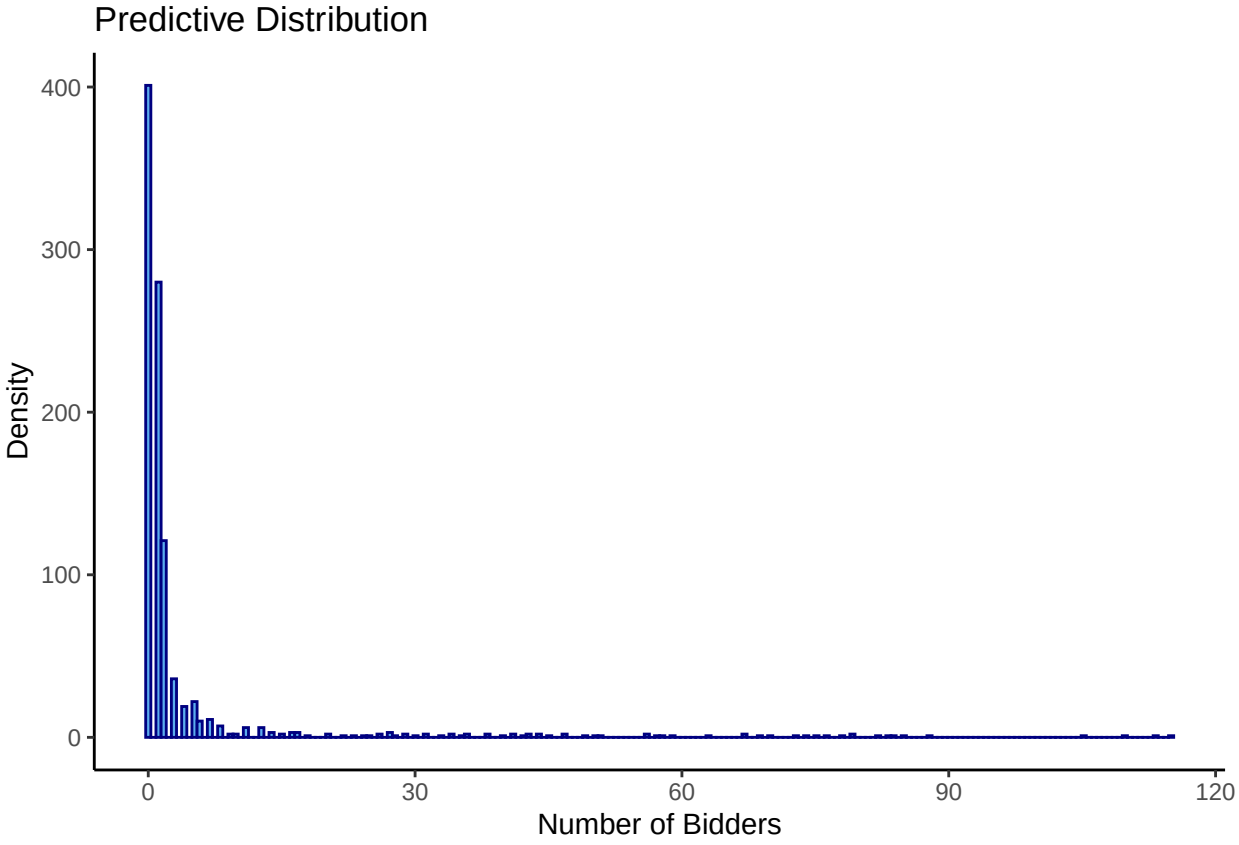
$\frac{p(\beta_p|y)}{p(\beta^{(i-1)}|y)} = exp\Big[log(p(\beta_p|y)) - log(p(\beta^{(i-1)}|y))$

4) With probability α set $\theta^{(i)} = \beta_p$ and $\beta^{(i)} = \beta^{(i-1)}$



In the above plots, the blue lines illustrate the convergence of each $\widetilde{\beta}$ sampled by the Random Walk Metropolis algorithm, and the red lines represent the actual posterior mode. More specifically, it could be assumed that the mode of all $\widetilde{\beta}$ converges to the actual mode, although it is not great in some cases.

Task d

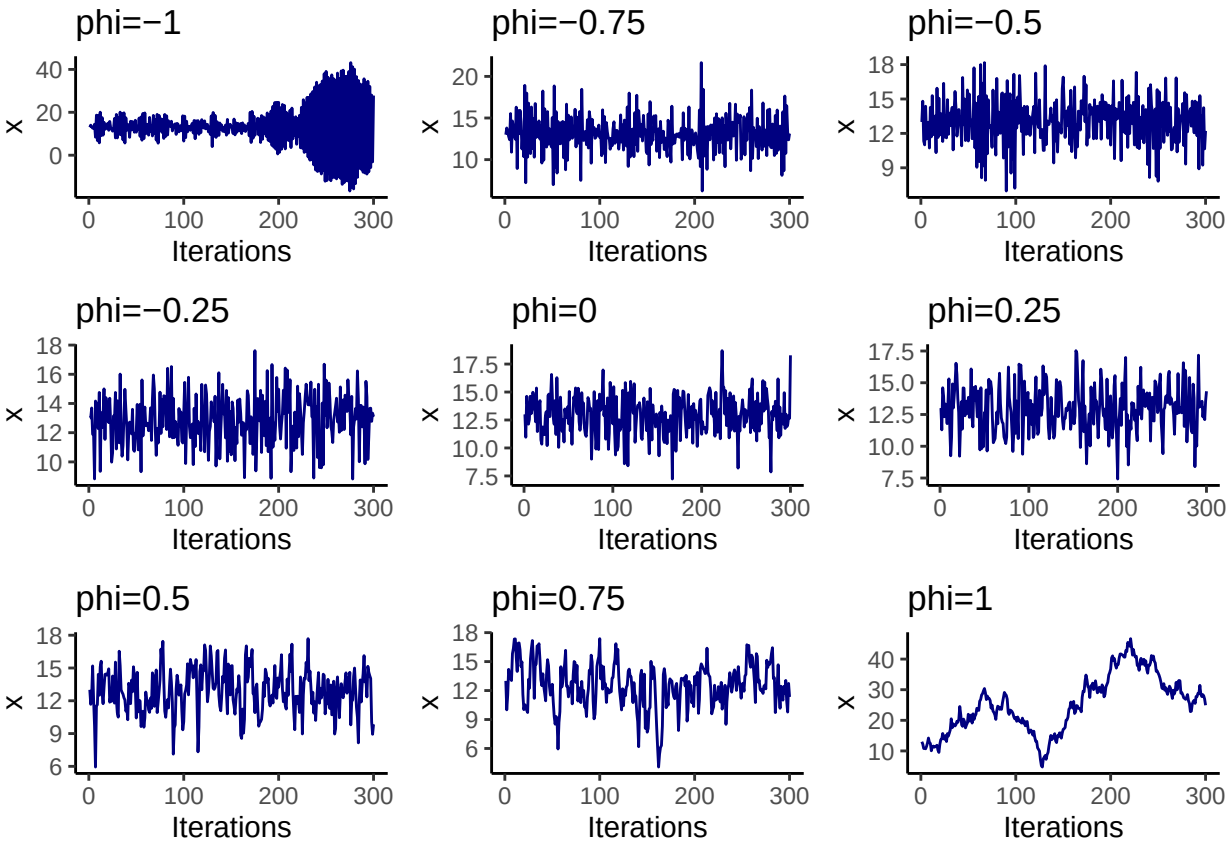


The probability of no bidders in this new auction is approximately 40%.

Assignment 3 *Time series models in Stan*

Task a

The AR(1)-process is $x_t = \mu + \phi(x_{t-1} - \mu) + \epsilon_t$, where $\epsilon_t \overset{\text{iid}}{\sim} N(0, \sigma^2)$. The initial value of x_1 is μ and then the process simulates values for x_t for $t = 2, 3, ..., T$ and return the vector $x_{1:T}$ containing all time points. The given parameters are $\mu = 13$; $\sigma^2 = 3$ and $T = 300$.



The autoregressive model specifies that the output variable depends linearly on its own previous values and on a stochastic term. In the above plots, when ϕ has a negative value and is close to 0, then the process still looks like white noise, but as ϕ approaches 1, there seems to be a positive correlated with the previous terms.

https://en.wikipedia.org/wiki/Autoregressive_model#Explicit_mean/difference_form_of_AR(1)_process

Task b

i)

Table 5: Table for phi=0.2

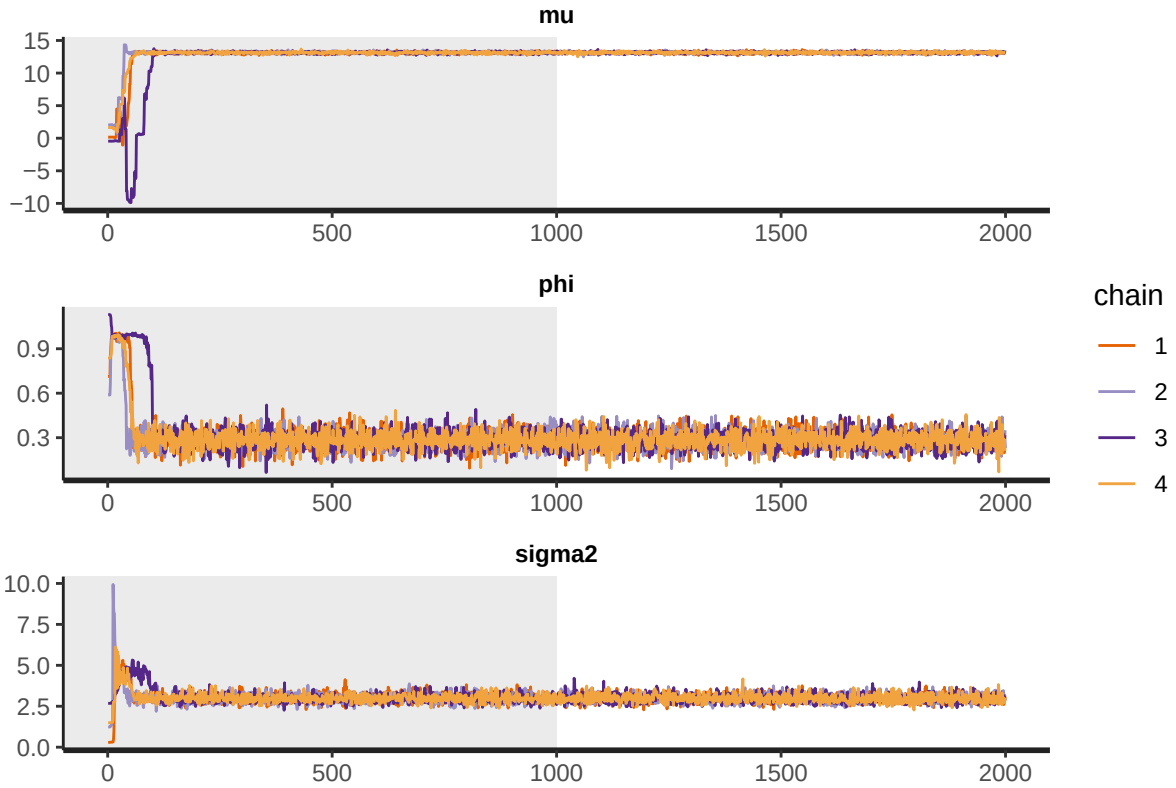
	mean	2.5%	97.5%	n_eff
mu	13.1313577	12.847777	13.4015866	3630.727
sigma2	2.9822928	2.548480	3.5116601	3444.398
phi	0.2859452	0.176058	0.3980682	3539.958

Table 6: Table for phi=0.95

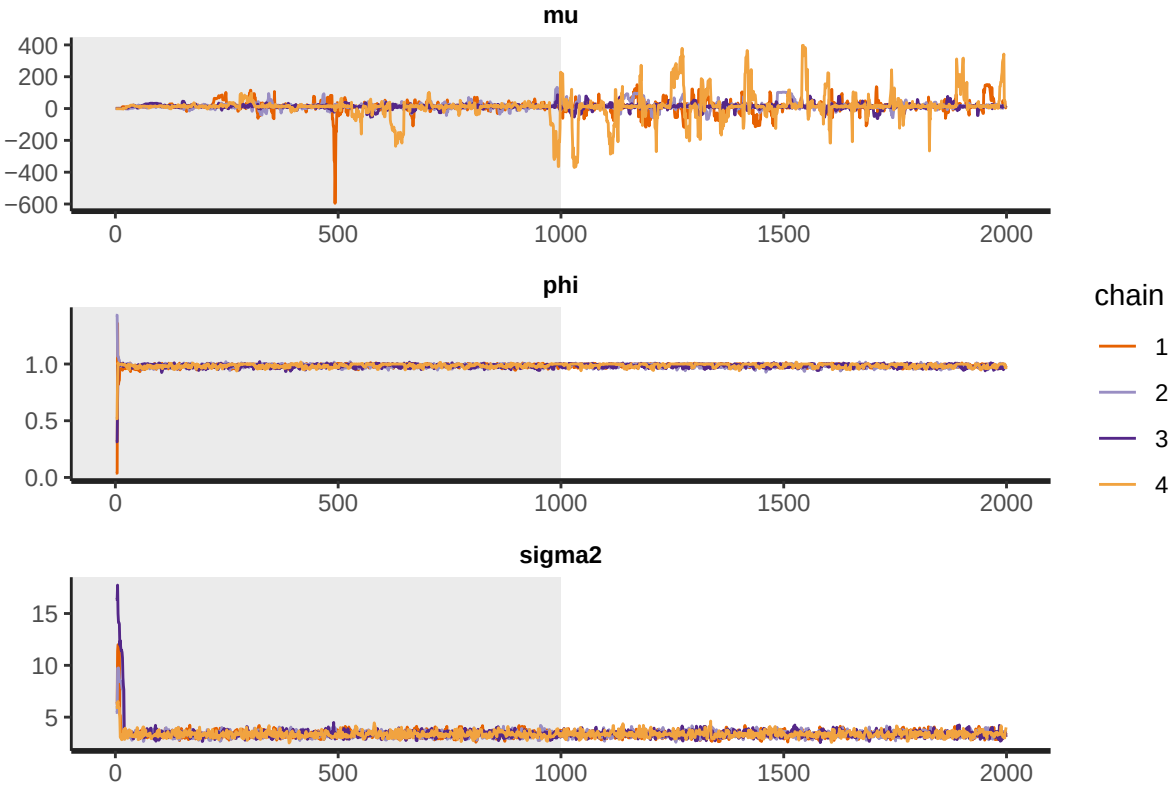
	mean	2.5%	97.5%	n_eff
mu	17.0385695	-79.5703024	145.321794	332.6698
sigma2	3.3502708	2.8547802	3.960345	1429.9033
phi	0.9855417	0.9550611	1.004276	429.0814

From the above tables, the posterior's means are similar to the actual value for each process, except in one case. However, the number of effective posterior samples for the three inferred parameters differ; the number of effective posterior samples is much greater when ϕ equals 0.2 than when ϕ is 0.95. This is because the 95% credible intervals are smaller when ϕ equals 0.2. However, the 95% credible interval of ϕ when ϕ is 0.95 has very strict lower and upper bounds, and this might be the reason for a low number of effective posterior samples. Both processes simulated the means of ϕ and σ^2 values pretty well. Moreover, μ is estimated close to its actual value when ϕ is 0.2, compared to μ when ϕ equals 0.95, which differs significantly; it was expected because of its "large" 95% credible interval.

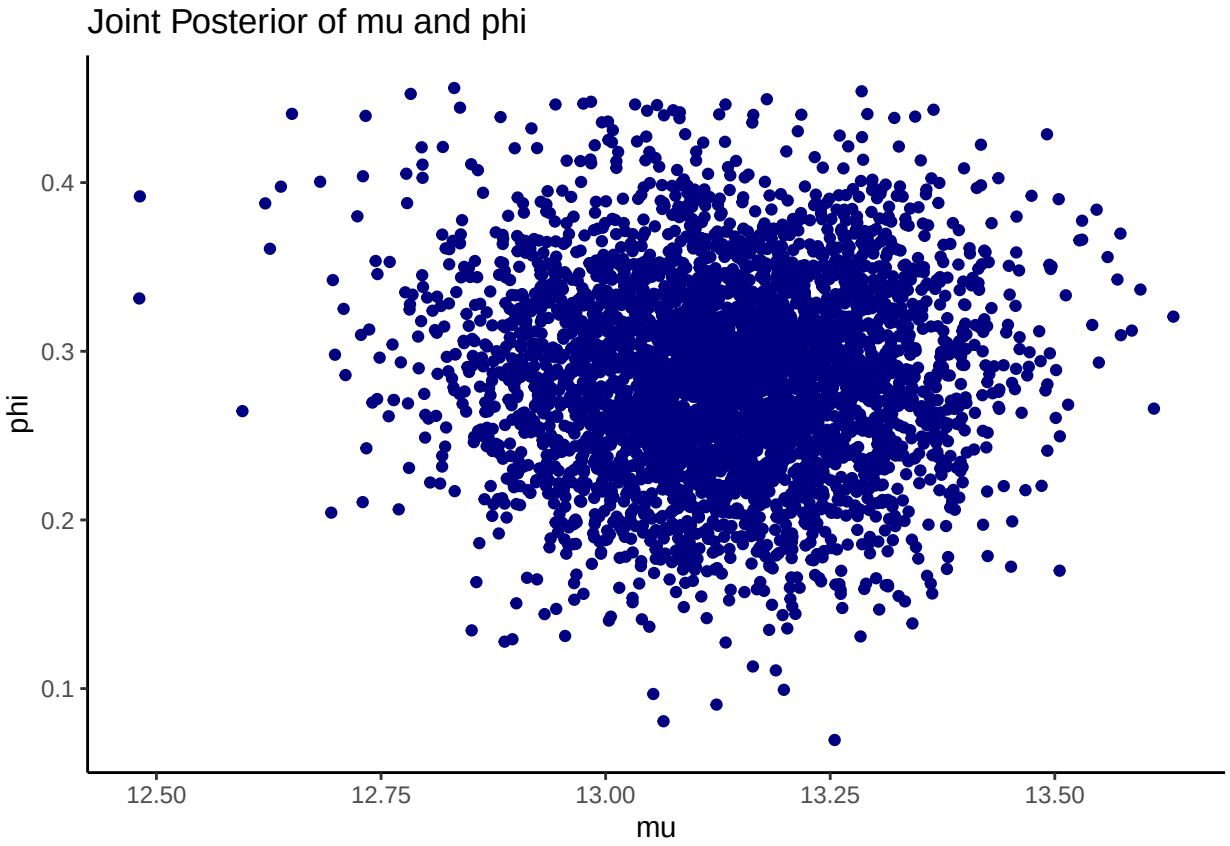
ii)



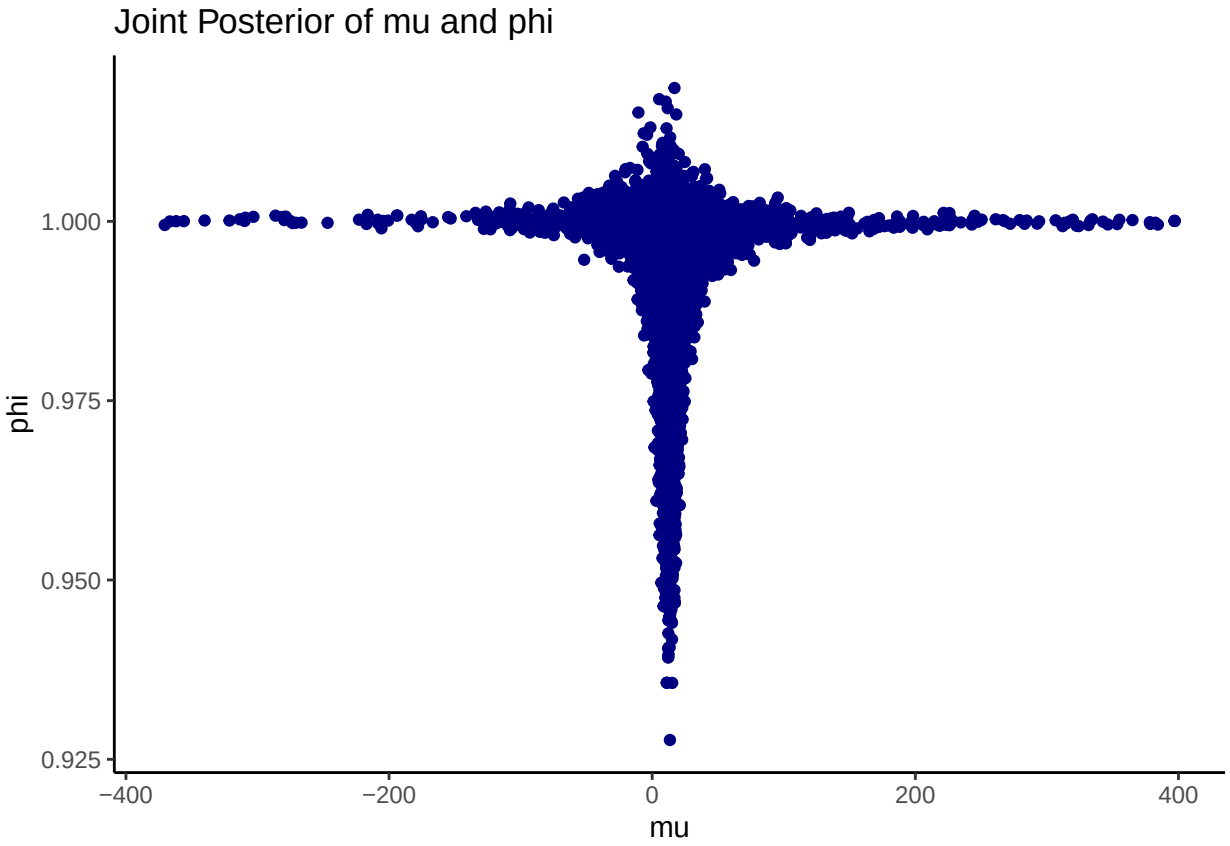
The above traceplots represent the convergence of the samplers when ϕ equals 0.2. After approximately 150 iterations all values seems to convergence to the actual ones. (The gray area illustrates the warm-up iterations).



The above traceplots represent the convergence of the samplers when ϕ equals 0.95. After approximately 150 iterations all values seems to convergence to the actual ones. An interesting feature is that μ has extreme values in same cases, which is expected because of its "big" 95% credible interval. (The gray area illustrates the warm-up iterations).



In the plot of the joint posterior of μ and ϕ when ϕ is 0.2, it is evident that most of the observations are gathered around in the region of the actual values of μ and ϕ.



In the plot of the joint posterior of μ and ϕ when ϕ is 0.95, the observations are plotted close to the actual value of ϕ. However, the observations are spread because of the extremely low and upper bound of the 95% credible interval of μ.

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
knitr::opts_chunk$set(tidy.opts = list(width.cutoff = 60), tidy = TRUE)
library(ggplot2)
library(mvtnorm)
library(gridExtra)
library(rstan)
# ___________________Assignment 1___________________#

# Reading & preparing data
precipitation <- as.data.frame(readRDS("Precipitation.rds"))
colnames(precipitation) <- "records"
log_records <- log(precipitation$records)
# Task 1a
set.seed(1234567890)

# setting up prior parameters
n <- length(log_records)
n_0 <- 1
mu_0 <- mean(log_records)
sigma2_0 <- var(log_records)
tau2_0 <- 1

# normal model with conditionally conjugate prior mu_prop
# <- rnorm(n = 1, mean = mu_0, sqrt(tau2_0)) sigma2_prop <-
# n_0*sigma2_0 / rchisq(1,n_0)

# sigma2 starting value
sigma2 <- sigma2_0

# vectors to store the samples
mu_gibbs <- c()
sigma2_gibbs <- c(sigma2)

N <- 1000
for (i in 1:N) {

    # calculating parameters
    w <- (n/sigma2)/((n/sigma2) + (1/tau2_0))
    mu_n <- w * mean(log_records) + (1 - w) * mu_0
    tau2_n <- 1/(n/sigma2 + 1/tau2_0)

    # calculating mu
    mu <- rnorm(n = 1, mean = mu_n, sd = tau2_n)

    # temporaly store the previous sigma2
    previous_sigma2 <- sigma2

    # calculating parameter
    n_n <- n_0 + n

    # calculating sigma2
```

```
    sigma2 <- (n_n * ((n_0 * sigma2_0 + sum((log_records - mu)^2))/n_n))/rchisq(1,
        n_n)

    # conditions of how to store the samples
    if (i == 1) {
        mu_gibbs <- c(mu)
        sigma2_gibbs <- c(sigma2)
    } else {
        mu_gibbs <- append(mu_gibbs, c(mu, mu))
        sigma2_gibbs <- append(sigma2_gibbs, c(previous_sigma2,
            sigma2))
    }
}

# calculating autocorrelation for mu
mu_r <- acf(mu_gibbs, main = "Autocorrelation of mu", col = "red4",
    lwd = 2)

# callculating the inefficiency factor of mu
IF_mu <- 1 + 2 * sum(mu_r$acf[-1])

# calculating autocorrelation for sigma2
sigma2_r <- acf(sigma2_gibbs, main = "Autocorrelation of sigma2",
    col = "red4", lwd = 2)

# callculating the inefficiency factor of sigma2
IF_sigma2 <- 1 + 2 * sum(sigma2_r$acf[-1])

# df with the inefficiency factors of mu & sigma
IFs_df <- data.frame(mu = IF_mu, sigma2 = IF_sigma2)
rownames(IFs_df) <- c("Inefficiency Factors")
knitr::kable(IFs_df)

# df for plot
plot_df_traj <- data.frame(mu = mu_gibbs, sigma2 = sigma2_gibbs)

# plotting the first 70 samples plotting all the samples
# would be a mess
plot_df_traj <- plot_df_traj[1:70, ]

# trajectories of the sampled Markov chains.
ggplot(plot_df_traj) + geom_path(aes(x = mu, y = sigma2), color = "navy") +
    geom_point(aes(x = mu[1], y = sigma2[1]), color = "red2",
        size = 2) + geom_point(aes(x = mu[70], y = sigma2[70]),
    color = "green2", size = 2) + ylab("sigma2") + xlab("mu") +
    theme_classic()

# Task b

set.seed(1234567890)

# predicted draws
predictions <- rnorm(nrow(precipitation), mu_gibbs, sqrt(sigma2_gibbs))
```

```r
# kernel density estimation
density_actual <- density(precipitation$records)
density_pred <- density(exp(predictions))

# df for plot
plot_df_dens <- data.frame(actual_x = density_actual$x, actual_y = density_actual$y,
    pred_x = density_pred$x, pred_y = density_pred$y)

ggplot(plot_df_dens) + geom_line(aes(x = actual_x, y = actual_y,
    color = "navy")) + geom_line(aes(x = pred_x, y = pred_y,
    color = "red2")) + theme(legend.position = "right") + scale_color_manual(values = c("navy",
    "red2"), name = "", labels = c("Actual Values", "Predicted Values")) +
    xlab("Value") + ylab("Density") + theme_classic()

# ___________________Assignment 2_____________________#

# Reading data
ebay <- read.table("eBayNumberOfBidderData.dat", header = TRUE)
# Task a

# glm function, + 0 in order to do not input the covariate
# Const
model <- glm(formula = nBids ~ . - Const, data = ebay, family = poisson)

# print the summary of the model
summary(model)
# Task b

# the below code snippets from a demo of logistic
# regression in Lecture 6

# target value
y <- ebay$nBids

# prior inputs Select which covariates/features to include
X <- as.matrix(ebay[2:10])
Xnames <- colnames(X)

Npar <- dim(X)[2]

# Setting up the prior
mu <- as.matrix(rep(0, Npar))  # Prior mean vector
Sigma <- 100 * solve(t(X) %*% X)  # Prior covariance matrix

# Functions that returns the log posterior for the poisson
# and probit regression.  First input argument of this
# function must be the parameters we optimize on, i.e. the
# regression coefficients beta.

LogPostPoisson <- function(betas, y, X, mu, Sigma) {
    linPred <- X %*% betas
    # loglikelihood of Poisson regression
    logLik <- sum(linPred * y - exp(linPred))
```

```
    if (abs(logLik) == Inf)
        logLik = -20000  # Likelihood is not finite, stear the optimizer away from here!
    logPrior <- dmvnorm(betas, mu, Sigma, log = TRUE)

    return(logLik + logPrior)
}

# Select the initial values for beta
initVal <- matrix(0, Npar, 1)

# The argument control is a list of options to the
# optimizer optim, where fnscale=-1 means that we minimize
# the negative log posterior. Hence, we maximize the log
# posterior.
OptimRes <- optim(initVal, LogPostPoisson, gr = NULL, y, X, mu,
    Sigma, method = c("BFGS"), control = list(fnscale = -1),
    hessian = TRUE)

names(OptimRes$par) <- Xnames  # Naming the coefficient by covariates
approxPostStd <- sqrt(diag(-solve(OptimRes$hessian)))  # Computing approximate standard deviations.
names(approxPostStd) <- Xnames  # Naming the coefficient by covariates
# print('The posterior mode is:') print(OptimRes$par)
# print('The approximate posterior standard deviation is:')
approxPostStd <- sqrt(diag(-solve(OptimRes$hessian)))
# print(approxPostStd)

invHessian <- round(-solve(OptimRes$hessian), 5)
invHessian <- data.frame(invhes = invHessian)
colnames(invHessian) <- c("Constant", "PowerSeller", "VerifyID",
    "Sealed", "MinBlem", "MajBlem", "LargNeg", "LogBook", "MinBidShare")
knitr::kable(invHessian)

df_post_mode <- data.frame(post_mode = OptimRes$par)
colnames(df_post_mode) <- c("Value")
rownames(df_post_mode) <- c("Constant", "PowerSeller", "VerifyID",
    "Sealed", "MinBlem", "MajBlem", "LargNeg", "LogBook", "MinBidShare")
knitr::kable(df_post_mode)

df_approxPostStd <- data.frame(approxPostStd = sqrt(diag(-solve(OptimRes$hessian))))
colnames(df_approxPostStd) <- c("Value")
rownames(df_approxPostStd) <- c("Constant", "PowerSeller", "VerifyID",
    "Sealed", "MinBlem", "MajBlem", "LargNeg", "LogBook", "MinBidShare")
knitr::kable(df_approxPostStd)
# Task c

set.seed(1234567890)

RWMSampler <- function(N, logPostFunc, init_beta, constant, cov_mat,
    target_val, features, mu) {

    init_sample = rmvnorm(1, init_beta, cov_mat)

    # matrix to store the betas
```

```
    betas <- matrix(nrow = N, ncol = length(init_sample))

    betas[1, ] <- init_sample

    for (i in 2:N) {

        # sample proposal
        sample_proposal <- as.vector(rmvnorm(1, betas[i - 1,
            ], constant * cov_mat))

        # LogPostPoisson <- function(betas,y,X,mu,Sigma)
        # calculating the new posterior
        post_new <- logPostFunc(sample_proposal, target_val,
            features, mu, cov_mat)

        # calculating the old posterior
        post_old <- logPostFunc(betas[i - 1, ], target_val, features,
            mu, cov_mat)

        # acceptance probability
        a <- min(1, exp(post_new - post_old))

        u <- runif(1, 0, 1)

        if (u < a) {
            betas[i, ] <- sample_proposal
        } else {
            betas[i, ] <- betas[i - 1, ]
        }
    }
    return(betas)
}


RWMbetas <- RWMSampler(1000, LogPostPoisson, runif(9, 0, 1),
    0.35, -solve(OptimRes$hessian), ebay$nBids, as.matrix(ebay[2:10]),
    OptimRes$par)

plot_df_RWM <- data.frame(RWMbetas)
colnames(plot_df_RWM) <- c("Constant", "PowerSeller", "VerifyID",
    "Sealed", "Minblem", "MajBlem", "LargNeg", "LogBook", "MinBidShare")

plot_df_RWM$Iterations <- 1:1000

p1 <- ggplot(plot_df_RWM) + geom_line(aes(x = Iterations, y = Constant),
    color = "navy") + geom_hline(yintercept = model$coefficients[1],
    color = "red3") + theme_classic()

p2 <- ggplot(plot_df_RWM) + geom_line(aes(x = Iterations, y = PowerSeller),
    color = "navy") + geom_hline(yintercept = model$coefficients[2],
    color = "red3") + theme_classic()

p3 <- ggplot(plot_df_RWM) + geom_line(aes(x = Iterations, y = VerifyID),
```

```
    color = "navy") + geom_hline(yintercept = model$coefficients[3],
    color = "red3") + theme_classic()

p4 <- ggplot(plot_df_RWM) + geom_line(aes(x = Iterations, y = Sealed),
    color = "navy") + geom_hline(yintercept = model$coefficients[4],
    color = "red3") + theme_classic()

p5 <- ggplot(plot_df_RWM) + geom_line(aes(x = Iterations, y = Minblem),
    color = "navy") + geom_hline(yintercept = model$coefficients[5],
    color = "red3") + theme_classic()

p6 <- ggplot(plot_df_RWM) + geom_line(aes(x = Iterations, y = MajBlem),
    color = "navy") + geom_hline(yintercept = model$coefficients[6],
    color = "red3") + theme_classic()

p7 <- ggplot(plot_df_RWM) + geom_line(aes(x = Iterations, y = LargNeg),
    color = "navy") + geom_hline(yintercept = model$coefficients[7],
    color = "red3") + theme_classic()

p8 <- ggplot(plot_df_RWM) + geom_line(aes(x = Iterations, y = LogBook),
    color = "navy") + geom_hline(yintercept = model$coefficients[8],
    color = "red3") + theme_classic()

p9 <- ggplot(plot_df_RWM) + geom_line(aes(x = Iterations, y = MinBidShare),
    color = "navy") + geom_hline(yintercept = model$coefficients[9],
    color = "red3") + theme_classic()

grid.arrange(p1, p2, p3, p4, p5, p6, p7, p8, p9, nrow = 3)

# Task d

set.seed(1234567890)

# given auction
auction <- c(1, 1, 0, 1, 0, 1, 0, 1.2, 0.8)

nbidders = c()

for (i in 1:nrow(RWMbetas)) {
    nbidders[i] <- rpois(1, exp(RWMbetas[i, ] %*% auction))
}

nbidders <- data.frame(nbidders)

zero_bidders <- which(nbidders$nbidders == 0)
prob <- length(zero_bidders)/(nrow(nbidders))

ggplot(nbidders, aes(x = nbidders)) + geom_histogram(bins = 200,
    color = "navy", fill = "steelblue2") + ggtitle("Predictive Distribution") +
    xlab("Number of Bidders") + ylab("Density") + theme_classic()

# ___________________Assignment 3___________________#
```

```r
# Task a

set.seed(12345)

# AR(1)-process
ar_process <- function(t, mu, phi, sigma2) {
    x <- c()
    x[1] <- mu
    for (i in 2:t) {
        x[i] <- mu + phi * (x[i - 1] - mu) + rnorm(1, 0, sqrt(sigma2))
    }
    return(x)
}

# given parameters
mu <- 13
sigma2 <- 3
t <- 300
phi <- seq(-1, 1, 0.25)

res <- as.data.frame(sapply(phi, function(phi) ar_process(t,
    mu, phi, sigma2)))

res$iterations <- 1:t

# Time series plots of different phis

p1 <- ggplot(res) + geom_line(aes(x = iterations, y = V1), color = "navy") +
    ggtitle("phi=-1") + xlab("Iterations") + ylab("x") + theme_classic()

p2 <- ggplot(res) + geom_line(aes(x = iterations, y = V2), color = "navy") +
    ggtitle("phi=-0.75") + xlab("Iterations") + ylab("x") + theme_classic()

p3 <- ggplot(res) + geom_line(aes(x = iterations, y = V3), color = "navy") +
    ggtitle("phi=-0.5") + xlab("Iterations") + ylab("x") + theme_classic()

p4 <- ggplot(res) + geom_line(aes(x = iterations, y = V4), color = "navy") +
    ggtitle("phi=-0.25") + xlab("Iterations") + ylab("x") + theme_classic()

p5 <- ggplot(res) + geom_line(aes(x = iterations, y = V5), color = "navy") +
    ggtitle("phi=0") + xlab("Iterations") + ylab("x") + theme_classic()

p6 <- ggplot(res) + geom_line(aes(x = iterations, y = V6), color = "navy") +
    ggtitle("phi=0.25") + xlab("Iterations") + ylab("x") + theme_classic()

p7 <- ggplot(res) + geom_line(aes(x = iterations, y = V7), color = "navy") +
    ggtitle("phi=0.5") + xlab("Iterations") + ylab("x") + theme_classic()

p8 <- ggplot(res) + geom_line(aes(x = iterations, y = V8), color = "navy") +
    ggtitle("phi=0.75") + xlab("Iterations") + ylab("x") + theme_classic()

p9 <- ggplot(res) + geom_line(aes(x = iterations, y = V9), color = "navy") +
    ggtitle("phi=1") + xlab("Iterations") + ylab("x") + theme_classic()
```

```r
grid.arrange(p1, p2, p3, p4, p5, p6, p7, p8, p9, nrow = 3)

# parts of the below code snippets from a demo of
# Stanmodel.R provided by the lecturer

# Task b

set.seed(1234567890)

# stan_model
StanModel = "
data {
  int<lower=0> N;
  vector[N] x;
}
parameters {
  real mu;
  real<lower=0> sigma2;
  real phi;
}
model {
  for(i in 2:N){
    x[i] ~ normal( mu + phi*(x[i-1]-mu), sqrt(sigma2));
  }
}"

# given parameters
phi1 <- 0.2
phi2 <- 0.95

# ar with given parameters
res1 <- ar_process(t, mu, phi1, sigma2)
res2 <- ar_process(t, mu, phi2, sigma2)

data1 <- list(N = t, x = res1)
data2 <- list(N = t, x = res2)

warmup <- 1000
niter <- 2000

fit1 <- stan(model_code = StanModel, data = data1, warmup = warmup,
    iter = niter, chains = 4)
fit2 <- stan(model_code = StanModel, data = data2, warmup = warmup,
    iter = niter, chains = 4)

fit1_summary <- summary(fit1)
phi1_stats <- fit1_summary$summary[1:3, c(1, 4, 8, 9)]

knitr::kable(phi1_stats, caption = "Table for phi=0.2")

fit2_summary <- summary(fit2)
phi2_stats <- fit2_summary$summary[1:3, c(1, 4, 8, 9)]
```

```r
knitr::kable(phi2_stats, caption = "Table for phi=0.95")

traceplot(fit1, pars = c("mu", "phi", "sigma2"), inc_warmup = TRUE,
    nrow = 3)

traceplot(fit2, pars = c("mu", "phi", "sigma2"), inc_warmup = TRUE,
    nrow = 3)

draws1 <- extract(fit1)
draws2 <- extract(fit2)

df_draws1 <- data.frame(mu = draws1$mu, phi = draws1$phi)
df_draws2 <- data.frame(mu = draws2$mu, phi = draws2$phi)

ggplot(df_draws1) + geom_point(aes(x = mu, y = phi), color = "navy") +
    ggtitle("Joint Posterior of mu and phi") + theme_classic()

ggplot(df_draws2) + geom_point(aes(x = mu, y = phi), color = "navy") +
    ggtitle("Joint Posterior of mu and phi") + theme_classic()
```