Bayesian Learning: Laboratory 3

Nicolas Taba (nicta839), Yuki Washio (timwa902)

19/05/2021

Gibbs sampler for a normal model

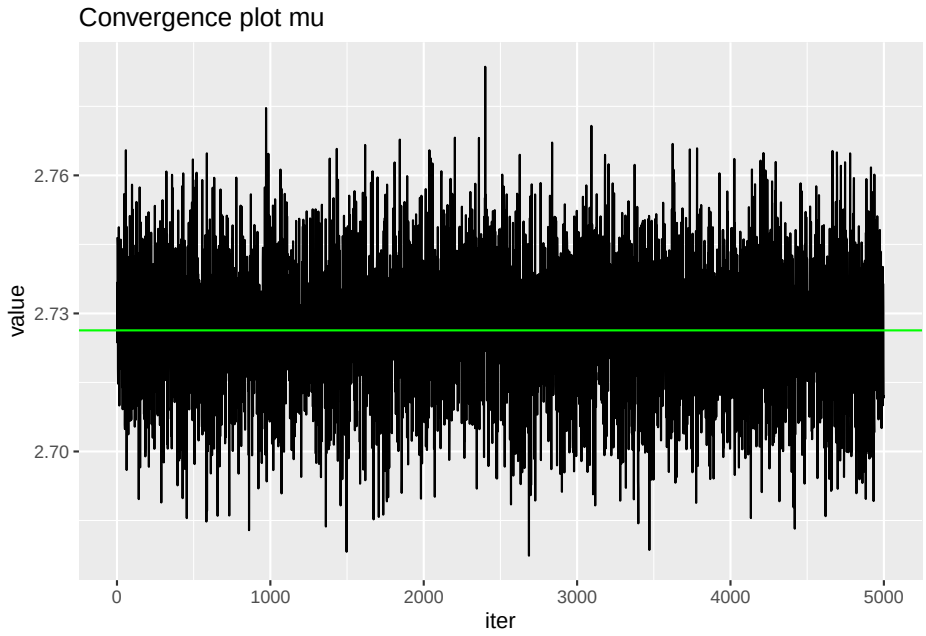
Gibbs sampler implementation

```
# rm(list = ls())

########### A ################
library(ggplot2)
library(mvtnorm)
```

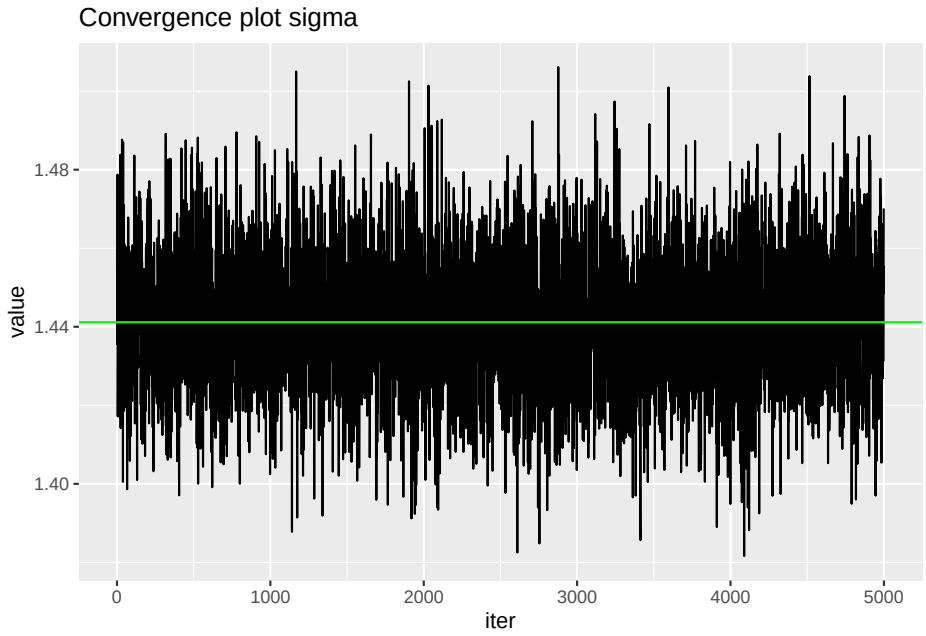
```
## Warning: package 'mvtnorm' was built under R version 4.0.3
```

```
# working directory change as needed
setwd("C:/Users/nicol/Documents/MSc/MSc_Statistics_and_Machine_Learning/Bayesian Learning 732A73/lab3")


# Reading data and transform to log
data_raw <- read.table("rainfall.dat")
data_log <- unlist(log(data_raw))

# Scaled inverse chisq
#  Inverse chisquare
rInvChisq <- function(draws, n, tau_sq){
  #  n are the degrees of freedom
  X <- rchisq(draws, n)
  sample <- (tau_sq * n)/X
  return(sample)
}

# posterior for mu (lecture 7)
mu_post <- function(x, var, mu_0, tau_0){
  n <- length(x)
  tau_n <- 1 / ((n/var) + (1 / tau_0))
  w <- (n/var) / ((n/var) + (1 / tau_0))
  mu_n <- w * mean(x) + (1-w) * mu_0

  return(rnorm(1, mu_n, sqrt(tau_n)))
}

# posterior for sigma (lecture 7)
var_post <- function(x, mu, v_0, var_0){
  n <- length(x)
  v_n <- v_0 + n
```

```r
  variance <- (v_0*var_0 + sum((x-mu)^2)) / v_n
  return(rInvChisq(1, v_n, variance))
}

# initialize values
mu_0 <- 0
sigma_0 <- 1
tau_0 <- 1
nu_0 <- 1
draws <- 5000


# Gibbs sampler
gibbs_sampler <- function(data, mu_0, sigma_0, tau_0, nu_0, draws){
  mu_val <- c(mu_0, rep(0, draws))
  sigma_val <- c(sigma_0, rep(0, draws))
  # tau_val <- c(tau_0, rep(0, draws))
  # nu_val <- c(nu_0, rep(0, draws))
  dist_val <- c(rnorm(1, mu_0, sigma_0), rep(0, draws))

  for(i in 2:draws){
    mu_val[i] <- mu_post(x = data_log, var = sigma_val[i-1], mu_0 = mu_val[i-1], tau_0 = tau_0)
    sigma_val[i] <- var_post(x = data_log, mu = mu_val[i], v_0 = (nu_0+length(data)), var_0 = sigma_0)
    dist_val[i] <- rnorm(1, mu_val[i], sqrt(sigma_val[i]))

    # # Update tau and nu (why do we get correlated draws?) -> (because tau and nu are not randomly sam
    # tau_val[i] <- 1 / ((length(data)/sigma_val[i-1]) + (1 / tau_val[i-1]))
    # nu_val[i] <- nu_val[i-1] + length(data)
  }
  final_df <- data.frame(mu_sample = mu_val, sigma_sample = sigma_val, norm_dist = dist_val)
  return(final_df)
}

# Gibbs sample
sample_gibbs <- gibbs_sampler(data = data_log, mu_0 = mu_0, sigma_0 = sigma_0, tau_0 = tau_0, nu_0 = nu
df_sample_gibbs <- data.frame(sample_gibbs[2:(draws-1), ], x = 2:(draws-1))

plot_mu <- ggplot(data =df_sample_gibbs, aes(x = x))+
  geom_line(aes(y = mu_sample))+
  geom_hline(yintercept = mean(df_sample_gibbs$mu_sample), color = "green")+
  labs(title = "Convergence plot mu", x = "iter", y = "value")

plot_sigma <- ggplot(data =df_sample_gibbs, aes(x = x))+
  geom_line(aes(y = sigma_sample))+
  geom_hline(yintercept = mean(df_sample_gibbs$sigma_sample), color = "green")+
  labs(title = "Convergence plot sigma", x = "iter", y = "value")

plot_mu
```




```
plot_sigma
```




```
# autocorrelation
# First and last values removed as outliers
mu_autocorr <- acf(sample_gibbs$mu_sample[2:5000], plot = FALSE)
sigma_autocorr <- acf(sample_gibbs$sigma_sample[2:5000], plot = FALSE)
# Inefficiency factor
# Remove first value to get lag from k=1
IF_mu <- 1 + 2 * sum(mu_autocorr$acf[-1])
IF_sigma <- 1 + 2 * sum(sigma_autocorr$acf[-1])
cat("The inefficiency factor for mu is: ", IF_mu,"\n")
```

```
## The inefficiency factor for mu is:  0.9692904
```

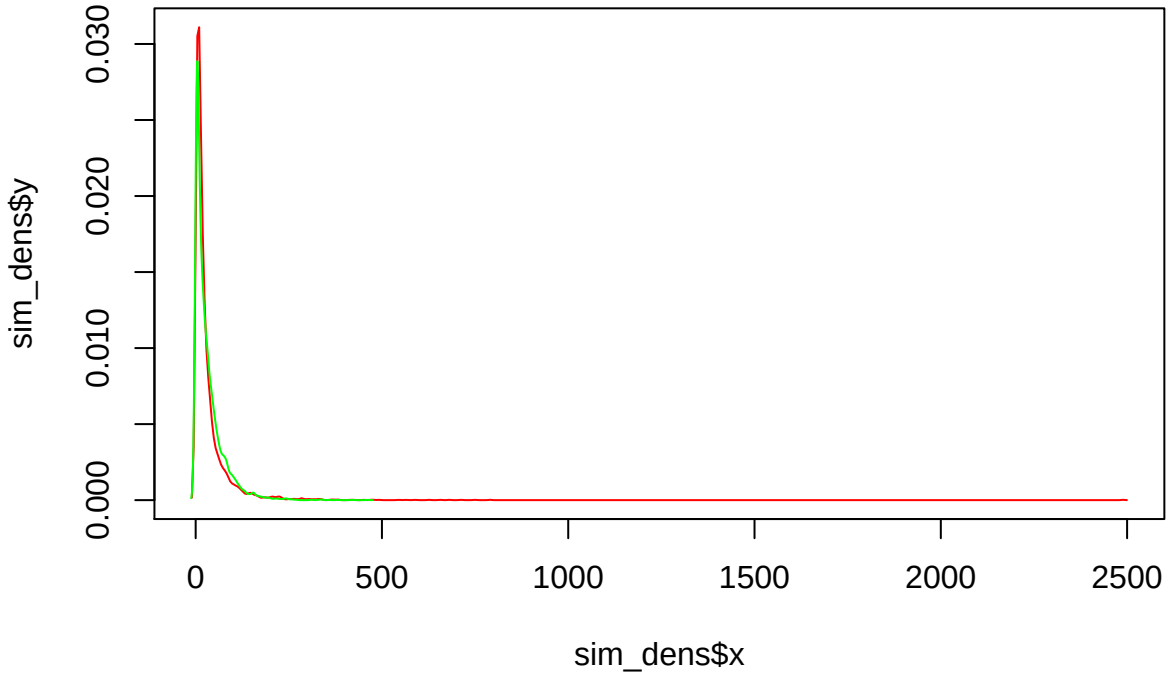
```r
cat("The inefficiency factor for sigma is: ", IF_sigma)
```

```
## The inefficiency factor for sigma is:  0.9394642
```

We visually assess that the convergence plot indicate that the Gibbs sampler for μ ans σ show that the values converge. This assessment is further confirmed by the small values of the inefficiency factor that are one more marker of convergence for Gibbs sampling.

Plotting densities

```r
data_dens <- density(exp(data_log))
sim_dens <- density(exp(df_sample_gibbs$norm_dist))

plot(x = sim_dens$x, y = sim_dens$y, col = "red", type = "l")
lines(x = data_dens$x, y = data_dens$y, col = "green")
```



In this plot, the red line represents the simulated density whereas the green line represents the original data. From this plat, we can see that both densities agree with each other although our simulated data overestimates the probability mass around the mode.

Metropolis Random Walk for Poisson regression

GLM

```
############ A ############
data_ebay <- read.table("eBayNumberOfBidderData.dat", stringsAsFactors = FALSE, header = TRUE)
data_ebay_glm <- data_ebay[, -2]

#fit glm model
glm_model <- glm(formula = nBids~., data = data_ebay_glm, family = "poisson")

summary(glm_model)
```

```
##
## Call:
## glm(formula = nBids ~ ., family = "poisson", data = data_ebay_glm)
##
## 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 coefficicents that are marked as significant are those with three stars in the summary. These coefficients are "Intercept", "VerifyID", "Sealed", "LogBook" and "MinBidShare". Although the "MinBlem" coefficient has low significance, it might be worthwhile further investigating how much its significance is.

Bayesian analysis of Poisson regression

```
library(mvtnorm)

log_posterior <- function(Beta, X, y, mu, sigma){

  loglik <- sum(-log(factorial(y)) + X%*%Beta * y - exp(X%*%Beta))
  log_prior <- dmvnorm(Beta, mu, sigma, log=TRUE)
```

```
  #log posterior is loglik + log_prior instead of lik*prior because of the logarithm
  return(loglik + log_prior)
}

# set up the data
covariates = as.matrix(data_ebay[, -1])
target <- as.vector(data_ebay[, 1])
N <- ncol(covariates)
mu <- rep(0, N)
sigma <- (100 * solve(t(covariates)%*%covariates))
init <- rep(0, N)

res <- optim(init, log_posterior, X=covariates, y=target, mu=mu, sigma=sigma, method = "BFGS", control =

coefficients <- res$par
J <- -solve(res$hessian)
colnames(J) <- colnames(data_ebay)[2:ncol(data_ebay)]
rownames(J) <- colnames(data_ebay)[2:ncol(data_ebay)]


# Draw
beta_samples <- as.matrix(rmvnorm(n=1000, mean = coefficients, sigma = J))
beta_estimates <- apply(beta_samples, 2, mean)

cat("The estimates of the betas using Bayesian approach are:\n", beta_estimates)
```

```
## The estimates of the betas using Bayesian approach are:
##  1.07098 -0.02225978 -0.3966435 0.4447058 -0.05299007 -0.2199956 0.06989023 -0.1208613 -1.893103
```

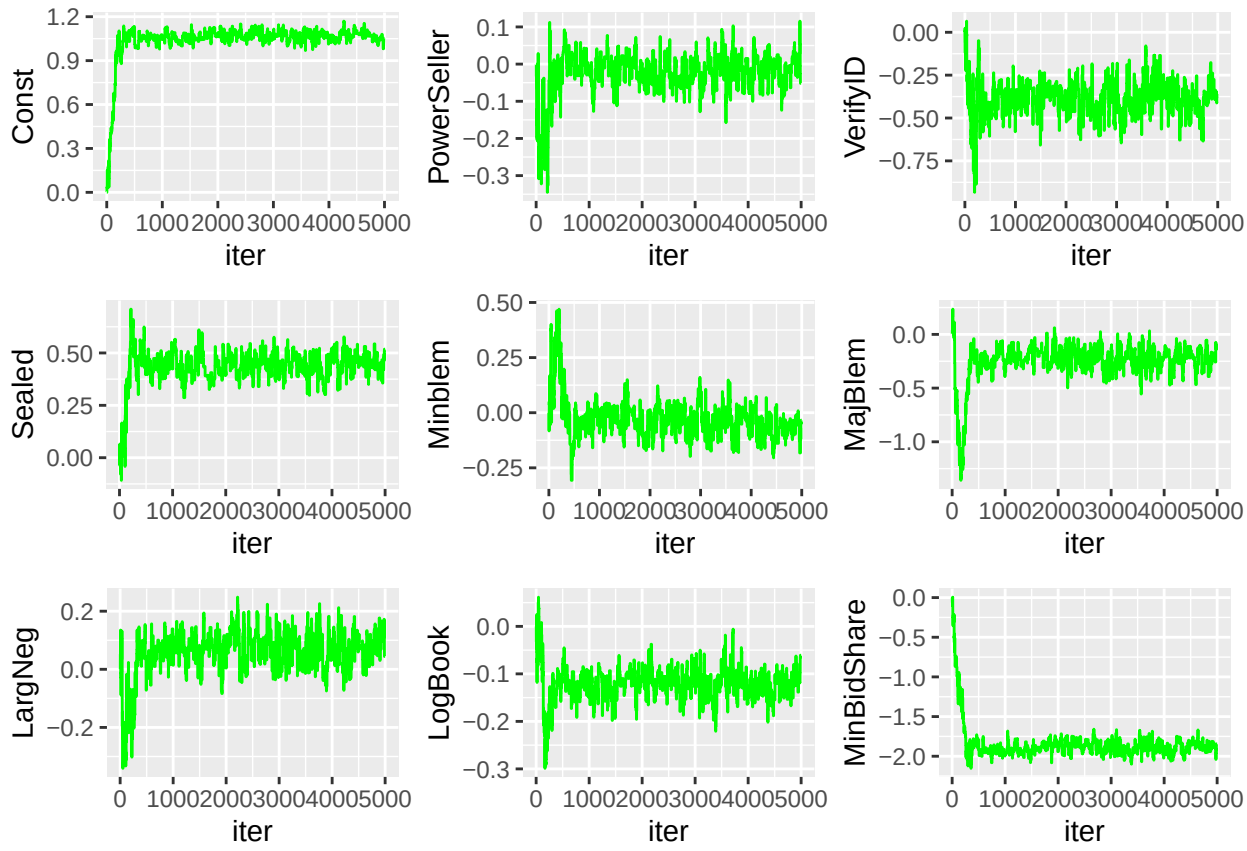
Using a similar code than in the previous lab, we can see that our estimates of the coefficients agree with the glm method applied in the previous question.

Random Walk Metropolis Implementation

```
############# C ############
# Implementation of sampler
Metro_sampler <- function(draws, func, c, mu){
  #initialize generated coefficients
  coefficients <- matrix(0, nrow = draws, ncol = N)
  coefficients[1, ] <- mu


  for(i in 2:draws){
    # proposal
    temp <- as.vector(rmvnorm(1, mean = as.vector(coefficients[i-1, ]), c * as.matrix(sigma_posterior))
    #acceptance probability, log used to avoid overflow issues
    log_prob <- exp(func(temp) - func(coefficients[i-1, ]))
    # accept-reject
    a <- min(1, log_prob)
    u <- runif(1)
    if(u<=a){
      coefficients[i, ] <- temp
    }else{
      coefficients[i, ] <- coefficients[i-1, ]
```

```
    }
  }
  return(coefficients)
}

# posterior function
posterior_distrib <- function(variables){
  log_post <- dmvnorm(variables, beta_estimates, sigma_posterior, log=TRUE)
  return(log_post)
}

# Variables
betas <- rep(0, N)
sigma_posterior <- J
c <- 1

# prior
mu <- rep(0, N)
sigma_prior <-(100 * solve(t(covariates)%*%covariates))
draws <- 5000

# Sampling
new_beta <- Metro_sampler(draws = draws, func= posterior_distrib, c = c, mu = mu)
```

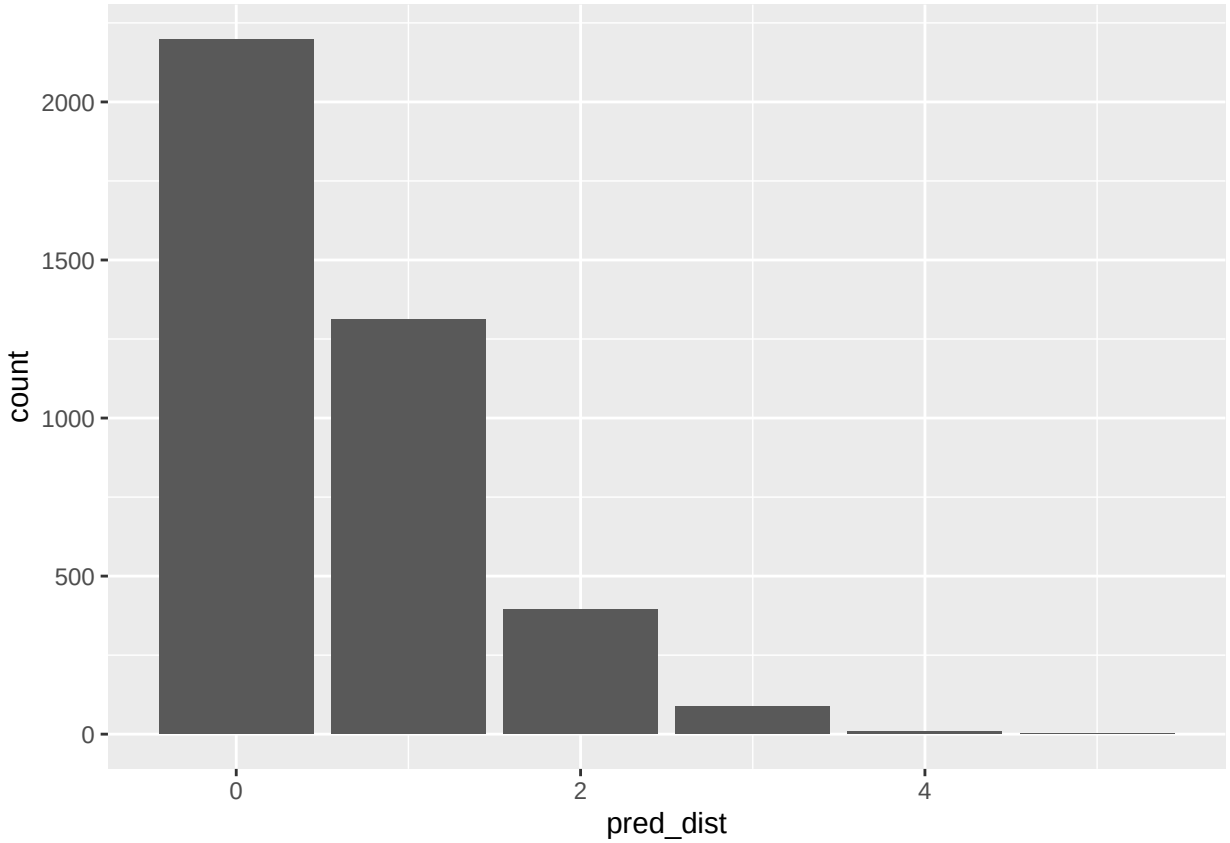
```
#plotting
library(gridExtra)
df_plot_cov <- data.frame(new_beta)
colnames(df_plot_cov) <- colnames(J)

# Plot fun
plot_fun <- function(col, data){
  ggplot(data=data, mapping = aes(x = 1:nrow(data)))+
    geom_line(mapping = aes(y = data[, col]), col = "green")+
    labs(x = "iter", y = col)
}

# prepare grobs for grid.arrange
names <- colnames(df_plot_cov)
grobs <- lapply(X = names, FUN = plot_fun, data = df_plot_cov)

grid.arrange(grobs = grobs, ncol = 3)
```



From the convergence graphs produced, we can see that all chains have converged after a burn-in period of a little more than 1000 iterations.

SImulation and probability of no-bidders

```
params <- c(1,1,1,1,0,1,0,1,0.7) # added 1 for const/bias
artif_data <- exp(new_beta[1000:nrow(new_beta), ] %*% params)

pred_dist <- sapply(artif_data, rpois, n=1)
df_pred_dist <- data.frame(x = 1:length(pred_dist), pred_dist)

pred_dist_plot <- ggplot()+
  geom_bar(data = df_pred_dist, aes(x=pred_dist))

pred_dist_plot
```




```
prob_0 <- sum(pred_dist == 0)/ length(pred_dist)

cat("The probability of having no bidders is: ", prob_0)
```

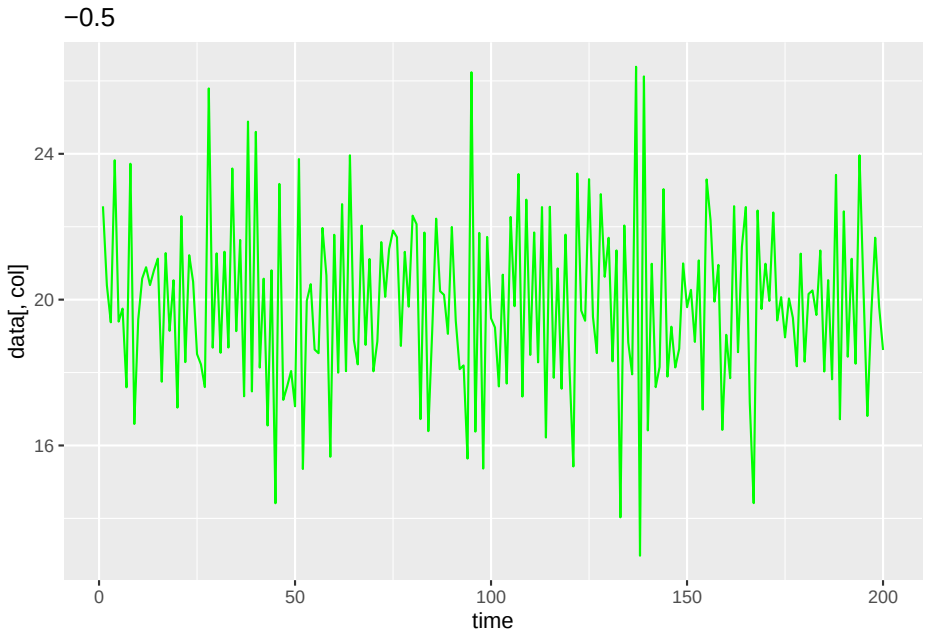
```
## The probability of having no bidders is:  0.5493627
```

Time series model with STAN

SImulation from AR(1) process

```
################## A ##################
AR <- function(phi){
  # set parameters
  mu <- 20
  sigma2 <- 4
  t <- 200

  # initialize
  output <- rep(0, t)
  x_1 <- mu
  output[1] <- mu + rnorm(1, 0, sqrt(sigma2))

  # remaining of the process
  for(i in 2:t){
    output[i] <- mu + phi*(output[i-1] - mu) + rnorm(1, 0, sqrt(sigma2))
  }
  return(output)
}

# testing different values of phi
test_phi <- c(-0.5, 0, 0.3, 0.9, 1)
tests <- sapply(test_phi, AR)
column_names <- test_phi

# Data frame
df_ar <- data.frame(tests)
colnames(df_ar) = column_names


# Plot fun
plot_fun_ar <- function(col, data){
  ggplot(data=data, mapping = aes(x = 1:nrow(data)))+
    geom_line(mapping = aes(y = data[, col]), col = "green")+
    labs(x = "time", title = col)
}

# prepare grobs for grid.arrange
names <- colnames(df_ar)
grobs <- lapply(X = names, FUN = plot_fun_ar, data = df_ar)

# grid.arrange(grobs = grobs, ncol = 2)
grobs
```

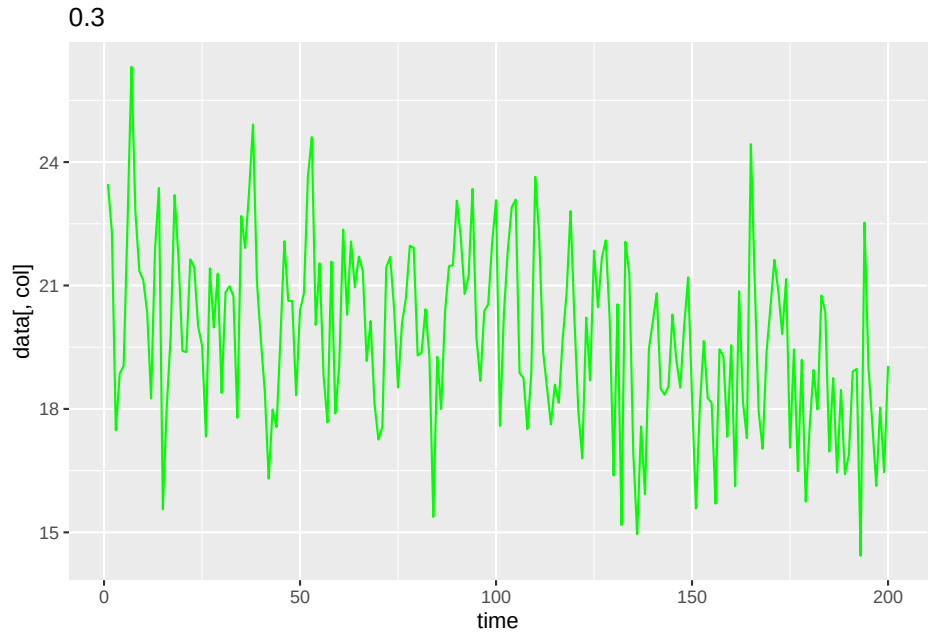
```
## [[1]]
```


−0.5
24
20
16
data[, col]
0
50
100
150
200
time

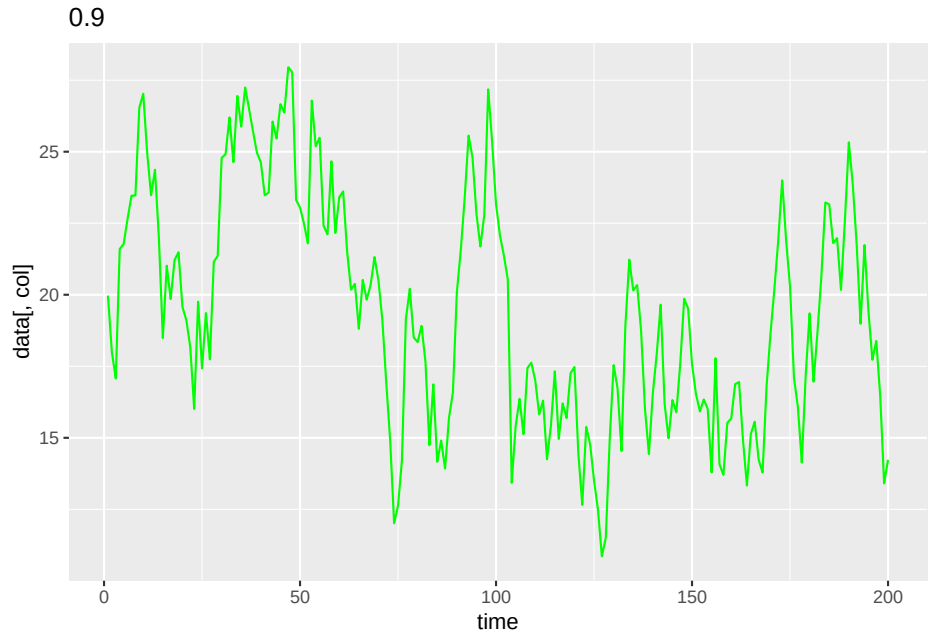
```
##
## [[2]]
```


0
24
20
16
data[, col]
0
50
100
150
200
time

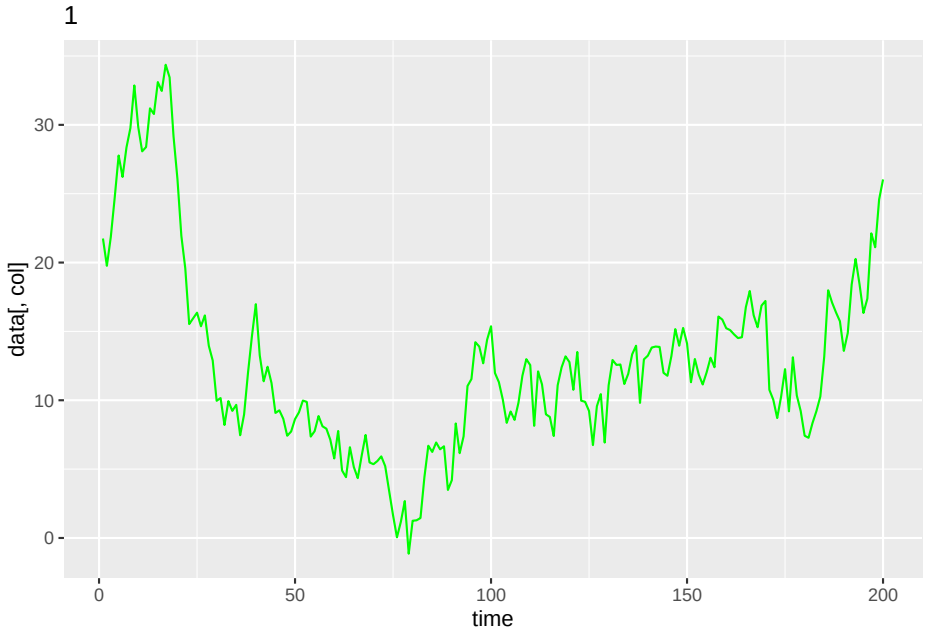
```
##
## [[3]]
```


0.3
data[, col]
24
21
18
15
0
50
100
150
200
time

```
##
## [[4]]
```


0.9
data[, col]
25
20
15
0
50
100
150
200
time

```
##
## [[5]]
```



The smaller the value of ϕ the faster we see fluctuation in the AR process. When ϕ is close to 1, the series is more strongly correlated with the previous value. As ϕ gets closer to 0, this correlation decreases. When the parameter ϕ is equal to 0, we loose the second term of the AR(1) process and the variations are only explained by the random noise without any correlation with the previous point.

Simulate 2 AR(1) processes

```
#################### B ################
library(rstan)
```

```
## Loading required package: StanHeaders
```

```
## rstan (Version 2.21.2, GitRev: 2e1f913d3ca3)
```

```
## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
```

```
## Do not specify '-march=native' in 'LOCAL_CPPFLAGS' or a Makevars file
```

```
# synthetic data
x <- AR(0.3)
y <- AR(0.9)

# stan model
StanModel <- "
data {
  int<lower=0> N;
  vector[N] h;
}

parameters {
  real mu;
  real phi;
```

```
  real<lower=0> sigma;
}

model {
  mu ~ normal(0, 100);
  phi ~ normal(0, 1);
  sigma ~ normal(1, 10);
  h[2:N] ~ normal(mu + phi * (h[1:(N - 1)] - mu), sigma);
}"

data_x <- list(N=length(x), h = x)
data_y <- list(N=length(y), h = y)
warmup <- 1000
niter <- 5000

#fit stan model
fit_x <- stan(model_code=StanModel, data=data_x, warmup=warmup, iter=niter, chains=4)
fit_y <- stan(model_code=StanModel, data=data_y, warmup=warmup, iter=niter, chains=4)
```

```
## Warning: There were 5 divergent transitions after warmup. See
## http://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
```

```
# summary
summary_x <- print(summary(fit_x)$summary)
```

```
##               mean      se_mean         sd         2.5%          25%
## mu      19.8526307 0.0017266890 0.21443773   19.4219157   19.7124711
## phi      0.2868418 0.0005556332 0.06873668    0.1529671    0.2404666
## sigma    2.1250297 0.0008554453 0.10785839    1.9254885    2.0499561
## lp__  -248.0392641 0.0137615798 1.25537260 -251.3480385 -248.5979112
##                50%          75%        97.5%     n_eff      Rhat
## mu      19.8541586   19.9921923   20.2668320 15423.187 1.0001511
## phi      0.2866994    0.3329738    0.4205587 15303.850 1.0000684
## sigma    2.1211545    2.1932563    2.3507580 15897.299 1.0002088
## lp__  -247.7201479 -247.1310233 -246.6300529  8321.636 0.9999554
```

We present here the summary of the process using $\phi = 0.3$. The boundaries of the intervals are respectively presented in the columns 2.5% and 97.5%. The number of effective posterior samples are marked under n_eff. We note here that the 95% CI for μ is quite small compared to the one for ϕ. In order to assess the convergence of the samplers, we turn to the last column Rhat. From the documentation, this value "compares the between- and within-chain estimates for the model parameters". We if these estimates don't agree, the value will be larger than 1. It is recommended to discard samples with Rhat greater than 1.05. In our case, the estimates agree and we are confident about convergence.

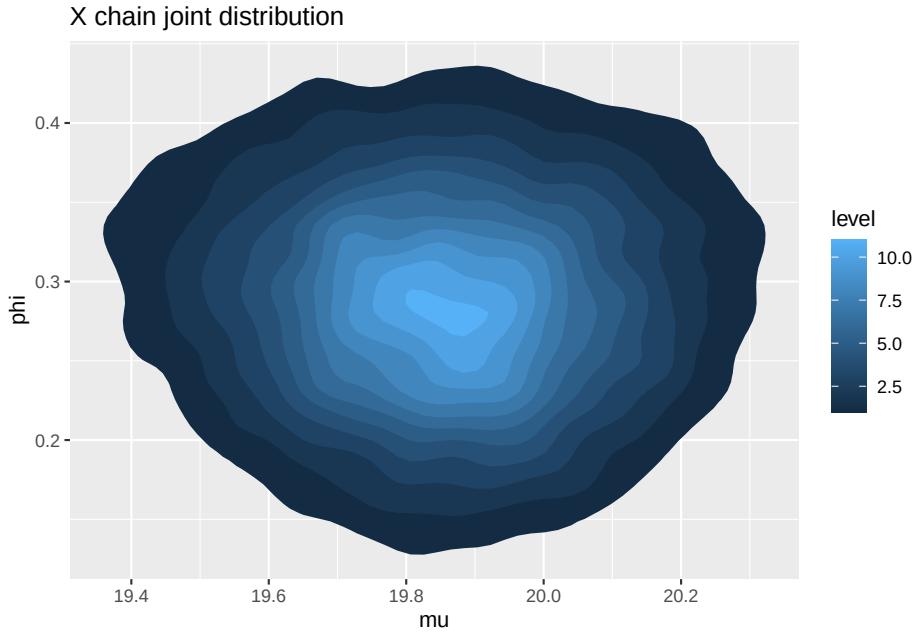
```
# summary
summary_y <- print(summary(fit_y)$summary)
```

```
##               mean      se_mean         sd         2.5%         25%
## mu      22.5652225 0.0401566835 2.20463901   18.1011150   21.602237
## phi      0.9080688 0.0004581712 0.03485464    0.8408974    0.884399
## sigma    1.9156116 0.0009711571 0.09906694    1.7345690    1.846610
## lp__  -227.9196124 0.0213451431 1.34809025 -231.3715323 -228.557161
##                50%          75%        97.5%     n_eff     Rhat
```

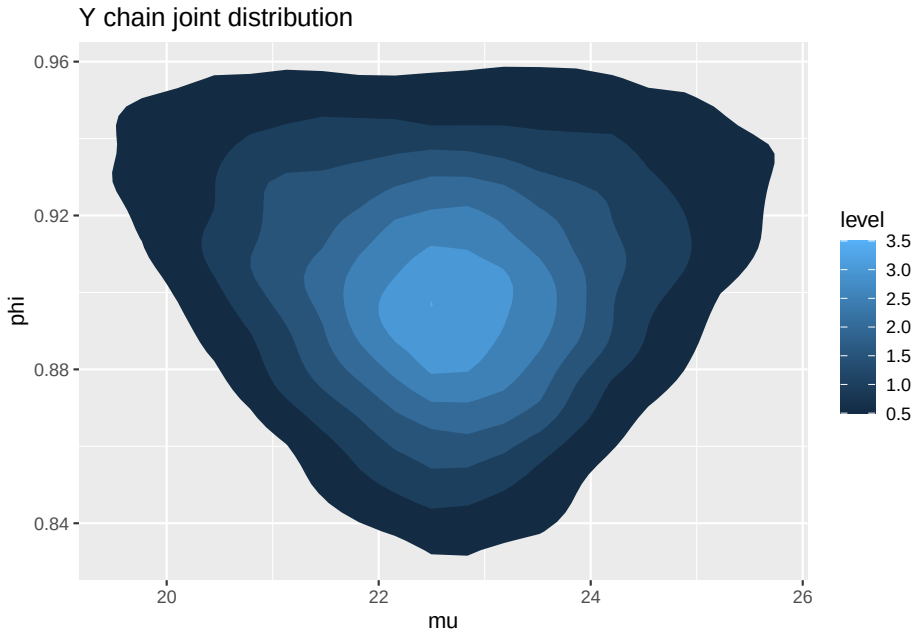
```
## mu      22.6184352   23.6089191   26.5319125  3014.111 1.000958
## phi      0.9073442    0.9313198    0.9789895  5787.163 1.000515
## sigma    1.9115744    1.9803124    2.1215906 10405.873 1.000422
## lp__  -227.5671615 -226.9170054 -226.3578070  3988.777 1.001143
```

Here we have the summary of the process using $\phi = 0.9$. Using the same assessment as previously, we here have values of Rhat greater than 1 for both mu and phi. However, these values fall below the recommended 1.05 and we accept the sample estimates.

```
# get posterior samples
post_x <- extract(fit_x)
post_y <- extract(fit_y)

# Plot joint distributions
df_x <- data.frame(mu = post_x$mu, phi = post_x$phi)
df_y <- data.frame(mu = post_y$mu, phi = post_y$phi)

   # Show the area only
plot_jointx <- ggplot(data = df_x, aes(x=mu, y=phi) ) +
  stat_density_2d(aes(fill = ..level..), geom = "polygon") +
  labs(title = "X chain joint distribution", x = "mu", y = "phi")

plot_jointy <- ggplot(data = df_y, aes(x=mu, y=phi) ) +
  stat_density_2d(aes(fill = ..level..), geom = "polygon") +
  labs(title = "Y chain joint distribution", x = "mu", y = "phi")


plot_jointx
```




```
plot_jointy
```



The joint posterior plot of the X-chain ($\phi = 0.3$) shows an ellipsoid shape that can easily be modeled by a bivariate normal distribution. The estimate of the mean is easier to assess for symmetry reasons. The marked small CI interval for ϕ can be seen here. The joint posterior of the other chain presents a skewed distribution that cannot be easily modeled by a bivariate normal. We would need a more complex model in order to properly model this. Furthermore, the CI for ϕ is large in this case as reflected in this distribution.

Appendix: All code for this report

```
knitr::opts_chunk$set(echo = TRUE)
# rm(list = ls())

########### A ###############
library(ggplot2)
library(mvtnorm)
# working directory change as needed
setwd("C:/Users/nicol/Documents/MSc/MSc_Statistics_and_Machine_Learning/Bayesian Learning 732A73/lab3")


# Reading data and transform to log
data_raw <- read.table("rainfall.dat")
data_log <- unlist(log(data_raw))

# Scaled inverse chisq
#  Inverse chisquare
rInvChisq <- function(draws, n, tau_sq){
  #  n are the degrees of freedom
  X <- rchisq(draws, n)
  sample <- (tau_sq * n)/X
  return(sample)
}

# posterior for mu (lecture 7)
mu_post <- function(x, var, mu_0, tau_0){
  n <- length(x)
  tau_n <- 1 / ((n/var) + (1 / tau_0))
  w <- (n/var) / ((n/var) + (1 / tau_0))
  mu_n <- w * mean(x) + (1-w) * mu_0

  return(rnorm(1, mu_n, sqrt(tau_n)))
}

# posterior for sigma (lecture 7)
var_post <- function(x, mu, v_0, var_0){
  n <- length(x)
  v_n <- v_0 + n
  variance <- (v_0*var_0 + sum((x-mu)^2)) / v_n
  return(rInvChisq(1, v_n, variance))
}

# initialize values
mu_0 <- 0
sigma_0 <- 1
tau_0 <- 1
nu_0 <- 1
draws <- 5000


# Gibbs sampler
gibbs_sampler <- function(data, mu_0, sigma_0, tau_0, nu_0, draws){
  mu_val <- c(mu_0, rep(0, draws))
```

```r
  sigma_val <- c(sigma_0, rep(0, draws))
  # tau_val <- c(tau_0, rep(0, draws))
  # nu_val <- c(nu_0, rep(0, draws))
  dist_val <- c(rnorm(1, mu_0, sigma_0), rep(0, draws))

  for(i in 2:draws){
    mu_val[i] <- mu_post(x = data_log, var = sigma_val[i-1], mu_0 = mu_val[i-1], tau_0 = tau_0)
    sigma_val[i] <- var_post(x = data_log, mu = mu_val[i], v_0 = (nu_0+length(data)), var_0 = sigma_0)
    dist_val[i] <- rnorm(1, mu_val[i], sqrt(sigma_val[i]))

    # # Update tau and nu (why do we get correlated draws?) -> (because tau and nu are not randomly sam
    # tau_val[i] <- 1 / ((length(data)/sigma_val[i-1]) + (1 / tau_val[i-1]))
    # nu_val[i] <- nu_val[i-1] + length(data)
  }
  final_df <- data.frame(mu_sample = mu_val, sigma_sample = sigma_val, norm_dist = dist_val)
  return(final_df)
}

# Gibbs sample
sample_gibbs <- gibbs_sampler(data = data_log, mu_0 = mu_0, sigma_0 = sigma_0, tau_0 = tau_0, nu_0 = nu
df_sample_gibbs <- data.frame(sample_gibbs[2:(draws-1), ], x = 2:(draws-1))

plot_mu <- ggplot(data =df_sample_gibbs, aes(x = x))+
  geom_line(aes(y = mu_sample))+
  geom_hline(yintercept = mean(df_sample_gibbs$mu_sample), color = "green")+
  labs(title = "Convergence plot mu", x = "iter", y = "value")

plot_sigma <- ggplot(data =df_sample_gibbs, aes(x = x))+
  geom_line(aes(y = sigma_sample))+
  geom_hline(yintercept = mean(df_sample_gibbs$sigma_sample), color = "green")+
  labs(title = "Convergence plot sigma", x = "iter", y = "value")

plot_mu
plot_sigma

# autocorrelation
# First and last values removed as outliers
mu_autocorr <- acf(sample_gibbs$mu_sample[2:5000], plot = FALSE)
sigma_autocorr <- acf(sample_gibbs$sigma_sample[2:5000], plot = FALSE)
# Inefficiency factor
# Remove first value to get lag from k=1
IF_mu <- 1 + 2 * sum(mu_autocorr$acf[-1])
IF_sigma <- 1 + 2 * sum(sigma_autocorr$acf[-1])
cat("The inefficiency factor for mu is: ", IF_mu,"\n")
cat("The inefficiency factor for sigma is: ", IF_sigma)

data_dens <- density(exp(data_log))
sim_dens <- density(exp(df_sample_gibbs$norm_dist))

plot(x = sim_dens$x, y = sim_dens$y, col = "red", type = "l")
lines(x = data_dens$x, y = data_dens$y, col = "green")

############## A ############
```

```r
data_ebay <- read.table("eBayNumberOfBidderData.dat", stringsAsFactors = FALSE, header = TRUE)
data_ebay_glm <- data_ebay[, -2]

#fit glm model
glm_model <- glm(formula = nBids~., data = data_ebay_glm, family = "poisson")

summary(glm_model)
library(mvtnorm)

log_posterior <- function(Beta, X, y, mu, sigma){

  loglik <- sum(-log(factorial(y)) + X%*%Beta * y - exp(X%*%Beta))
  log_prior <- dmvnorm(Beta, mu, sigma, log=TRUE)

  #log posterior is loglik + log_prior instead of lik*prior because of the logarithm
  return(loglik + log_prior)
}

# set up the data
covariates = as.matrix(data_ebay[, -1])
target <- as.vector(data_ebay[, 1])
N <- ncol(covariates)
mu <- rep(0, N)
sigma <- (100 * solve(t(covariates)%*%covariates))
init <- rep(0, N)

res <- optim(init, log_posterior, X=covariates, y=target, mu=mu, sigma=sigma, method = "BFGS", control =

coefficients <- res$par
J <- -solve(res$hessian)
colnames(J) <- colnames(data_ebay)[2:ncol(data_ebay)]
rownames(J) <- colnames(data_ebay)[2:ncol(data_ebay)]


# Draw
beta_samples <- as.matrix(rmvnorm(n=1000, mean = coefficients, sigma = J))
beta_estimates <- apply(beta_samples, 2, mean)

cat("The estimates of the betas using Bayesian approach are:\n", beta_estimates)
############# C ############
# Implementation of sampler
Metro_sampler <- function(draws, func, c, mu){
  #initialize generated coefficients
  coefficients <- matrix(0, nrow = draws, ncol = N)
  coefficients[1, ] <- mu


  for(i in 2:draws){
    # proposal
    temp <- as.vector(rmvnorm(1, mean = as.vector(coefficients[i-1, ]), c * as.matrix(sigma_posterior))
    #acceptance probability, log used to avoid overflow issues
    log_prob <- exp(func(temp) - func(coefficients[i-1, ]))
    # accept-reject
```

```
    a <- min(1, log_prob)
    u <- runif(1)
    if(u<=a){
      coefficients[i, ] <- temp
    }else{
      coefficients[i, ] <- coefficients[i-1, ]
    }
  }
  return(coefficients)
}

# posterior function
posterior_distrib <- function(variables){
  log_post <- dmvnorm(variables, beta_estimates, sigma_posterior, log=TRUE)
  return(log_post)
}

# Variables
betas <- rep(0, N)
sigma_posterior <- J
c <- 1

# prior
mu <- rep(0, N)
sigma_prior <-(100 * solve(t(covariates)%*%covariates))
draws <- 5000

# Sampling
new_beta <- Metro_sampler(draws = draws, func= posterior_distrib, c = c, mu = mu)
#plotting
library(gridExtra)
df_plot_cov <- data.frame(new_beta)
colnames(df_plot_cov) <- colnames(J)

# Plot fun
plot_fun <- function(col, data){
  ggplot(data=data, mapping = aes(x = 1:nrow(data)))+
    geom_line(mapping = aes(y = data[, col]), col = "green")+
    labs(x = "iter", y = col)
}

# prepare grobs for grid.arrange
names <- colnames(df_plot_cov)
grobs <- lapply(X = names, FUN = plot_fun, data = df_plot_cov)

grid.arrange(grobs = grobs, ncol = 3)
params <- c(1,1,1,1,0,1,0,1,0.7) # added 1 for const/bias
artif_data <- exp(new_beta[1000:nrow(new_beta), ] %*% params)

pred_dist <- sapply(artif_data, rpois, n=1)
df_pred_dist <- data.frame(x = 1:length(pred_dist), pred_dist)

pred_dist_plot <- ggplot()+
```

```r
  geom_bar(data = df_pred_dist, aes(x=pred_dist))

pred_dist_plot

prob_0 <- sum(pred_dist == 0)/ length(pred_dist)

cat("The probability of having no bidders is: ", prob_0)
################## A ##################
AR <- function(phi){
  # set parameters
  mu <- 20
  sigma2 <- 4
  t <- 200

  # initialize
  output <- rep(0, t)
  x_1 <- mu
  output[1] <- mu + rnorm(1, 0, sqrt(sigma2))

  # remaining of the process
  for(i in 2:t){
    output[i] <- mu + phi*(output[i-1] - mu) + rnorm(1, 0, sqrt(sigma2))
  }
  return(output)
}

# testing different values of phi
test_phi <- c(-0.5, 0, 0.3, 0.9, 1)
tests <- sapply(test_phi, AR)
column_names <- test_phi

# Data frame
df_ar <- data.frame(tests)
colnames(df_ar) = column_names


# Plot fun
plot_fun_ar <- function(col, data){
  ggplot(data=data, mapping = aes(x = 1:nrow(data)))+
    geom_line(mapping = aes(y = data[, col]), col = "green")+
    labs(x = "time", title = col)
}

# prepare grobs for grid.arrange
names <- colnames(df_ar)
grobs <- lapply(X = names, FUN = plot_fun_ar, data = df_ar)

# grid.arrange(grobs = grobs, ncol = 2)
grobs

#################### B #################
library(rstan)
```

```
# synthetic data
x <- AR(0.3)
y <- AR(0.9)

# stan model
StanModel <- "
data {
  int<lower=0> N;
  vector[N] h;
}

parameters {
  real mu;
  real phi;
  real<lower=0> sigma;
}

model {
  mu ~ normal(0, 100);
  phi ~ normal(0, 1);
  sigma ~ normal(1, 10);
  h[2:N] ~ normal(mu + phi * (h[1:(N - 1)] - mu), sigma);
}"

data_x <- list(N=length(x), h = x)
data_y <- list(N=length(y), h = y)
warmup <- 1000
niter <- 5000

#fit stan model
fit_x <- stan(model_code=StanModel, data=data_x, warmup=warmup, iter=niter, chains=4)
fit_y <- stan(model_code=StanModel, data=data_y, warmup=warmup, iter=niter, chains=4)

# summary
summary_x <- print(summary(fit_x)$summary)
# summary
summary_y <- print(summary(fit_y)$summary)
# get posterior samples
post_x <- extract(fit_x)
post_y <- extract(fit_y)

# Plot joint distributions
df_x <- data.frame(mu = post_x$mu, phi = post_x$phi)
df_y <- data.frame(mu = post_y$mu, phi = post_y$phi)

   # Show the area only
plot_jointx <- ggplot(data = df_x, aes(x=mu, y=phi) ) +
  stat_density_2d(aes(fill = ..level..), geom = "polygon") +
  labs(title = "X chain joint distribution", x = "mu", y = "phi")

plot_jointy <- ggplot(data = df_y, aes(x=mu, y=phi) ) +
  stat_density_2d(aes(fill = ..level..), geom = "polygon") +
  labs(title = "Y chain joint distribution", x = "mu", y = "phi")
```

```
plot_jointx
plot_jointy
```