Lab 3

Alejo Perez Gomez(alepe026), Martynas Lukosevicius(marlu207)

Gibbs Sampling for a normal model

a) implementation of Gibbs Sampler

Let us implement a Gibbs sampler function to draw samples given $ln(y_1), ...ln(y_2) \mid \mu, \sigma^2$. We have as priors $\mu \sim \mathcal{N}(\mu, \sigma^2)$ and $\sigma^2 \sim Inv - \chi^2(\nu, \sigma^2)$. We will assume as initial values for $\mu_o = 0, \nu_0 = 1, \sigma^2 = 1, \tau^2 = 1$. After that, we will make 3 functions. *rmu_norm* will sample from the full conditional posterior:

$$p(\mu \mid \sigma^2, Y) \sim \mathcal{N}(\mu_n, \tau_n^2)$$

where

$$\frac{1}{\tau_n^2} = \frac{n}{\sigma^2} + \frac{1}{\tau_0^2}$$

being $n = lenght(datapoints)$ and

$$\mu_n = w\bar{Y} + (1 - w)\mu_o$$

being $\bar{Y} = \sum Y_i$

$$w = \frac{\frac{n}{\sigma^2}}{\frac{n}{\sigma^2} + \frac{1}{\tau_o^2}}$$

r_full_cond_var in turn, will use *r_inv_chi_draw* to draw from the full conditional distribution of σ^2:

$$p(\sigma^2 \mid \mu, Y) \sim Inv - \chi^2(\frac{\nu_n, \nu_o\sigma_o^2 + \sum_{i=1}^{n}(X_i - \mu)^2}{\nu_n})$$

being $\nu_n = n + \nu_o$

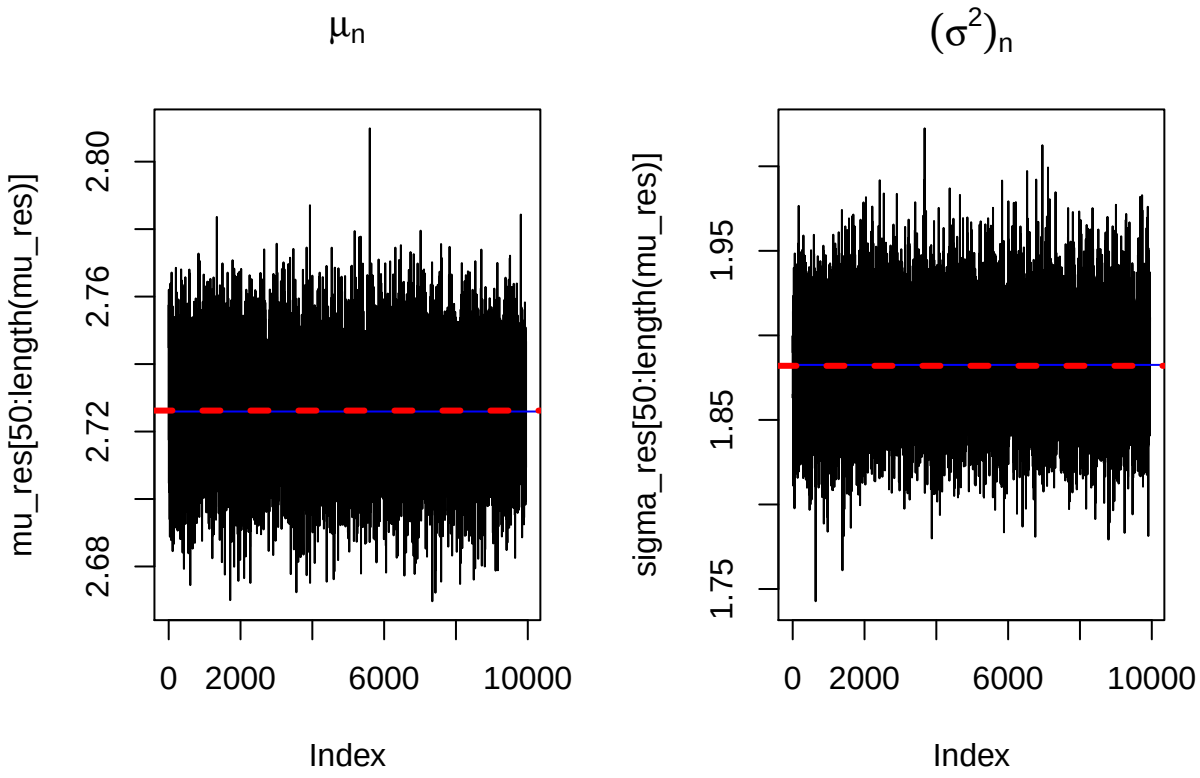
```
rmu_norm <- function(mu_0, tau_0_sqrt, sigma_sqrt, x_bar, n){

  inv_tau_n_sqrt <- (n/sigma_sqrt) + (1/tau_0_sqrt)
  w <- (n/sigma_sqrt) / ((n/sigma_sqrt) + (1/tau_0_sqrt))
  mu_n <- (w*x_bar) + (1-w)*mu_0

  return(rnorm(1, mu_n, sqrt(1/inv_tau_n_sqrt)))
}


r_inv_chi_draw <- function(dof,var_cust){
  X <- rchisq(1, dof)
  return((dof*var_cust)/X)
}


r_full_cond_var <- function(data, vn, mu,v0, var0){
```

```
  n <- length(data)
  sum_data <- sum((data-mu)^2)
  var_c <- ((v0*var0)+sum_data)/(n+v0)
  return(r_inv_chi_draw(vn,var_c))
}
```

Therefore, our Gibbs sampler will draw first a μ_n sample using the sampled values of the last iteration of μ_{n-1} and σ^2_{n-1}. A sample σ^2_n will in this same iterration draw from its posterior conditional by using μ_n.

```
rainfall_data <- log(unlist(read.table("rainfall.dat", header=FALSE)))
x_bar <- mean(rainfall_data)
n <- length(rainfall_data)

mu_0 <- 0
sigma_0 <- 1
tau_0 <- 1
v_0 <- 1

n_it <- 10000

gibbs <- function(n_it, mu_0, sigma_0, tau_0, x_bar, n, v_0, data){
  mu_vec <- rep(NA, n_it+1)
  mu_vec[1] <- mu_0
  sigma_vec <- rep(NA, n_it+1)
  norm_vec <- rep(NA, n_it+1)
  sigma_vec[1] <- sigma_0
  norm_vec[1] <- rnorm(1,0,1)

  for(i in 2:(n_it+1)){

    mu_vec[i] <- rmu_norm(mu_0 = mu_vec[i-1], tau_0_sqrt = tau_0, sigma_sqrt = sigma_vec[i-1], x_bar = x
    sigma_vec[i] <- r_full_cond_var(data = data, vn = (v_0+n), mu = mu_vec[i], v0 = v_0, var0 = sigma_0
    norm_vec[i] <- rnorm(1,mu_vec[i],sqrt(sigma_vec[i]))
  }

  return(list(mu_n=mu_vec, sigma_n=sigma_vec, r_data = norm_vec))
}

res <- gibbs(n_it, mu_0, sigma_0, tau_0, x_bar, n, v_0, data=rainfall_data)
mu_res <- res$mu_n
sigma_res <- res$sigma_n
```

We can afterwards see that the converged estimates for μ and σ^2 converge to the mean and variance of the sample. Note we excluded the burn out period in the plot.



Let us assess the Inefficiency Factors of the sample. This inefficiency factor is interpreted as the numerical variance of the posterior mean from the MCMC chain to the variance of the posterior mean from hypothetical independent draws.

```
a_Gibbs <- acf(sigma_res, plot = FALSE)

IF_Gibbs <- 1+2*sum(a_Gibbs$acf[-1])
cat("ineficiency sigma:", IF_Gibbs)
```

```
## ineficiency sigma: 0.9487603
```

```
a_Gibbs <- acf(mu_res, plot = FALSE)

IF_Gibbs <- 1+2*sum(a_Gibbs$acf[-1])
cat("\n ineficiency mu:", IF_Gibbs)
```

```
##
##  ineficiency mu: 1.038652
```

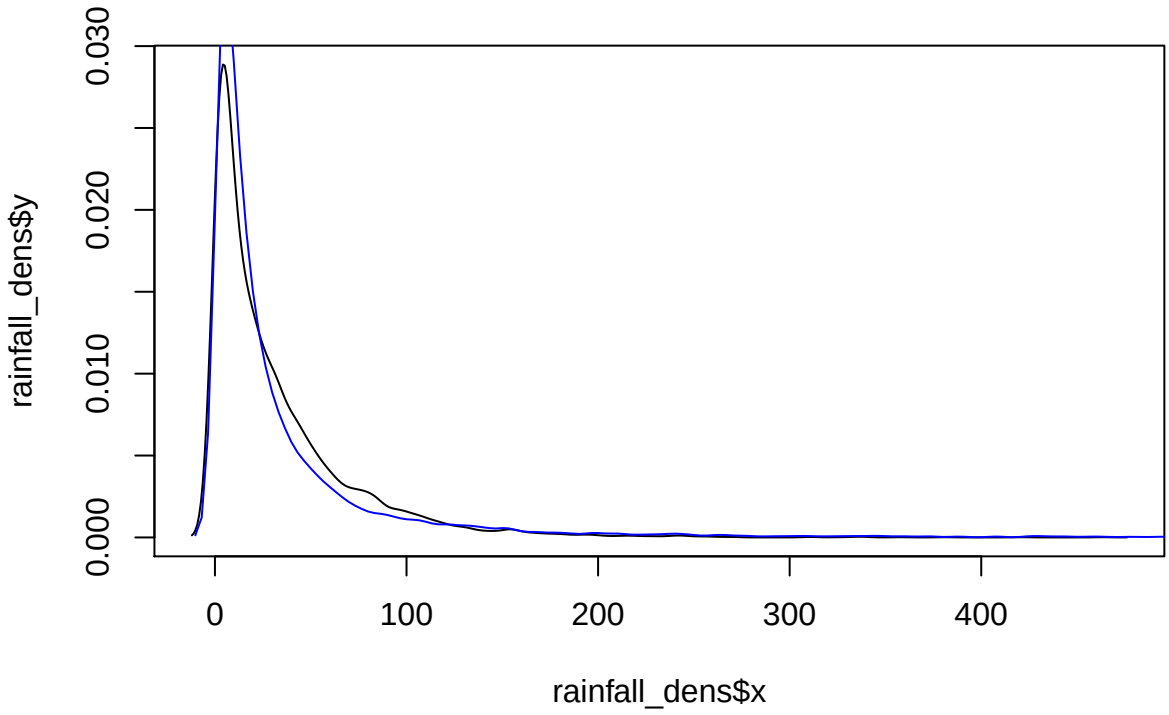
This low IF leads us to think that our sampler is efficient in the sense that samples are not overly correlated.

b) plot of density estimate of the data and simulated posterior draws

In this part we will be estimating the kernel density of the actual data and comparing it with the estimated kernel density out of the posterior simulated draws. We can see in the plot both shapes fit each other

noticeably leading us to think the simulation has capture the underlying data distribution. Notice the aplication of exponential function in both the data and the simulation as they have the log function applied beforehand.

```
rainfall_dens <- density(exp(rainfall_data))
simulated_dens <- density(exp(res$r_data))

plot(x = rainfall_dens$x, y = rainfall_dens$y, type="l")
lines(x = simulated_dens$x, y = simulated_dens$y, col = "blue")
```



2) Metropolis Random Walk for Poisson regression

a)

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

Maximum likelihood estimates of β estimated with glm function (family = poisson)

```
ebay_data <- read.table("eBayNumberOfBidderData.dat", header=TRUE)
ebay_data_wo_const <- ebay_data[,-2]

glm(nBids~., family = poisson, data = ebay_data_wo_const)
```

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

```
##
## Coefficients:
## (Intercept)  PowerSeller     VerifyID       Sealed      Minblem      MajBlem
##     1.07244     -0.02054     -0.39452      0.44384     -0.05220     -0.22087
##     LargNeg      LogBook  MinBidShare
##     0.07067     -0.12068     -1.89410
##
## Degrees of Freedom: 999 Total (i.e. Null);  991 Residual
## Null Deviance:       2151
## Residual Deviance: 867.5     AIC: 3610
```

significant covariates are VerifyId, Sealed, MinBidShare

b)

prior: $\beta \sim N[0, 100 * (X^T X)^{-1}]$

approximate posterior: $\beta|y \sim N(\tilde{\beta}, J_y^{-1}(\tilde{\beta}))$

posterior mode and jacobian matrix will be obtained using BFGS optimization algorithm, maximizing Log Posterior

$$log(p(\beta|X,y)) = (\sum_i log(exp(-exp(X\beta) * \frac{1}{y_i!} * (exp(X\beta)^{y_i}))) + log(p(\beta))$$

```
# Setting up the prior
library(mvtnorm)
```

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

```
Nfeat <- ncol(ebay_data)-1
mu <- as.matrix(rep(0,Nfeat)) # Prior mean vector
init_val <- matrix(0,Nfeat,1)
X <- as.matrix(ebay_data[,-1])
y <- as.matrix(ebay_data[1])
cov_prior <- 100 * solve(t(X) %*% X)

LogPostpoisson <- function(betas,y,X,mu,Sigma){
  lambda <- exp(X%*%betas)
  logLik <- sum(log(exp(-lambda) * (1/factorial(y)) * (lambda^y)))
  logPrior <- dmvnorm(betas, mu, Sigma, log=TRUE)

  return(logLik + logPrior)
}

# Optimization for the beta estimates Hessian

OptimRes <- optim(init_val,LogPostpoisson, gr =NULL, y = y, X = X, mu = mu, Sigma = cov_prior, method=c

beta_estimates <- OptimRes$par
hessian <-  OptimRes$hessian
hessian_inv <- solve(-hessian)
```

Const	PowerSeller	VerifyID	Sealed	Minblem	MajBlem	LargNeg	LogBook	MinBidShare
1.07	-0.0205	-0.393	0.444	-0.0525	-0.221	0.0707	-0.12	-1.89

As we can see from results of task a) posterior betas gained from BFGS optimisation are close to betas gained using glm function

c)

for metropolis hasting algorithm proposal density will be $\theta_p|\theta^{i-1} \sim N(\theta^{(i-1)}, c * \Sigma)$

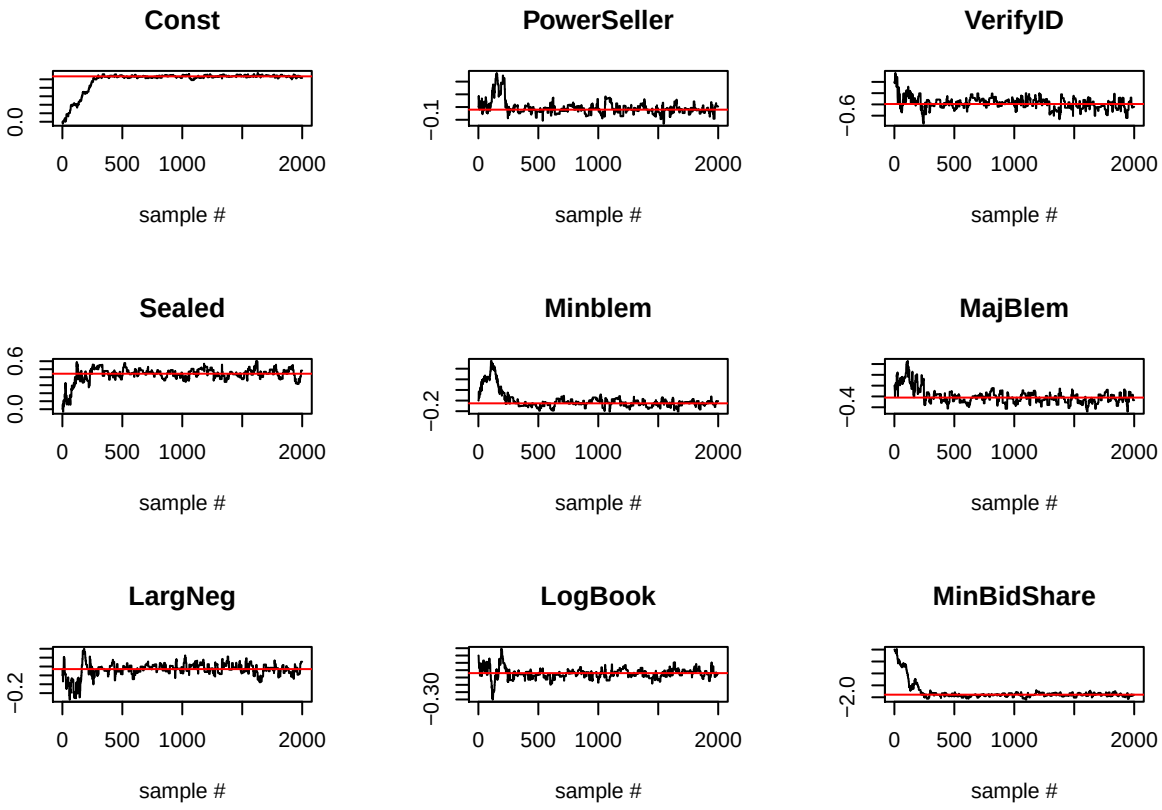
```
metropolis <- function(dens, c, n){
  var_c <- c * hessian_inv
  x0 <- rep(0,9)
  x <- matrix(0,n,9)
  x[1,] <- x0
  alphas <- vector()

  for (t in 2:n) {
    unif <- runif(1)
    rnd <- rmvnorm(1, x0, var_c)
    alpha <- min(1, exp(dens(rnd) - dens(x0))) ## its simetric
    if(unif<alpha){
      x0 <- rnd
    }
    x[t,] <- x0
    #alphas <- append(alphas, alpha)
  }
  return(x)
}
```

density we want to sample from is $\beta|y \sim N(\tilde{\beta}, J_y^{-1}(\tilde{\beta}))$

```
posterior <- function(x){
 return(dmvnorm(x, beta_estimates ,hessian_inv, log = TRUE))
}
```

```
metropolis_samples <-  metropolis(posterior,1,2000)
```

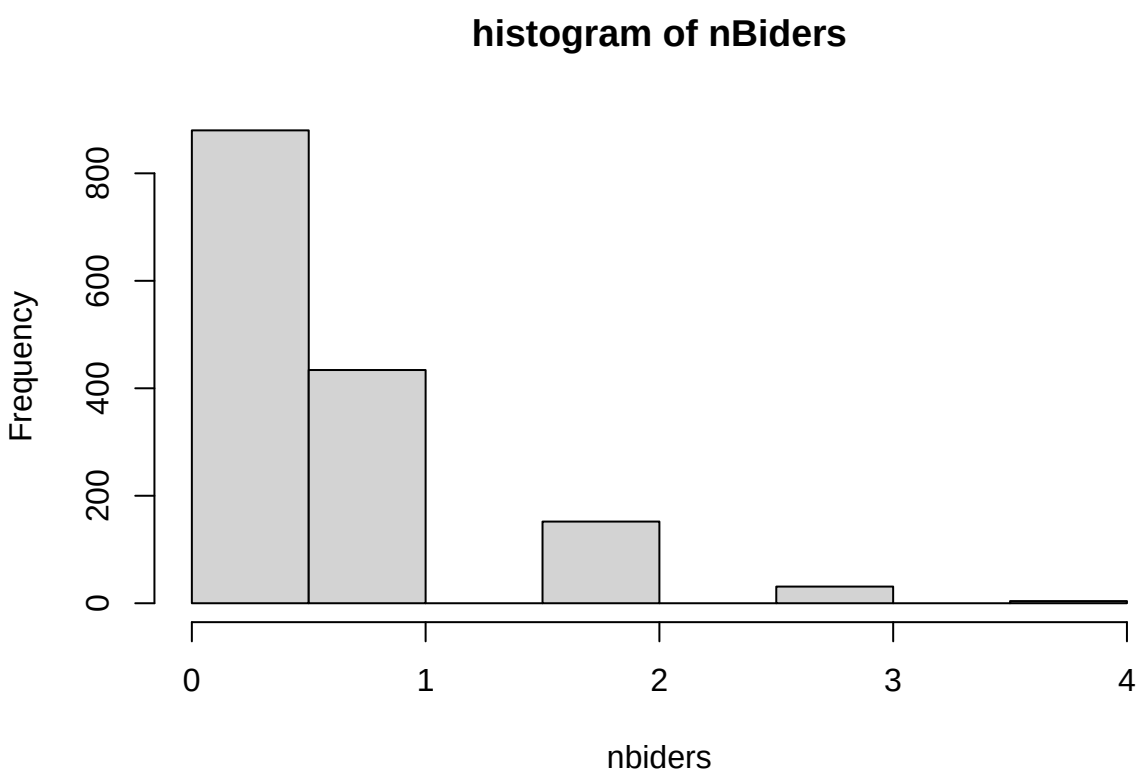
```
#cols <- c("black", "red", "blue", "forestgreen", "purple", "yellow", "darkgreen", "dodgerblue1", "gold
par(mfrow=c(3,3))
for (i in 1:9) {
  plot(metropolis_samples[,i], type = "l", main = colnames(ebay_data)[-1][i], xlab = "sample #", ylab="
  abline(h = beta_estimates[i], col = "red")
}
```



From graphs we can see that approx first 500 samples are burn in period, after it samples converged to β modes

d)

```
new_x <- c(1,1,1,1,0,1,0,1,0.7)
lambdas <- exp(metropolis_samples[500:nrow(metropolis_samples) ,] %*% new_x) # getting rid of burn in p
gen_data <- sapply(lambdas, rpois, n = 1) # sampling from poisson
hist(gen_data, main = "histogram of nBiders", xlab = "nbiders")
```




```
prob_no_bidds <- sum(gen_data == 0)/length(gen_data)
```

probability of no bidders = 0.5862758

Time series models in Stan

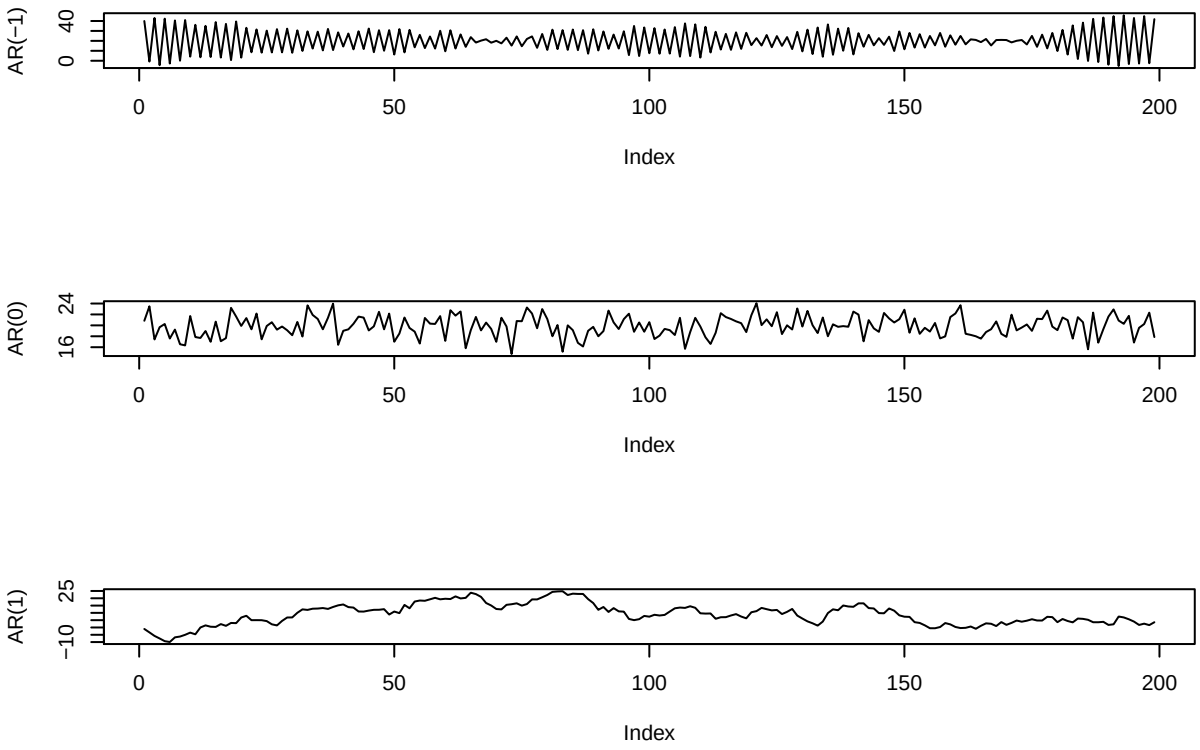
a) Implementation of the AR(1)-process simulation

Given the AR(1)-process.

$$x_t = \mu + \phi(x_{t-1} - \mu) + \varepsilon_t, \varepsilon_t \sim \mathcal{N}(0, \sigma^2)$$

```
AR <- function(phi){
  mu <- 20
  var_c <- 4
  n_T <- 200
  x <- 0
  res <- vector()
    for (i in 2:n_T) {
     x <- mu + phi*(x-mu)+rnorm(1,0,sqrt(var_c))
     res <- append(res,x)
    }
  return(res)
}
```

Down below, we will simulate with a *number of iterations n_T= 200* and $\phi_1 = -1, \phi_2 = 0, \phi_3 = 1$. From the plots we can notice that this ϕ parameters roughly controls the period of fluctuation in an inversely proportional basis. Intuitively, this concrete autoregressive process is regulated by the parameter ϕ. In the stationary range of this model, we can observe the correlation towards the previous sample increases when the parameter approaches 1, because it puts a considerable weight on the previous sample rather than the normal noise. When the parameter get closer to 0 though, the succession of samples is just identically distributed by normal distribution plus a constant. We can conclude this stochastic process is higly dependent on the very first sample observed, that will be correlated with the lagged subsequent observations controlled by the ϕ parameter.



b)

Once the simulation is executed, we will be using $\phi_x = 0.3, \phi_3 = 0.9$ to simulate by means of *stan*. Please find attached the code below contained in the **stan_model.stan** file.

```
#data {
#  int<lower=0> N;
#  vector[N] y;
#}
#parameters {
#  real mu;
#  real phi;
#  real<lower=0> sigma;
#}
#model {
#  for (n in 2:N)
```

```
#    y[n] ~ normal(mu + phi*(y[n-1]-mu), sigma);
#}
```

The time series with the required parameters will be generated below.

```
synthetic_x <- list(N = 199,
                    y = AR(0.3))
synthetic_y <- list(N = 199,
                    y = AR(0.9))
```

Let us fit the stan model to carry out the simulation from the posterior distribution to make Bayesian inference on the parameters (π, μ, σ^2) by means of Stan library. The respective prior probabilities will be proportioned by the default ones by Stan (Notice the lower constraint for σ^2 preventing it to be lower than 0).

```
library(rstan)
```

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

```
## Warning: package 'StanHeaders' was built under R version 4.0.5
```

```
## Loading required package: ggplot2
```

```
## 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
```

```
fit <- stan(file = 'stan_model.stan', data = synthetic_x, verbose = FALSE)
```

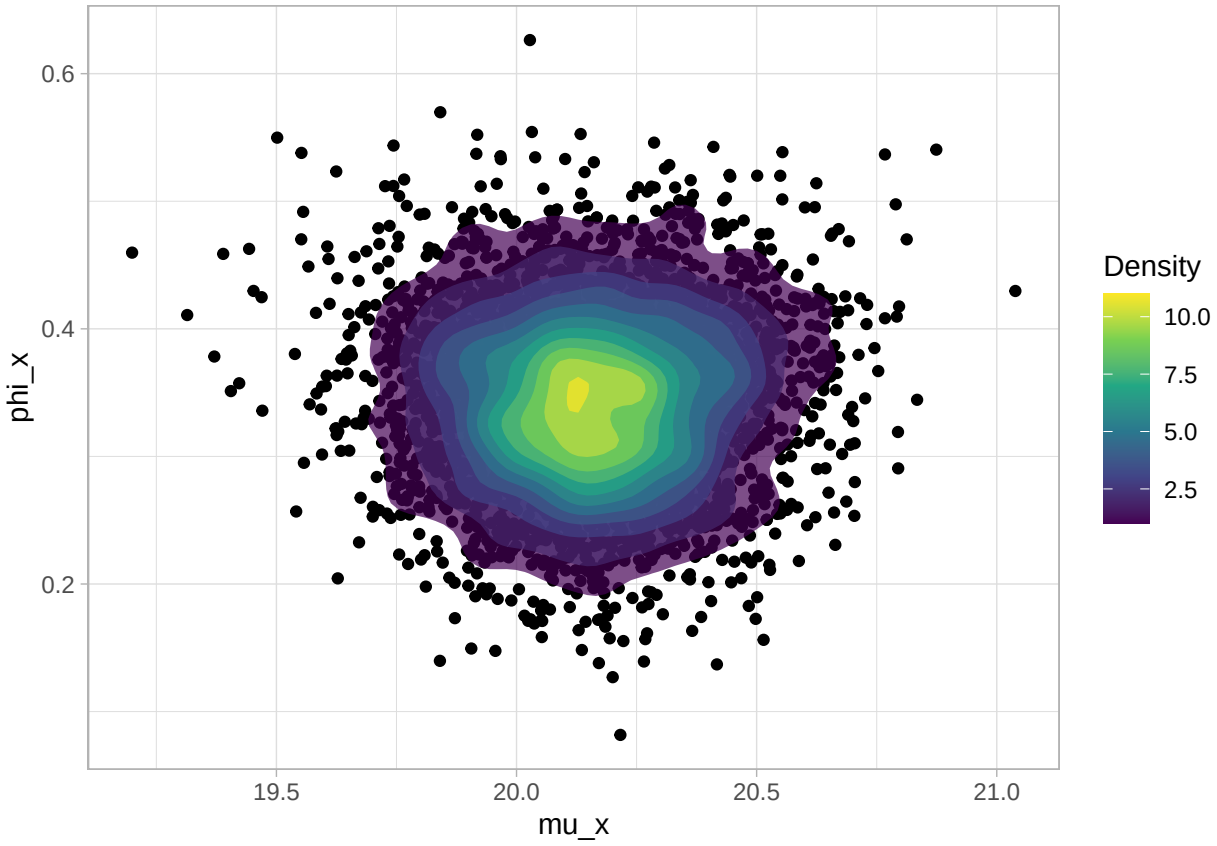
```
## Warning in readLines(file, warn = TRUE): incomplete final line found on 'C:
## \Users\alejo\Downloads\stan_model.stan'
```

```
summary_fit <- summary(fit)
```

In the table below are specified several parameters inferred for the synthetic data X ($\phi_x = 0.3$): namely mean, 95% confidence interval for such parameters, effective sample size and split Rhats ("the potential scale reduction derived from all chains after splitting each chain in half and treating the halves as chains", according to library docs).

	mean	2.5%	97.5%	n_eff	Rhat
mu	20.1556849	19.7431500	20.5822061	3215.202	1.000146
phi	0.3446566	0.2162694	0.4788666	3921.134	0.999939
sigma	1.9779265	1.7990539	2.1762425	3857.678	1.000991

By extracting the posterior draws of μ, ϕ, we will plot the joint posterior distribution. We can see by the plot outputted that the shape could be nearly similar to an ellipsoid, possibly approximated by a multivariate normal. As it is not spread rather unevenly along any axis we could not state its covariance matrix is more prominent in that respective direction. The rounded shape of it makes the mean more predictable through this stationary MCMC approximation of the stochastic process for values around the yellow zone for both parameters where most of the mass is contained.



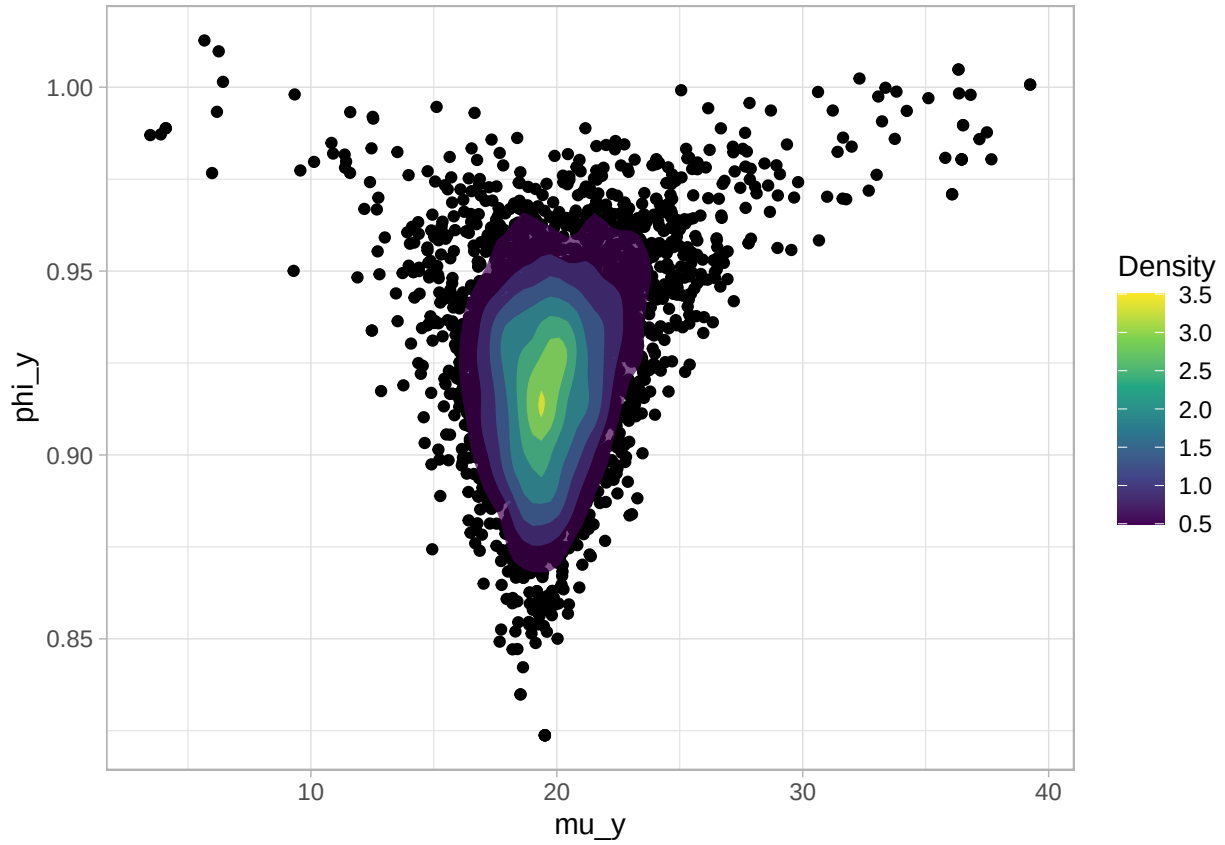
Likewise, the same process of fitting and plot of the joint posterior will be done with the *synthetic data* y. In this case, the posterior joint distribution would not be approximated with a multivariate gaussian since the shape is much more irregular (primarily for the ϕ axis). As the regulation ϕ parameter puts more weight in the previous sample, it makes much harder to make inference in both parameters since their axis hold more variation as the values are more span and have more presence of outliers than the previous dataset. The intervals that keep more density of points are the specified in the CI (in table) It is also remarkable that when $\phi = 0.9$, μ gets more variability and viceversa when $\mu = 20$.

```
## Warning in readLines(file, warn = TRUE): incomplete final line found on 'C:
## \Users\alejo\Downloads\stan_model.stan'
```

```
## Warning: There were 2 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
```

	mean	2.5%	97.5%	n_eff	Rhat
mu	19.8752707	15.4455746	25.7148315	722.4996	1.003779
phi	0.9232931	0.8733746	0.9776983	1331.7752	1.000746
sigma	1.9621368	1.7751348	2.1658883	2424.5347	1.001597



Appendix: All code for this report

```
rmu_norm <- function(mu_0, tau_0_sqrt, sigma_sqrt, x_bar, n){

  inv_tau_n_sqrt <- (n/sigma_sqrt) + (1/tau_0_sqrt)
  w <- (n/sigma_sqrt) / ((n/sigma_sqrt) + (1/tau_0_sqrt))
  mu_n <- (w*x_bar) + (1-w)*mu_0

  return(rnorm(1, mu_n, sqrt(1/inv_tau_n_sqrt)))
}


r_inv_chi_draw <- function(dof,var_cust){
  X <- rchisq(1, dof)
  return((dof*var_cust)/X)
}
```

```
r_full_cond_var <- function(data, vn, mu,v0, var0){
  n <- length(data)
  sum_data <- sum((data-mu)^2)
  var_c <- ((v0*var0)+sum_data)/(n+v0)
  return(r_inv_chi_draw(vn,var_c))
}

rainfall_data <- log(unlist(read.table("rainfall.dat", header=FALSE)))
x_bar <- mean(rainfall_data)
n <- length(rainfall_data)

mu_0 <- 0
sigma_0 <- 1
tau_0 <- 1
v_0 <- 1

n_it <- 10000

gibbs <- function(n_it, mu_0, sigma_0, tau_0, x_bar, n, v_0, data){
  mu_vec <- rep(NA, n_it+1)
  mu_vec[1] <- mu_0
  sigma_vec <- rep(NA, n_it+1)
  norm_vec <- rep(NA, n_it+1)
  sigma_vec[1] <- sigma_0
  norm_vec[1] <- rnorm(1,0,1)

  for(i in 2:(n_it+1)){

    mu_vec[i] <- rmu_norm(mu_0 = mu_vec[i-1], tau_0_sqrt = tau_0, sigma_sqrt = sigma_vec[i-1], x_bar = x
    sigma_vec[i] <- r_full_cond_var(data = data, vn = (v_0+n), mu = mu_vec[i], v0 = v_0, var0 = sigma_0
    norm_vec[i] <- rnorm(1,mu_vec[i],sqrt(sigma_vec[i]))
  }

  return(list(mu_n=mu_vec, sigma_n=sigma_vec, r_data = norm_vec))
}

res <- gibbs(n_it, mu_0, sigma_0, tau_0, x_bar, n, v_0, data=rainfall_data)
mu_res <- res$mu_n
sigma_res <- res$sigma_n
par(mfrow=c(1,2))
plot(mu_res[50:length(mu_res)], type="l", main=bquote(mu[n]))
abline(h=c(mean(mu_res),mean(rainfall_data)), col=c("blue", "red"), lty=c(1,2), lwd=c(1, 3))
plot(sigma_res[50:length(mu_res)], type="l", main=bquote((sigma^2)[n]))
abline(h=c(mean(sigma_res),sd(rainfall_data)^2), col=c("blue", "red"), lty=c(1,2), lwd=c(1, 3))


a_Gibbs <- acf(sigma_res, plot = FALSE)

IF_Gibbs <- 1+2*sum(a_Gibbs$acf[-1])
cat("ineficiency sigma:", IF_Gibbs)


a_Gibbs <- acf(mu_res, plot = FALSE)
```

```r
IF_Gibbs <- 1+2*sum(a_Gibbs$acf[-1])
cat("\n ineficiency mu:", IF_Gibbs)
rainfall_dens <- density(exp(rainfall_data))
simulated_dens <- density(exp(res$r_data))

plot(x = rainfall_dens$x, y = rainfall_dens$y, type="l")
lines(x = simulated_dens$x, y = simulated_dens$y, col = "blue")
ebay_data <- read.table("eBayNumberOfBidderData.dat", header=TRUE)
ebay_data_wo_const <- ebay_data[,-2]

glm(nBids~., family = poisson, data = ebay_data_wo_const)
# Setting up the prior
library(mvtnorm)


Nfeat <- ncol(ebay_data)-1
mu <- as.matrix(rep(0,Nfeat)) # Prior mean vector
init_val <- matrix(0,Nfeat,1)
X <- as.matrix(ebay_data[,-1])
y <- as.matrix(ebay_data[1])
cov_prior <- 100 * solve(t(X) %*% X)

LogPostpoisson <- function(betas,y,X,mu,Sigma){
  lambda <- exp(X%*%betas)
  logLik <- sum(log(exp(-lambda) * (1/factorial(y)) * (lambda^y)))
  logPrior <- dmvnorm(betas, mu, Sigma, log=TRUE)

  return(logLik + logPrior)
}

# Optimization for the beta estimates Hessian

OptimRes <- optim(init_val,LogPostpoisson, gr =NULL, y = y, X = X, mu = mu, Sigma = cov_prior, method=c

beta_estimates <- OptimRes$par
hessian <-  OptimRes$hessian
hessian_inv <- solve(-hessian)
knitr::kable(t(signif(beta_estimates,3)), col.names = colnames(ebay_data)[-1])
metropolis <- function(dens, c, n){
  var_c <- c * hessian_inv
  x0 <- rep(0,9)
  x <- matrix(0,n,9)
  x[1,] <- x0
  alphas <- vector()

  for (t in 2:n) {
    unif <- runif(1)
    rnd <- rmvnorm(1, x0, var_c)
    alpha <- min(1, exp(dens(rnd) - dens(x0))) ## its simetric
    if(unif<alpha){
      x0 <- rnd
    }
    x[t,] <- x0
```

```
    #alphas <- append(alphas, alpha)
  }
  return(x)
}
posterior <- function(x){
 return(dmvnorm(x, beta_estimates ,hessian_inv, log = TRUE))
}
metropolis_samples <-  metropolis(posterior,1,2000)
#cols <- c("black", "red", "blue", "forestgreen", "purple", "yellow", "darkgreen", "dodgerblue1", "gold
par(mfrow=c(3,3))
for (i in 1:9) {
  plot(metropolis_samples[,i], type = "l", main = colnames(ebay_data)[-1][i], xlab = "sample #", ylab="
  abline(h = beta_estimates[i], col = "red")
}
new_x <- c(1,1,1,1,0,1,0,1,0.7)
lambdas <- exp(metropolis_samples[500:nrow(metropolis_samples) ,] %*% new_x) # getting rid of burn in p
gen_data <- sapply(lambdas, rpois, n = 1) # sampling from poisson
hist(gen_data, main = "histogram of nBiders", xlab = "nbiders")
prob_no_bidds <- sum(gen_data == 0)/length(gen_data)
AR <- function(phi){
  mu <- 20
  var_c <- 4
  n_T <- 200
  x <- 0
  res <- vector()
    for (i in 2:n_T) {
     x <- mu + phi*(x-mu)+rnorm(1,0,sqrt(var_c))
     res <- append(res,x)
    }
  return(res)
}

par(mfrow=c(3,1))
plot(AR(-1), type = "l")
plot(AR(0), type = "l")
plot(AR(1), type = "l")

#data {
#  int<lower=0> N;
#  vector[N] y;
#}
#parameters {
#  real mu;
#  real phi;
#  real<lower=0> sigma;
#}
#model {
#  for (n in 2:N)
#    y[n] ~ normal(mu + phi*(y[n-1]-mu), sigma);
#}

synthetic_x <- list(N = 199,
                    y = AR(0.3))
```

```r
synthetic_y <- list(N = 199,
                    y = AR(0.9))

library(rstan)
fit <- stan(file = 'stan_model.stan', data = synthetic_x, verbose = FALSE)
summary_fit <- summary(fit)

knitr::kable(summary_fit$summary[-4,c(-2,-3,-5,-6,-7)])
postdata <- extract(fit)

df_plot <- data.frame(mu = postdata$mu, phi = postdata$phi)

ggplot(df_plot, aes(x = mu, y = phi)) +
geom_point() +
stat_density_2d(aes(fill = stat(level)), geom = "polygon", alpha = 0.7) +
labs(y = "phi_x", x = "mu_x") +
scale_fill_viridis_c(name = "Density") +
theme_light()

fit1 <- stan(file = 'stan_model.stan', data = synthetic_y, verbose = FALSE)
summary_fit1 <- summary(fit1)
knitr::kable(summary_fit1$summary[-4,c(-2,-3,-5,-6,-7)])
postdata1 <- extract(fit1)

df_plot <- data.frame(mu = postdata1$mu, phi = postdata1$phi)

ggplot(df_plot, aes(x = mu, y = phi)) +
geom_point() +
stat_density_2d(aes(fill = stat(level)), geom = "polygon", alpha = 0.7) +
labs(y = "phi_y", x = "mu_y") +
scale_fill_viridis_c(name = "Density") +
theme_light()
```