Bertil Wegmann
Dept of Computer and Information Science
Linköping University

Bayesian Learning

Computer Lab 1

You are recommended to use R for solving the labs.
You work and submit your labs in pairs, but both of you should contribute equally and understand all parts of your solutions.
`It is not allowed to share exact solutions` with other student pairs.
The submitted lab reports will be verified through OURIGINAL and indications of plagiarism will be investigated by the Disciplinary Board.
Submit your solutions via LISAM, no later than April 17 at 23:59.

Please note the following about the format of the submitted lab report:
1. The lab report should include all solutions and plots to the stated problems with necessary comments.
2. Submit the lab report with your code attached to the solution of each sub-problem (1a), 1b),...) in **one** PDF document.
3. Submit a separate file containing all code.

---

1. *Daniel Bernoulli*
   Let $y_1, ..., y_n|\theta \sim \text{Bern}(\theta)$, and assume that you have obtained a sample with $s = 22$ successes in $n = 70$ trials. Assume a $\text{Beta}(\alpha_0, \beta_0)$ prior for θ and let $\alpha_0 = \beta_0 = 8$.

   (a) Draw 10000 random values (`nDraws = 10000`) from the posterior $\theta|y \sim \text{Beta}(\alpha_0 + s, \beta_0 + f)$, where $y = (y_1, \ldots, y_n)$, and verify graphically that the posterior mean $E[\theta|y]$ and standard deviation $SD[\theta|y]$ converges to the true values as the number of random draws grows large. [Hint: `use rbeta()` to draw random values and make graphs of the sample means and standard deviations of θ as a function of the accumulating number of drawn values].

   (b) Draw 10000 random values from the posterior to compute the posterior probability
   $\Pr(\theta > 0.3|y)$ and compare with the exact value from the Beta posterior. [Hint: `use pbeta()`].

   (c) Draw 10000 random values from the posterior of the odds $\phi = \frac{\theta}{1-\theta}$ by using the previous random draws from the Beta posterior for θ and plot the posterior distribution of ϕ. [Hint: `hist()` and `density()` can be utilized].

2. *Log-normal distribution and the Gini coefficient.*
   Assume that you have asked 8 randomly selected persons about their monthly income (in thousands Swedish Krona) and obtained the following eight observations: 33, 24, 48, 32, 55, 74, 23, and 17. A common model for non-negative continuous

variables is the log-normal distribution. The log-normal distribution $\log\mathcal{N}(\mu, \sigma^2)$ has density function

$$p(y|\mu, \sigma^2) = \frac{1}{y \cdot \sqrt{2\pi\sigma^2}} \exp\left[-\frac{1}{2\sigma^2}(\log y - \mu)^2\right],$$

where $y > 0$, $-\infty < \mu < \infty$ and $\sigma^2 > 0$. The log-normal distribution is related to the normal distribution as follows: if $y \sim \log\mathcal{N}(\mu, \sigma^2)$ then $\log y \sim \mathcal{N}(\mu, \sigma^2)$. Let $y_1, ..., y_n|\mu, \sigma^2 \overset{iid}{\sim} \log\mathcal{N}(\mu, \sigma^2)$, where $\mu = 3.6$ is assumed to be known but σ^2 is unknown with non-informative prior $p(\sigma^2) \propto 1/\sigma^2$. The posterior for σ^2 is the $Inv - \chi^2(n, \tau^2)$ distribution, where

$$\tau^2 = \frac{\sum_{i=1}^{n}(\log y_i - \mu)^2}{n}.$$

(a) Draw 10000 random values from the posterior of σ^2 by assuming $\mu = 3.6$ and plot the posterior distribution.

(b) The most common measure of income inequality is the Gini coefficient, G, where $0 \leq G \leq 1$. $G = 0$ means a completely equal income distribution, whereas $G = 1$ means complete income inequality (see e.g. Wikipedia for more information about the Gini coefficient). It can be shown that $G = 2\Phi\left(\sigma/\sqrt{2}\right) - 1$ when incomes follow a $\log\mathcal{N}(\mu, \sigma^2)$ distribution. $\Phi(z)$ is the cumulative distribution function (CDF) for the standard normal distribution with mean zero and unit variance. Use the posterior draws in a) to compute the posterior distribution of the Gini coefficient G for the current data set.

(c) Use the posterior draws from b) to compute a 95% equal tail credible interval for G. A 95% equal tail credible interval (a, b) cuts off 2.5% percent of the posterior probability mass to the left of a, and 2.5% to the right of b.

(d) Use the posterior draws from b) to compute a 95% Highest Posterior Density Interval (HPDI) for G. Compare the two intervals in (c) and (d). [Hint: do a kernel density estimate of the posterior of G using the `density` function in R with default settings, and use that kernel density estimate to compute the HPDI. Note that you need to order/sort the estimated density values to obtain the HPDI.].

3. *Bayesian inference for the concentration parameter in the von Mises distribution.* This exercise is concerned with directional data. The point is to show you that the posterior distribution for somewhat weird models can be obtained by plotting it over a grid of values. The data points are observed wind directions at a given location on ten different days. The data are recorded in degrees:

$$(20, 314, 285, 40, 308, 314, 299, 296, 303, 326),$$

where North is located at zero degrees (see Figure 1 on the next page, where the angles are measured clockwise). To fit with Wikipedia's description of probability distributions for circular data we convert the data into radians $-\pi \leq y \leq \pi$. The 10 observations in radians are

$$(-2.79, 2.33, 1.83, -2.44, 2.23, 2.33, 2.07, 2.02, 2.14, 2.54).$$

Assume that these data points conditional on (μ, κ) are independent observations from the following von Mises distribution:

$$p(y|\mu,\kappa) = \frac{\exp\left[\kappa \cdot \cos(y-\mu)\right]}{2\pi I_0(\kappa)},\ -\pi \leq y \leq \pi,$$

where $I_0(\kappa)$ is the modified Bessel function of the first kind of order zero [see `?besselI` in R]. The parameter μ $(-\pi \leq \mu \leq \pi)$ is the mean direction and $\kappa > 0$ is called the concentration parameter. Large κ gives a small variance around μ, and vice versa. Assume that μ is known to be 2.4. Let $\kappa \sim$ Exponential$(\lambda = 0.5)$ a priori, where λ is the rate parameter of the exponential distribution (so that the mean is $1/\lambda$).

(a) Derive the expression for what the posterior $p(\kappa|y,\mu)$ is proportional to. Hence, derive the function $f(\kappa)$ such that $p(\kappa|y,\mu) \propto f(\kappa)$. Then, plot the posterior distribution of κ for the wind direction data over a fine grid of κ values. [Hint: you need to normalize the posterior distribution of κ so that it integrates to one.]

(b) Find the (approximate) posterior mode of κ from the information in a).

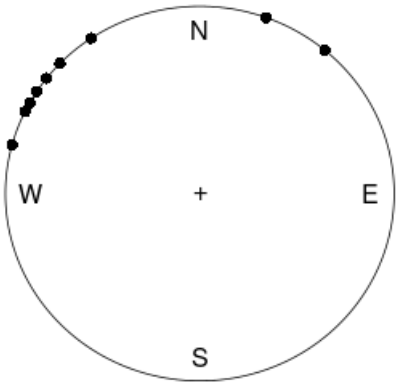


Figure 1: The wind direction data. Angles are measured clock-wise starting from North.

GOOD LUCK!

Bayesian Learning Lab 1

Mohamed Ali - Mohal954

2023-05-06

Question 1 Daniel Bernoulli

let $y_1, ..., y_n|\theta \sim Bern(\theta)$ and assume that you have obtained a sample with s = 22 successes in n = 70 trials. Assume a $Beta(\alpha_0; \beta_0)$ prior for θ and let $\alpha_0 = \beta_0 = 8$.

First we calculate the mean and the standar deviation from the below equation to compare the with sample means and standar deviations of θ as function of the accumulating number of drawn values.

$$E(\theta|y) = a/a+b$$
$$E(\theta|y) = ab/(a+b)^2(a+b+1)$$

A

Draw 10000 random values (nDraws = 10000) from the posterior $\theta|y \sim \beta(\alpha_0 + s, \beta_0 + f)$, where y = (y1,..,yn); and verify graphically that the posterior mean $E(\theta|y)$ and standard deviation $SD(\theta|y)$ converges to the true values as the number of random draws grows large.
[Hint: use rbeta() to draw random values and make graphs of the sample means and standard deviations of θ as a function of the accumulating number of drawn values].

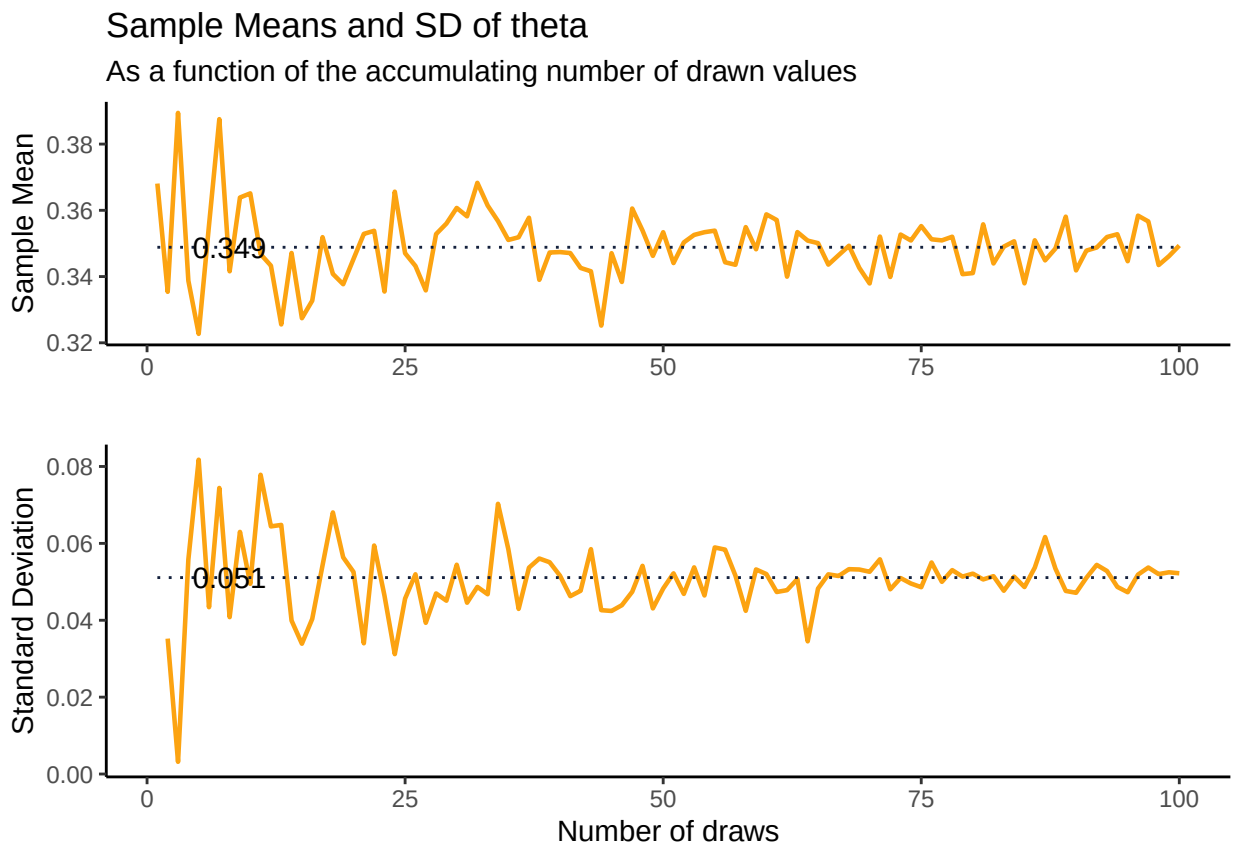
The figure below shows how the values of the Mean and Sd converges to the true values as the number of draws grows large

```
ber_fun<-function(n_d,n,s,a,b) {
  #Initial Value of the function givan from the question
  t_n = n
  s   = s
  f   = n-s
  a   = a
  b   = b
  #Beta(alpha+s,beta+s)
  a_new = a+s
  b_new = b+f
  Mean_true= a_new/(a_new+b_new)
  #we take the sqrt to get the Sd insted of the Var
  Sd_true=  sqrt((a_new*b_new)/(((a_new+b_new)^2) * (a_new+b_new+1)))
  # Beta(alpha+s,beta+s)
  mean_theta = c()
  sd_theta = c()
```

```
  n_draws = 1:n_d
  #For loop to fill the values of mean and sd based on the draws
  for (i in 1:n_d){
    mean_theta[i]=mean(rbeta(i,a_new,b_new))
    sd_theta[i]=sd(rbeta(i,a_new,b_new))}
  #Binding everything togther
  df<-cbind.data.frame(n_draws,mean_theta,sd_theta)
  #Plot of the mean
  mean<-ggplot(df,aes(x=n_draws))+geom_line(aes(y=mean_theta)
                                            , color='#FCA311', size=.8)+
              geom_line(aes(y=Mean_true), color='#14213D',linetype=3)+
              annotate(geom = "text", x = 8, y = Mean_true,
              label = paste0(format(round(Mean_true, 3), nsmall = 3)))+
              labs(title = 'Sample Means and SD of theta ',
         subtitle = 'As a function of the accumulating number of drawn values',
                           x= ' ', y='Sample Mean')+ theme_classic()
  #Plot of the Sd
  sd<-ggplot(df,aes(x=n_draws))+geom_line(aes(y=sd_theta),
                                          color='#FCA311', size=.8)+
              geom_line(aes(y=Sd_true), color='#14213D',linetype=3)+
              annotate(geom = "text", x = 8, y = Sd_true,
              label = paste0(format(round(Sd_true, 3), nsmall = 3)))+
              labs(x= 'Number of draws', y='Standard Deviation')+
              theme_classic()
  #grid.arrange(mean,sd)
  return(grid.arrange(mean,sd))
}


ber_fun(100,70,22,8,8)
```

```
## Warning: Using ‘size‘ aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use ‘linewidth‘ instead.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```

```
## Warning: Removed 1 row containing missing values (‘geom_line()‘).
```



B

Draw 10000 random values from the posterior to compute the posterior probability $Pr(\theta > 0.3|y)$ and compare with the exact value from the Beta posterior. [Hint: use pbeta()].

First we find the value from the beta posterior using the function *pbeta* we get the value of theta which can be used to compute the probability that a random variable from a beta distribution is less than or equal to a given value, or greater than a given value, depending on the value of the lower.tail argument. in our case we use lower.tail as False because we need the values grater than.

```
prob<-function(n_d,n,s,a,b,prob){
        t_n = n
        s   = s
        f   = n-s
        a   = a
        b   = b
        #Beta(alpha+s,beta+s)
        a_new = a+s
        b_new = b+f
        p<-pbeta(prob,a_new,b_new,lower.tail = F)
        post<-mean(rbeta(n_d,a_new,b_new)>prob)
        return(list(paste('random values from the posterior with the given condition',post)
        ,paste('The exact value from the Beat posterior', round(p,2))))}
prob(100,70,22,8,8,.3)
```

```
## [[1]]
## [1] "random values from the posterior with the given condition 0.85"
##
## [[2]]
## [1] "The exact value from the Beat posterior 0.83"
```

```
#Example Code from lec

# ##############################################################################################
# # Generates samples from the joint posterior distribution of the parameters
# # in the x1,....xn iid Normal(theta,sigma^2) model with
# # prior p(theta,sigma^2) propto 1/sigma^2
# ##############################################################################################
# NormalNonInfoPrior <- function(NDraws,Data){
#   Datamean <- mean(Data)
#   s2 <- var(Data)
#   n <- length(Data)
#   PostDraws <- matrix(0,NDraws,2)
#   PostDraws[,2] <- ((n-1)*s2)/rchisq(NDraws,n-1)
#   PostDraws[,1] <- rnorm(NDraws,mean=Datamean,sd=sqrt(PostDraws[,2]/n))
#
#   return(PostDraws)
# }
#
# Nobs <- 10000
# Ndraws <- 10000
# Data <- rnorm(Nobs,5,10)  # Sampling Nobs observations from the N(5,10) density##
# PostDraws <- NormalNonInfoPrior(Ndraws,Data) # Generating draws from the joint posterior of mu and si
# hist(PostDraws[,1])           # Plotting the histogram of mu-draws
# hist(PostDraws[,2])       # Plotting the histogram of sigma^2-draws
#
# # Examples of probability calculations
# mean(PostDraws[,1]>4.9 & PostDraws[,1]<5.1) # Approximate posterior probability of 4.9 < mu < 5.1
# mean(PostDraws[,2]>99 & PostDraws[,2]<101)  # Approximate posterior probability of 99 < sigma^2 < 101
#
#
# ##############################################################################################
# # Generate samples from the joint posterior distribution of theta=(theta_1,...,theta_K)
# # for the multinomial model with K categories and a Dirichlet prior for theta.
# ##############################################################################################
# Dirichlet <- function(NDraws,y,alpha){
#   K <- length(alpha)
#   xDraws <- matrix(0,NDraws,K)
#   thetaDraws <- matrix(0,NDraws,K) # Matrix where the posterior draws of theta are stored
#   for (j in 1:K){
#     xDraws[,j] <- rgamma(NDraws,shape=alpha[j]+y[j],rate=1)
#   }
#   for (ii in 1:NDraws){
#     thetaDraws[ii,] <- xDraws[ii,]/sum(xDraws[ii,])
#   }
#   return(thetaDraws)
# }
```

```
#
# ############   Setting up data and prior  ##################
# y <- c(180,230,62,41) # Data of counts for each category
# p <- y/sum(y)
# alpha_const <- 1
# alpha <- alpha_const*c(15,15,10,10) # Dirichlet prior hyperparameters
# NDraws <- 10000 # Number of posterior draws
#
# ############   Posterior sampling from Dirichlet  ##################
# thetaDraws <- Dirichlet(NDraws,y,alpha)
#
# K <- length(y)
# ############ Summary statistics from the posterior sample #########
# for (k in 1:K){
#   mean(thetaDraws[,k])
#   sqrt(var(thetaDraws[,k]))
# }
#
# sum(thetaDraws[,2]>thetaDraws[,1])/NDraws # p(theta2>theta1 | y)
# # Posterior probability that Android has largest share, i.e. p(theta_2 > max(theta_1,theta_3,theta_4)
# Index_max <- matrix(0,NDraws,1)
# for (ii in 1:NDraws){
# Index_max[ii,1] <- which.max(thetaDraws[ii,])
# }
# mean(Index_max==2)
#
# # Plot histograms of the posterior draws
# plot.new() # Opens a new graphical window
# par(mfrow = c(2,2)) # Splits the graphical window in four parts (2-by-2 structure)
# hist(thetaDraws[,1],25) # Plots the histogram of theta[,1] in the upper left subgraph
# hist(thetaDraws[,2],25)
# hist(thetaDraws[,3],25)
# hist(thetaDraws[,4],25)

# marginal likelihood for the model
# beta(post_alpha,post_beta)/beta(alpha,beta) # Ratio of beta functions
```

C

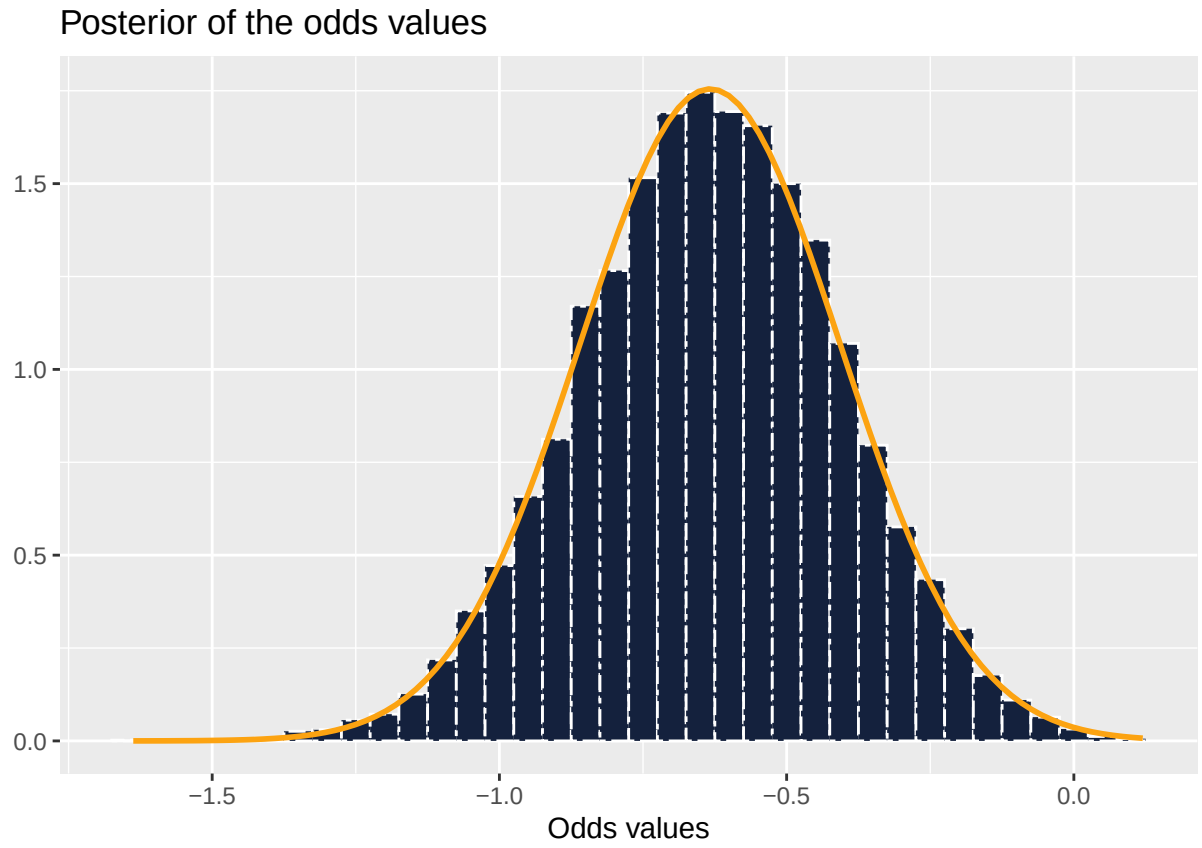
Draw 10000 random values from the posterior of the odds $\phi = \frac{\theta}{1-\theta}$ by using the previous random draws from the Beta posterior for $\theta$and plot the posterior distribution of ϕ.
[Hint: hist() and density() can be utilized].

Now we want to draw 10000 random values from the posterior of the odds

```
grp<-function(n_d,n,s,a,b){
        t_n = n
        s   = s
        f   = n-s
        a   = a
        b   = b
        #Beta(alpha+s,beta+s)
```

```
        a_new = a+s
        b_new = b+f
        res<-rbeta(n_d,a_new,b_new)
        res2<-data.frame(x=log(res/(1-res)))
        plt<-ggplot(res2,aes(x=x))+geom_histogram(aes(y=..density..),
                                    color="white",linetype=6,
                                    fill='#14213D',binwidth = 0.05)+
                                    labs(x='Odds values',y=' ',
                                    title ='Posterior of the odds values')+
                                    stat_function(fun = dnorm,
                                        args = list(mean = mean(res2$x),
                                        sd = sd(res2$x)),
                                        color = "#FCA311", size = 1)
        return(plt)}

grp(10000,70,22,8,8)
```

```
## Warning: The dot-dot notation (‘..density..‘) was deprecated in ggplot2 3.4.0.
## i Please use ‘after_stat(density)‘ instead.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```



Question 2 Log-Normal distribution and Gini Coefficient

Assume that you have asked 8 randomly selected persons about their monthly income (in thousands Swedish Krona) and obtained the following eight observations: 33, 24, 48, 32, 55, 74, 23, and 17. A common model for non-negative continuous variables is the log-normal distribution.
The log-normal distribution $logN(\mu, \sigma^2)$ has density function:

$$p(y|\mu, \sigma^2) = \frac{1}{y.\sqrt{2\pi\sigma^2}} exp[-\frac{1}{2\sigma^2}(logy - \mu)^2]$$

where $y > 0, -inf < \mu < inf$ and σ^2 The log-normal distribution is related to the normal distribution as follows: if $y \sim log\ N(\mu, \sigma^2)$ then $log\ y \sim N(\mu, \sigma^2)$. Let $y_1, ..., y_n|\mu, \sigma^2 \sim logN(\mu, \sigma^2)$, where $\mu = 3.6$ is assumed to be known but σ^2 is unknown with non-informative prior $p(\sigma^2) \propto 1/\sigma^2$ The posterior for σ^2 is the $inv - \chi^2(n, \tau^2)$ distribution, where:

$$\tau^2 = \frac{\sum_{i=1}^{n}(logy_i - \mu^2)^2}{n}$$

A

Draw 10000 random values from the posterior of σ^2 by assuming $\mu = 3{:}6$ and plot the posterior distribution.

First we find the value of tau, the question assume that we have 8 randomly selected persons thus we have n = 8 and we have the value of mu = 3.6 Given, to find the value of tau we use this equation:

$$\tau^2 \ = \ \sum_{1}^{n}(log\ y_i \ - \mu)^2/n$$

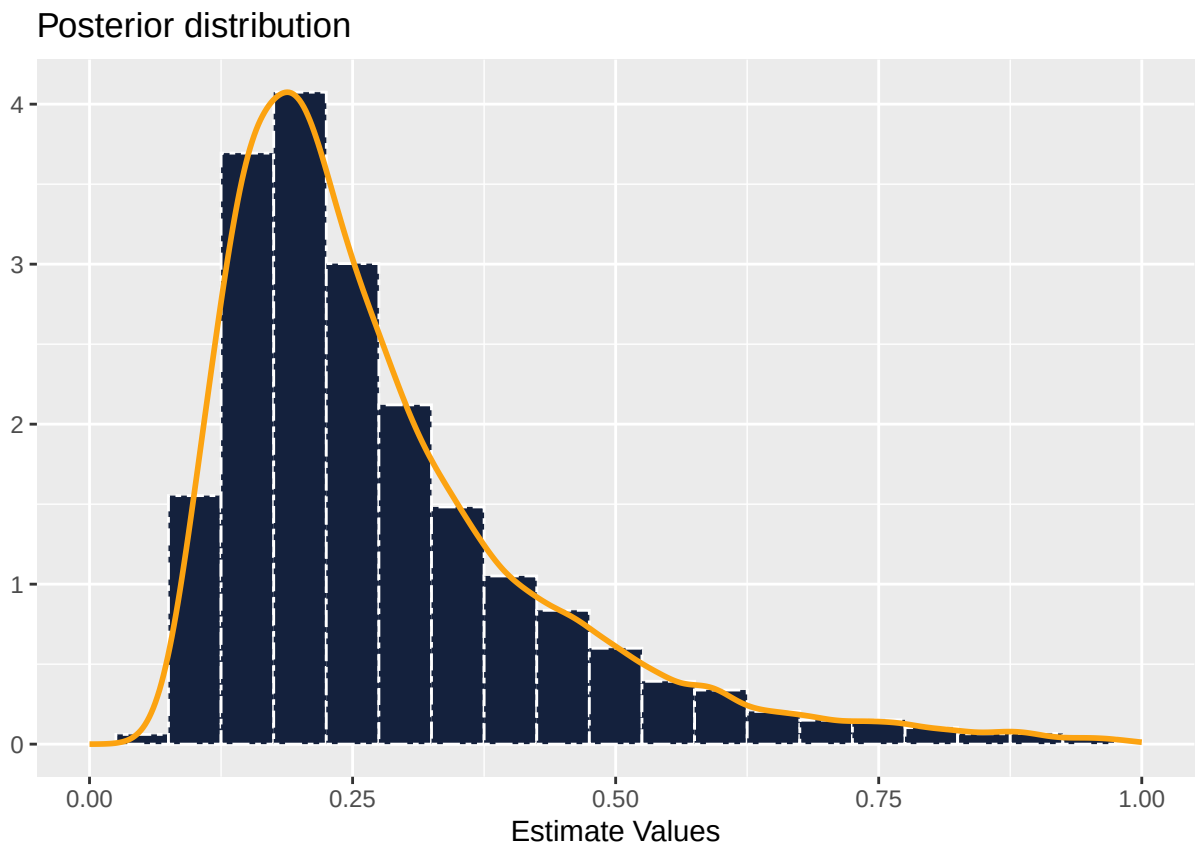
’.
Note A non-informative prior is a prior distribution that is chosen to express little to no prior information about the parameters. Non-informative priors are often chosen to avoid introducing bias or strong assumptions into the model, and to allow the data to have a greater influence on the posterior distribution. Examples of non-informative priors include the uniform distribution, the Jeffreys prior, and the reference prior.
It’s important to note that a non-informative prior is not necessarily a prior with no information at all, but rather one that expresses a minimal amount of information that is consistent with our knowledge and beliefs before observing the data. In practice, the choice of prior distribution often depends on the specific problem and the available prior knowledge.
in the question we have inverse chi distribution is our posterior, the inverse-chi-squared distribution (or inverted-chi-square distribution[1]) is a continuous probability distribution of a positive-valued random variable. It is closely related to the chi-squared distribution. It arises in Bayesian inference, where it can be used as the prior and posterior distribution for an unknown variance of the normal distribution.

the inverse chi distribution is The inverse chi-square distribution with degrees of freedom n and scale parameter s^2 is closely related to the inverse gamma distribution with shape parameter α = n/2 and scale parameter $\beta = 1/(2s^2)$.
In fact, if X is Inv-χ^2(n,s^2) distributed, then Y = (ns^2)/X is Inv-γ(α,β) distributed, and vice versa. then we use the function rinvgamma from r and we change on the pramters shape and scale.

```
y<-c(33,24,48,32,55,74,23,17)
#n is the sample size
n=8
#the number of draws wanted
nDraws=10000
```

```r
mu= 3.6
logy= (log(y)-mu)^2
tau2=sum(logy)/n
# then we use the function rinvgamma from r and we change on the pramters shape and scale.
df<-data.frame(x=rinvgamma(nDraws, shape=n/2, scale=n*tau2/2))
plt<-ggplot(df,aes(x=x))+geom_histogram(aes(y=..density..),color="white",
                                        linetype=6,
                                  fill='#14213D',binwidth = 0.05)+
                                        labs(x='Estimate Values',y=' ',
                                        title ='Posterior distribution')+
                                        xlim(0,1)+
  geom_density(color = "#FCA311", size = 1)
plt
```

```
## Warning: Removed 125 rows containing non-finite values (‘stat_bin()‘).

## Warning: Removed 125 rows containing non-finite values (‘stat_density()‘).

## Warning: Removed 2 rows containing missing values (‘geom_bar()‘).
```



B

The most common measure of income inequality is the Gini coeffcient, G, where 0 <= G <= 1. G = 0 means a completely equal income distribution, whereas G = 1 means complete income inequality (see

e.g. Wikipedia for more information about the Gini coeffcient).
It can be shown that $G = 2\phi(\sigma/\sqrt{2}) - 1$ when incomes follow a *log* $N(\mu, \sigma^2)$ distribution. $\phi(z)$ is the cumulative distribution function (CDF) for the standard normal distribution with mean zero and unit variance. Use the posterior draws in a) to compute the posterior distribution of the Gini coeffcient G for the current data set.

We define The Gini coefficient as a measure of inequality in a distribution, typically used to measure income inequality. It ranges from 0 (perfect equality, where everyone has the same income) to 1 (perfect inequality, where one person has all the income).
A Gini coefficient of 0.5, for example, indicates that 50% of the population has 50% of the total income, while the other 50% of the population has the remaining 50% of the income.

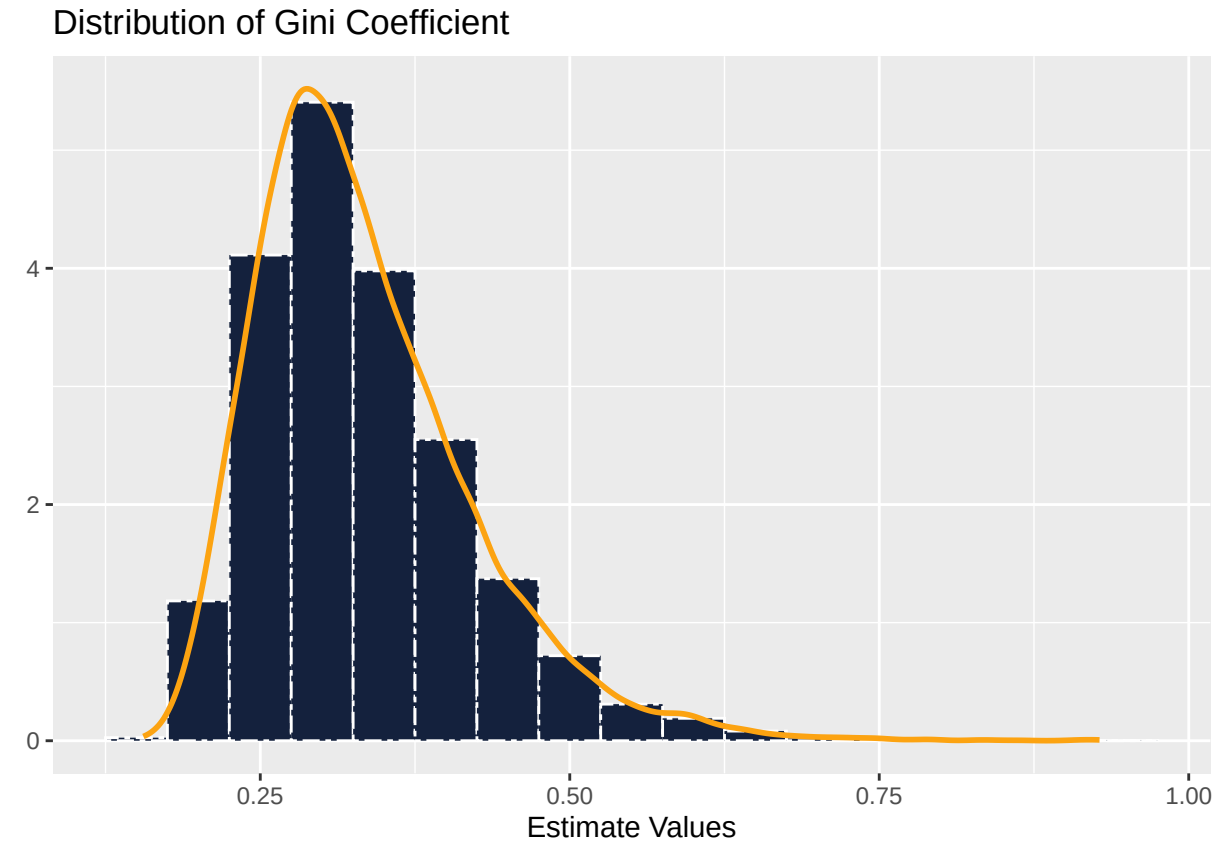
Steps:

1- We find the value of $\phi(\sigma/\sqrt{2})$ by using the values of σ^2 from the estimated values in A using the formula $\sigma/\sqrt{2}$.
2- We use the function *pnorm* to find the values of ϕ where:
- q: the quantile(s) at which to evaluate the CDF.
mean: the mean of the normal distribution (default value is 0).
- sd: the standard deviation of the normal distribution (default value is 1).
- lower.tail: a logical value indicating whether to compute the lower tail probability (TRUE, default) or the upper tail probability (FALSE).
- log.p: a logical value indicating whether to return the natural logarithm of the probability density (TRUE) or the probability density (FALSE, default).
Note that: Quantiles are points in a probability distribution that divide the distribution into intervals of equal probability.In our case we use the probabilities from the function $x = rinvgamma(nDraws, shape = n/2, scale = n * tau2/2)$.

3- We calculate the value of G by pluggin 1 and 2.

```
y<-c(33,24,48,32,55,74,23,17)
#n is the sample size
n=length(y)
#the number of draws wanted
nDraws=10000
mu= 3.6
logy= (log(y)-mu)^2
tau2=sum(logy)/n
# then we use the function rinvgamma from r and we change on the pramters shape and scale.
df<-data.frame(x=rinvgamma(nDraws, shape=n/2, scale=n*tau2/2))
# Calculating the value of Phi_arg which is simply the squar root of the estimated sigma
df$phi_arg<-df$x/sqrt(2)
# Calculating the value of Gini index by appling the formula G=2*phi(sigma/sqrt(2))-1

# Xgrid
# x1_grid <- seq(min(X[,2]),max(X[,2]),0.1)
# Mu_draws <- matrix(0,length(x1_grid),2)
# for (ii in 1:length(x1_grid)){
# CurrMu <- BostonRes$betaSample %*% c(1,x1_grid[ii],XNewHouse[-1:-2])
# Mu_draws[ii,] <- quantile(CurrMu,probs=c(0.025,0.975))
# }
# plot(x1_grid,Mu_draws[,1],"n",main="95 % posterior probability intervals as a function of crim",
```

```r
# xlab="crim", ylab="",ylim=c(10,30))
# lines(x1_grid,Mu_draws[,1],col="blue")
# lines(x1_grid,Mu_draws[,2],col="blue")

df$G<-2*pnorm(sqrt(df$phi_arg),0,1)-1
#Ploting the Gini distribution
ggplot(df,aes(x=G))+geom_histogram(aes(y=..density..),color="white",
                                   linetype=6,
                                   fill='#14213D',binwidth = 0.05)+
  labs(x='Estimate Values',y=' ',
       title ='Distribution of Gini Coefficient')+
  geom_density(color = "#FCA311", size = 1)
```



C

Use the posterior draws from b) to compute a 95% equal tail credible interval for G. A 95% equal tail credible interval (a; b) cuts off 2:5% percent of the posterior probability mass to the left of a, and 2:5% to the right of b.

To find the 95% CI we use the function *qnorm*()in R with mean and Sd drived from the Gini distribution we found in the previous data. here we indicates that 0,025 is the 2,5% cut off point for both uppbe and lower limits.

```r
sample_mean= mean(df$G)
sample_sd= sd(df$G)
# Compute the 95% confidence interval
lower_ci <- qnorm(0.025, mean = sample_mean, sd = sample_sd)
upper_ci <- qnorm(1-0.025, mean = sample_mean, sd = sample_sd)

#In case of utility fun
# nIter <- 1e4
# Mean_log_mu <- mean(log(rnorm(nIter,mean = 92,sd = 2)))
# ExpectedUtility <- function(c, Mean_log_mu){
#   EU <- 60 + sqrt(c)*Mean_log_mu - c
#   return(EU)
# }
#
# cGrid <- seq(0,20,by = 0.01)
# EU <- rep(NA,length(cGrid),1)
# count <- 0
# for (c in cGrid){
#   count <- count + 1
#   EU[count] = ExpectedUtility(c, Mean_log_mu)
# }
# plot(cGrid, EU, type = "l")
# cOpt = cGrid[which.max(EU)] # This is the optimal c
# points(cOpt,ExpectedUtility(c=cOpt, Mean_log_mu), col = "red",pch=19)
# cOpt
# Print the confidence interval
cat("95% confidence interval: [", round(lower_ci, 2), ",", round(upper_ci, 2), "]", "\n")
```

```
## 95% confidence interval: [ 0.16 , 0.5 ]
```

D

Use the posterior draws from b) to compute a 95% Highest Posterior Density Interval (HPDI) for G. Compare the two intervals in (c) and (d). [Hint: do a kernel density estimate of the posterior of G using the density function in R with default settings, and use that kernel density estimate to compute the HPDI. Note that you need to order/sort the estimated density values to obtain the HPDI.].

We define the Highest Posterior Density (HPD) interval is a type of confidence interval in Bayesian inference that contains the most credible values for a parameter based on the observed data. Specifically, it is the narrowest interval that contains a specified proportion (usually 95% or 99%) of the posterior distribution of the parameter.
To do so First we generate a sample using the posterior mean and Sd then We use the function HDI to find the 95% CI from library(HDInterval).

```r
library(HDInterval)
```

```
## Warning: package 'HDInterval' was built under R version 4.1.3
```

```r
#First we generate a sample using the posterior mean and Sd
post_G <- rnorm(n = 10000, mean = mean(df$G), sd = sd(df$G))
post_G_density <- density(post_G)

#We use the function HDI to find the 95% CI
hdpi=hdi(post_G, conf = 0.95)

cat("95% equal tail interavl for G : [", round(lower_ci, 2), ",",
    round(upper_ci, 2), "]", "\n")
```

```
## 95% equal tail interavl for G : [ 0.16 , 0.5 ]
```

```r
cat("95% Highest Posterioe Density Interval for G: [", round(hdpi[1], 2), ",",
    round(hdpi[2], 2), "]", "\n")
```

```
## 95% Highest Posterioe Density Interval for G: [ 0.17 , 0.5 ]
```

```r
# Simulate and find the pi dist
# nSim <- 1e5
# x_pred <- matrix(0,nSim,1)
# for (jj in 1:nSim){
#   mu_draw <- rnorm(1,mean = 92,sd = 2)
#   x_pred[jj,1] <- rnorm(1,mean = mu_draw,sd = sqrt(50))
# }
# plot(density(x_pred),type="l",main="Posterior distribution of x_{n+1}",
#xlab="x_{n+1}",ylab="")
```

Question 3 Bayesian inference for the concentration parameter in the von Mises distribution

This exercise is concerned with directional data. The point is to show you that the posterior distribution for somewhat weird models can be obtained by plotting it over a grid of values. The data points are observed wind directions at a given location on ten different days.
The data are recorded in degrees: (20, 314, 285, 40, 308, 314, 299, 296, 303, 326) where North is located at zero degrees (see Figure 1 on the next page, where the angles are measured clockwise).
To fit with Wikipedia's description of probability distributions for circular data we convert the data into radians $-\pi <= y <= \pi$ .The 10 observations in radians are (-2.79, 2.33, 1.83, -2.44, 2.23, 2.33, 2.07, 2.02, 2.14, 2.54). Assume that these data points conditional on $(\mu, k]$ are independent observations from the following von Mises distribution:

$$p(y|\mu,k) = \frac{exp[k.cos(y.\mu)]}{2\pi I_0(k)}, \ -\pi <= y <= \pi$$

where $I_0(k)$ is the modified Bessel function of the first kind of order zero [see ?besselI in R]. The parameter $\mu(-\pi <= y <= \pi)$ is the mean direction and k > 0 is called the concentration parameter. Large k gives a small variance around μ, and vice versa.
Assume that μ is known to be 2:4. Let $k \sim Exponential(\lambda = 0.5)$ a priori, where λ is the rate parameter of the exponential distribution (so that the mean is $1/\lambda$).

A

Derive the expression for what the posterior $p(k|y,\mu)$ is proportional to. Hence, derive the function f (k) such that $p(k|y,\mu) \propto f(k)$.
Then, plot the posterior distribution of k for the wind direction data over a fine grid of k values. [Hint: you need to normalize the posterior distribution of k so that it integrates to one.]

B

Find the (approximate) posterior mode of k from the information in a).

First to we drive our joint distribution to find the liklehood function from the model distribution.
We have our model distribution $p(y|\mu,k)$ defined as:

$$P(y|\mu,k) = \prod_{i=1}^{n} \frac{1}{2\pi I_0(k)} * exp(k * Cos(y-\mu))$$
$$= \frac{1}{(2\pi)^n I_0(k)^n} * exp(k * Cos(\sum_{i=1}^{n} y_i - \mu))$$

We have the prior k follows the $exp(\lambda = 0.5)$

$$p(k) = 0.5 * exp(-\lambda * k)$$

now we drive the postrior distribution using bayse formula

$$p(\theta|x) \propto p(x|\theta)\ p(\theta)$$

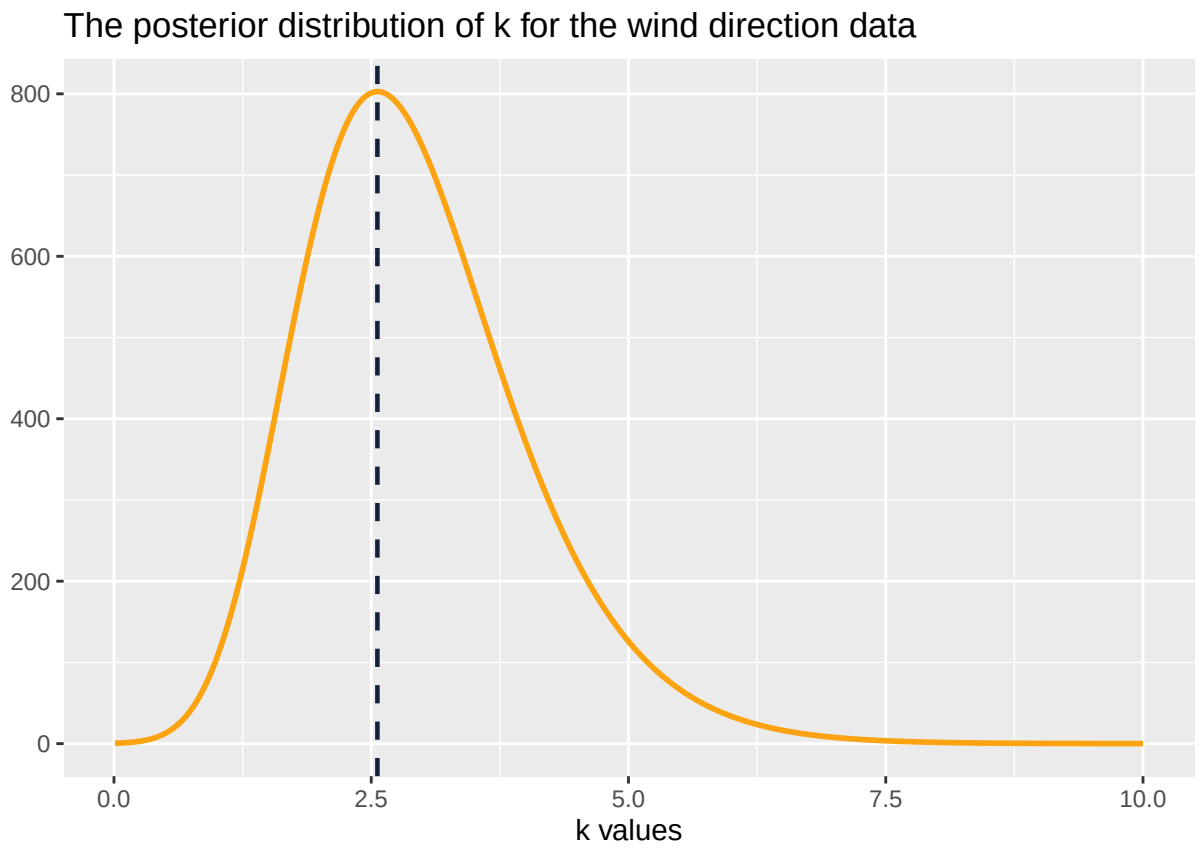
We plug the liklehood function drived above with the prior we get:

$$p(k|\mu,y) \propto \frac{1}{(2\pi)^n I_0(k)^n} * exp(k * Cos(\sum_{i=1}^{n} y_i - \mu))\ .\ 0.5 * exp(-\lambda * k)$$
$$\propto \frac{1}{I_0(k)^n} * exp(k * Cos(\sum_{i=1}^{n} y_i - \mu) - \frac{k}{2})$$

Now we implement the code in R:

```
# The y and mu values given from the question
y<-c(-2.79,2.33,1.83,-2.44,2.23,2.33,2.07,2.02,2.14,2.54)
mu<- 2.51
n=length(y)

#We generate a sequeance of k values by 0.1
k<-seq(0.01,10,by=0.01)

#The drived posterior function can be expressed by
#(0.5/(besselI(k,0))^n)*exp(k*sum(cos(y-mu))-(k*0.5)
df<- data.frame(x=k,y=(0.5/(besselI(k,0))^n)*exp(k*sum(cos(y-mu))-(k*0.5)))

#Plotting function:
plt<-ggplot(df,aes(x=x,y=y))+geom_line(aes(),
                                       color="#FCA311",linetype=1,size=1)+
  labs(x='k values',y=' ',
```

```
      title ='The posterior distribution of k for the wind direction data')+
        geom_vline(xintercept =df$x[which.max(df$y)],
                   # we use df$x[which.max(df$y)] to find the mode
                   linetype = "dashed", color = "#14213D",size=.8)
plt
```




```
#compare the probs
#mean(Betas[,2]>0 & Betas[,3]>0)
#For betas we find the quatiles and interpretated the CI
# Effect_x1x2 <- Betas[,6]
# plot(density(Effect_x1x2),main="Posterior distribution",xlab="beta_5", ylab="")
# #quantile(Effect_x1x2,probs=c(0.025,0.975))
```

References:

1- Bertil Wegmann (2023). Bayesian Learning [Lecture notes]. 732A73, Department of Computer and Information Science,LiU University.

Code Appendix

```r
knitr::opts_chunk$set(echo = TRUE)
library(ggplot2)
library(gridExtra)
library(LaplacesDemon)
ber_fun<-function(n_d,n,s,a,b) {
  #Initial Value of the function givan from the question
  t_n = n
  s   = s
  f   = n-s
  a   = a
  b   = b
  #Beta(alpha+s,beta+s)
  a_new = a+s
  b_new = b+f
  Mean_true= a_new/(a_new+b_new)
  #we take the sqrt to get the Sd insted of the Var
  Sd_true=  sqrt((a_new*b_new)/(((a_new+b_new)^2) * (a_new+b_new+1)))
  # Beta(alpha+s,beta+s)
  mean_theta = c()
  sd_theta = c()
  n_draws = 1:n_d
  #For loop to fill the values of mean and sd based on the draws
  for (i in 1:n_d){
    mean_theta[i]=mean(rbeta(i,a_new,b_new))
    sd_theta[i]=sd(rbeta(i,a_new,b_new))}
  #Binding everything togther
  df<-cbind.data.frame(n_draws,mean_theta,sd_theta)
  #Plot of the mean
  mean<-ggplot(df,aes(x=n_draws))+geom_line(aes(y=mean_theta)
                                            , color='#FCA311', size=.8)+
              geom_line(aes(y=Mean_true), color='#14213D',linetype=3)+
              annotate(geom = "text", x = 8, y = Mean_true,
              label = paste0(format(round(Mean_true, 3), nsmall = 3)))+
              labs(title = 'Sample Means and SD of theta ',
         subtitle = 'As a function of the accumulating number of drawn values',
                           x= ' ', y='Sample Mean')+ theme_classic()
  #Plot of the Sd
  sd<-ggplot(df,aes(x=n_draws))+geom_line(aes(y=sd_theta),
                                          color='#FCA311', size=.8)+
              geom_line(aes(y=Sd_true), color='#14213D',linetype=3)+
              annotate(geom = "text", x = 8, y = Sd_true,
              label = paste0(format(round(Sd_true, 3), nsmall = 3)))+
              labs(x= 'Number of draws', y='Standard Deviation')+
              theme_classic()
  #grid.arrange(mean,sd)
  return(grid.arrange(mean,sd))
}


ber_fun(100,70,22,8,8)
prob<-function(n_d,n,s,a,b,prob){
        t_n = n
        s   = s
```

```
        f   = n-s
        a   = a
        b   = b
        #Beta(alpha+s,beta+s)
        a_new = a+s
        b_new = b+f
        p<-pbeta(prob,a_new,b_new,lower.tail = F)
        post<-mean(rbeta(n_d,a_new,b_new)>prob)
        return(list(paste('random values from the posterior with the given condition',post)
        ,paste('The exact value from the Beat posterior', round(p,2))))}
prob(100,70,22,8,8,.3)

#Example Code from lec

# ##############################################################################################
# # Generates samples from the joint posterior distribution of the parameters
# # in the x1,....xn iid Normal(theta,sigma^2) model with
# # prior p(theta,sigma^2) propto 1/sigma^2
# ##############################################################################################
# NormalNonInfoPrior <- function(NDraws,Data){
#   Datamean <- mean(Data)
#   s2 <- var(Data)
#   n <- length(Data)
#   PostDraws <- matrix(0,NDraws,2)
#   PostDraws[,2] <- ((n-1)*s2)/rchisq(NDraws,n-1)
#   PostDraws[,1] <- rnorm(NDraws,mean=Datamean,sd=sqrt(PostDraws[,2]/n))
#
#   return(PostDraws)
# }
#
# Nobs <- 10000
# Ndraws <- 10000
# Data <- rnorm(Nobs,5,10)  # Sampling Nobs observations from the N(5,10) density##
# PostDraws <- NormalNonInfoPrior(Ndraws,Data) # Generating draws from the joint posterior of mu and si
# hist(PostDraws[,1])           # Plotting the histogram of mu-draws
# hist(PostDraws[,2])       # Plotting the histogram of sigma^2-draws
#
# # Examples of probability calculations
# mean(PostDraws[,1]>4.9 & PostDraws[,1]<5.1) # Approximate posterior probability of 4.9 < mu < 5.1
# mean(PostDraws[,2]>99 & PostDraws[,2]<101)  # Approximate posterior probability of 99 < sigma^2 < 101
#
#
# ##############################################################################################
# # Generate samples from the joint posterior distribution of theta=(theta_1,...,theta_K)
# # for the multinomial model with K categories and a Dirichlet prior for theta.
# ##############################################################################################
# Dirichlet <- function(NDraws,y,alpha){
#   K <- length(alpha)
#   xDraws <- matrix(0,NDraws,K)
#   thetaDraws <- matrix(0,NDraws,K) # Matrix where the posterior draws of theta are stored
#   for (j in 1:K){
#     xDraws[,j] <- rgamma(NDraws,shape=alpha[j]+y[j],rate=1)
#   }
```

```
#   for (ii in 1:NDraws){
#     thetaDraws[ii,] <- xDraws[ii,]/sum(xDraws[ii,])
#   }
#   return(thetaDraws)
# }
#
# ############   Setting up data and prior  ##################
# y <- c(180,230,62,41) # Data of counts for each category
# p <- y/sum(y)
# alpha_const <- 1
# alpha <- alpha_const*c(15,15,10,10) # Dirichlet prior hyperparameters
# NDraws <- 10000 # Number of posterior draws
#
# ############   Posterior sampling from Dirichlet  ##################
# thetaDraws <- Dirichlet(NDraws,y,alpha)
#
# K <- length(y)
# ########### Summary statistics from the posterior sample #########
# for (k in 1:K){
#   mean(thetaDraws[,k])
#   sqrt(var(thetaDraws[,k]))
# }
#
# sum(thetaDraws[,2]>thetaDraws[,1])/NDraws # p(theta2>theta1 | y)
# # Posterior probability that Android has largest share, i.e. p(theta_2 > max(theta_1,theta_3,theta_4)
# Index_max <- matrix(0,NDraws,1)
# for (ii in 1:NDraws){
# Index_max[ii,1] <- which.max(thetaDraws[ii,])
# }
# mean(Index_max==2)
#
# # Plot histograms of the posterior draws
# plot.new() # Opens a new graphical window
# par(mfrow = c(2,2)) # Splits the graphical window in four parts (2-by-2 structure)
# hist(thetaDraws[,1],25) # Plots the histogram of theta[,1] in the upper left subgraph
# hist(thetaDraws[,2],25)
# hist(thetaDraws[,3],25)
# hist(thetaDraws[,4],25)

# marginal likelihood for the model
# beta(post_alpha,post_beta)/beta(alpha,beta) # Ratio of beta functions
grp<-function(n_d,n,s,a,b){
        t_n = n
        s   = s
        f   = n-s
        a   = a
        b   = b
        #Beta(alpha+s,beta+s)
        a_new = a+s
        b_new = b+f
        res<-rbeta(n_d,a_new,b_new)
        res2<-data.frame(x=log(res/(1-res)))
        plt<-ggplot(res2,aes(x=x))+geom_histogram(aes(y=..density..),
```

```
                                    color="white",linetype=6,
                                    fill='#14213D',binwidth = 0.05)+
                                    labs(x='Odds values',y=' ',
                                    title ='Posterior of the odds values')+
                                    stat_function(fun = dnorm,
                                        args = list(mean = mean(res2$x),
                                        sd = sd(res2$x)),
                                        color = "#FCA311", size = 1)
        return(plt)}

grp(10000,70,22,8,8)
y<-c(33,24,48,32,55,74,23,17)
#n is the sample size
n=8
#the number of draws wanted
nDraws=10000
mu= 3.6
logy= (log(y)-mu)^2
tau2=sum(logy)/n
# then we use the function rinvgamma from r and we change on the pramters shape and scale.
df<-data.frame(x=rinvgamma(nDraws, shape=n/2, scale=n*tau2/2))
plt<-ggplot(df,aes(x=x))+geom_histogram(aes(y=..density..),color="white",
                                        linetype=6,
                                    fill='#14213D',binwidth = 0.05)+
                                        labs(x='Estimate Values',y=' ',
                                        title ='Posterior distribution')+
                                        xlim(0,1)+
  geom_density(color = "#FCA311", size = 1)
plt
y<-c(33,24,48,32,55,74,23,17)
#n is the sample size
n=length(y)
#the number of draws wanted
nDraws=10000
mu= 3.6
logy= (log(y)-mu)^2
tau2=sum(logy)/n
# then we use the function rinvgamma from r and we change on the pramters shape and scale.
df<-data.frame(x=rinvgamma(nDraws, shape=n/2, scale=n*tau2/2))
# Calculating the value of Phi_arg which is simply the squar root of the estimated sigma
df$phi_arg<-df$x/sqrt(2)
# Calculating the value of Gini index by appling the formula G=2*phi(sigma/sqrt(2))-1

# Xgrid
# x1_grid <- seq(min(X[,2]),max(X[,2]),0.1)
# Mu_draws <- matrix(0,length(x1_grid),2)
# for (ii in 1:length(x1_grid)){
# CurrMu <- BostonRes$betaSample %*% c(1,x1_grid[ii],XNewHouse[-1:-2])
# Mu_draws[ii,] <- quantile(CurrMu,probs=c(0.025,0.975))
# }
# plot(x1_grid,Mu_draws[,1],"n",main="95 % posterior probability intervals as a function of crim",
# xlab="crim", ylab="",ylim=c(10,30))
# lines(x1_grid,Mu_draws[,1],col="blue")
```

```r
# lines(x1_grid,Mu_draws[,2],col="blue")

df$G<-2*pnorm(sqrt(df$phi_arg),0,1)-1
#Ploting the Gini distribution
ggplot(df,aes(x=G))+geom_histogram(aes(y=..density..),color="white",
                                   linetype=6,
                                   fill='#14213D',binwidth = 0.05)+
  labs(x='Estimate Values',y=' ',
       title ='Distribution of Gini Coefficient')+
  geom_density(color = "#FCA311", size = 1)
sample_mean= mean(df$G)
sample_sd= sd(df$G)
# Compute the 95% confidence interval
lower_ci <- qnorm(0.025, mean = sample_mean, sd = sample_sd)
upper_ci <- qnorm(1-0.025, mean = sample_mean, sd = sample_sd)

#In case of utility fun
# nIter <- 1e4
# Mean_log_mu <- mean(log(rnorm(nIter,mean = 92,sd = 2)))
# ExpectedUtility <- function(c, Mean_log_mu){
#   EU <- 60 + sqrt(c)*Mean_log_mu - c
#   return(EU)
# }
#
# cGrid <- seq(0,20,by = 0.01)
# EU <- rep(NA,length(cGrid),1)
# count <- 0
# for (c in cGrid){
#   count <- count + 1
#   EU[count] = ExpectedUtility(c, Mean_log_mu)
# }
# plot(cGrid, EU, type = "l")
# cOpt = cGrid[which.max(EU)] # This is the optimal c
# points(cOpt,ExpectedUtility(c=cOpt, Mean_log_mu), col = "red",pch=19)
# cOpt
# Print the confidence interval
cat("95% confidence interval: [", round(lower_ci, 2), ",", round(upper_ci, 2), "]", "\n")

library(HDInterval)

#First we generate a sample using the posterior mean and Sd
post_G <- rnorm(n = 10000, mean = mean(df$G), sd = sd(df$G))
post_G_density <- density(post_G)

#We use the function HDI to find the 95% CI
hdpi=hdi(post_G, conf = 0.95)

cat("95% equal tail interavl for G : [", round(lower_ci, 2), ",",
    round(upper_ci, 2), "]", "\n")

cat("95% Highest Posterioe Density Interval for G: [", round(hdpi[1], 2), ",",
    round(hdpi[2], 2), "]", "\n")
```

```r
# Simulate and find the pi dist
# nSim <- 1e5
# x_pred <- matrix(0,nSim,1)
# for (jj in 1:nSim){
#   mu_draw <- rnorm(1,mean = 92,sd = 2)
#   x_pred[jj,1] <- rnorm(1,mean = mu_draw,sd = sqrt(50))
# }
# plot(density(x_pred),type="l",main="Posterior distribution of x_{n+1}",
#xlab="x_{n+1}",ylab="")

# The y and mu values given from the question
y<-c(-2.79,2.33,1.83,-2.44,2.23,2.33,2.07,2.02,2.14,2.54)
mu<- 2.51
n=length(y)

#We generate a sequeance of k values by 0.1
k<-seq(0.01,10,by=0.01)

#The drived posterior function can be expressed by
#(0.5/(besselI(k,0))^n)*exp(k*sum(cos(y-mu))-(k*0.5)
df<- data.frame(x=k,y=(0.5/(besselI(k,0))^n)*exp(k*sum(cos(y-mu))-(k*0.5)))

#Plotting function:
plt<-ggplot(df,aes(x=x,y=y))+geom_line(aes(),
                                       color="#FCA311",linetype=1,size=1)+
  labs(x='k values',y=' ',
       title ='The posterior distribution of k for the wind direction data')+
         geom_vline(xintercept =df$x[which.max(df$y)],
                    # we use df$x[which.max(df$y)] to find the mode
                    linetype = "dashed", color = "#14213D",size=.8)
plt
#compare the probs
#mean(Betas[,2]>0 & Betas[,3]>0)
#For betas we find the quatiles and interpretated the CI
# Effect_x1x2 <- Betas[,6]
# plot(density(Effect_x1x2),main="Posterior distribution",xlab="beta_5", ylab="")
# #quantile(Effect_x1x2,probs=c(0.025,0.975))


######## Example code y
# Mu <- Betas[,1] + Betas[,2]*50 + Betas[,3]*50**2 + Betas[,4]*25 + Betas[,5]*25**2 + Betas[,6]*50*25
# Sigma <- sqrt(Sigma2)
# y_Vals <- rnorm(10000,Mu,Sigma)
# plot(density(y_Vals),main="Posterior predictive distribution of y",xlab="y", ylab="")
######## Example code pi
# x_obs <- as.vector(c(1,40,1))
# lin_pred <- Betas%*%x_obs
# p_i <- exp(lin_pred)/(1+exp(lin_pred))
# plot(density(p_i),type="l",main="Posterior distribution of p_i",xlab="p_i",ylab="")

######

# Example code Pr(Y)
```

```
# T_y <- max(y)
# T_y_rep <- matrix(0,nIter,1)
# Mu <- Betas %*% t(X)
# for (ii in 1:nIter){
# y_Vals <- rnorm(length(y),Mu[ii,],Sigma[ii])
# T_y_rep[ii,1] <- max(y_Vals)
# }
# mean(T_y_rep >= T_y)
# New val x
# alpha_n <- 2326
# beta_n <- 7
# theta <- rgamma(1e4,shape = alpha_n,rate = beta_n)
# Q_6 <- rpois(1e4,theta)
# hist(Q_6,main="Posterior distribution",xlab="Q_6",ylab="")
# mean(Q_6 > 350)
```

Bertil Wegmann
Dept of Computer and Information Science
Linköping University

Bayesian Learning

Computer Lab 2

You are recommended to use R for solving the labs.
You work and submit your labs in pairs, but both of you should contribute equally and understand all parts of your solutions.
`It is not allowed to share exact solutions` with other student pairs.
The submitted lab reports will be verified through URKUND and indications of plagiarism will be investigated by the Disciplinary Board.
Submit your solutions via LISAM, no later than May 2 at 23:59.

Please note the following about the format of the submitted lab report:
1. The lab report should include all solutions and plots to the stated problems with necessary comments.
2. Submit the lab report with your code attached to the solution of each sub-problem (1a), 1b),...) in **one** PDF document.
3. Submit a separate file containing all code.

---

1. **Linear and polynomial regression**
   The dataset `Linkoping2022.xlsx` contains daily average temperatures (in degree Celcius) in Linköping over the course of the year 2022. Use the function read_xlsx(), which is included in the R package readxl (install.packages("readxl")), to import the dataset in R. The response variable is *temp* and the covariate *time* that you need to create yourself is defined by

$$time = \frac{\text{the number of days since the beginning of the year}}{365}.$$

   A Bayesian analysis of the following quadratic regression model is to be performed:

$$temp = \beta_0 + \beta_1 \cdot time + \beta_2 \cdot time^2 + \varepsilon, \; \varepsilon \overset{iid}{\sim} N(0, \sigma^2).$$

   (a) Use the conjugate prior for the linear regression model. The prior hyperparameters μ_0, Ω_0, ν_0 and σ_0^2 shall be set to sensible values. Start with $\mu_0 = (0, 100, -100)^T$, $\Omega_0 = 0.01 \cdot I_3$, $\nu_0 = 1$ and $\sigma_0^2 = 1$. Check if this prior agrees with your prior opinions by simulating draws from the joint prior of all parameters and for every draw compute the regression curve. This gives a collection of regression curves; one for each draw from the prior. Does the collection of curves look reasonable? If not, change the prior hyperparameters until the collection of prior regression curves agrees with your prior beliefs about the regression curve.
   [Hint: R package `mvtnorm` can be used and your *Inv*-χ^2 simulator of random draws from Lab 1.]

(b) Write a function that **simulate draws from the joint posterior distribution** of β_0, β_1,β_2 and σ^2.

i. Plot a histogram for each marginal posterior of the parameters.

ii. Make a scatter plot of the temperature data and overlay a curve for the posterior median of the regression function $f(time) = \mathrm{E}[temp|time] = \beta_0 + \beta_1 \cdot time + \beta_2 \cdot time^2$, i.e. the median of $f(time)$ is computed for every value of $time$. In addition, overlay curves for the 90% equal tail posterior probability intervals of $f(time)$, i.e. the 5 and 95 posterior percentiles of $f(time)$ is computed for every value of $time$. Does the posterior probability intervals contain most of the data points? Should they?

(c) It is of interest to locate the $time$ with the highest expected temperature (i.e. the $time$ where $f(time)$ is maximal). Let's call this value $\tilde{x}$. Use the simulated draws in (b) to simulate from the **posterior distribution of** $\tilde{x}$. [Hint: the regression curve is a quadratic polynomial. Given each posterior draw of β_0, β_1 and β_2, you can find a simple formula for $\tilde{x}$.]

(d) Say now that you want to **estimate a polynomial regression of order** 10, but you suspect that higher order terms may not be needed, and you worry about overfitting the data. Suggest a suitable prior that mitigates this potential problem. You do not need to compute the posterior. Just write down your prior. [Hint: the task is to specify μ_0 and Ω_0 in a suitable way.]

2. **Posterior approximation for classification with logistic regression**
The dataset `WomenAtWork.dat` contains $n = 132$ observations on the following eight variables related to women:

Variable	Data type	Meaning	Role
Work	Binary	Whether or not the woman works	Response y
Constant	1	Constant to the intercept	Feature
HusbandInc	Numeric	Husband's income	Feature
EducYears	Counts	Years of education	Feature
ExpYears	Counts	Years of experience	Feature
Age	Counts	Age	Feature
NSmallChild	Counts	Number of child ≤ 6 years in household	Feature
NBigChild	Counts	Number of child > 6 years in household	Feature

(a) Consider the logistic regression model:

$$\Pr(y = 1|\mathbf{x}, \beta) = \frac{\exp\left(\mathbf{x}^T\beta\right)}{1 + \exp\left(\mathbf{x}^T\beta\right)},$$

where y equals 1 if the woman works and 0 if she does not. $\mathbf{x}$ is a 7-dimensional vector containing the seven features (including a 1 to model the intercept). The goal is to approximate the posterior distribution of the parameter vector β with a multivariate normal distribution

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

where $\tilde{\beta}$ is the posterior mode and $J(\tilde{\beta}) = -\frac{\partial^2 \ln p(\beta|\mathbf{y})}{\partial\beta\partial\beta^T}|_{\beta=\tilde{\beta}}$ is the negative of the observed Hessian evaluated at the posterior mode. Note that $\frac{\partial^2 \ln p(\beta|\mathbf{y})}{\partial\beta\partial\beta^T}$ is

a 7×7 matrix with second derivatives on the diagonal and cross-derivatives $\frac{\partial^2 \ln p(\beta|\mathbf{y})}{\partial \beta_i \partial \beta_j}$ on the off-diagonal. You can compute this derivative by hand, but we will let the computer do it numerically for you. Calculate both $\tilde{\beta}$ and $J(\tilde{\beta})$ by using the `optim` function in R. [Hint: You may use code snippets from my demo of logistic regression in Lecture 6.] Use the prior $\beta \sim \mathcal{N}(0, \tau^2 I)$, where $\tau = 2$.

Present the numerical values of $\tilde{\beta}$ and $J_{\mathbf{y}}^{-1}(\tilde{\beta})$ for the `WomenAtWork` data. Compute an approximate 95% equal tail posterior probability interval for the regression coefficient to the variable NSmallChild. Would you say that this feature is of importance for the probability that a woman works?

[Hint: You can verify that your estimation results are reasonable by comparing the posterior means to the maximum likelihood estimates, given by: `glmModel <- glm(Work ~ 0 + ., data = WomenAtWork, family = binomial)`.]

(b) Use your normal approximation to the posterior from (a). Write a function that simulate draws from the posterior predictive distribution of $\Pr(y = 0|\mathbf{x})$, where the values of $\mathbf{x}$ corresponds to a 40-year-old woman, with two children (4 and 7 years old), 11 years of education, 7 years of experience, and a husband with an income of 18. Plot the posterior predictive distribution of $\Pr(y = 0|\mathbf{x})$ for this woman.

[Hints: The R package `mvtnorm` will be useful. Remember that $\Pr(y = 0|\mathbf{x})$ can be calculated for each posterior draw of β.]

(c) Now, consider 13 women which all have the same features as the woman in (b). Rewrite your function and plot the posterior predictive distribution for the number of women, out of these 13, that are not working. [Hint: Simulate from the binomial distribution, which is the distribution for a sum of Bernoulli random variables.]

GOOD LUCK!
BEST, BERTIL

Bayesian learning Lab 2

Mohamed Ali - Mohal954

2023-05-10

Linear and polynomial regression

The dataset TempLambohov.txt contains daily average temperatures (in degree Celcius) at Lambohov, Linköping over the course of the year 2019.
The response variable is temp and the covariate is

$$time = \frac{the\ number\ of\ days\ since\ the\ beginning\ of\ the\ year}{365}$$

A Bayesian analysis of the following quadratic regression model is to be performed:

$$temp = \beta_0 + \beta_1.time\ + e,\ e \sim N(0, \sigma^2)$$

To answer this question a conjugate prior for the linear regression model will be used in which: The joint prior for $\beta\ and\ \sigma^2$:

$$\begin{aligned} \beta|\sigma^2 &\sim\ \mathcal{N}(\mu_0, \sigma^2{\Omega_0}^{-1}) \\ \sigma^2 &\sim Inv - \chi^2(v_0, \sigma^2) \end{aligned}$$

While the posterior:

$$\begin{aligned} \beta|\sigma^2, y &\sim\ \mathcal{N}(\mu_n, \sigma^2{\Omega_n}^{-1}) \\ \sigma^2 &\sim Inv - \chi^2(v_n, {\sigma^2}_n) \\ where: & \\ \mu_n &= (X'X\Omega_0)^{-1}\ (X'X\hat{\beta} + \Omega_0\mu_0) \\ \Omega_n &= X'X + \Omega_0 \\ v_n &= v_0 + n \\ v_n\sigma^2 &= v_n\sigma^2 + (y'y + \mu_0'\Omega_0\mu_0 - \mu_n'\Omega_n\mu_n) \end{aligned}$$

First we start by reading the files and then we Create the covariate_time vaiable as (the number of days sinvce the beginning of the year 365).

```
#reading the files
df<-read_xlsx("Linkoping2022.xlsx")
#Creating the covariate_time vaiable as
#(the number of days sinvce the beginning of the year / 365)
a<- df$datetime
#begunning of the year
b<- '2022-01-01'
a<-format.POSIXlt(strptime(a,'%Y-%m-%d'))
```

```
b<-format.POSIXlt(strptime(b,'%Y-%m-%d'))
#time diff from the
x<-as.vector(difftime(a,b,units='days'))
df$cov_tm<-x/365
```

A

Use the conjugate prior for the linear regression model. The prior hyperparameters $\mu_0, \Omega_0, v_0, \sigma_0^2$ shall be set to sensible values. Start with $\mu_0 = (-10, 100, -100)^T, \Omega_0 = 0.02.I_3, v_0 = 3, \sigma_0^2 = 2$.
Check if this prior agrees with your prior opinions by simulating draws from the joint prior of all parameters and for every draw compute the regression curve.
This gives a collection of regression curves; one for each draw from the prior.
Does the collection of curves look reasonable? If not, change the prior hyperparameters until the collection of prior regression curves agrees with your prior beliefs about the regression curve. [Hint: R package mvtnorm can be used and your $inv - \chi^2$ simulator of random draws from Lab 1.].

We assume to use a conjugate prior from the linear regression in Lec 5 , we have been given the prior hyperparamteres as follow:

```
#We assume to use a conjugate prior from
#the linear regression in Lec 5 ,
#we have been given the prior hyperparamteres as follow:
mu_0= as.matrix(c(0,100,-100),ncol=3)
omega_0=0.01*diag(3)
v_0=1
segma2_0=1
n= length(df$temp)
ndraws=10
```

We have the joint prior for beta and segma2 defined as $\beta|\sigma^2 \sim \mathcal{N}(\mu_0, \sigma^2\Omega_0^{-1})$ and $\sigma^2 \sim Inv - \chi^2(v_o, \sigma_0^2)$ follows Inv-Chi(v0,sigma2_0).

First we draw our random samaple from inv-chi2 using the below defined function from Lec 3 slide 5

```
# Step 1: Draw X folow chi2(n - 1)
draw_chi_sq <- function(n) {
  return(rchisq(1, df = n - 1))
}
# Step 2: Compute sigma2 = (n - 1) * s² / X
compute_sigma_sq <- function(n, segma2_0, X) {
  return((n - 1) * segma2_0 / X)
}

# simulation
segma_estimation <- function(n, mu_0, segma2_0, ndraws) {
  results <- c()

  for (i in 1:ndraws) {
    X <- draw_chi_sq(n)
    sigma_sq <- compute_sigma_sq(n, segma2_0, X)
```

```
    results[i] <- sigma_sq
  }
  return(results)
}

sigma2<-segma_estimation(n, mu_0, segma2_0, ndraws)
```

Now we estimate the betas values using the formula $\beta|\sigma^2 \sim \mathcal{N}(\mu_0, \sigma^2\Omega_0^{-1})$ and we fit the regression based ontemp = beta0 + beta1 time + beta 2 time^2 + erorr.

*Note we have our error follows the normal distrbution by 0 and σ^2.

```
for (i in 1:length(sigma2)) {
  e<- rnorm(1,0,sigma2[i])
  res<-rmvnorm(1,mu_0,sigma2[i]*omega_0)
  temp= x=res[1,1]+res[1,2]*df$cov_tm+res[1,3]*df$cov_tm^2+e
  df[[paste0("temp_fit_",sigma2[i])]]<-temp
}
```

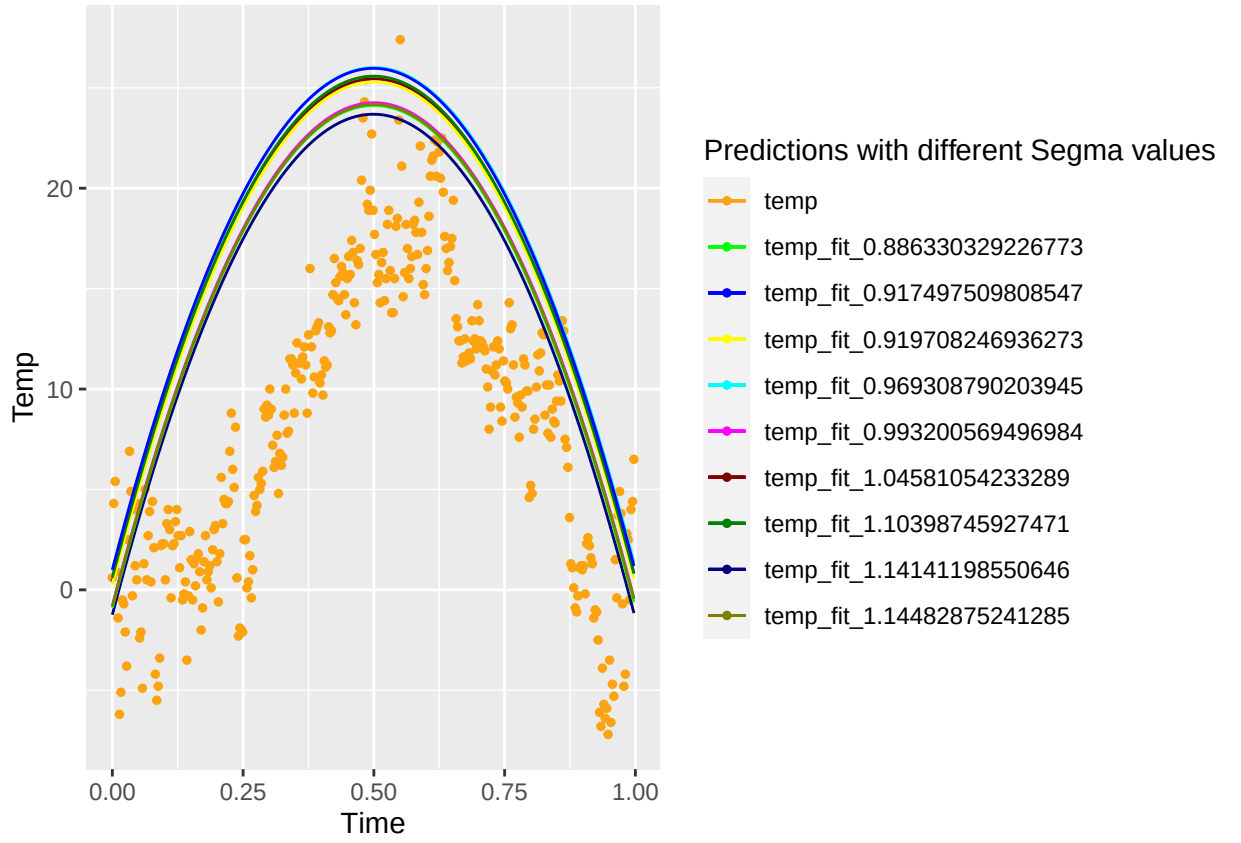
After we done with the draws now for every draw compute the regression curve.

```
## Example function
# x1_grid <- seq(min(X[,2]),max(X[,2]),0.1)
# Mu_draws <- matrix(0,length(x1_grid),2)
# for (ii in 1:length(x1_grid)){
# Curr_x <- c(1,x1_grid[ii],x1_grid[ii]**2,27,27**2,x1_grid[ii]*27)
# CurrMu <- Betas %*% Curr_x
# Mu_draws[ii,] <- quantile(CurrMu,probs=c(0.025,0.975))
# }
# plot(x1_grid,Mu_draws[,1],"n",main="95 % posterior
# probability intervals as a function of x1",
# xlab="x1", ylab="",ylim=c(0,500))
# lines(x1_grid,Mu_draws[,1],col="blue")
# lines(x1_grid,Mu_draws[,2],col="blue")


# Define a vector of colors
colors <-c("#FCA311", "#00FF00", "#0000FF", "#FFFF00", "#00FFFF",
           "#FF00FF", "#800000", "#008000", "#000080", "#808000",
           "#800080", "#008080", "#808080", "#FFC0CB", "#FFA500",
           "#FFD700", "#A52A2A", "#7FFF00", "#FF1493", "#00BFFF")

# Plot with different colored lines
plt <- ggplot(df, aes(x = cov_tm, y = temp)) +
  geom_point(aes(color = factor('temp')), size = 1)

for (i in names(df)[-c(1:4, ncol(df))]) {
  plt <- plt + geom_line(aes_string(y = i, color = factor(i)), linetype = 1)
}
```

```
## Warning: `aes_string()` was deprecated in ggplot2 3.0.0.
```

```
## i Please use tidy evaluation idioms with `aes()`.
## i See also `vignette("ggplot2-in-packages")` for more information.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
# Map colors to the lines
plt <- plt +
  scale_color_manual(values = colors) +
  labs(x = 'Time', y = 'Temp',color='Predictions with different Segma values')
plt
```



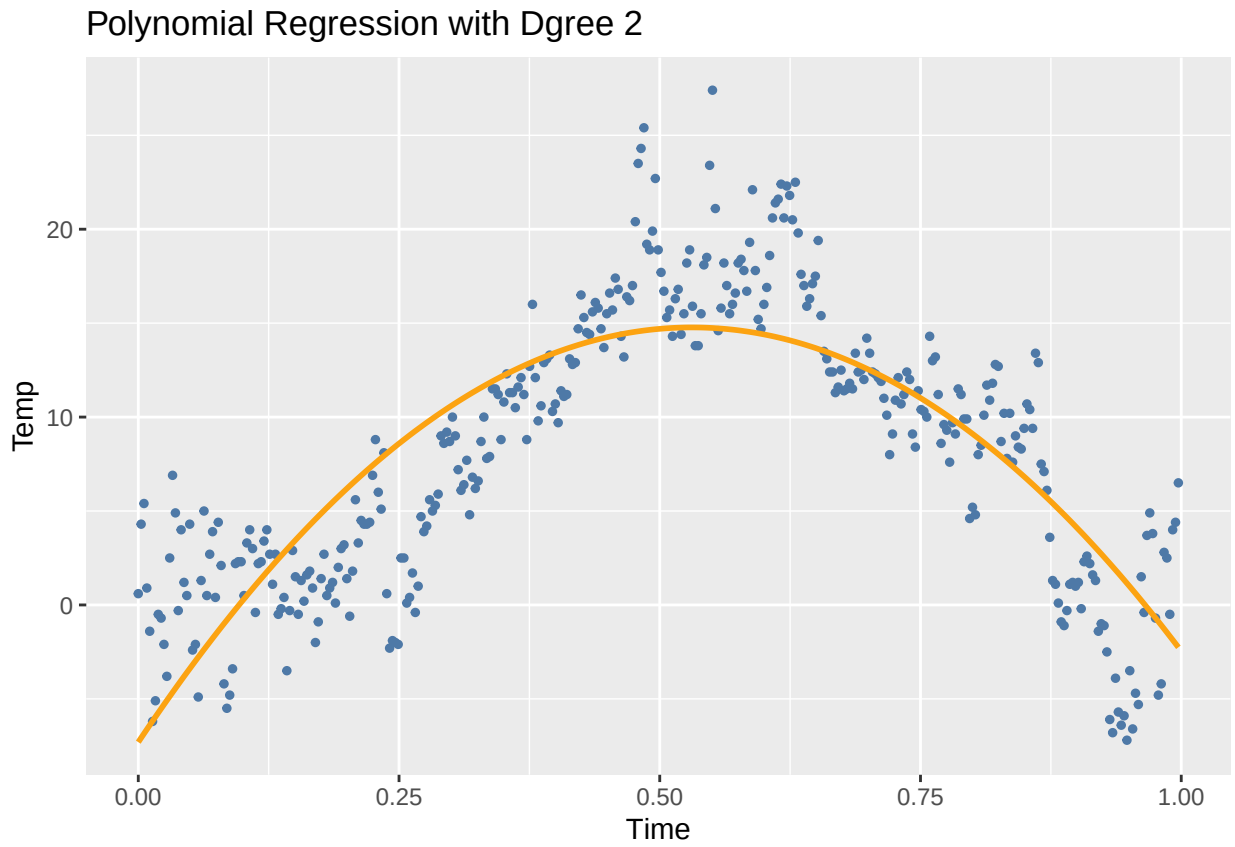
Comparing the above chart with our prior beliefs which we can see by running the polynomial model using the *lm* function in R with df = 2 we can see the fitted regression line with to some extend follow our regression line when σ^2 is equal to *1.037*.

```
degree <- 2  # Set the degree of the polynomial
x= df$cov_tm
y= df$temp
model <- lm(y ~ poly(x, degree, raw = TRUE))
df_plt<- data.frame(x=x,y=y)
z=predict(model)
# Print the model summary
```

```
# Plot the data and regression line
plt <- ggplot(df_plt, aes(x = x, y = y)) +
  geom_point(color = "#4E79A7", size = 1)+
  geom_line(aes(y = z), color = "#FCA311",size=1 ,linetype = 1)+
  labs(x = 'Time', y = 'Temp'
       ,title ='Polynomial Regression with Dgree 2')
```

```
## Warning: Using ‘size‘ aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use ‘linewidth‘ instead.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```

```
plt
```



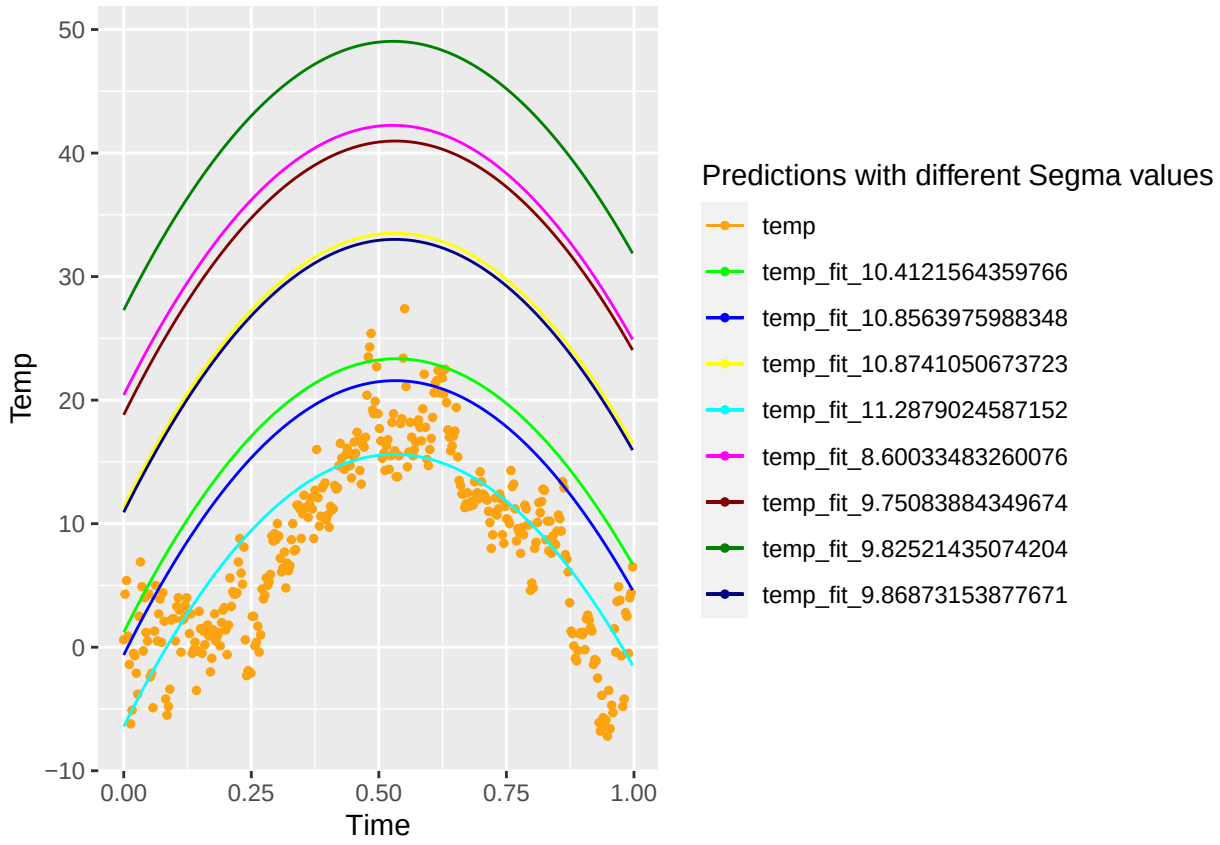

```
summary(model)
```

```
##
## Call:
## lm(formula = y ~ poly(x, degree, raw = TRUE))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
```

```
## -10.6557  -2.8525  -0.1874   2.5052  12.6580
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                   -7.3039     0.6582  -11.10   <2e-16 ***
## poly(x, degree, raw = TRUE)1  83.1568     3.0492   27.27   <2e-16 ***
## poly(x, degree, raw = TRUE)2 -78.3093     2.9600  -26.46   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.215 on 362 degrees of freedom
## Multiple R-squared:  0.6726, Adjusted R-squared:  0.6708
## F-statistic: 371.9 on 2 and 362 DF,  p-value: < 2.2e-16
```

Now we draw simulations using the modified μ_0 from the model results we have.

```
mu_news= as.matrix(c(7.3,83.1,-78.3),ncol=3)
segma2_new=10
sigma2<-segma_estimation(n,mu_news, segma2_new, ndraws)

for (i in 1:length(sigma2)) {
  e<- rnorm(1,0,sigma2[i])
  res<-rmvnorm(1,mu_news,sigma2[i]*omega_0)
  temp= x=res[1,1]+res[1,2]*df$cov_tm+res[1,3]*df$cov_tm^2+e
  df[[paste0("temp_fit_",sigma2[i])]]<-temp
}

# Define a vector of colors
colors <-c("#FCA311", "#00FF00", "#0000FF", "#FFFF00", "#00FFFF",
          "#FF00FF", "#800000", "#008000", "#000080", "#808000",
         "#800080", "#008080", "#808080", "#FFC0CB", "#FFA500",
        "#FFD700", "#A52A2A", "#7FFF00", "#FF1493", "#00BFFF")

# Plot with different colored lines
plt <- ggplot(df, aes(x = cov_tm, y = temp)) +
  geom_point(aes(color = factor('temp')), size = 1)

for (i in names(df)[-c(1:15, ncol(df))]) {
  plt <- plt + geom_line(aes_string(y = i, color = factor(i)), linetype = 1)
}

# Map colors to the lines
plt <- plt +
  scale_color_manual(values = colors) +
  labs(x = 'Time', y = 'Temp',color='Predictions with different Segma values')
plt
```



B

Write a function that simulate draws from the joint posterior distribution of $\beta_0, \beta_1, \beta_2, \sigma^2$. i- Plot a histogram for each marginal posterior of the parameters.
ii- Make a scatter plot of the temperature data and overlay a curve for the posterior median of the regression function $f(time) = E[temp|time] = \beta_0 + \beta_1.time + \beta_2.time^2$, i.e. the median of $f(time)$ is computed for every value of time.

In addition, overlay curves for the 95% equal tail posterior probability intervals of $f(time)$, i.e. the 2.5 and 97.5 posterior percentiles of $f(time)$ is computed for every value of time. Does the posterior probability intervals contain most of the data points? Should they?

As we want to estimate the uncertainty in the model parameters. Simulating from the joint posterior allows us to obtain a set of plausible values for all the parameters in the model, taking into account the observed data and prior information. to do so we have our non-informative prior:

$$p(\beta, \sigma^2) \propto \sigma^-1$$

Our joint posterior of β and σ^2:

$$\begin{aligned}
\beta|\sigma^2, y &\sim \mathcal{N}(\hat{\beta}, \sigma^2(X\text{t}X)^{-1}) \\
\sigma^2|y &\sim Inv - \chi^{\in}(n-k, s^2) \\
where &: \\
k &= number\ of\ \beta s\ in\ our\ case\ 3 \\
\hat{\beta} &= (X\text{t}X)^{-1}X\text{t}y \\
s^2 &= \frac{1}{n-k}(y - X\hat{\beta})\text{t}(y - X\hat{\beta})
\end{aligned}$$

Thus to simulate from the joint posterior we need to simulate from:
1- $p(\sigma|y$.
2- $p(\beta|\sigma^2, y)$.

And then we find the marginal posterior of β:

$$\beta|y \sim t_n - k(\hat{\beta}, s^2(X\text{t}X)^{-}1)$$

There for to draw a sample from the joint posterior distribution of $\beta_0, \beta_2, \beta_3$ *and* σ^2 we need first to calculate the value of $\hat{\beta} = (X\text{t}X)^{-}1X\text{t}y$:

```
y<-as.matrix(df$temp)
x<-as.matrix(cbind(1,df$cov_tm,df$cov_tm^2))
n<- length(y)
k=3
beta_ht<-(solve(t(x)%*%x))%*%(t(x)%*%y)
s2<-(1/(n-k))*t((y-(x%*%beta_ht)))%*%(y-(x%*%beta_ht))
```

Then we calculate the values of Ω_n , v_n , μ_n *and* ${\sigma^2}_n$ we use the same format as in eq 1

```
omega_n <- t(x) %*% x+omega_0
df_<-(n-k)
lmbda_<-s2
v_n <- v_0 + n
mu_n <- solve(t(x) %*% x + omega_0) %*% (t(x) %*% x %*% beta_ht + omega_0 %*% mu_0)
sigma2_n <- (v_0 * segma2_0 + (t(y) %*% y + t(mu_0) %*% omega_0
                               %*% mu_0 - t(mu_n) %*% omega_n %*% mu_n)) / v_n
```

Now we generate the valuse of β from $t_{n-k}(\hat{\beta},\ s^2(X\text{t}X)^{-1})$ using μ_n as our delta and σ_n^2 as sigma. in R we use function. *mvtnorm::rmvt*:

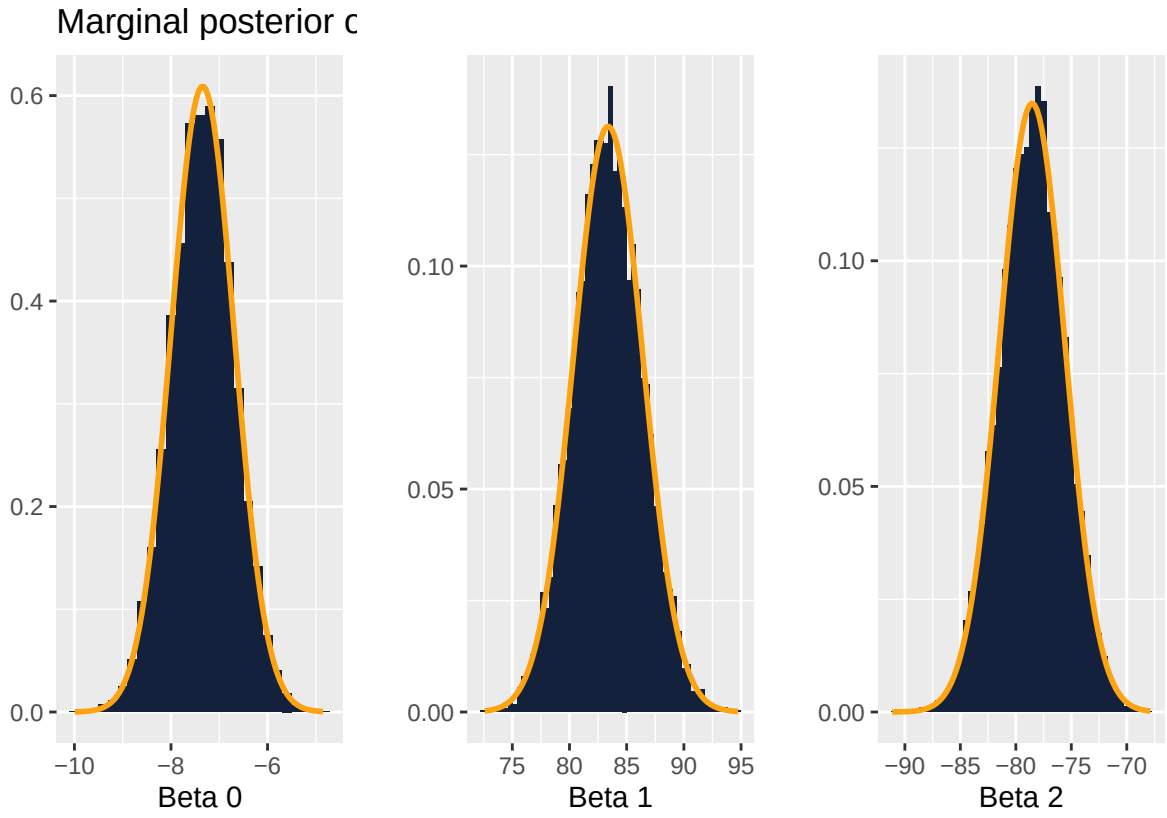
```
res<-data.frame(mvtnorm::rmvt(10000,delta=mu_n,df=df_,sigma=sigma2_n[1,1]*solve(t(x)%*%x)))
```

i/ Now we Plot a histogram for each marginal posterior of the parameters $\beta's$.

```
#We store the value of betas in a df and
#then we use this data to plot the histogarm of the betas
res<-data.frame(mvtnorm::rmvt(10000,delta=mu_n,
                              df=df_,sigma=sigma2_n[1,1]*solve(t(x)%*%x)))
```

```r
names(res)<-c('b_0','b_1','b_2')
plt1 <- ggplot(res,aes(x = b_0)) +geom_histogram(aes(y=..density..),
                  linetype=1,
                  fill='#14213D',binwidth = 0.2)+
  labs(x='Beta 0',y=' ',title ='Marginal posterior of the parameters')+
                stat_function(fun = dnorm, args = list(mean = mean(res$b_0),
                              sd = sd(res$b_0)),
                color = "#FCA311", size = 1)

plt2 <- ggplot(res,aes(x = b_1)) +geom_histogram(aes(y=..density..),
                                                 linetype=1,
                                                 fill='#14213D',binwidth = 0.4)+
  labs(x='Beta 1',y=' ')+
  stat_function(fun = dnorm, args = list(mean = mean(res$b_1),
                                         sd = sd(res$b_1)),
                color = "#FCA311", size = 1)
plt3 <- ggplot(res,aes(x = b_2)) +geom_histogram(aes(y=..density..),
                                                 linetype=1,
                                                 fill='#14213D',binwidth = 0.5)+
  labs(x='Beta 2',y=' ',)+
  stat_function(fun = dnorm, args = list(mean = mean(res$b_2),
                                         sd = sd(res$b_2)),
                color = "#FCA311", size = 1)

plt1+plt2+plt3
```

```
## Warning: The dot-dot notation (`..density..`) was deprecated in ggplot2 3.4.0.
## i Please use `after_stat(density)` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

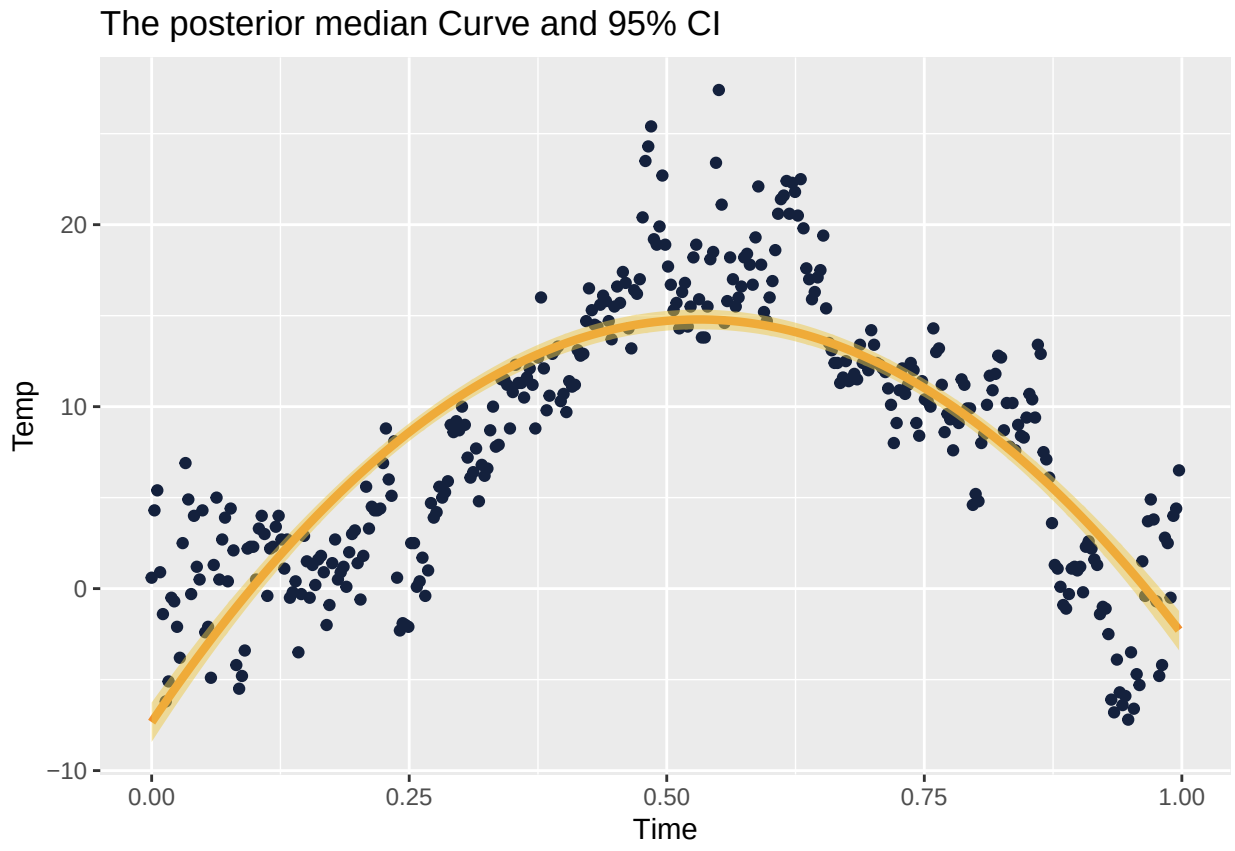


ii/ Make a scatter plot of the temperature data and overlay a curve for the posterior median of the regression function $f(time) = E[temp|time] = \beta_0 + \beta_1.time + \beta_2.time^2$.
First we need to calculate the $f(time)$ by using the results from i we can define:

```
# Calculating the median value point
median=as.matrix(apply(res, 2, median))

# we fint the regression  model P(time)=beta_0+beta_1*time+beta_2*time2
predicted_response <- x%*% median

#storing the median values
posterior_median <- apply(predicted_response, 1, median)

#Finding the predicted interval
prd_int <- data.frame(nrow = n, nrow = 2)
colnames(prd_int) <- c("CI_lower","CI_upper")
preds<- as.matrix(res)%*%t(x)
for(i in 1:nrow(x)){
  data_t <- preds[,i]
  #Here we have 95% CI using the function quantile
  prd_int[i,] <- quantile(data_t, probs = c(0.05,0.95))
}
```

By using the results from above we can fit the curve line for the posterior median and 95% CI on the scatter plot of the temperature data as below:

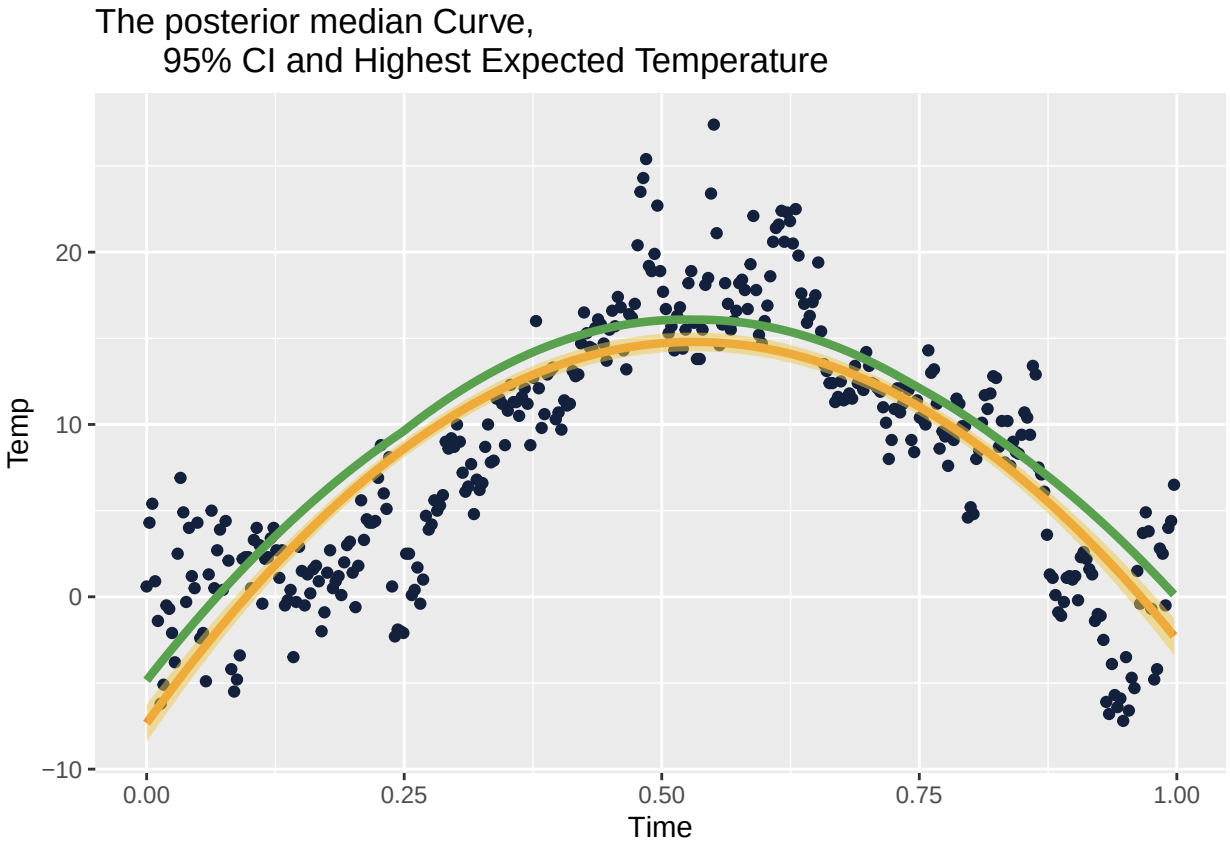
```r
# Storing the data in one data frame

plt_df=data.frame(x=df$cov_tm,y=df$temp,med=posterior_median)
plt_df= cbind(plt_df,prd_int)
# Calculate posterior median of the predicted response


plt <- ggplot(plt_df, aes(x = x, y = y)) +
  geom_point(color = "#14213D", size = 1.5)+
  geom_line(aes(y = med), color = "#F28E2B", linetype = 1,size=1.5)+
  geom_ribbon(aes(ymin = CI_lower, ymax = CI_upper)
              , alpha = 0.5,fill = "#EDC948")+
  labs(x = 'Time', y = 'Temp'
       ,title ='The posterior median Curve and 95% CI')
plt
```



C

It is of interest to locate the time with the highest expected temperature (i.e. the time where $f(time)$ is maximal). Let's call this value $\bar{x}$. Use the simulated draws in (b) to simulate from the posterior distribution of $\bar{x}$.
[Hint: the regression curve is a quadratic polynomial. Given each posterior draw of $\beta_0, \beta_1, \beta_2$, you can find a simple formula for $\bar{x}$.]. To simulate from the posterior distribution of the highest expected temperature $\bar{X}$ we can use the all possible prediction values that we generated in the task before, then by taking the max value we get the point point wise highest expected temperature over time:

```r
# Storing the data in one data frame
#Initite the storing vector
het<-c()

#Startingh the for loop
for (i in 1:nrow(x)) {
  het[i]<-max(preds[,i])
}

# binding the data into te ploting data frame

plt_df= cbind(plt_df,het)

#Ploting the data

plt <- ggplot(plt_df, aes(x = x, y = y)) +
  geom_point(color = "#14213D", size = 1.5)+
  geom_line(aes(y = het), color = "#59A14F", linetype = 1,size=1.5)+
  geom_line(aes(y = med), color = "#F28E2B", linetype = 1,size=1.5)+
  geom_ribbon(aes(ymin = CI_lower, ymax = CI_upper)
              , alpha = 0.5,fill = "#EDC948")+
  labs(x = 'Time', y = 'Temp'
       ,title ='The posterior median Curve,
       95% CI and Highest Expected Temperature'
       ,color = "Line Legend") +
  scale_color_manual(values = c("#14213D","#59A14F","#F28E2B","#EDC948")
                     , labels = c("1","2","3","4"))+
  theme(legend.position="bottom")
plt
```



D

Say now that you want to estimate a polynomial regression of order 8, but you suspect that higher order terms may not be needed, and you worry about overfitting the data. Suggest a suitable prior that mitigates this potential problem. You do not need to compute the posterior. Just write down your prior. [Hint: the task is to specify μ_0 and Ω_0 in a suitable way.].
To mitigate the problem of over fitting when estimating a polynomial resgression of order 10 without being worry of the over fitting, one can use Smoothness/Shrinkage/Regularization of the prior by introducing λ a penalization parameter (See lec 5 Slide 9 Lasso) in this method we have:

$$\beta_j|\sigma^2 \sim N(o, \frac{\sigma^2}{\lambda})$$

Here we have a large values of λ gives smoother fit. More shrinkage. where:

$$\mu_0 = 0$$
$$\Omega_0 = \lambda I$$

Which equivalent to *Penalized Likelihood*:

$$-2log\ p(\beta|\sigma^2, y, x) \propto (y - X\beta)'(y - X\beta) + \lambda\beta'\beta$$

Thus, the Posterior mean/mode gives ridge regressoin estimator:

$$\tilde{\beta} = (X\text{t}X + \lambda I)^{-1} X\text{t}y$$
$$if\ X\text{t}X = I$$
$$Then$$
$$\tilde{\beta} = \frac{1}{(1+\lambda)}\hat{\beta}$$

We might also be interested to determine the lambda, which could be by performing a cross validation on the test data pf using the Bayesian inference, where to us a prior for λ. we have this hierarchical setup:

$$\begin{aligned} y|\beta, \sigma^2, x &\sim N(X\beta, \sigma^2 I_n) \\ \beta|\sigma^2, \lambda &\sim N(0, \sigma^2 \lambda^{-1} I_m) \\ \sigma^2 &\sim Inv - \chi^2(v_o, \sigma_o^2) \\ \lambda &\sim Inv - \chi^2(\eta_0, \lambda_0) \\ so,\ \mu_o = 0,\ \Omega &= \lambda I_m \end{aligned}$$

and we have the joint posterior of β, σ^2, and λ is:

$$\begin{aligned} \beta|\sigma^2, \lambda, y &\sim N(\mu_n, \sigma^2 \Omega_n^{-1}) \\ \sigma^2|\lambda, y &\sim Inv - \chi^2(v_n, \sigma_n^2) \\ p(\lambda|y) &\propto \sqrt{\frac{|\Omega_0|}{|X^T X + \Omega_0|}} (\frac{v_n \sigma_n^2}{2})^{\frac{-v_n}{2}} . p(\lambda) \\ where,\ \Omega_0 &= \lambda I_m\ and\ p(\lambda)\ is\ the\ prior\ for\ \lambda\ and: \\ \mu_n &= (X^T X + \Omega_0)^{-1} X^T y \\ \Omega_n &= X^T X + \Omega_0 \\ v_n &= v_0 + n \\ v_n \sigma_n^2 &= v_0 \sigma_0^2 + y^T y - \mu_0^T \Omega_n \mu_n \end{aligned}$$

Posterior approximation for classifcation with logistic regression

A

Firstly we want to calculate the value of $\tilde{\beta}$ the posterior mode and the negative of the observed 7x7 hessian evaluated at the posterior mode $J(\tilde{\beta}) = -\frac{\partial^2\ lnp(\beta|y)}{\partial\beta\partial\beta^T}|_{\beta=\tilde{\beta}}$.
Using code snippets from my demo of logistic regression in Lecture 6. First we want to calculate the vale of $\tilde{\beta}$ and $J(\tilde{\beta}) = -\frac{\partial^2\ lnp(\beta|y)}{\partial\beta\partial\beta^T}|_{\beta=\tilde{\beta}}$ by using the *optim*, Note that we have $\tau = 2$ and $\beta \sim N(0, \frac{1}{\lambda} I)$.

```
# First we want to calculate the vale of beta and the Jacopiang
#inv of beta by using thge optim function and the code from the lec notes
# Note that we have tau = 2 and prior beta follows N(0,tau^2I)

### Select Logistic or Probit regression and install packages ###
Probit <- 0

### Prior and data inputs ###
Covs <- c(2:8) # Select which covariates/features to include
```

2. **Posterior approximation for classification with logistic regression**
The dataset `WomenAtWork.dat` contains $n = 168$ observations on the following eight variables related to women:

Variable	Data type	Meaning	Role
Work	Binary	Whether or not the woman works	Response y
Constant	1	Constant to the intercept	Feature
HusbandInc	Numeric	Husband's income	Feature
EducYears	Counts	Years of education	Feature
ExpYears	Counts	Years of experience	Feature
Age	Counts	Age	Feature
NSmallChild	Counts	Number of child ≤ 6 years in household	Feature
NBigChild	Counts	Number of child > 6 years in household	Feature

(a) Consider the logistic regression model:

$$\Pr(y = 1|\mathbf{x}, \beta) = \frac{\exp\left(\mathbf{x}^T\beta\right)}{1 + \exp\left(\mathbf{x}^T\beta\right)},$$

where y equals 1 if the woman works and 0 if she does not. $\mathbf{x}$ is a 7-dimensional vector containing the seven features (including a 1 to model the intercept). The goal is to approximate the posterior distribution of the parameter vector β with a multivariate normal distribution

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

where $\tilde{\beta}$ is the posterior mode and $J(\tilde{\beta}) = -\frac{\partial^2 \ln p(\beta|\mathbf{y})}{\partial\beta\partial\beta^T}|_{\beta=\tilde{\beta}}$ is the negative of the observed Hessian evaluated at the posterior mode. Note that $\frac{\partial^2 \ln p(\beta|\mathbf{y})}{\partial\beta\partial\beta^T}$ is a 7×7 matrix with second derivatives on the diagonal and cross-derivatives

Figure 1: Alt Text

$\frac{\partial^2 \ln p(\beta|\mathbf{y})}{\partial \beta_i \partial \beta_j}$ on the off-diagonal. You can compute this derivative by hand, but we will let the computer do it numerically for you. Calculate both $\tilde{\beta}$ and $J(\tilde{\beta})$ by using the `optim` function in R. [Hint: You may use code snippets from my demo of logistic regression in Lecture 6.] Use the prior $\beta \sim \mathcal{N}(0, \tau^2 I)$, where $\tau = 5$.

Present the numerical values of $\tilde{\beta}$ and $J_{\mathbf{y}}^{-1}(\tilde{\beta})$ for the `WomenAtWork` data. Compute an approximate 95% equal tail posterior probability interval for the regression coefficient to the variable NSmallChild. Would you say that this feature is of importance for the probability that a woman works?

[Hint: You can verify that your estimation results are reasonable by comparing the posterior means to the maximum likelihood estimates, given by: `glmModel <- glm(Work ~ 0 + ., data = WomenAtWork, family = binomial)`.]

(b) Use your normal approximation to the posterior from (a). Write a function that simulate draws from the posterior predictive distribution of $\Pr(y = 1|\mathbf{x})$, where the values of $\mathbf{x}$ corresponds to a 43-year-old woman, with two children (7 and 10 years old), 12 years of education, 8 years of experience, and a husband with an income of 20. Plot the posterior predictive distribution of $\Pr(y = 1|\mathbf{x})$ for this woman.

[Hints: The R package `mvtnorm` will be useful. Remember that $\Pr(y = 1|\mathbf{x})$ can be calculated for each posterior draw of β.]

(c) Now, consider 11 women which all have the same features as the woman in (b). Rewrite your function and plot the posterior predictive distribution for the number of women, out of these 11, that are working. [Hint: Simulate from the binomial distribution, which is the distribution for a sum of Bernoulli random variables.]

Figure 2: Alt Text

```
standardize <- F # If TRUE,covariates/features are standardized
                 #to mean 0 and variance 1
lambda <- 4 # scaling factor for the prior of beta in our case tau = 2

# Loading out data set
wat<-read.table("WomenAtWork.dat",header = T) # read data from file
Nobs <- dim(wat)[1] # number of observations
y <- wat[1] # y=1 if the women is working, otherwise y=0.
X <- as.matrix(wat[,Covs]) # Covs matrix 7*7
Xnames <- colnames(X)
# Standraizing the covs matrix
if (standardize){
  Index <- 2:(length(Covs)-1)
  X[,Index] <- scale(X[,Index])
}
Npar <- dim(X)[2]

#############################################################################
# This is to add y variable as binary response and adding
#intercept, for now it's not needed
# for (ii in 1:Nobs){
#   if (wat$quality[ii] > 5){
#     y[ii] <- 1
#   }
# }
# wat <- data.frame(intercept=rep(1,Nobs),wat) # add intercept
#############################################################################

# Setting up the prior
mu <- as.matrix(rep(0,Npar)) # Prior mean vector
Sigma <- (1/lambda)*diag(Npar) # Prior covariance matrix

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

LogPostLogistic <- function(betas,y,X,mu,Sigma){
  linPred <- X%*%betas;
  logLik <- sum( linPred*y - log(1 + 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)
}

LogPostProbit <- function(betas,y,X,mu,Sigma){
  linPred <- X%*%betas;
  SmallVal <- .Machine$double.xmin
  logLik <- sum(y*log(pnorm(linPred)+SmallVal) +
                  (1-y)*log(1-pnorm(linPred)+SmallVal) )
  logPrior <- dmvnorm(betas, mu, Sigma, log=TRUE);
  return(logLik + logPrior)
```

```
}

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

if (Probit==1){
  logPost = LogPostProbit;
} else{
  logPost = LogPostLogistic;
}

# 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,logPost,gr=NULL,y,X,mu,
                  Sigma,method=c("BFGS"),control=list(fnscale=-1),hessian=TRUE)

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

```
## [1] "The posterior mode is:"
```

```
print(OptimRes$par)
```

```
##              [,1]
## [1,] -0.056386120
## [2,] -0.031827146
## [3,]  0.122159465
## [4,]  0.117428313
## [5,] -0.034825257
## [6,] -0.843613303
## [7,] -0.004864958
## attr(,"names")
## [1] "Constant"    "HusbandInc"  "EducYears"   "ExpYears"    "Age"
## [6] "NSmallChild" "NBigChild"
```

```
print('The Hessian Matrix:')
```

```
## [1] "The Hessian Matrix:"
```

```
print(OptimRes$hessian)
```

```
##              [,1]        [,2]         [,3]         [,4]        [,5]        [,6]
## [1,]   -30.680116   -529.3142   -327.50822   -236.69284  -1090.0231   -6.677466
## [2,]  -529.314175 -13341.9063  -6766.54274  -4843.93682 -21997.1466 -120.512938
## [3,]  -327.508217  -6766.5427  -4173.35502  -2920.20417 -13322.2994  -89.131991
## [4,]  -236.692843  -4843.9368  -2920.20417  -3210.07312 -10128.0426  -52.468406
```

```
## [5,] -1090.023088 -21997.1466 -13322.29943 -10128.04260 -46241.4185 -222.328139
## [6,]    -6.677466   -120.5129    -89.13199    -52.46841   -222.3281  -12.483127
## [7,]   -35.842534   -680.3382   -427.53675   -263.56474  -1376.1948   -7.970026
##              [,7]
## [1,]   -35.842534
## [2,]  -680.338176
## [3,]  -427.536753
## [4,]  -263.564737
## [5,] -1376.194769
## [6,]    -7.970026
## [7,]   -95.588839
```

```
print('The approximate posterior standard deviation is:')
```

```
## [1] "The approximate posterior standard deviation is:"
```

```
print(approxPostStd)
```

```
##    Constant  HusbandInc   EducYears    ExpYears         Age NSmallChild
##  0.48111702  0.02091553  0.07054393  0.03223060  0.01981162  0.32730151
##   NBigChild
##  0.14263857
```

Now we compare with the results from the regression model: *glmModel<- glm(Work ~ 0 + ., data = WomenAtWork, family = binomial).*

```
glmModel<- glm(Work ~ 0 + ., data = wat, family = binomial)
summary(glmModel)
```

```
##
## Call:
## glm(formula = Work ~ 0 + ., family = binomial, data = wat)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3210  -0.9799   0.4423   0.9707   1.9131
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## Constant     0.02263    1.93083   0.012 0.990649
## HusbandInc  -0.03796    0.02229  -1.703 0.088573 .
## EducYears    0.18447    0.10007   1.844 0.065253 .
## ExpYears     0.12132    0.03353   3.618 0.000297 ***
## Age         -0.04858    0.03323  -1.462 0.143686
## NSmallChild -1.56485    0.51078  -3.064 0.002187 **
## NBigChild   -0.02526    0.17716  -0.143 0.886618
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
```

```
##     Null deviance: 182.99  on 132  degrees of freedom
## Residual deviance: 146.73  on 125  degrees of freedom
## AIC: 160.73
##
## Number of Fisher Scoring iterations: 4
```

The Coefficients of model results about shows relationship between each predictor variable and the log-odds of the outcome variable. variables like *HusbandInc,Age,NSmallChild and NBigChild* have a negative impact on the y variable in which the direction of the relationship, while the rest have a positive impact, our results from the teammate tell the same direction of the model estimates (i.e variables with negative impact *HusbandInc,Age,NSmallChild and NBigChild*).
However the magnitude of the coefficient which indicates the strength of the relationship is a bit different.

Now we Compute an approximate 95% equal tail posterior probability interval for the regression coefficient to the variable NSmallChild.

```
# We use the function rmvnorm to generate the
#variates using OptimRes$par as our mean and approxPostStd as sigma
postmode<-as.matrix(OptimRes$par[,1])
poststd<-solve(-OptimRes$hessian)
watvar<-data.frame(rmvnorm(n=10000,mean = postmode, sigma = poststd))

#For a 95% CI, you would typically calculate
#the lower and upper bounds at quantiles 0.025 and 0.975, respectively.
print('An approximate 95% equal tail posterior probability
      interval for the regression coeffcient to the variable NSmallChild is:')
```

```
## [1] "An approximate 95% equal tail posterior probability\n      interval for the regression coeffcie
```

```
print(quantile(watvar[,6],c(0.025,.975)))
```

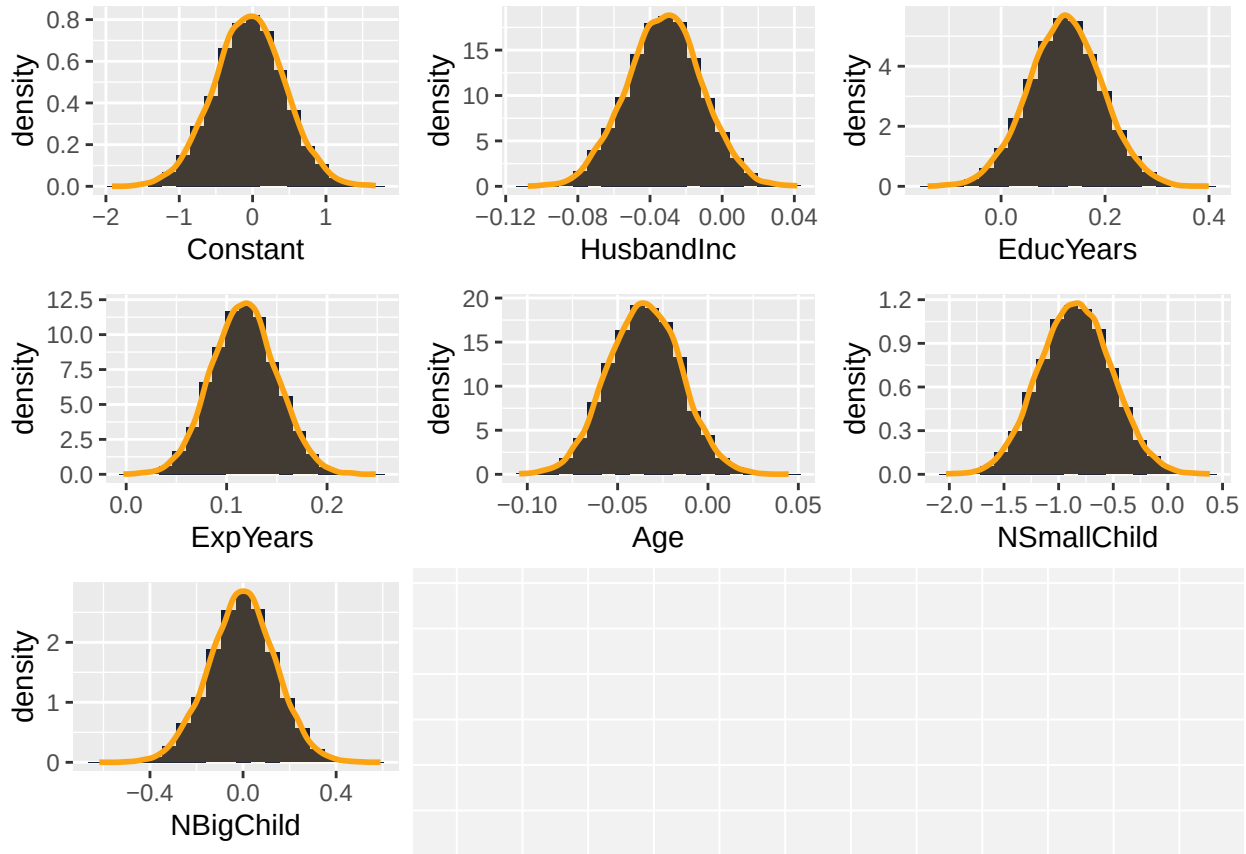
```
##       2.5%      97.5%
## -1.4837914 -0.1907636
```

The results of the CI tell us that the NSmallChild regression coefficient have a significant impact on the log-odds of the outcome variable as we can see the coefficient fall between the lower and the upper bound of the CI, the direction of this impact in negative and have the highest magnitude in the model coefficients.

B

Now we write a function that simulate draws from the posterior predictive distribution of Pr(y = 0|x),where the values of x corresponds to a 40-year-old woman, with two children (4 and 7 years old), 11 years of education, 7 years of experience, and a husband with an income of 18. Plot the posterior predictive distribution of Pr(y = 0|x) for this woman.
First we estimate the values of β using the normal approximation to the posterior from (a) above, the below graph shows the distribution of the estimated βs:

```
betas<-rmvnorm(n=10000,mean = postmode, sigma = poststd)
# ploting the beta distribution

p_data<- as.data.frame(betas)
colnames(p_data)<-colnames(wat[2:8])
names<-colnames(p_data)
p_fun<- function(coln){
          plt <- ggplot(p_data,aes_string(x = coln)) +
              geom_histogram(aes(y=..density..),linetype=1
                             ,fill='#14213D',bins = 20)+
              geom_density(alpha=.2,color="#FCA311",size=1,fill="#FCA311")
plt
}

plot(arrangeGrob(grobs = lapply(names, p_fun)))
```



Now we Write a function that simulate draws from the posterior predictive distribution of Pr(y = 0jx),where the values of x.

```
pred_prob<- function(ndraws,x_new){
        ### Select Logistic or Probit regression and install packages ###
        Probit <- 0
        ### Prior and data inputs ###
        Covs <- c(2:8) # Select which covariates/features to include
        standardize <- F # If TRUE, covariates/features
```

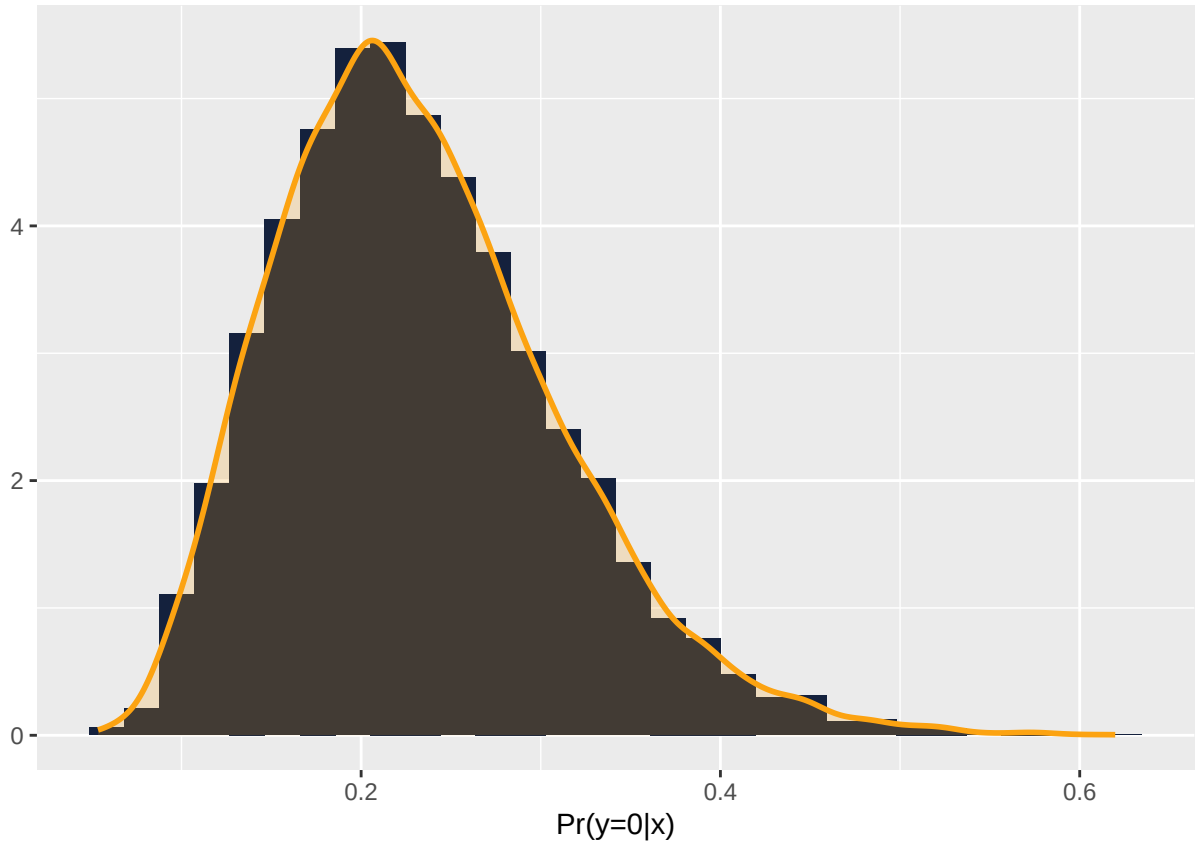
```r
#are standardized to mean 0 and variance 1
lambda <- 2 # scaling factor for the prior of beta in our case tau = 2
# Loading out data set
wat<-read.table("WomenAtWork.dat",header = T) # read data from file
Nobs <- dim(wat)[1] # number of observations
y <- wat[1] # y=1 if the women is working, otherwise y=0.
X <- as.matrix(wat[,Covs]) # Covs matrix 7*7
Xnames <- colnames(X)
# Standraizing the covs matrix
if (standardize){
  Index <- 2:(length(Covs)-1)
  X[,Index] <- scale(X[,Index])
}
Npar <- dim(X)[2]
# Setting up the prior
mu <- as.matrix(rep(0,Npar)) # Prior mean vector
Sigma <- (lambda)^2 *diag(Npar) # Prior covariance matrix
LogPostLogistic <- function(betas,y,X,mu,Sigma){
  linPred <- X%*%betas;
  logLik <- sum( linPred*y - log(1 + 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)
}
# Not in use we change the value to 0 at the beginning of the code
#########################################################################
LogPostProbit <- function(betas,y,X,mu,Sigma){
  linPred <- X%*%betas;
  SmallVal <- .Machine$double.xmin
  logLik <- sum(y*log(pnorm(linPred)+SmallVal) +
                  (1-y)*log(1-pnorm(linPred)+SmallVal))
  logPrior <- dmvnorm(betas, mu, Sigma, log=TRUE);
  return(logLik + logPrior)
}
#########################################################################
# Select the initial values for beta
initVal <- matrix(0,Npar,1)
if (Probit==1){
  logPost = LogPostProbit;
} else{
  logPost = LogPostLogistic;
}
# 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,logPost,gr=NULL,y,X,mu,Sigma,method=c("BFGS")
                  ,control=list(fnscale=-1),hessian=TRUE)

postmode<-as.matrix(OptimRes$par[,1])
poststd<- solve(-OptimRes$hessian)
```

```
        x_new<-as.matrix(x_new,ncol=1)
        betas<-rmvnorm(n=ndraws,mean = postmode, sigma = poststd)
        # Finding the value y givan the new Xs
        pr_y<-data.frame(x=betas%*%x_new)
        # Finding the probabilities using the logistics function
        pr_y$x_logit<-1/(1+exp(-pr_y$x))
        # Ploting the dataset
        plt <- ggplot(pr_y,aes(x = x_logit)) +geom_histogram(aes(y=..density..),
                  linetype=1, fill='#14213D')+
                  geom_density(alpha=.2,color="#FCA311",size=1,fill="#FCA311")+
                  labs(x='Pr(y=0|x)',y=' ',)
        plt
}
```

```
pred_prob(10000,c(1,18,11,7,40,1,1))
```

```
##
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

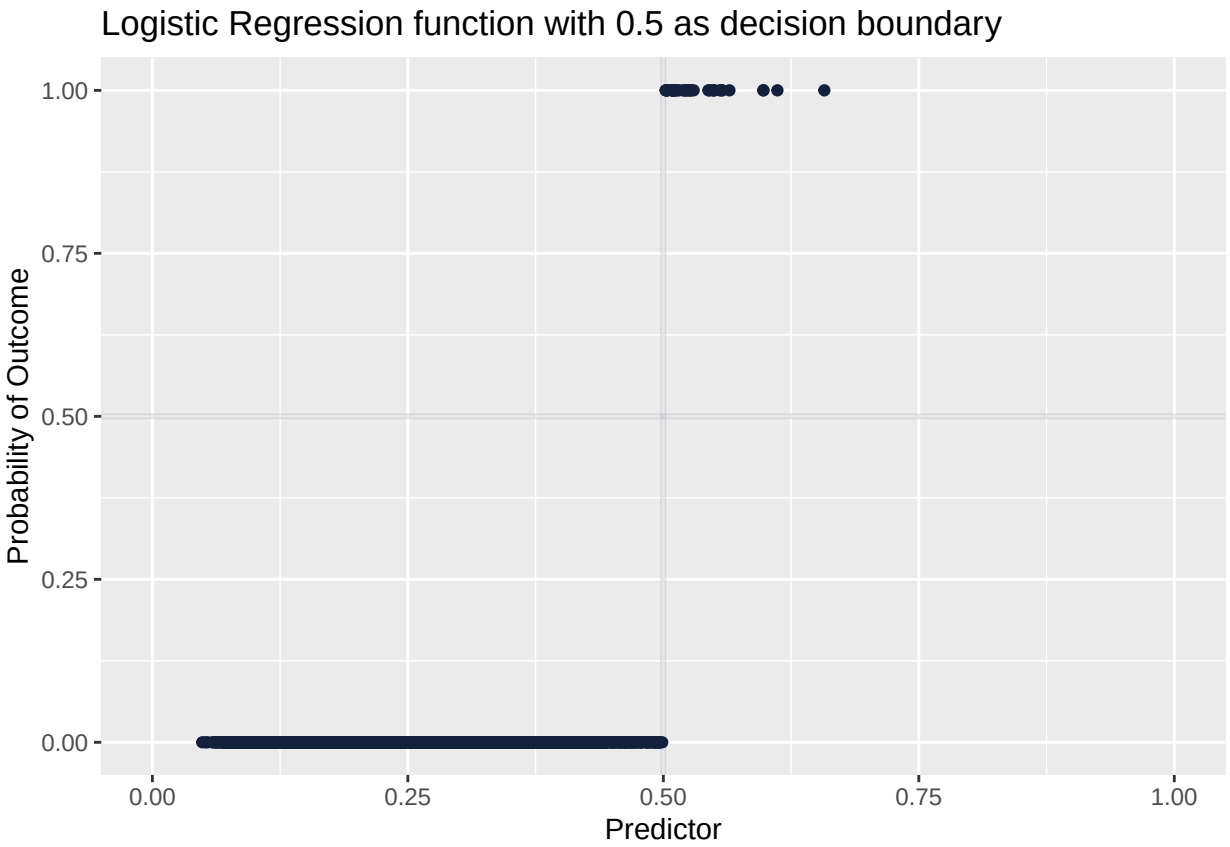


consider 13 women which all have the same features as the woman in (b).Rewrite your function and plot the posterior predictive distribution for the number of women, out of these 13, that are not working.

```
pred_prob2<- function(ndraws,x_new){
      ### Select Logistic or Probit regression and install packages ###
      Probit <- 0
      ### Prior and data inputs ###
      Covs <- c(2:8) # Select which covariates/features to include
      standardize <- F # If TRUE, covariates/features are
                       #standardized to mean 0 and variance 1
      lambda <- 2 # scaling factor for the prior of beta in our case tau = 2
      # Loading out data set
      wat<-read.table("WomenAtWork.dat",header = T) # read data from file
      Nobs <- dim(wat)[1] # number of observations
      y <- wat[1] # y=1 if the women is working, otherwise y=0.
      X <- as.matrix(wat[,Covs]) # Covs matrix 7*7
      Xnames <- colnames(X)
      # Standraizing the covs matrix
      if (standardize){
        Index <- 2:(length(Covs)-1)
        X[,Index] <- scale(X[,Index])
      }
      Npar <- dim(X)[2]
      # Setting up the prior
      mu <- as.matrix(rep(0,Npar)) # Prior mean vector
      Sigma <- (lambda)^2 *diag(Npar) # Prior covariance matrix
      LogPostLogistic <- function(betas,y,X,mu,Sigma){
        linPred <- X%*%betas;
        logLik <- sum( linPred*y - log(1 + 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)
      }
      # Not in use we change the value to 0 at the beginning of the code
      ################################################################################
      LogPostProbit <- function(betas,y,X,mu,Sigma){
        linPred <- X%*%betas;
        SmallVal <- .Machine$double.xmin
        logLik <- sum(y*log(pnorm(linPred)+SmallVal) +
                        (1-y)*log(1-pnorm(linPred)+SmallVal))
        logPrior <- dmvnorm(betas, mu, Sigma, log=TRUE);
        return(logLik + logPrior)
      }
      ################################################################################
      # Select the initial values for beta
      initVal <- matrix(0,Npar,1)
      if (Probit==1){
        logPost = LogPostProbit;
      } else{
        logPost = LogPostLogistic;
      }
      # 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,logPost,gr=NULL,y,X,mu,Sigma,method=c("BFGS")
                          ,control=list(fnscale=-1),hessian=TRUE)

        postmode<-as.matrix(OptimRes$par[,1])
        poststd<- solve(-OptimRes$hessian)

        x_new<-as.matrix(x_new,ncol=1)
        betas<-rmvnorm(n=ndraws,mean = postmode, sigma = poststd)
        # Finding the value y givan the new Xs
        pr_y<-data.frame(x=betas%*%x_new)
        # Finding the probabilities using the logistics function
        pr_y$x_logit<-1/(1+exp(-pr_y$x))
        #Adding the clasifier
        pr_y$job_flag <- ifelse(pr_y$x_logit <= 0.5, 0, 1)
        plt <- ggplot(pr_y, aes(x = x_logit, y = job_flag)) +
          geom_point(colour="#14213D") +
          # stat_smooth(method="glm",colour="#FCA311",
          #             alpha = 0.5, se=FALSE, fullrange=TRUE,
          #             method.args = list(family=binomial)) +
          xlab("Predictor") + xlim(c(0,1))+
          ylab("Probability of Outcome") +
          ggtitle("Logistic Regression function with 0.5 as decision boundary")+
    geom_vline(aes(xintercept = 0.5), color = "#14213D",size=1, alpha = 0.1) +
     geom_hline(aes(yintercept = 0.5), color = "#14213D",size=1, alpha = 0.1)

        plt
  }
```

```
pred_prob2(10000,c(1,18,11,7,40,1,1))
```



References:

1- Bertil Wegmann (2023). Bayesian Learning [Lecture notes]. 732A73, Department of Computer and Information Science,LiU University.

Code Appendix

```
knitr::opts_chunk$set(echo = TRUE)
set.seed(123)
library(ggplot2)
library(mvtnorm)
library(readxl)
library(LaplacesDemon)
library(patchwork)
library(gridExtra)
#reading the files
df<-read_xlsx("Linkoping2022.xlsx")
#Creating the covariate_time vaiable as
#(the number of days sinvce the beginning of the year / 365)
a<- df$datetime
#begunning of the year
```

```r
b<- '2022-01-01'
a<-format.POSIXlt(strptime(a,'%Y-%m-%d'))
b<-format.POSIXlt(strptime(b,'%Y-%m-%d'))
#time diff from the
x<-as.vector(difftime(a,b,units='days'))
df$cov_tm<-x/365
#We assume to use a conjugate prior from
#the linear regression in Lec 5 ,
#we have been given the prior hyperparamteres as follow:
mu_0= as.matrix(c(0,100,-100),ncol=3)
omega_0=0.01*diag(3)
v_0=1
segma2_0=1
n= length(df$temp)
ndraws=10

# Step 1: Draw X folow chi2(n - 1)
draw_chi_sq <- function(n) {
  return(rchisq(1, df = n - 1))
}
# Step 2: Compute sigma2 = (n - 1) * s² / X
compute_sigma_sq <- function(n, segma2_0, X) {
  return((n - 1) * segma2_0 / X)
}

# simulation
segma_estimation <- function(n, mu_0, segma2_0, ndraws) {
  results <- c()

  for (i in 1:ndraws) {
    X <- draw_chi_sq(n)
    sigma_sq <- compute_sigma_sq(n, segma2_0, X)
    results[i] <- sigma_sq
  }
  return(results)
}

sigma2<-segma_estimation(n, mu_0, segma2_0, ndraws)
for (i in 1:length(sigma2)) {
  e<- rnorm(1,0,sigma2[i])
  res<-rmvnorm(1,mu_0,sigma2[i]*omega_0)
  temp= x=res[1,1]+res[1,2]*df$cov_tm+res[1,3]*df$cov_tm^2+e
  df[[paste0("temp_fit_",sigma2[i])]]<-temp
}
## Example function
# x1_grid <- seq(min(X[,2]),max(X[,2]),0.1)
# Mu_draws <- matrix(0,length(x1_grid),2)
# for (ii in 1:length(x1_grid)){
# Curr_x <- c(1,x1_grid[ii],x1_grid[ii]**2,27,27**2,x1_grid[ii]*27)
# CurrMu <- Betas %*% Curr_x
# Mu_draws[ii,] <- quantile(CurrMu,probs=c(0.025,0.975))
# }
# plot(x1_grid,Mu_draws[,1],"n",main="95 % posterior
```

```
# probability intervals as a function of x1",
# xlab="x1", ylab="",ylim=c(0,500))
# lines(x1_grid,Mu_draws[,1],col="blue")
# lines(x1_grid,Mu_draws[,2],col="blue")


# Define a vector of colors
colors <-c("#FCA311", "#00FF00", "#0000FF", "#FFFF00", "#00FFFF",
           "#FF00FF", "#800000", "#008000", "#000080", "#808000",
           "#800080", "#008080", "#808080", "#FFC0CB", "#FFA500",
           "#FFD700", "#A52A2A", "#7FFF00", "#FF1493", "#00BFFF")

# Plot with different colored lines
plt <- ggplot(df, aes(x = cov_tm, y = temp)) +
  geom_point(aes(color = factor('temp')), size = 1)

for (i in names(df)[-c(1:4, ncol(df))]) {
  plt <- plt + geom_line(aes_string(y = i, color = factor(i)), linetype = 1)
}

# Map colors to the lines
plt <- plt +
  scale_color_manual(values = colors) +
  labs(x = 'Time', y = 'Temp',color='Predictions with different Segma values')
plt
degree <- 2  # Set the degree of the polynomial
x= df$cov_tm
y= df$temp
model <- lm(y ~ poly(x, degree, raw = TRUE))
df_plt<- data.frame(x=x,y=y)
z=predict(model)
# Print the model summary


# Plot the data and regression line
plt <- ggplot(df_plt, aes(x = x, y = y)) +
  geom_point(color = "#4E79A7", size = 1)+
  geom_line(aes(y = z), color = "#FCA311",size=1 ,linetype = 1)+
  labs(x = 'Time', y = 'Temp'
       ,title ='Polynomial Regression with Dgree 2')
plt
summary(model)

mu_news= as.matrix(c(7.3,83.1,-78.3),ncol=3)
segma2_new=10
sigma2<-segma_estimation(n,mu_news, segma2_new, ndraws)

for (i in 1:length(sigma2)) {
  e<- rnorm(1,0,sigma2[i])
  res<-rmvnorm(1,mu_news,sigma2[i]*omega_0)
  temp= x=res[1,1]+res[1,2]*df$cov_tm+res[1,3]*df$cov_tm^2+e
  df[[paste0("temp_fit_",sigma2[i])]]<-temp
}
```

```
# Define a vector of colors
colors <-c("#FCA311", "#00FF00", "#0000FF", "#FFFF00", "#00FFFF",
          "#FF00FF", "#800000", "#008000", "#000080", "#808000",
         "#800080", "#008080", "#808080", "#FFC0CB", "#FFA500",
        "#FFD700", "#A52A2A", "#7FFF00", "#FF1493", "#00BFFF")

# Plot with different colored lines
plt <- ggplot(df, aes(x = cov_tm, y = temp)) +
  geom_point(aes(color = factor('temp')), size = 1)

for (i in names(df)[-c(1:15, ncol(df))]) {
  plt <- plt + geom_line(aes_string(y = i, color = factor(i)), linetype = 1)
}

# Map colors to the lines
plt <- plt +
  scale_color_manual(values = colors) +
  labs(x = 'Time', y = 'Temp',color='Predictions with different Segma values')
plt
y<-as.matrix(df$temp)
x<-as.matrix(cbind(1,df$cov_tm,df$cov_tm^2))
n<- length(y)
k=3
beta_ht<-(solve(t(x)%*%x))%*%(t(x)%*%y)
s2<-(1/(n-k))*t((y-(x%*%beta_ht)))%*%(y-(x%*%beta_ht))
omega_n <- t(x) %*% x+omega_0
df_<-(n-k)
lmbda_<-s2
v_n <- v_0 + n
mu_n <- solve(t(x) %*% x + omega_0) %*% (t(x) %*% x %*% beta_ht + omega_0 %*% mu_0)
sigma2_n <- (v_0 * segma2_0 + (t(y) %*% y + t(mu_0) %*% omega_0
                               %*% mu_0 - t(mu_n) %*% omega_n %*% mu_n)) / v_n
res<-data.frame(mvtnorm::rmvt(10000,delta=mu_n,df=df_,sigma=sigma2_n[1,1]*solve(t(x)%*%x)))
#We store the value of betas in a df and
#then we use this data to plot the histogarm of the betas
res<-data.frame(mvtnorm::rmvt(10000,delta=mu_n,
                              df=df_,sigma=sigma2_n[1,1]*solve(t(x)%*%x)))


names(res)<-c('b_0','b_1','b_2')
plt1 <- ggplot(res,aes(x = b_0)) +geom_histogram(aes(y=..density..),
                  linetype=1,
                  fill='#14213D',binwidth = 0.2)+
  labs(x='Beta 0',y=' ',title ='Marginal posterior of the parameters')+
                stat_function(fun = dnorm, args = list(mean = mean(res$b_0),
                              sd = sd(res$b_0)),
                color = "#FCA311", size = 1)

plt2 <- ggplot(res,aes(x = b_1)) +geom_histogram(aes(y=..density..),
                                                 linetype=1,
                                                 fill='#14213D',binwidth = 0.4)+
  labs(x='Beta 1',y=' ')+
  stat_function(fun = dnorm, args = list(mean = mean(res$b_1),
```

```
                                        sd = sd(res$b_1)),
                color = "#FCA311", size = 1)
plt3 <- ggplot(res,aes(x = b_2)) +geom_histogram(aes(y=..density..),
                                                 linetype=1,
                                                 fill='#14213D',binwidth = 0.5)+
  labs(x='Beta 2',y=' ',)+
  stat_function(fun = dnorm, args = list(mean = mean(res$b_2),
                                        sd = sd(res$b_2)),
                color = "#FCA311", size = 1)

plt1+plt2+plt3

# Calculating the median value point
median=as.matrix(apply(res, 2, median))

# we fint the regression  model P(time)=beta_0+beta_1*time+beta_2*time2
predicted_response <- x%*% median

#storing the median values
posterior_median <- apply(predicted_response, 1, median)

#Finding the predicted interval
prd_int <- data.frame(nrow = n, nrow = 2)
colnames(prd_int) <- c("CI_lower","CI_upper")
preds<- as.matrix(res)%*%t(x)
for(i in 1:nrow(x)){
  data_t <- preds[,i]
  #Here we have 95% CI using the function quantile
  prd_int[i,] <- quantile(data_t, probs = c(0.05,0.95))
}
# Storing the data in one data frame

plt_df=data.frame(x=df$cov_tm,y=df$temp,med=posterior_median)
plt_df= cbind(plt_df,prd_int)
# Calculate posterior median of the predicted response


plt <- ggplot(plt_df, aes(x = x, y = y)) +
  geom_point(color = "#14213D", size = 1.5)+
  geom_line(aes(y = med), color = "#F28E2B", linetype = 1,size=1.5)+
  geom_ribbon(aes(ymin = CI_lower, ymax = CI_upper)
              , alpha = 0.5,fill = "#EDC948")+
  labs(x = 'Time', y = 'Temp'
       ,title ='The posterior median Curve and 95% CI')
plt
# Storing the data in one data frame
#Initite the storing vector
het<-c()

#Startingh the for loop
for (i in 1:nrow(x)) {
  het[i]<-max(preds[,i])
}
```

```r
# binding the data into te ploting data frame

plt_df= cbind(plt_df,het)

#Ploting the data

plt <- ggplot(plt_df, aes(x = x, y = y)) +
  geom_point(color = "#14213D", size = 1.5)+
  geom_line(aes(y = het), color = "#59A14F", linetype = 1,size=1.5)+
  geom_line(aes(y = med), color = "#F28E2B", linetype = 1,size=1.5)+
  geom_ribbon(aes(ymin = CI_lower, ymax = CI_upper)
              , alpha = 0.5,fill = "#EDC948")+
  labs(x = 'Time', y = 'Temp'
       ,title ='The posterior median Curve,
       95% CI and Highest Expected Temperature'
       ,color = "Line Legend") +
  scale_color_manual(values = c("#14213D","#59A14F","#F28E2B","#EDC948")
                     , labels = c("1","2","3","4"))+
  theme(legend.position="bottom")
plt
# First we want to calculate the vale of beta and the Jacopiang
#inv of beta by using thge optim function and the code from the lec notes
# Note that we have tau = 2 and prior beta follows N(0,tau^2I)

### Select Logistic or Probit regression and install packages ###
Probit <- 0

### Prior and data inputs ###
Covs <- c(2:8) # Select which covariates/features to include
standardize <- F # If TRUE,covariates/features are standardized
                 #to mean 0 and variance 1
lambda <- 4 # scaling factor for the prior of beta in our case tau = 2

# Loading out data set
wat<-read.table("WomenAtWork.dat",header = T) # read data from file
Nobs <- dim(wat)[1] # number of observations
y <- wat[1] # y=1 if the women is working, otherwise y=0.
X <- as.matrix(wat[,Covs]) # Covs matrix 7*7
Xnames <- colnames(X)
# Standraizing the covs matrix
if (standardize){
  Index <- 2:(length(Covs)-1)
  X[,Index] <- scale(X[,Index])
}
Npar <- dim(X)[2]

#############################################################################
# This is to add y variable as binary response and adding
#intercept, for now it's not needed
# for (ii in 1:Nobs){
#   if (wat$quality[ii] > 5){
#     y[ii] <- 1
#   }
```

```
# }
# wat <- data.frame(intercept=rep(1,Nobs),wat) # add intercept
################################################################################

# Setting up the prior
mu <- as.matrix(rep(0,Npar)) # Prior mean vector
Sigma <- (1/lambda)*diag(Npar) # Prior covariance matrix

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

LogPostLogistic <- function(betas,y,X,mu,Sigma){
  linPred <- X%*%betas;
  logLik <- sum( linPred*y - log(1 + 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)
}

LogPostProbit <- function(betas,y,X,mu,Sigma){
  linPred <- X%*%betas;
  SmallVal <- .Machine$double.xmin
  logLik <- sum(y*log(pnorm(linPred)+SmallVal) +
                  (1-y)*log(1-pnorm(linPred)+SmallVal) )
  logPrior <- dmvnorm(betas, mu, Sigma, log=TRUE);
  return(logLik + logPrior)
}

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

if (Probit==1){
  logPost = LogPostProbit;
} else{
  logPost = LogPostLogistic;
}

# 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,logPost,gr=NULL,y,X,mu,
                  Sigma,method=c("BFGS"),control=list(fnscale=-1),hessian=TRUE)

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

```
print('The Hessian Matrix:')
print(OptimRes$hessian)
print('The approximate posterior standard deviation is:')
print(approxPostStd)
glmModel<- glm(Work ~ 0 + ., data = wat, family = binomial)
summary(glmModel)
# We use the function rmvnorm to generate the
#variates using OptimRes$par as our mean and approxPostStd as sigma
postmode<-as.matrix(OptimRes$par[,1])
poststd<-solve(-OptimRes$hessian)
watvar<-data.frame(rmvnorm(n=10000,mean = postmode, sigma = poststd))

#For a 95% CI, you would typically calculate
#the lower and upper bounds at quantiles 0.025 and 0.975, respectively.
print('An approximate 95% equal tail posterior probability
      interval for the regression coeffcient to the variable NSmallChild is:')
print(quantile(watvar[,6],c(0.025,.975)))
betas<-rmvnorm(n=10000,mean = postmode, sigma = poststd)
# ploting the beta distribution

p_data<- as.data.frame(betas)
colnames(p_data)<-colnames(wat[2:8])
names<-colnames(p_data)
p_fun<- function(coln){
          plt <- ggplot(p_data,aes_string(x = coln)) +
              geom_histogram(aes(y=..density..),linetype=1
                             ,fill='#14213D',bins = 20)+
              geom_density(alpha=.2,color="#FCA311",size=1,fill="#FCA311")
plt
}

plot(arrangeGrob(grobs = lapply(names, p_fun)))
pred_prob<- function(ndraws,x_new){
        ### Select Logistic or Probit regression and install packages ###
        Probit <- 0
        ### Prior and data inputs ###
        Covs <- c(2:8) # Select which covariates/features to include
        standardize <- F # If TRUE, covariates/features
        #are standardized to mean 0 and variance 1
        lambda <- 2 # scaling factor for the prior of beta in our case tau = 2
        # Loading out data set
        wat<-read.table("WomenAtWork.dat",header = T) # read data from file
        Nobs <- dim(wat)[1] # number of observations
        y <- wat[1] # y=1 if the women is working, otherwise y=0.
        X <- as.matrix(wat[,Covs]) # Covs matrix 7*7
        Xnames <- colnames(X)
        # Standraizing the covs matrix
        if (standardize){
          Index <- 2:(length(Covs)-1)
          X[,Index] <- scale(X[,Index])
        }
        Npar <- dim(X)[2]
        # Setting up the prior
```

```
        mu <- as.matrix(rep(0,Npar)) # Prior mean vector
        Sigma <- (lambda)^2 *diag(Npar) # Prior covariance matrix
        LogPostLogistic <- function(betas,y,X,mu,Sigma){
          linPred <- X%*%betas;
          logLik <- sum( linPred*y - log(1 + 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)
        }
        # Not in use we change the value to 0 at the beginning of the code
        ##########################################################################
        LogPostProbit <- function(betas,y,X,mu,Sigma){
          linPred <- X%*%betas;
          SmallVal <- .Machine$double.xmin
          logLik <- sum(y*log(pnorm(linPred)+SmallVal) +
                          (1-y)*log(1-pnorm(linPred)+SmallVal))
          logPrior <- dmvnorm(betas, mu, Sigma, log=TRUE);
          return(logLik + logPrior)
        }
        ##########################################################################
        # Select the initial values for beta
        initVal <- matrix(0,Npar,1)
        if (Probit==1){
          logPost = LogPostProbit;
        } else{
          logPost = LogPostLogistic;
        }
        # 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,logPost,gr=NULL,y,X,mu,Sigma,method=c("BFGS")
                          ,control=list(fnscale=-1),hessian=TRUE)

        postmode<-as.matrix(OptimRes$par[,1])
        poststd<- solve(-OptimRes$hessian)

        x_new<-as.matrix(x_new,ncol=1)
        betas<-rmvnorm(n=ndraws,mean = postmode, sigma = poststd)
        # Finding the value y givan the new Xs
        pr_y<-data.frame(x=betas%*%x_new)
        # Finding the probabilities using the logistics function
        pr_y$x_logit<-1/(1+exp(-pr_y$x))
        # Ploting the dataset
        plt <- ggplot(pr_y,aes(x = x_logit)) +geom_histogram(aes(y=..density..),
                    linetype=1, fill='#14213D')+
                    geom_density(alpha=.2,color="#FCA311",size=1,fill="#FCA311")+
                    labs(x='Pr(y=0|x)',y=' ',)
        plt
  }
pred_prob(10000,c(1,18,11,7,40,1,1))
  pred_prob2<- function(ndraws,x_new){
```

```
### Select Logistic or Probit regression and install packages ###
Probit <- 0
### Prior and data inputs ###
Covs <- c(2:8) # Select which covariates/features to include
standardize <- F # If TRUE, covariates/features are
                 #standardized to mean 0 and variance 1
lambda <- 2 # scaling factor for the prior of beta in our case tau = 2
# Loading out data set
wat<-read.table("WomenAtWork.dat",header = T) # read data from file
Nobs <- dim(wat)[1] # number of observations
y <- wat[1] # y=1 if the women is working, otherwise y=0.
X <- as.matrix(wat[,Covs]) # Covs matrix 7*7
Xnames <- colnames(X)
# Standraizing the covs matrix
if (standardize){
  Index <- 2:(length(Covs)-1)
  X[,Index] <- scale(X[,Index])
}
Npar <- dim(X)[2]
# Setting up the prior
mu <- as.matrix(rep(0,Npar)) # Prior mean vector
Sigma <- (lambda)^2 *diag(Npar) # Prior covariance matrix
LogPostLogistic <- function(betas,y,X,mu,Sigma){
  linPred <- X%*%betas;
  logLik <- sum( linPred*y - log(1 + 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)
}
# Not in use we change the value to 0 at the beginning of the code
#################################################################################
LogPostProbit <- function(betas,y,X,mu,Sigma){
  linPred <- X%*%betas;
  SmallVal <- .Machine$double.xmin
  logLik <- sum(y*log(pnorm(linPred)+SmallVal) +
                  (1-y)*log(1-pnorm(linPred)+SmallVal))
  logPrior <- dmvnorm(betas, mu, Sigma, log=TRUE);
  return(logLik + logPrior)
}
#################################################################################
# Select the initial values for beta
initVal <- matrix(0,Npar,1)
if (Probit==1){
  logPost = LogPostProbit;
} else{
  logPost = LogPostLogistic;
}
# 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,logPost,gr=NULL,y,X,mu,Sigma,method=c("BFGS")
                          ,control=list(fnscale=-1),hessian=TRUE)

        postmode<-as.matrix(OptimRes$par[,1])
        poststd<- solve(-OptimRes$hessian)

        x_new<-as.matrix(x_new,ncol=1)
        betas<-rmvnorm(n=ndraws,mean = postmode, sigma = poststd)
        # Finding the value y givan the new Xs
        pr_y<-data.frame(x=betas%*%x_new)
        # Finding the probabilities using the logistics function
        pr_y$x_logit<-1/(1+exp(-pr_y$x))
        #Adding the clasifier
        pr_y$job_flag <- ifelse(pr_y$x_logit <= 0.5, 0, 1)
        plt <- ggplot(pr_y, aes(x = x_logit, y = job_flag)) +
          geom_point(colour="#14213D") +
          # stat_smooth(method="glm",colour="#FCA311",
          #             alpha = 0.5, se=FALSE, fullrange=TRUE,
          #             method.args = list(family=binomial)) +
          xlab("Predictor") + xlim(c(0,1))+
          ylab("Probability of Outcome") +
          ggtitle("Logistic Regression function with 0.5 as decision boundary")+
    geom_vline(aes(xintercept = 0.5), color = "#14213D",size=1, alpha = 0.1) +
     geom_hline(aes(yintercept = 0.5), color = "#14213D",size=1, alpha = 0.1)

        plt
  }
################################################################################
        # Example function
        # LogPost <- function(theta,n,Sumx3){
        #
        #     logLik <- n*log(theta) -  Sumx3*theta;
        #     logPrior <- 2*log(theta) - 4*theta;
        #
        #     return(logLik + logPrior)
        #   }
        #   theta_grid <- seq(0.01,2.5,0.01)
        #   PostDens_propto <- exp(LogPost(theta_grid,5,4.084))
        #   PostDens <- PostDens_propto/(0.01*sum(PostDens_propto))
        #   plot(theta_grid,PostDens,main="Posterior distribution"
        #        ,xlab="theta", ylab="")
        #
        # n <- 5
        # Sumx3 <- 4.084
        # OptRes <- optim(0.5,LogPost,gr=NULL,n,Sumx3,method=c("L-BFGS-B")
        #                 ,lower=0.1,control=list(fnscale=-1),hessian=TRUE)
        #
        # plot(theta_grid,PostDens,col="blue",main="Posterior distribution"
        #      ,xlab="theta", ylab="")
        # lines(theta_grid,dnorm(theta_grid
        #         ,mean = OptRes$par,sd = sqrt(-1/OptRes$hessian)),col="red")
        # legend("topleft", legend=c("Approximation", "Exact"
        #                            ,col=c("red", "blue"), lty=1:2, cex=0.8))
```

```
        ##########################################################################
pred_prob2(10000,c(1,18,11,7,40,1,1))
```

Bertil Wegmann
Dept of Computer and Information Science
Linköping University

Bayesian Learning

Computer Lab 3

You are recommended to use R for solving the labs.
You work and submit your labs in pairs, but both of you should contribute equally and understand all parts of your solutions.
`It is not allowed to share exact solutions` with other student pairs.
The submitted lab reports will be verified through OURIGINAL and indications of plagiarism will be investigated by the Disciplinary Board.
Submit your solutions via LISAM, no later than May 16 at 23:59.

Please note the following about the format of the submitted lab report:
1. The lab report should include all solutions and plots to the stated problems with necessary comments.
2. Submit the lab report with your code attached to the solution of each sub-problem (1a), 1b),...) in **one** PDF document.
3. Submit a separate file containing all code.

1. **Gibbs sampler for a normal model**
   The dataset `Precipitation.rds` consists of daily records of weather with rain or snow (in units of mm) from the beginning of 1962 to the end of 2008 in a certain area. Assume the natural log of the daily precipitation $\{y_1, ..., y_n\}$ to be independent normally distributed, $\ln y_1, ..., \ln y_n | \mu, \sigma^2 \overset{iid}{\sim} \mathcal{N}(\mu, \sigma^2)$, where both μ and σ^2 are unknown. Let $\mu \sim \mathcal{N}(\mu_0, \tau_0^2)$ independently of $\sigma^2 \sim Inv\text{-}\chi^2(\nu_0, \sigma_0^2)$.

   (a) Implement (code!) a Gibbs sampler that simulates from the joint posterior $p(\mu, \sigma^2 | \ln y_1, ..., \ln y_n)$. The full conditional posteriors are given on the slides from Lecture 7. Evaluate the convergence of the Gibbs sampler by calculating the Inefficiency Factors (IFs) and by plotting the trajectories of the sampled Markov chains.

   (b) Plot the following in one figure: 1) a histogram or kernel density estimate of the daily precipitation $\{y_1, ..., y_n\}$. 2) The resulting posterior predictive density $p(\tilde{y}|y_1, \ldots, y_n)$ using the simulated posterior draws from (a). How well does the posterior predictive density agree with this data?

2. **Metropolis Random Walk for Poisson regression**
   Consider the following Poisson regression model

$$y_i | \beta \overset{iid}{\sim} \text{Poisson}\left[\exp\left(\mathbf{x}_i^T \beta\right)\right], \; i = 1, ..., n,$$

   where y_i is the count for the ith observation in the sample and x_i is the p-dimensional vector with covariate observations for the ith observation. Use the data set

`eBayNumberOfBidderData.dat`. This dataset contains observations from 1000 eBay auctions of coins. The response variable is **nBids** and records the number of bids in each auction. The remaining variables are features/covariates ($\mathbf{x}$):

- **Const** (for the intercept)
- **PowerSeller** (equal to 1 if the seller is selling large volumes on eBay)
- **VerifyID** (equal to 1 if the seller is a verified seller by eBay)
- **Sealed** (equal to 1 if the coin was sold in an unopened envelope)
- **MinBlem** (equal to 1 if the coin has a minor defect)
- **MajBlem** (equal to 1 if the coin has a major defect)
- **LargNeg** (equal to 1 if the seller received a lot of negative feedback from customers)
- **LogBook** (logarithm of the book value of the auctioned coin according to expert sellers. Standardized)
- **MinBidShare** (ratio of the minimum selling price (starting price) to the book value. Standardized).

(a) Obtain the maximum likelihood estimator of β in the Poisson regression model for the eBay data [Hint: `glm.R`, don't forget that `glm()` adds its own intercept so don't input the covariate Const]. Which covariates are significant?

(b) Let's do a Bayesian analysis of the Poisson regression. Let the prior be $\beta \sim \mathcal{N}\left[\mathbf{0}, 100 \cdot (\mathbf{X}^T\mathbf{X})^{-1}\right]$, where $\mathbf{X}$ is the $n \times p$ covariate matrix. This is a commonly used prior, which is called Zellner's g-prior. Assume first that the posterior density is approximately multivariate normal:

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

where $\tilde{\beta}$ is the posterior mode and $J_{\mathbf{y}}(\tilde{\beta})$ is the negative Hessian at the posterior mode. $\tilde{\beta}$ and $J_{\mathbf{y}}(\tilde{\beta})$ can be obtained by numerical optimization (`optim.R`) exactly like you already did for the logistic regression in Lab 2 (but with the log posterior function replaced by the corresponding one for the Poisson model, which you have to code up.).

(c) Let's simulate from the actual posterior of β using the Metropolis algorithm and compare the results with the approximate results in b). Program a general function that uses the Metropolis algorithm to generate random draws from an *arbitrary* posterior density. In order to show that it is a general function for any model, we denote the vector of model parameters by θ. Let the proposal density be the multivariate normal density mentioned in Lecture 8 (random walk Metropolis):

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

where $\Sigma = J_{\mathbf{y}}^{-1}(\tilde{\beta})$ was obtained in b). The value c is a tuning parameter and should be an input to your Metropolis function. The user of your Metropolis function should be able to supply her own posterior density function, not necessarily for the Poisson regression, and still be able to use your Metropolis function. This is not so straightforward, unless you have come across *function objects* in `R`. The note **HowToCodeRWM.pdf** in Lisam describes how you can do this in R.

Now, use your new Metropolis function to sample from the posterior of β in the Poisson regression for the eBay dataset. Assess MCMC convergence by graphical methods.

(d) Use the MCMC draws from c) to simulate from the predictive distribution of the number of bidders in a new auction with the characteristics below. Plot the predictive distribution. What is the probability of no bidders in this new auction?

- **PowerSeller** = 1
- **VerifyID** = 0
- **Sealed** = 1
- **MinBlem** = 0
- **MajBlem** = 1
- **LargNeg** = 0
- **LogBook** = 1.2
- **MinBidShare** = 0.8

3. **Time series models in Stan**

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

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

for given values of μ, ϕ and σ^2. Start the process at $x_1 = \mu$ and then simulate values for x_t for $t = 2, 3 \ldots, T$ and return the vector $x_{1:T}$ containing all time points. Use $\mu = 13, \sigma^2 = 3$ and $T = 300$ and look at some different realizations (simulations) of $x_{1:T}$ for values of ϕ between -1 and 1 (this is the interval of ϕ where the AR(1)-process is stationary). Include a plot of at least one realization in the report. What effect does the value of ϕ have on $x_{1:T}$?

(b) Use your function from a) to simulate two AR(1)-processes, $x_{1:T}$ with $\phi = 0.2$ and $y_{1:T}$ with $\phi = 0.95$. Now, treat your simulated vectors as synthetic data, and treat the values of μ, ϕ and σ^2 as unknown parameters. Implement Stan-code that samples from the posterior of the three parameters, using suitable non-informative priors of your choice. [Hint: Look at the time-series models examples in the Stan user's guide/reference manual, and note the different parameterization used here.]

i. Report the posterior mean, 95% credible intervals and the number of effective posterior samples for the three inferred parameters for each of the simulated AR(1)-process. Are you able to estimate the true values?

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

GOOD LUCK!
BEST, BERTIL

Bayesian Learning Lab 3

Mohamed Ali - Mohal954

2023-05-15

Gibbs sampler for a normal model

A

Implement (code!) a Gibbs sampler that simulates from the joint posterior $p(\mu, \sigma^2|lny_1, ..., lnyn)$. The full conditional posteriors are given on the slides from Lecture 7.
Evaluate the convergence of the Gibbs sampler by calculating the Inefficiency Factors (IFs) and by plotting the trajectories of the sampled Markov chains.
The Gibbs Sampling algorithm:

1- Choose inital values $\theta_2^0, \theta_3^0, ..., \theta_n^0$.
2- Repeat for j=1,. . . ,N:
- Draw θ_1^j from $p(\theta_1|\theta_2^{j-1}, ..., \theta_k^{j-1})$.
- Draw θ_2^j from $p(\theta_2|\theta_1^j, ..., \theta_k^{j-1})$.
.
.
.
3- Draw θ_k^j from $p(\theta_k|\theta_1^j, ..., \theta_{k-1}^j)$.
4- Return draws: θ^1,. . . ,θ^N, where θ^j=(θ_1^j,. . . ,θ_k^j).

$$\bar{\theta} = 1/N \sum_{t=1}^{N} \theta^t$$

$$var(\bar{\theta}) = \frac{\sigma^2}{N}$$

Autocorrelated samples :

$$var(\bar{\theta}) = \frac{\sigma^2}{N}(1 + 2\sum_{k=1}^{\inf} \rho_k)$$

Convergence diagnostics:

$$var(\bar{\theta}) = \frac{\sigma^2}{N}(1 + 2\sum_{k=1}^{\inf} \rho_k)$$

$$IF = 1 + 2\sum_{k=1}^{\inf} \rho_k$$

$$ESS = N/IF$$

We start by building our function using the Slides notes we can define the Normal Model with Conditionally conjugate prior as:

$$\mu \sim N(\mu_0, \tau_0^2)\sigma \quad \sim Inv - \chi^2(v_0^2, \sigma_0^2)$$

And the full conditional posteriors:

$$\mu|\sigma^2, x \sim N(\mu_n, \tau_n^2)$$
$$\sigma^2|\mu, x \sim Inv - \chi^2(v_n, \frac{v_0\sigma^2 + \sum_{i=1}^{n}(x_i - \mu)^2}{n + v_0})$$

Where we have:

$$\mu_n = w\bar{x} + (1-w)\mu_0$$
$$w = \frac{\frac{n}{\sigma^2}}{\frac{n}{\sigma^2} + \frac{1}{\tau_0^2}}$$
$$\frac{1}{\tau_n^2} = \frac{n}{\sigma^2} + \frac{1}{\tau_0^2}$$
$$v_n = v_0 + n$$

We start by defining the above variables in R, Note that as we have $p(lny_1, ..., lnyn|\mu, \sigma^2) \sim N(\mu, \sigma^2)$ we will transform our variable by taking the natural log of the daily precipitation.

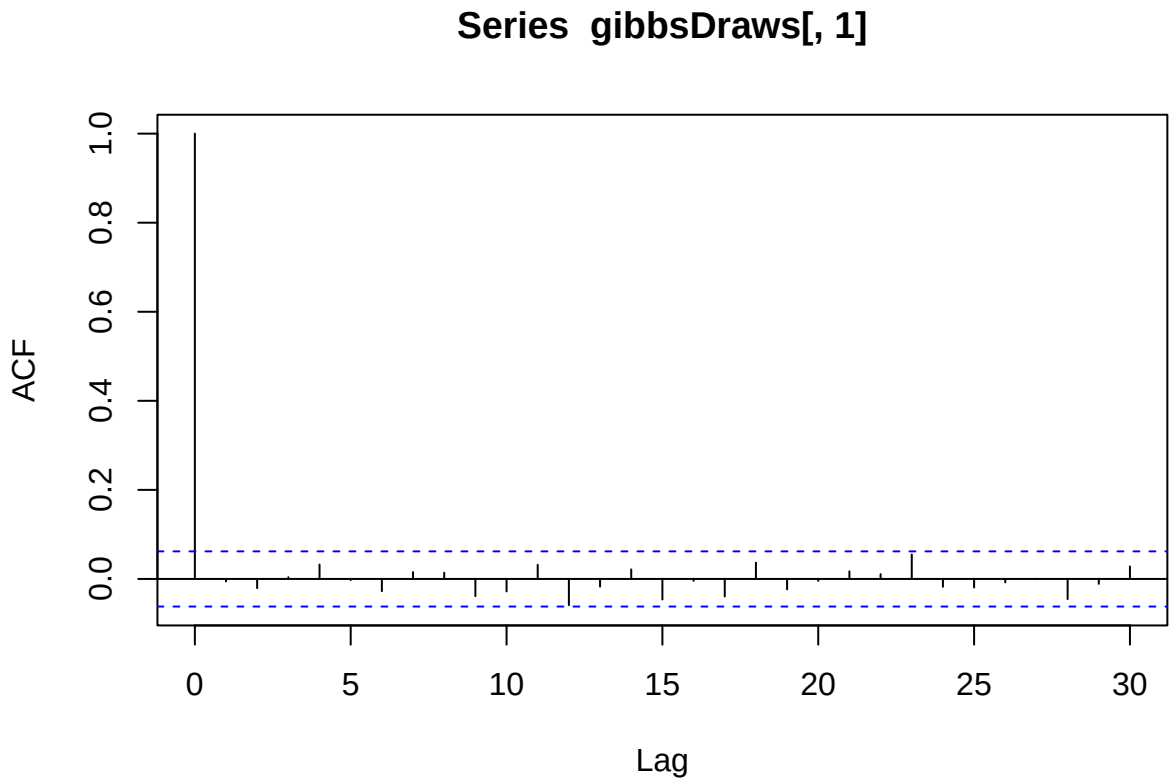
```
# we start by reading our dataset using the readRDS command in R
df <- data.frame(x=readRDS("Precipitation.rds"))
# in the task we looking at natural log of the daily precipitation
#lny1,lny2,lny3,...lnyn follows N(mu,sigma)
# We add a new var called logx
df$logx<- log(df$x)
sigma2_0<- var(df$logx)  # Sample var
tau2_0<- 1# arbirtry initail value let it be 1
n<- nrow(df)# Sample size
mu_0<- mean(df$logx)#sample mean
w<- (n/sigma2_0)/((n/sigma2_0) + (1/tau2_0))# value used to calculate mu_n
v_0<- 1 # arbirtry initail value let it be 1
v_n<- v_0+n
mu_n<- w*(mean(df$logx))+ (1-w)*mu_0
 ###### This part of the code has been modified######
tau2_n<- 1/((n/sigma2_0)+(1/tau2_0))################## shoud be 1/tau
#####################################################
nDraws<- 1000
gibbsDraws <- matrix(0,nDraws,2)
```

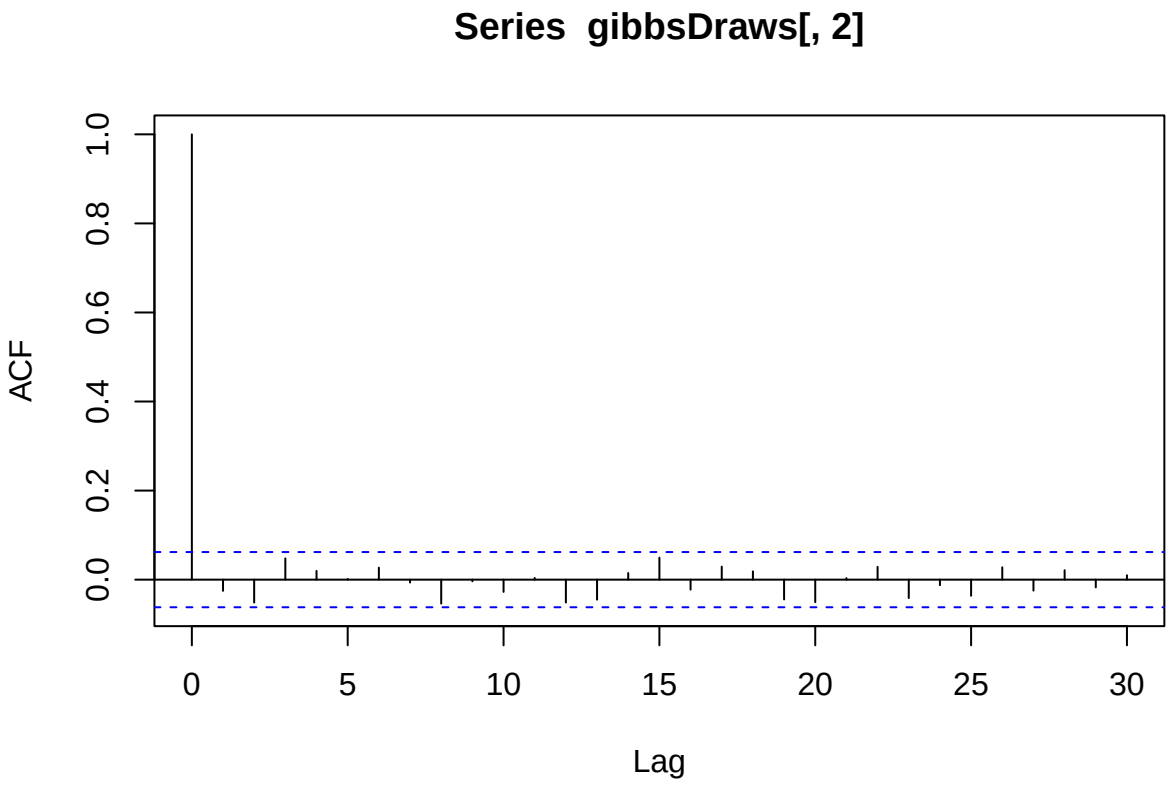
Now we write a code to simulates from the joint posterior using Gibbs sampler.

```
### From lec notes we can use this part of the code
  ###### This part of the code has been modified######
scale=sigma2_0 ########################################
for (i in 1:nDraws){###################################
  w<- (n/scale)/((n/scale) + (1/tau2_0)) #############
  mu_n<- w*(mean(df$logx))+ (1-w)*mu_0 ############### We need to update those parameters as well
  tau2_n<- 1/((n/scale)+(1/tau2_0))##################
  ####################################################
  # Update theta1 ----> mu given theta2
  theta1 <- rnorm(1, mean = mu_n, sd = (tau2_n))
```

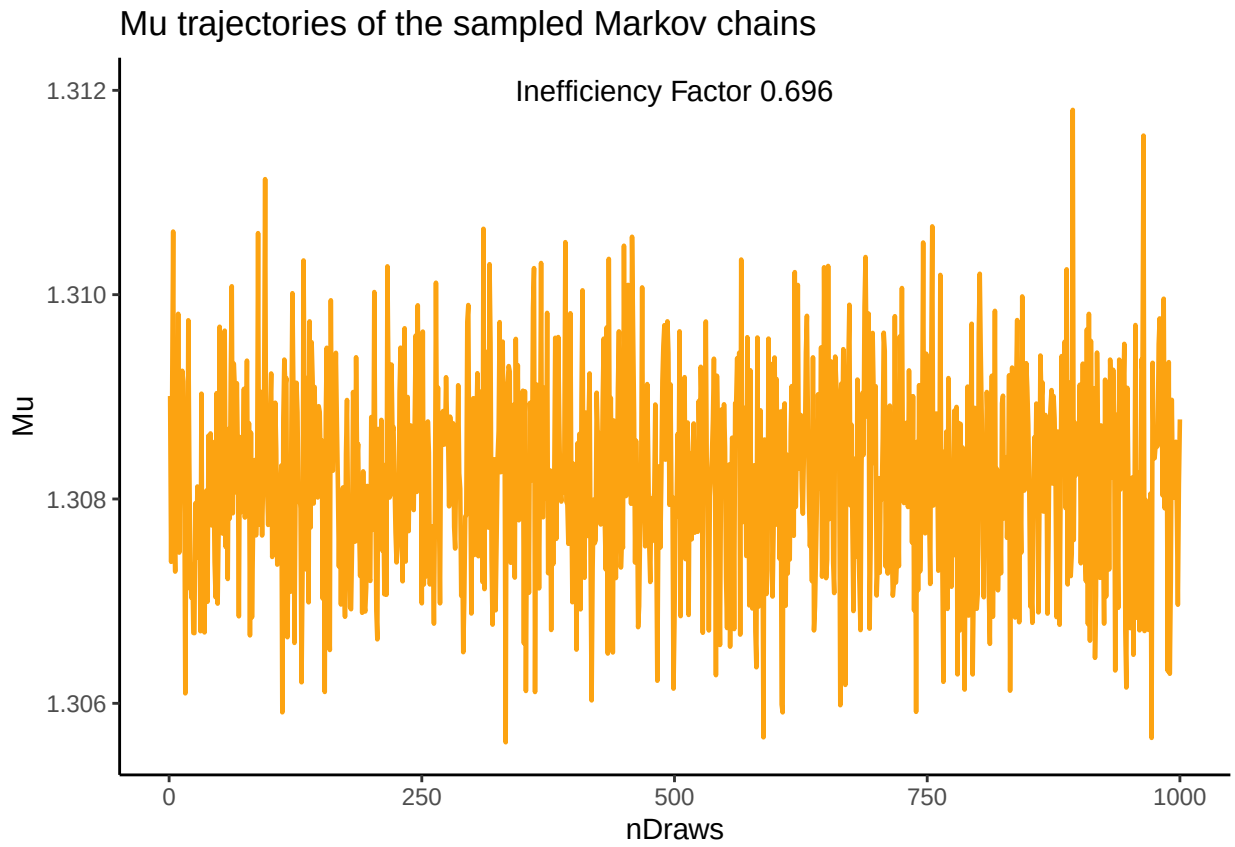
```
  gibbsDraws[i,1] <- theta1

  scale<- ((v_0*sigma2_0)+(sum((df$logx-theta1)^2)))/(n+v_0)
  # Update theta2 ----> sigma2 given theta1
  theta2 <- rinvchisq(1,v_n,scale=scale)
  gibbsDraws[i,2] <- theta2
}
```

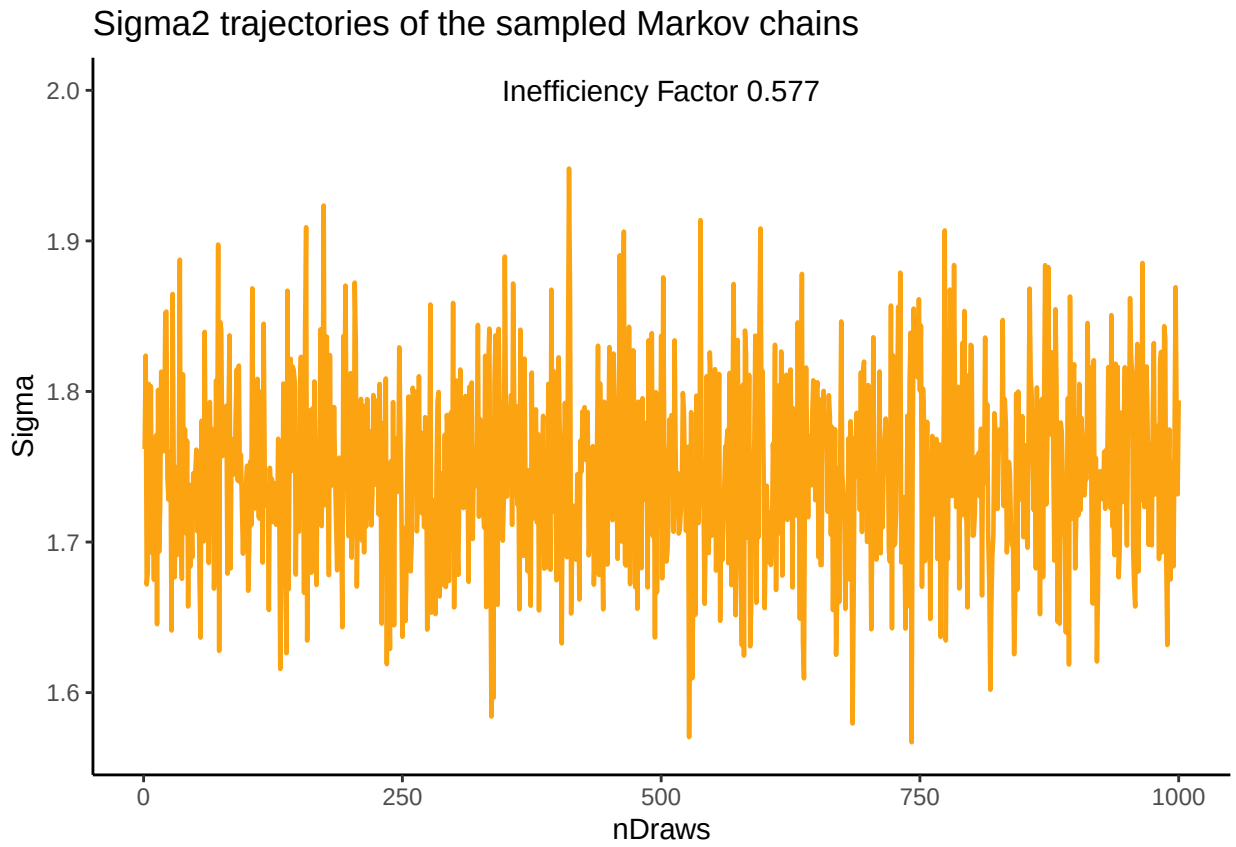
To Evaluate the convergence of the Gibbs sampler we calculate the Inefficiency Factors (IFs) $1 + 2\sum_{k=1}^{\infty} \rho_k$ - where ρ_k is the autocorrelation at lag k - and then we plot the trajectories of the sampled Markov chains.






```
## Warning: Using ‘size‘ aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use ‘linewidth‘ instead.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```


Mu trajectories of the sampled Markov chains
Inefficiency Factor 0.696
1.312
1.310
1.308
1.306
Mu
0
250
500
750
1000
nDraws

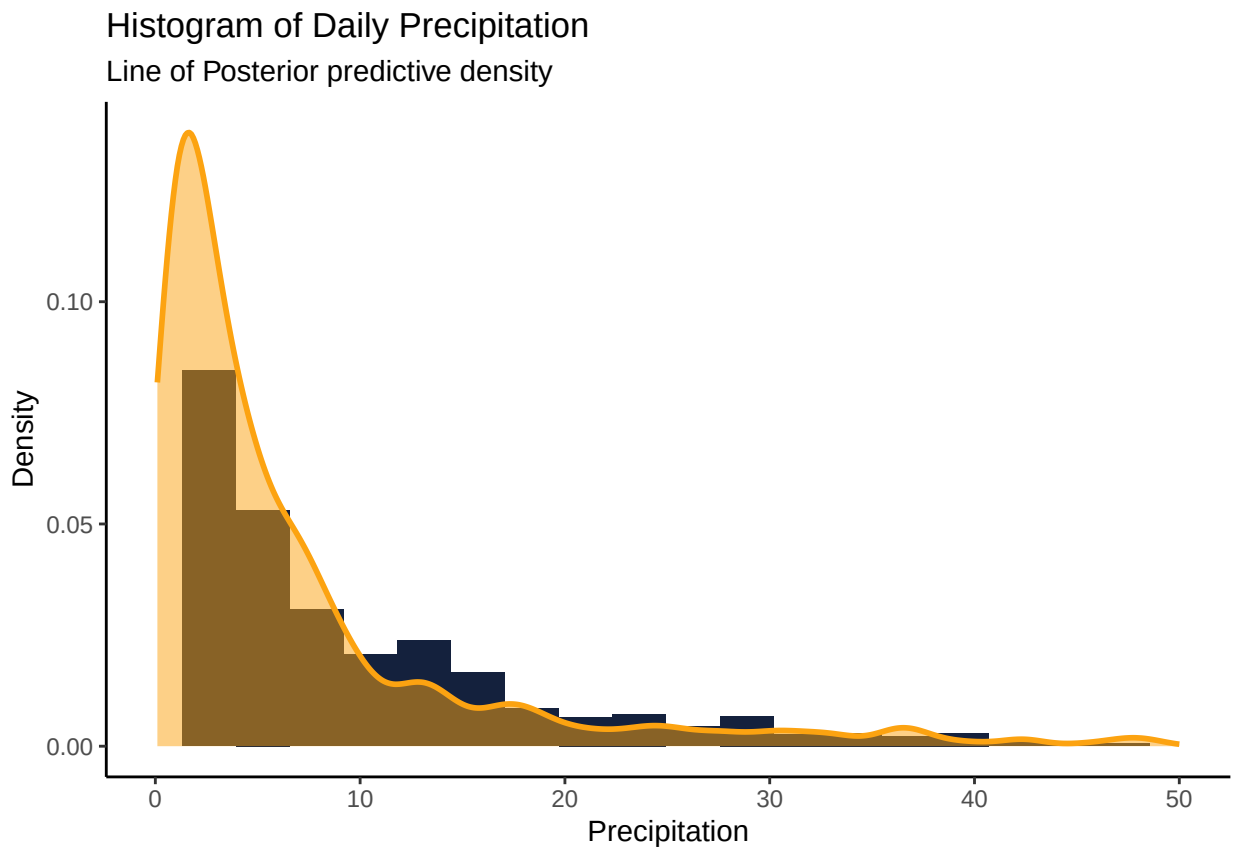


The graph above shows the convergence of the Gibbs sampler and the Inefficiency Factors, the IF tell us number of iterations needed to get an independent sample from the posterior distribution, as we can see μ and σ^2 have 0,69 and 0,57 IF respectively, which tell us a better the convergence of the Gibbs sampler for the σ^2.

B

Plot the following in one figure:
1) a histogram or kernel density estimate of the daily precipitation $(y_1, ..., y_n)$.
2)The resulting posterior predictive density $p(\tilde{y}|y_1, ..., y_n)$ using the simulated posterior draws from (a).
How well does the posterior predictive density agree with this data?

To do we draw a sample from rnorm with mean and sd from the posterior results we got in (a).



Metropolis Random Walk for Poisson regression

A

Obtain the maximum likelihood estimator of β in the Poisson regression model for the eBay data [Hint: glm.R, don't forget that glm() adds its own intercept so don't input the covariate Const].
Which covariates are significant?

```
##
## Call:
## glm(formula = nBids ~ ., family = poisson, data = df1[, -2])
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.5800  -0.7222  -0.0441   0.5269   2.4605
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.07244    0.03077  34.848  < 2e-16 ***
## PowerSeller -0.02054    0.03678  -0.558   0.5765
## VerifyID    -0.39452    0.09243  -4.268 1.97e-05 ***
## Sealed       0.44384    0.05056   8.778  < 2e-16 ***
## Minblem     -0.05220    0.06020  -0.867   0.3859
```

```
## MajBlem     -0.22087    0.09144  -2.416   0.0157 *
## LargNeg      0.07067    0.05633   1.255   0.2096
## LogBook     -0.12068    0.02896  -4.166 3.09e-05 ***
## MinBidShare -1.89410    0.07124 -26.588  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 2151.28  on 999  degrees of freedom
## Residual deviance:  867.47  on 991  degrees of freedom
## AIC: 3610.3
##
## Number of Fisher Scoring iterations: 5
```

The model reuslts shows the coefficients and their standard errors for each predictor. The "Estimate" shows the estimated effect of each predictor on the expected number of bids. A positive coefficient indicates that the predictor is associated with an increase in the expected number of bids, while a negative coefficient indicates a decrease.
The "Pr(>|z|)" column shows the p-value for each coefficient. If this value is less than 0.05, the coefficient is considered statistically significant.
The model suggests that the variables *VerifyID, Sealed, LogBook, and MinBidShare* have a significant effect on the expected number of bids (p-value less that 0.01 (p<.01) at $\alpha = 0.01$), while the others do not.

B

Let's do a Bayesian analysis of the Poisson regression. Let the prior be $\beta \sim N(0, 100\ (X^T X)^{-1})$,where X is the n x p covariate matrix. This is a commonly used prior, which is called Zellner's g-prior. Assume frst that the posterior density is approximately multivariate normal:

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

where $\tilde{\beta}$ is the posterior mode and $J_y^{-1}(\tilde{\beta})$ is the negative Hessian at the posterior mode. $\tilde{\beta}$ and $J_y^{-1}(\tilde{\beta})$ can be obtained by numerical optimization (optim.R) exactly like you already did for the logistic regression in Lab 2 (but with the log posterior function replaced by the corresponding one for the Poisson model, which you have to code up.).

```
## [1] "The posterior mode is:"
```

```
##          [,1]        [,2]      [,3]      [,4]        [,5]       [,6]       [,7]
## [1,] 1.069841 -0.02051246 -0.393006 0.4435555 -0.05246627 -0.2212384 0.07069683
##            [,8]      [,9]
## [1,] -0.1202177 -1.891985
## attr(,"names")
## [1] "Const"       "PowerSeller" "VerifyID"    "Sealed"      "Minblem"
## [6] "MajBlem"     "LargNeg"     "LogBook"     "MinBidShare"
```

```
## [1] "The Hessian Matrix:"
```

```
##             [,1]        [,2]          [,3]          [,4]          [,5]
##  [1,] -3634.2841 -1574.88862 -1.284330e+02 -5.054825e+02 -3.089573e+02
##  [2,] -1574.8886 -1574.88862 -6.049186e+01 -3.260764e+02 -1.044615e+02
##  [3,]  -128.4330   -60.49186 -1.284330e+02 -6.148277e+01 -9.865299e+00
##  [4,]  -505.4825  -326.07643 -6.148277e+01 -5.054825e+02  1.705303e-07
##  [5,]  -308.9573  -104.46148 -9.865299e+00  1.705303e-07 -3.089573e+02
##  [6,]  -126.9303   -68.96966  2.273737e-07  0.000000e+00  0.000000e+00
##  [7,]  -385.7170   -53.05278 -1.136868e-07  0.000000e+00 -3.390566e+01
##  [8,]  -638.9730    71.79073 -6.887776e+01 -1.287304e+02 -2.229706e+01
##  [9,]   729.8896   146.21556  2.380017e+01  8.975677e+01  5.518223e+01
##                [,6]          [,7]        [,8]       [,9]
##  [1,] -1.269303e+02 -3.857170e+02  -638.97297  729.88956
##  [2,] -6.896966e+01 -5.305278e+01    71.79073  146.21556
##  [3,]  2.273737e-07 -1.136868e-07   -68.87776   23.80017
##  [4,]  0.000000e+00  0.000000e+00  -128.73043   89.75677
##  [5,]  0.000000e+00 -3.390566e+01   -22.29706   55.18223
##  [6,] -1.269303e+02  0.000000e+00   -36.39914   34.16904
##  [7,]  0.000000e+00 -3.857170e+02  -220.23559  115.38523
##  [8,] -3.639914e+01 -2.202356e+02 -1930.07936  534.16381
##  [9,]  3.416904e+01  1.153852e+02   534.16381 -446.88731

## [1] "The approximate posterior standard deviation is:"

##       Const PowerSeller    VerifyID      Sealed     Minblem     MajBlem
##  0.03074837  0.03678418  0.09227871  0.05057448  0.06020470  0.09146070
##     LargNeg     LogBook MinBidShare
##  0.05634767  0.02895635  0.07109682
```

C

Random walk Metropolis algorithm:

- Initialize θ^0 and iterate for i= 1,2,..
  1- Sample proposal: $\theta_p|\theta^{(i-1)} \sim N(\theta^{(i-1)}, c.\Sigma)$.
  2- Compute the acceptance probability ($\alpha = min(1, \frac{p(\theta_p|y)}{\theta^{(i-1)}|y)})$).
  3- With probability α true set $\theta^i = \theta_p$ and $\theta^i = \theta^{i-1}$.

Note

The draw backs of the MH usual find it hard to find proposal distribution and getting too small step size give us too many rejections.
Alternately we can use HM (Hamiltonian Monto Carlo), which solve this by giving distant proposal and high acceptance probabilities.

Let's simulate from the actual posterior of β using the Metropolis algorithm and compare the results with the approximate results in b).
Program a general function that uses the Metropolis algorithm to generate random draws from an arbitrary posterior density. In order to show that it is a general function for any model, we denote the vector of model parameters by θ. Let the proposal density be the multivariate normal density mentioned in Lecture

8 (random walk Metropolis):

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

Where $\Sigma = J_y^{-1}(\tilde{\beta})$ was obtained in b). The value c is a tuning parameter and should be an input to your Metropolis function. The user of your Metropolis function should be able to supply her own posterior density function, not necessarily for the Poisson regression, and still be able to use your Metropolis function.
* Note that* on how to How to code up the Random Walk Metropolis Algorithm in R:
One of the input arguments of our *RWMSampler* function is *logPostFunc*. *logPostFunc* is a function object that computes the log posterior density at any value of the parameter vector.
This is needed when we compute the acceptance probability of the Metropolis algorithm. from (q2 a) we program the log posterior density, since logs are more stable and avoids problems with too small or large numbers (overflow).
The ratio of posterior densities in the Metropolis acceptance probability can be written as:

$$\frac{p(\theta_p|y)}{p(\theta^{i-1}|y)} = exp(log\ p(\theta_p|y) - log\ p(\theta^{i-1}|y))$$

The first argument our (log) posterior function is theta (theta_old,theta_new), the vector of parameters for which the posterior density is evaluated. You can of course use some other name for the variable, but it must be the first argument of your posterior density function.

Function RWMSampler

```
RWMSampler_old<- function(logPostFunc,nDraws,c,y,x,mu,Sigma){
  # First we buld our data frame of samples
  sample <- data.frame(matrix(nrow = nDraws, ncol = ncol(x)))
  colnames(sample) <- colnames(x)
  # The inital sample value c here represent a tuning parameter
  sample[1,] <- mvrnorm(1, posteriorMode, c*postCov)
  # Now we implement the Metropolis-Hastings
  #in which we generate samples from the proposal distribution in this case
  # We look at the results of the first sample
  #as theta_i-1 pluged in mvnorm to get theta_i and the we use the values in
  # our proposed logPostFunc
  counter <- 1
  i=1
  while (counter < nDraws) {
    theta_old<-as.numeric(sample[counter,])
    theta_new<-mvrnorm(1,theta_old,c*postCov)
  # We define th our accept/reject threshold
    th<-runif(1,0,1)
  # now we find the value of the target/proposed distribution
    proposed<- logPostFunc(theta_new,y = y,
                           x = x,
                           mu = posteriorMode,
                           Sigma = postCov)
    target<- logPostFunc(theta_old,y = y,
                         x = x,
                         mu = posteriorMode,
```

```
                        Sigma = postCov)
    # the ratio of posterior densities in the Metropolis acceptance probability
    if (th<min(1,exp(proposed-target))) {
      counter=counter+1
      sample[counter,]<-theta_new
    }
  }
 return(sample)
}
```

Old solution

```
RWMSampler<- function(logPostFunc,nDraws,c,y,x,mu,Sigma,brnin){
  # First we buld our data frame of samples
  sample <- data.frame(matrix(0,nrow = nDraws - brnin, ncol = ncol(x)))
  colnames(sample) <- colnames(x)
  # The inital sample value c here represent a tuning parameter
  # sample[1,] <- mvrnorm(1, posteriorMode, c*postCov)
  # Now we implement the Metropolis-Hastings
  #in which we generate samples from the proposal distribution in this case
  # We look at the results of the first sample
  #as theta_i-1 pluged in mvnorm to get theta_i and the we use the values in
  # our proposed logPostFunc

  theta_old<-as.numeric(sample[1,])
  for (i in 1:nDraws) {

    theta_new<-mvrnorm(1,theta_old,c*postCov)
    # We define th our accept/reject threshold
    th<-runif(1,0,1)
    # now we find the value of the target/proposed distribution
    proposed<- logPostFunc(theta_new,y = y,
                           x = x,
                           mu = posteriorMode,
                           Sigma = postCov)
    target<- logPostFunc(theta_old,y = y,
                         x = x,
                         mu = posteriorMode,
                         Sigma = postCov)
    # the ratio of posterior densities in the Metropolis acceptance probability
    if (th<exp(proposed-target)) {
      theta_old<-theta_new
    }
    if (i> brnin) {
      sample[i-brnin,]<-theta_old
    }
  }
  return(sample)
}
```

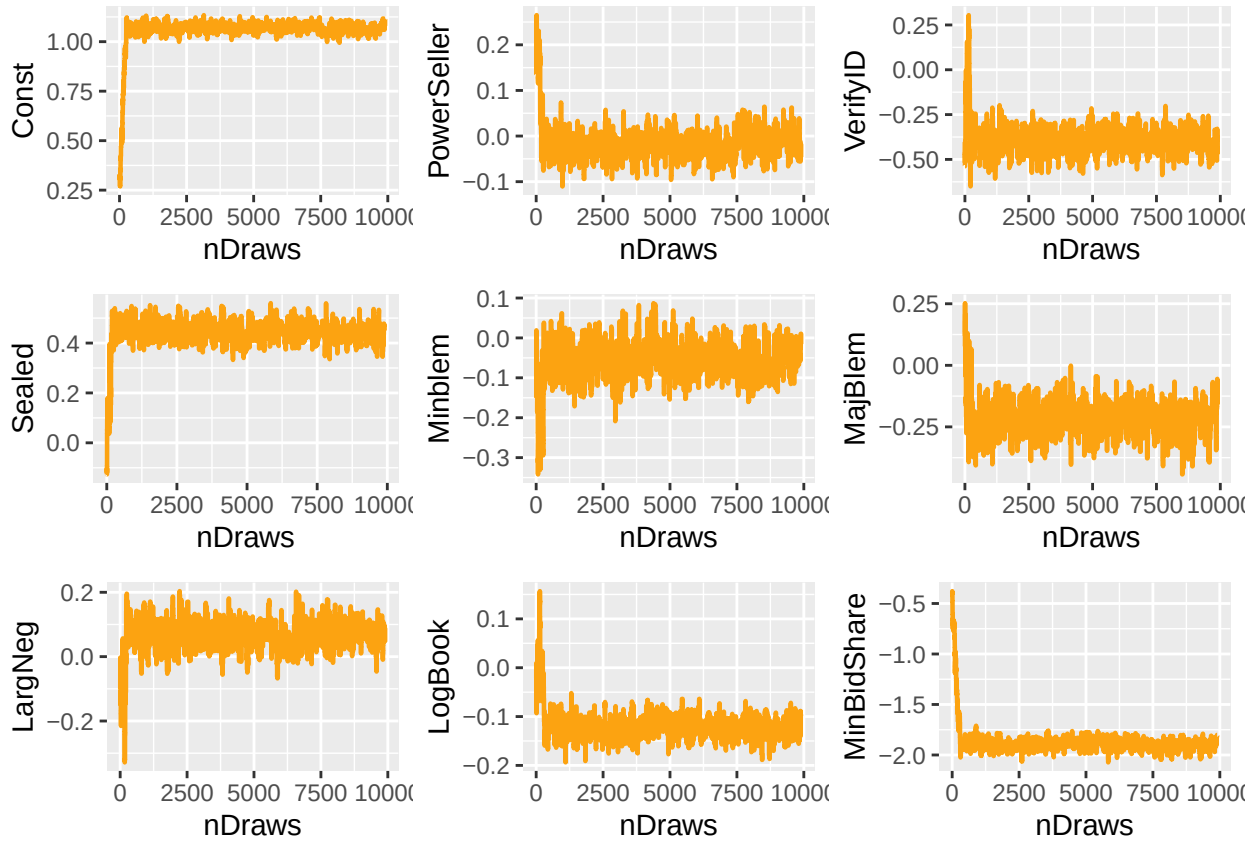
New Solution

- Now, use your new Metropolis function to sample from the posterior of β in the Poisson regression for the eBay dataset.

```
nDraws=10000
c=.5
brnin=100 ##adding this new parameter for buirnin interval
posteriorMode=OptimRes$par
postCov=solve(-OptimRes$hessian)
res <- RWMSampler(logPostFunc = logPossion,
                                  nDraws = nDraws,
                                    c=c,
                                    y = y,
                                    x = x,
                                    mu = posteriorMode,
                                    Sigma = postCov,
                                    brnin = brnin )
```

- Assess MCMC convergence by graphical methods.

```
## Warning: ‘aes_string()‘ was deprecated in ggplot2 3.0.0.
## i Please use tidy evaluation idioms with ‘aes()‘.
## i See also ‘vignette("ggplot2-in-packages")‘ for more information.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```

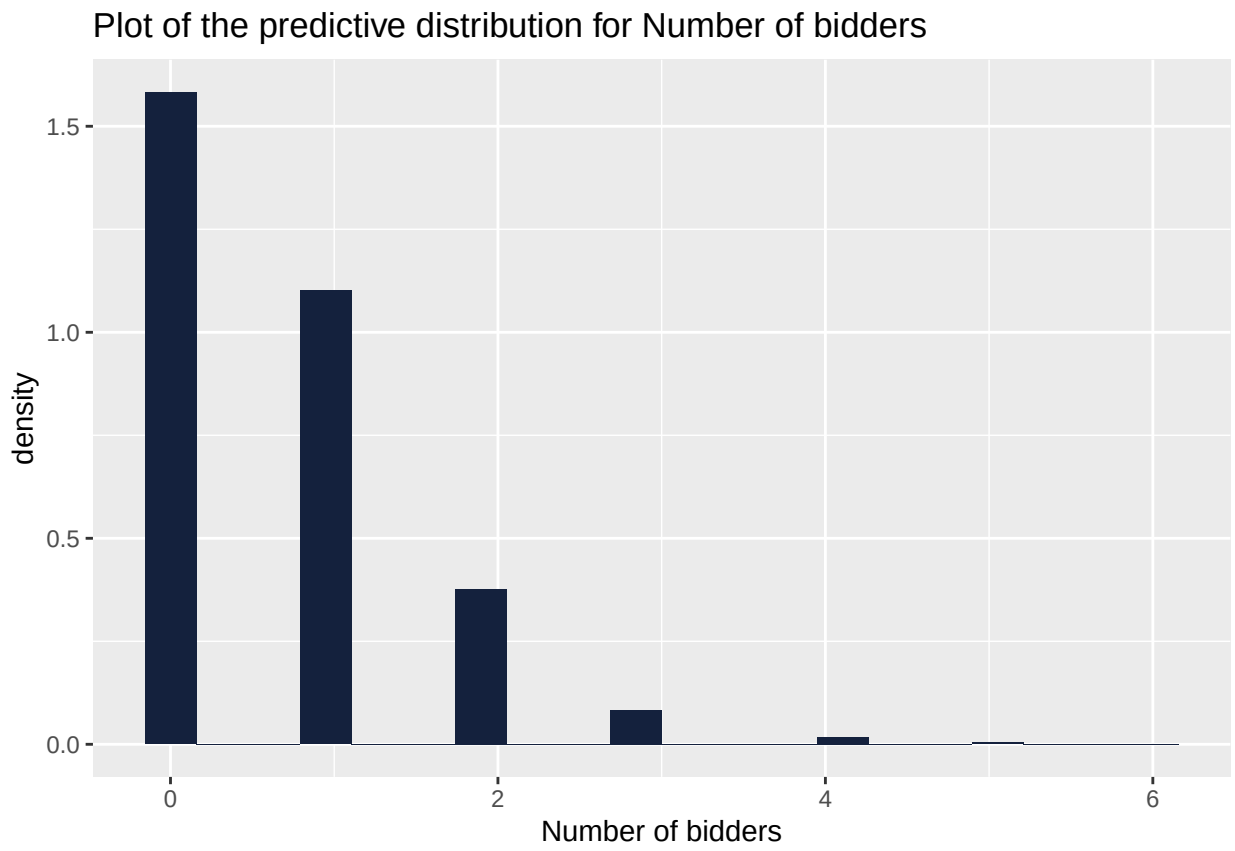


D

Use the MCMC draws from c) to simulate from the predictive distribution of the number of bidders in a new auction with the characteristics below. Plot the predictive distribution. What is the probability of no bidders in this new auction?
* PowerSeller = 1
* VerifyID = 0
* Sealed = 1
* MinBlem = 0
* MajBlem = 1
* LargNeg = 0
* LogBook = 1.2
* MinBidShare = 0.8

```
#First we estimate the betas from our RWMSampler function
betas<- as.matrix(res)
# Input data
x_new <- as.matrix(c(1,1,0,1,0,1,0,1.2,0.8))
##### This code has been modified
# calculating the bet for Poisson since we have our y follow poisson
beta_pois<-exp(betas  %*% x_new)
# Finding number of bidders using poisson function
nbiders<-data.frame(x=rpois(length(beta_pois),beta_pois))
```

Plotting the predictive distribution. What is the probability of no bidders in this new auction?



Time series models in Stan

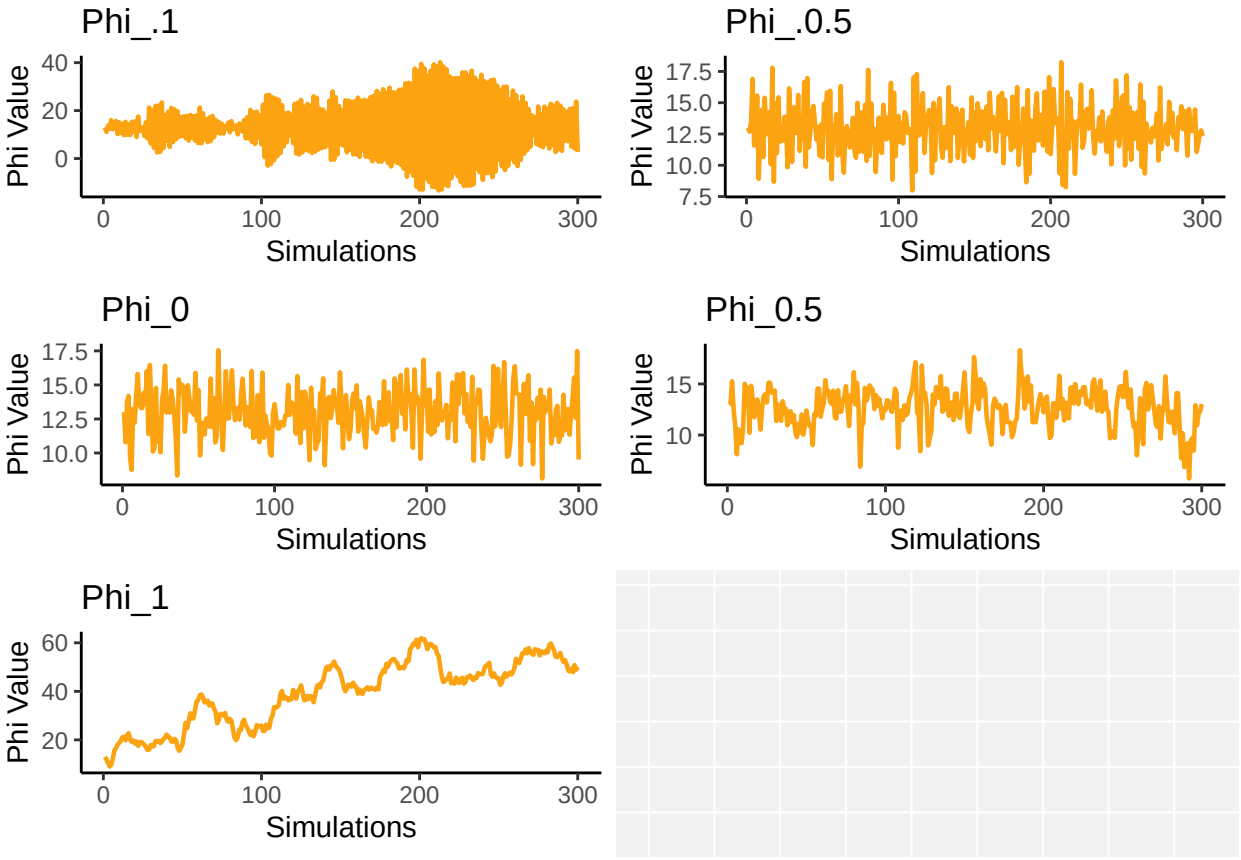
A

Write a function in R that simulates data from the AR(1)-process:
$x_t = \mu + \phi(x_{t-1} - \mu) + \epsilon_t$, $\epsilon_t \sim N(0, +\sigma^2)$ for given values of μ, ϕ and σ^2.
Start the process at x1 = μ and then simulate values for x_t for t= 1,2,3,. . . , T and return the vector x1:T containing all time points.
Use μ= 13; $\sigma^2 = 3$ and T = 300 and look at some different realizations (simulations) of $x_{1:t}$ for values of ϕ between -1 and 1 (this is the interval of ϕ where the AR(1)-process is stationary). Include a plot of at least one realization in the report.
What effect does the value of ϕ have on $x_{1:t}$?

```
ar_process<- function(phi,mu, sigma,t){
  x_t<-c()
  x_t[1]<-mu
  for(i in 2:t){
    e<- rnorm(1,0,sqrt(3))
    x_t[i]<-mu+(phi*(x_t[i-1]-mu))+e
  }
  return(x_t)
}
```

Now we Include a plot of at least one realization in the report. What effect does the value of ϕ have on $x_{1:t}$?

```
phi<-seq(-1,1,by=.5)
res<- list()
for (i in phi) {
  res1 <- ar_process(i, 13, 3, 300)
  res[[paste0("Phi_", i)]] <- res1
}

res<- data.frame(res)
names<-colnames(res)
p_fun<- function(coln){
  plt <- ggplot(res,aes(x = 1:300)) +
    geom_line(aes_string(y = coln),color='#FCA311', size=.8)+
  labs(title = coln,
       x= 'Simulations', y='Phi Value')+ theme_classic()
  plt
}

plot(arrangeGrob(grobs = lapply(names, p_fun)))
```



In the graph above, the value of phi greatly impacts the convergence of the sample chain. When phi is less than 0, it can lead to instability and hinder the convergence to a stationary distribution. This instability is evident when the chains appear to wander around or exhibit erratic behavior, indicating that the sampler has not yet converged.
However, it is notable that the chains demonstrate good mixing, meaning they efficiently explore the parameter space and cover a wide range of plausible values.
To achieve convergence, it may be necessary to increase the number of iterations.
It is important to note that when phi is equal to 1, the convergence appears to be reached more quickly compared to when phi is less than 0.

B

Use your function from a) to simulate two AR(1)-processes, $x_{1:T}$ with $\phi = 0.2$ and $y_{1:T}$ with $\phi = 0.95$. Now, treat your simulated vectors as synthetic data, and treat the values of , μ and σ^2 as unknown parameters. Implement Stancode that samples from the posterior of the three parameters, using suitable non-informative priors of your choice. *Notes from: (https://mc-stan.org/docs/stan-users-guide/autoregressive.html).*
We have A first-order autoregressive model (AR(1)) with normal noise takes each point y_n in a sequence y to be generated according to:

$$y_n \sim N(\alpha + \beta y_{n-i}, \sigma)$$

That is, the expected value of y_n is $\alpha + \beta y_{n-i}$, with noise scaled as σ.
With improper flat priors on the regression coefficients α and β and on the positively-constrained noise scale (σ), the Stan program for the AR(1) model is as follows:

```
data {
  int<lower=0> N;
```

```
  vector[N] y;
}
parameters {
  real alpha;
  real beta;
  real<lower=0> sigma;
}
model {
  for (n in 2:N) {
    y[n] ~ normal(alpha + beta * y[n-1], sigma);
  }
}
```

Adding the the Slicing for efficiency in which we silce the vector we can rewrite the above as:

```
data {
  int<lower=0> N;
  vector[N] y;
}
parameters {
  real alpha;
  real beta;
  real<lower=0> sigma;
}
model {
   y[2:N] ~ normal(alpha + beta * y[1:(N - 1)], sigma);
}
```

From lec notes we have Gibbs sampling for AR process:

If we have AP(p) process:

$$x_t = \mu + \phi_1(x_{t-1} - \mu) + ... + \phi_p(x_{t-p} - \mu) + \epsilon,\ \epsilon_t \sim N(0, \sigma^2)$$

let $\phi = (\phi_1, ..., \phi_p)'$, we have prior:

$$\begin{aligned} \mu &\sim Normal \\ \phi &\sim Multivariate\ Normal \\ \sigma^2 &\sim Scaled\ inverse\ \chi \end{aligned}$$

And the posterior can be simulated by Gibbs sampling:

$$\begin{aligned} \mu|\phi\ \sigma^2 &\sim Normal \\ \phi|\mu\ \sigma^2 &\sim Multivariate\ Normal \\ \sigma^2|\mu\ \phi &\sim Scaled\ inverse\ \chi \end{aligned}$$

StanModel

```
##### Example from notes #####
# library(rstan)
# y=c(4,5,6,4,0,2,5,3,8,6,10,8)
# N=length(y)
#
# StanModel = '
# data {
#   int<lower=0> N; // Number of observations
#   int<lower=0> y[N]; // Number of flowers
# }
# parameters {
#   real mu;
#   real<lower=0> sigma2;
# }
# model {
#   mu ~ normal(0,100); // Normal with mean 0, st.dev. 100
#   sigma2 ~ scaled_inv_chi_square(1,2); // Scaled-inv-chi2 with nu 1,sigma 2
#   for(i in 1:N){
#     y[i] ~ normal(mu,sqrt(sigma2));
#   }
# }'
#
# data <- list(N=N, y=y)
# warmup <- 1000
# niter <- 2000
# fit <- stan(model_code=StanModel,data=data, warmup=warmup,iter=niter,chains=4)
# # Print the fitted model
# print(fit,digits_summary=3)
# # Extract posterior samples
# postDraws <- extract(fit)
# # Do traceplots of the first chain
# par(mfrow = c(1,1))
# plot(postDraws$mu[1:(niter-warmup)],type="l",ylab="mu",main="Traceplot")
# # Do automatic traceplots of all chains
# traceplot(fit)
# # Bivariate posterior plots
# pairs(fit)
# ############################
StanModel = '
data {
  int<lower=0> N; // Number of observations
  vector[N] y;
}
parameters {
  real mu;
  real<lower=0> sigma2;
  real<lower=-1, upper=1> phi;
  //To enforce the estimation of a
  //stationary AR(1) process, the slope
  //coefficient beta may be constrained with bounds as follows.
}
model {
  mu ~ normal(0,100); // Normal with mean 0, st.dev. 100
```

```
  sigma2 ~ scaled_inv_chi_square(1,2); // Scaled-inv-chi2 with nu 1,sigma 2
  // Changing the model to be y_i-mu
  y[2:N] ~ normal(mu + phi * (y[1:(N - 1)]-mu), sqrt(sigma2^2));
}'
```

Model results

Part i Report the posterior mean, 95% credible intervals and the number of effective posterior samples for the three inferred parameters for each of the simulated AR(1)-process. Are you able to estimate the true values?

Giving $x_{1:T}$ with $\phi = 0.2$ and $y_{1:T}$ with $\phi = 0.95$, the model results can be shown as below.

1- *For $x_{1:T}$ with $\phi = 0.2$*

```
# Simulate of x
ar_x<-ar_process(.2, 13, 3, 300)

#From lec notes we have the stanmodel function defined as
y=ar_x
N=length(y)

data <- list(N=N, y=y)
warmup <- 1000
niter <- 2000
fitx <- stan(model_code=StanModel,data=data,warmup=warmup,iter=niter,chains=4)
```

```
##
## SAMPLING FOR MODEL '8874c61302b41c9e7aa22f87c38effdb' 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: 0.158 seconds (Warm-up)
## Chain 1:                0.065 seconds (Sampling)
## Chain 1:                0.223 seconds (Total)
```

```
## Chain 1:
##
## SAMPLING FOR MODEL '8874c61302b41c9e7aa22f87c38effdb' 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: 0.287 seconds (Warm-up)
## Chain 2:                0.106 seconds (Sampling)
## Chain 2:                0.393 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL '8874c61302b41c9e7aa22f87c38effdb' 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: 0.082 seconds (Warm-up)
## Chain 3:                0.093 seconds (Sampling)
## Chain 3:                0.175 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL '8874c61302b41c9e7aa22f87c38effdb' 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: 0.085 seconds (Warm-up)
## Chain 4:                0.072 seconds (Sampling)
## Chain 4:                0.157 seconds (Total)
## Chain 4:
```

```
# Print the fitted model
print(fitx,digits_summary=3)
```

```
## Inference for Stan model: 8874c61302b41c9e7aa22f87c38effdb.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##            mean se_mean    sd     2.5%      25%      50%      75%    97.5%
## mu       13.137   0.002 0.111   12.917   13.064   13.137   13.210   13.356
## sigma2    1.563   0.001 0.064    1.443    1.518    1.560    1.605    1.692
## phi       0.204   0.001 0.056    0.094    0.166    0.204    0.241    0.311
## lp__   -284.604   0.027 1.209 -287.783 -285.164 -284.296 -283.733 -283.249
##        n_eff  Rhat
## mu      3594 1.000
## sigma2  4056 0.999
## phi     4246 1.000
## lp__    2058 1.000
##
## Samples were drawn using NUTS(diag_e) at Sun May 28 20:39:57 2023.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```
samples_x <-extract(fitx, pars = c("mu", "sigma2", "phi"))
# Compute the mean
mean_x <- sapply(samples_x, mean)
# 95% credible intervals
intv_x <- sapply(samples_x,function(samples) quantile(samples,c(0.025, 0.975)))
# effective posterior samples
eff_samp_x <- summary(fitx)$summary[1:3,"n_eff"]
```

```r
result <- data.frame( Mean = mean_x,
                      Credible_Interval_Lower = intv_x[1, ],
                      Credible_Interval_Upper = intv_x[2, ],
                      Effective_Samples = eff_samp_x)
print(result)
```

```
##              Mean Credible_Interval_Lower Credible_Interval_Upper
## mu     13.1371909             12.91693642              13.3561165
## sigma2  1.5625755              1.44261786               1.6921687
## phi     0.2035987              0.09371789               0.3106965
##        Effective_Samples
## mu              3593.851
## sigma2          4055.560
## phi             4246.170
```

Based on the table above, the simulations performed using the Metropolis algorithm have provided estimates for the parameters μ, σ, and ϕ.
The estimated values are found to be approximately 11.6, 1.6, and 0.11, respectively.
These estimates are close to the true values used to simulate the AR(1) process, indicating that the sampling algorithm has been not effective in capturing the underlying parameter values. This suggests that the simulation has not successfully captured the characteristics of the data and has produced reliable estimates for the parameters of interest.

2- *For $y_{1:T}$ with $\phi = 0.95$*

```r
# Simulate of y
ar_y<-ar_process(.95, 13, 3, 300)
#From lec notes we have the stanmodel function defined as
#y=ar_y
N=length(y)

data <- list(N=N, y=y)
warmup <- 1000
niter <- 2000
fity <- stan(model_code=StanModel,data=data, warmup=warmup,iter=niter,chains=4)
```

```
##
## SAMPLING FOR MODEL '8874c61302b41c9e7aa22f87c38effdb' 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: 0.1 seconds (Warm-up)
## Chain 1:                0.066 seconds (Sampling)
## Chain 1:                0.166 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL '8874c61302b41c9e7aa22f87c38effdb' 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: 0.349 seconds (Warm-up)
## Chain 2:                0.083 seconds (Sampling)
## Chain 2:                0.432 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL '8874c61302b41c9e7aa22f87c38effdb' 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: 0.104 seconds (Warm-up)
## Chain 3:                0.076 seconds (Sampling)
## Chain 3:                0.18 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL '8874c61302b41c9e7aa22f87c38effdb' 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: 0.179 seconds (Warm-up)
## Chain 4:                0.065 seconds (Sampling)
## Chain 4:                0.244 seconds (Total)
## Chain 4:
```

```
# Print the fitted model
print(fity,digits_summary=3)
```

```
## Inference for Stan model: 8874c61302b41c9e7aa22f87c38effdb.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##            mean se_mean    sd     2.5%      25%      50%      75%    97.5%
## mu       13.137   0.002 0.114   12.920   13.060   13.139   13.213   13.360
## sigma2    1.565   0.001 0.065    1.449    1.519    1.563    1.608    1.703
## phi       0.200   0.001 0.057    0.093    0.160    0.199    0.238    0.314
## lp__   -284.672   0.027 1.235 -287.725 -285.222 -284.368 -283.762 -283.242
##        n_eff  Rhat
## mu      3853 1.001
## sigma2  4391 0.999
## phi     4325 1.000
## lp__    2037 1.001
##
## Samples were drawn using NUTS(diag_e) at Sun May 28 20:39:59 2023.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```r
samples_y <-extract(fity, pars = c("mu", "sigma2", "phi"))
# Compute the mean
mean_y <- sapply(samples_y, mean)
# 95% credible intervals
intv_y <- sapply(samples_y,function(samples) quantile(samples,c(0.025, 0.975)))
# effective posterior samples
eff_samp_y <- summary(fitx)$summary[1:3,"n_eff"]

result <- data.frame( Mean = mean_y,
                      Credible_Interval_Lower = intv_y[1, ],
                      Credible_Interval_Upper = intv_y[2, ],
                      Effective_Samples = eff_samp_y)
print(result)
```

```
##              Mean Credible_Interval_Lower Credible_Interval_Upper
## mu     13.1370383             12.91954935              13.3604494
## sigma2  1.5653937              1.44876753               1.7027939
## phi     0.2002713              0.09332073               0.3138576
##        Effective_Samples
## mu              3593.851
## sigma2          4055.560
## phi             4246.170
```

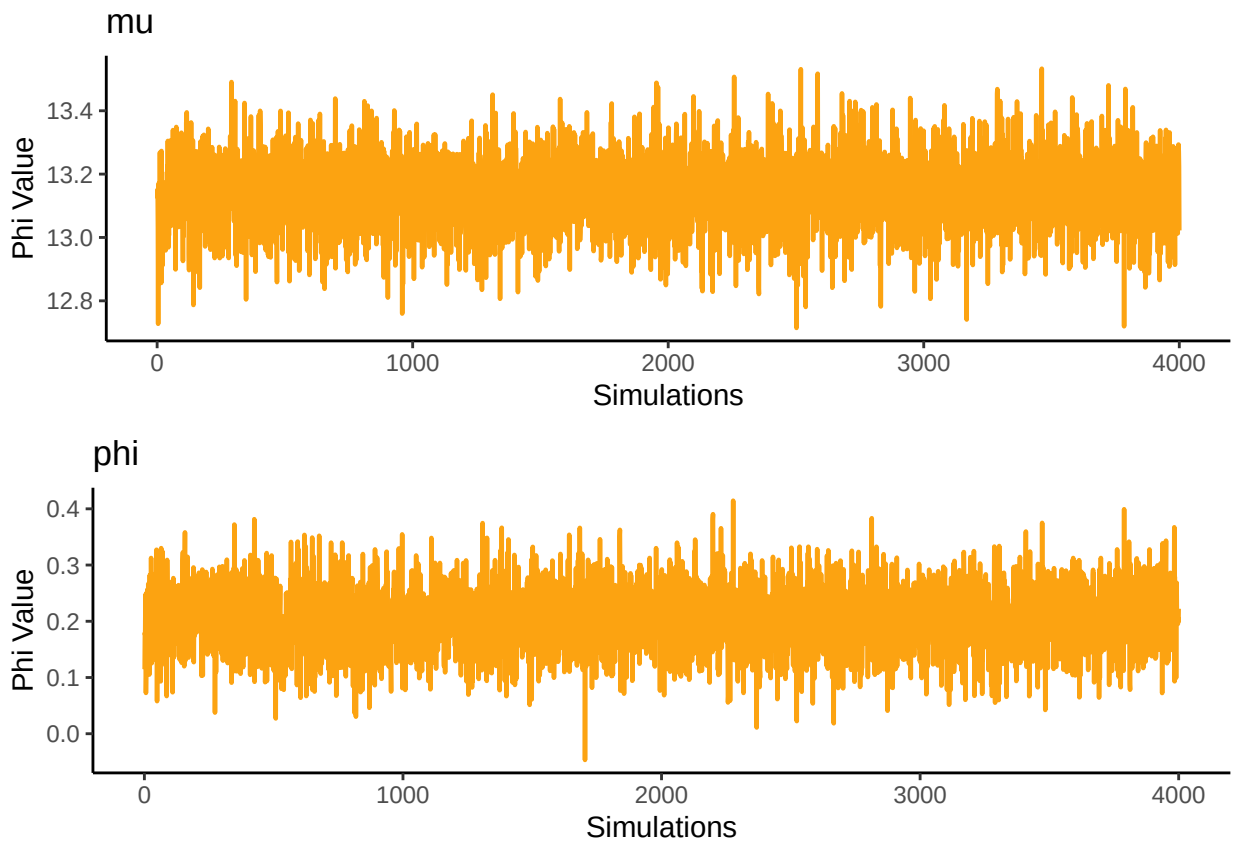
In the table above, it is observed that the second simulated AR(1) process exhibits some differences compared to the first one. While the estimates for σ and ϕ are relatively accurate, the estimate for μ appears to deviate slightly from the true value. This discrepancy suggests that the sampling algorithm may have encountered challenges in accurately capturing the true mean parameter.

There could be several reasons for this discrepancy. It is possible that the non-informative priors chosen for the parameters might not have been appropriate, leading to a bias in the estimation of μ. Additionally, the specific characteristics of the second AR(1) process, such as its underlying dynamics and data distribution, might have posed challenges for the sampling algorithm in accurately estimating μ.

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

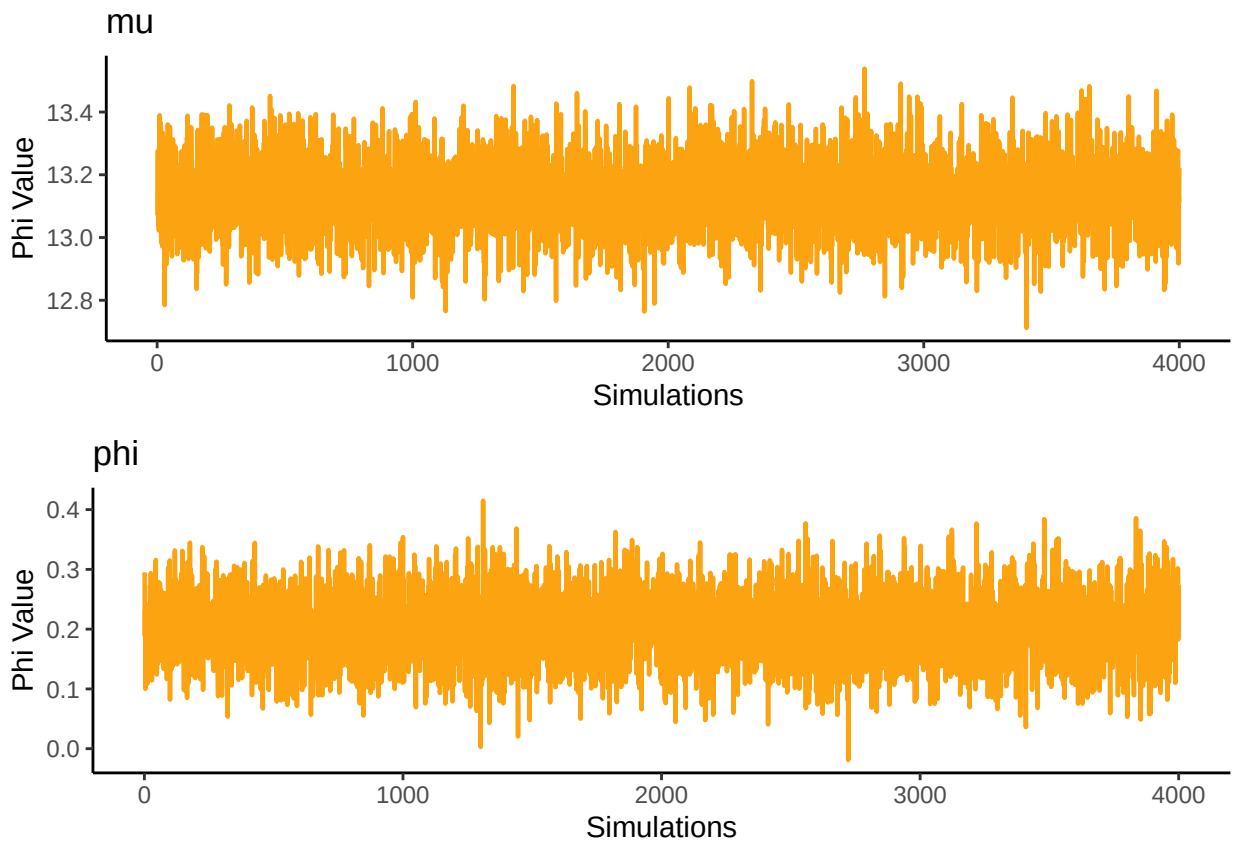
First sample: *For $x_{1:T}$ with $\phi = 0.2$*

```r
df_x <-as.data.frame(extract(fitx, pars = c("mu", "phi")))
names<-colnames(df_x)
ln<-length(df_x[,1])
p_fun<- function(coln){
  plt <- ggplot(df_x,aes(x = 1:ln)) +
    geom_line(aes_string(y = coln),color='#FCA311', size=.8)+
    labs(title = coln,
         x= 'Simulations', y='Phi Value')+ theme_classic()
  plt
}

plot(arrangeGrob(grobs = lapply(names, p_fun)))
```



Second sample: *For $y_{1:T}$ with $\phi = 0.95$*

```
df_y <-as.data.frame(extract(fity, pars = c("mu", "phi")))
names<-colnames(df_y)
ln<-length(df_y[,1])
p_fun<- function(coln){
  plt <- ggplot(df_y,aes(x = 1:ln)) +
    geom_line(aes_string(y = coln),color='#FCA311', size=.8)+
    labs(title = coln,
         x= 'Simulations', y='Phi Value')+ theme_classic()
  plt
}

plot(arrangeGrob(grobs = lapply(names, p_fun)))
```



Examining the above results, it is evident that the posterior estimates for both μ and ϕ exhibit convergence for each of the two data sets. This convergence indicates that the Markov chain has explored the parameter space sufficiently and reached a stable distribution.

References:

1- Stan Model

```
#### In case of normal
# library(rstan)
# y=c(4,5,6,4,0,2,5,3,8,6,10,8)
# N=length(y)
# StanModel = '
# data {
# int<lower=0> N; // Number of observations
# int<lower=0> y[N]; // Number of flowers
# }
# parameters {
# real mu;
# real<lower=0> sigma2;
# }
# model {
# mu ~ normal(0,100); // Normal with mean 0, st.dev. 100
# sigma2 ~ scaled_inv_chi_square(1,2); // Scaled-inv-chi2 with nu 1,sigma 2
```

```
# for(i in 1:N){
# y[i] ~ normal(mu,sqrt(sigma2));
# }
# }'
############ Multilevel normal
# StanModel <- '
# data {
# int<lower=0> N; // Number of observations
# int<lower=0> y[N]; // Number of flowers
# int<lower=0> P; // Number of plants
# }
# transformed data {
# int<lower=0> M; // Number of months
# M = N / P;
# }
# parameters {
# real mu;
# real<lower=0> sigma2;
# real mup[P];
# real sigmap2[P];
# }
# model {
# mu ~ normal(0,100); // Normal with mean 0, st.dev. 100
# sigma2 ~ scaled_inv_chi_square(1,2); // Scaled-inv-chi2 with nu 1, sigma 2
# for(p in 1:P){
# mup[p] ~ normal(mu,sqrt(sigma2));
# for(m in 1:M) {
# y[M*(p-1)+m] ~ normal(mup[p],sqrt(sigmap2[p]));
# }
# }
# }'
###############Possion
# StanModel <- '
# data {
# int<lower=0> N; // Number of observations
# int<lower=0> y[N]; // Number of flowers
# int<lower=0> P; // Number of plants
# }
# transformed data {
# int<lower=0> M; // Number of months
# M = N / P;
# }
# parameters {
# real mu;
# real<lower=0> sigma2;
# real mup[P];
# }
# model {
# mu ~ normal(0,100); // Normal with mean 0, st.dev. 100
# sigma2 ~ scaled_inv_chi_square(1,2); // Scaled-inv-chi2 with nu 1, sigma 2
# for(p in 1:P){
# mup[p] ~ lognormal(mu,sqrt(sigma2)); // Log-normal
# for(m in 1:M) {
```

```
# y[M*(p-1)+m] ~ poisson(mup[p]); // Poisson
# }
# }
# }'
```

1- Example code from lec notes

```
### The original Code:

# Direct vs Gibbs sampling for bivariate normal distribution

# Initial setting
mu1 <- 1
mu2 <- 1
rho <- 0.9
mu <- c(mu1,mu2)
Sigma = matrix(c(1,rho,rho,1),2,2)
nDraws <- 500 # Number of draws

library(MASS) # To access the mvrnorm() function

# Direct sampling from bivariate normal distribution
directDraws <- mvrnorm(nDraws, mu, Sigma)

# Gibbs sampling
gibbsDraws <- matrix(0,nDraws,2)
theta2 <- 0 # Initial value for theta2
for (i in 1:nDraws){

  # Update theta1 given theta2
  theta1 <- rnorm(1, mean = mu1 + rho*(theta2-mu2), sd = sqrt(1-rho**2))
  gibbsDraws[i,1] <- theta1

  # Update theta2 given theta1
  theta2 <- rnorm(1, mean = mu2 + rho*(theta1-mu1), sd = sqrt(1-rho**2))
  gibbsDraws[i,2] <- theta2

}

a_Direct <- acf(directDraws[,1])
a_Gibbs <- acf(gibbsDraws[,1])

IF_Gibbs <- 1+2*sum(a_Gibbs$acf[-1])

par(mfrow=c(2,4))

# DIRECT SAMPLING
plot(1:nDraws, directDraws[,1], type = "l",col="blue") # traceplot of direct draws

hist(directDraws[,1],col="blue") # histogram of direct draws
) # Cumulative mean value of theta1, direct draws
cusumData <- cumsum(directDraws[,1])/seq(1,nDraws
plot(1:nDraws, cusumData, type = "l", col="blue")
```

```
barplot(height = a_Direct$acf[-1],col="blue") # acf for direct draws

# GIBBS SAMPLING
plot(1:nDraws, gibbsDraws[,1], type = "l",col="red") # traceplot of Gibbs draws

hist(gibbsDraws[,1],col="red") # histogram of Gibbs draws
# Cumulative mean value of theta1, Gibbs draws
cusumData =  cumsum(gibbsDraws[,1])/seq(1,nDraws)
plot(1:nDraws, cusumData, type = "l", col="red")

barplot(height = a_Gibbs$acf[-1],col="red") # acf for Gibbs draws

# Plotting the cumulative path of estimates of Pr(theta1>0, theta2>0)
par(mfrow=c(2,1))
plot(cumsum(directDraws[,1]>0 & directDraws[,2]>0)/seq(1,nDraws),type="l",
     main='Direct draws', xlab='Iteration number', ylab='', ylim = c(0,1))
plot(cumsum(gibbsDraws[,1]>0 & gibbsDraws[,2]>0)/seq(1,nDraws),type="l",
     main='Gibbs draws', xlab='Iteration number', ylab='', ylim = c(0,1))


#######################################################
```

2- Stan Development Team. (2021). Stan user's guide. Retrieved from https://mc-stan.org/docs/2_27/stan-users-guide/index.html.
3- Bertil Wegmann (2023). Bayesian Learning [Lecture notes]. 732A73, Department of Computer and Information Science,LiU University.

Code Appendix

```
set.seed(123456)
knitr::opts_chunk$set(echo = TRUE)
library(LaplacesDemon)
library(mvtnorm)
library(MASS)
library(ggplot2)
library(gridExtra)
library(rstan)
# we start by reading our dataset using the readRDS command in R
df <- data.frame(x=readRDS("Precipitation.rds"))
# in the task we looking at natural log of the daily precipitation
#lny1,lny2,lny3,...lnyn follows N(mu,sigma)
# We add a new var called logx
df$logx<- log(df$x)
sigma2_0<- var(df$logx)  # Sample var
tau2_0<- 1# arbirtry initail value let it be 1
n<- nrow(df)# Sample size
mu_0<- mean(df$logx)#sample mean
w<- (n/sigma2_0)/((n/sigma2_0) + (1/tau2_0))# value used to calculate mu_n
v_0<- 1 # arbirtry initail value let it be 1
v_n<- v_0+n
```

```r
mu_n<- w*(mean(df$logx))+ (1-w)*mu_0
 ###### This part of the code has been modified######
tau2_n<- 1/((n/sigma2_0)+(1/tau2_0))################# shoud be 1/tau
#####################################################
nDraws<- 1000
gibbsDraws <- matrix(0,nDraws,2)



### From lec notes we can use this part of the code
  ###### This part of the code has been modified######
scale=sigma2_0 ######################################
for (i in 1:nDraws){#################################
  w<- (n/scale)/((n/scale) + (1/tau2_0)) #############
  mu_n<- w*(mean(df$logx))+ (1-w)*mu_0 ############### We need to update those parameters as well
  tau2_n<- 1/((n/scale)+(1/tau2_0))##################
  #####################################################
  # Update theta1 ----> mu given theta2
  theta1 <- rnorm(1, mean = mu_n, sd = (tau2_n))
  gibbsDraws[i,1] <- theta1

  scale<- ((v_0*sigma2_0)+(sum((df$logx-theta1)^2)))/(n+v_0)
  # Update theta2 ----> sigma2 given theta1
  theta2 <- rinvchisq(1,v_n,scale=scale)
  gibbsDraws[i,2] <- theta2
}
## Finding the acf plots and value
a_Gibbs_mu <- acf(gibbsDraws[,1])
a_Gibbs_sigma <- acf(gibbsDraws[,2])

IF_Gibbs_mu <- 1+2*sum(a_Gibbs_mu$acf[-1])

IF_Gibbs_sigma <- 1+2*sum(a_Gibbs_sigma$acf[-1])

### From lec notes we can use this part of the code
plot_df<- as.data.frame(cbind(1:nDraws,gibbsDraws))

# Plot for Mu trajectories of the sampled Markov chains
p1<-ggplot(plot_df,aes(x=V1))+geom_line(aes(y=V2), color='#FCA311', size=.8)+
  annotate(geom = "text", x = 500, y = 1.312,
           label = paste0("Inefficiency Factor ",format(round(IF_Gibbs_mu, 3),
                                                        nsmall = 3)))+
  labs(title = 'Mu trajectories of the sampled Markov chains',
       x= 'nDraws', y='Mu')+ theme_classic()

# Plot for Sigma trajectories of the sampled Markov chains
p2<-ggplot(plot_df,aes(x=V1))+geom_line(aes(y=V3), color='#FCA311', size=.8)+
  annotate(geom = "text", x = 500, y = 2,
           label = paste0("Inefficiency Factor ",format(round(IF_Gibbs_sigma, 3), nsmall = 3)))+
  labs(title = 'Sigma2 trajectories of the sampled Markov chains',
       x= 'nDraws', y='Sigma')+ theme_classic()
p1
p2
```

```r
#  mean and Sd from the results of the Gibbs sampler
#plot_df$V2 represent mean and plot_df$V3 Var
# The resulting posterior predictive density
post_pred<- rnorm(1000, mean = plot_df$V2, sd = sqrt(plot_df$V3))
y <- exp(post_pred)

# Now we plot histogarm of the data and the posterior predictive denstity

#data frame to store the density values
de_df<- data.frame(x=y)

ggplot() +
  geom_histogram(data=df,aes(x = x,y=..density..),linetype=1,
                 fill='#14213D',bins = 20)+
  geom_density(data=de_df,aes(x=y), color='#FCA311', size=1,
               fill='#FCA311',alpha=.5)+
  labs(title = "Histogram of Daily Precipitation",
       subtitle = "Line of Posterior predictive density",
       x = "Precipitation",
       y = "Density") + xlim(0.1,50)+ theme_classic()
#First we upload our data
df1<- read.table("eBayNumberOfBidderData.dat",header = T)
#We run a possion model using the function glm in r,
#note that we exclude the intercept and we use family possion
model <- glm(nBids ~ ., data = df1[,-2], family = poisson)
summary(model)
### Prior and data inputs ###
Covs <- c(2:10) # Select which covariates/features to include
standardize <- F # If TRUE, covariates/features are standardized to mean 0 and variance 1

Nobs <- dim(df1)[1] # number of observations
y <- df1$nBids # y=1 if the women is working, otherwise y=0.
x <- as.matrix(df1[,Covs]) # Covs matrix 7*7
Xnames <- colnames(x)
# Standraizing the covs matrix
if (standardize){
  Index <- 2:(length(Covs)-1)
  x[,Index] <- scale(x[,Index])
}
Npar <- dim(x)[2]
#############################################################################
# This is to add y variable as binary response and adding intercept,
#for now it's not needed
# for (ii in 1:Nobs){
#   if (wat$quality[ii] > 5){
#     y[ii] <- 1
#   }
# }
# wat <- data.frame(intercept=rep(1,Nobs),wat) # add intercept
#############################################################################
# Setting up the prior
mu <- c(rep(0,Npar)) # Prior mean vector
Sigma <- 100*solve(t(x)%*%x) # Prior covariance matrix
```

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

logPossion <- function(beta,y,x,mu,Sigma){
  logLik <- -sum(exp(x%*%beta)) + sum(y%*%(x%*%beta))# <---------------
  #We change on this line
  ################################################################################
  # The first term, -sum(exp(Xbeta)), calculates the sum of the exponential of
  # the linear predictor Xbeta over all observations. This term represents the
  # log-likelihood contribution from the expected
  #counts in the Poisson distribution.
  # The second term, sum(y(Xbeta)), calculates the sum of the
  #observed response variable
  # y multiplied by the linear predictor Xbeta over all
  #observations. This term represents
  # the log-likelihood contribution from
  #the observed counts in the Poisson distribution.
  # By subtracting the first term from the second term,
  #we obtain the log-likelihood of
  # the Poisson regression model given the data
  #and the regression coefficients.
  #if (abs(logLik) == Inf) logLik = -20000;
  # Likelihood is not finite, stear the optimizer away from here!
  ################################################################################
  if (abs(logLik) == Inf){
    logLik <- -20000
  }
  logPrior <- dmvnorm(beta, mu, Sigma, log=TRUE)

  return(logLik + logPrior)
}

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

# 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,
                  logPossion,
                  gr=NULL,
                  y=y,
                  x=x,
                  mu=mu,
                  Sigma=Sigma,
                  method=c("BFGS"),
                  control=list(fnscale=-1),
                  hessian=TRUE)
# Printing the results to the screen
names(OptimRes$par) <- Xnames # Naming the coefficient by covariates
# Computing approximate standard deviations.
```

```r
approxPostStd <- sqrt(diag(solve(-OptimRes$hessian)))

names(approxPostStd) <- Xnames # Naming the coefficient by covariates
print('The posterior mode is:')
print(OptimRes$par)
print('The Hessian Matrix:')
print(OptimRes$hessian)
print('The approximate posterior standard deviation is:')
print(approxPostStd)

RWMSampler_old<- function(logPostFunc,nDraws,c,y,x,mu,Sigma){
  # First we buld our data frame of samples
  sample <- data.frame(matrix(nrow = nDraws, ncol = ncol(x)))
  colnames(sample) <- colnames(x)
  # The inital sample value c here represent a tuning parameter
  sample[1,] <- mvrnorm(1, posteriorMode, c*postCov)
  # Now we implement the Metropolis-Hastings
  #in which we generate samples from the proposal distribution in this case
  # We look at the results of the first sample
  #as theta_i-1 pluged in mvnorm to get theta_i and the we use the values in
  # our proposed logPostFunc
  counter <- 1
  i=1
  while (counter < nDraws) {
    theta_old<-as.numeric(sample[counter,])
    theta_new<-mvrnorm(1,theta_old,c*postCov)
  # We define th our accept/reject threshold
    th<-runif(1,0,1)
  # now we find the value of the target/proposed distribution
    proposed<- logPostFunc(theta_new,y = y,
                           x = x,
                           mu = posteriorMode,
                           Sigma = postCov)
    target<- logPostFunc(theta_old,y = y,
                         x = x,
                         mu = posteriorMode,
                         Sigma = postCov)
    # the ratio of posterior densities in the Metropolis acceptance probability
    if (th<min(1,exp(proposed-target))) {
      counter=counter+1
      sample[counter,]<-theta_new
    }
  }
 return(sample)
}
RWMSampler<- function(logPostFunc,nDraws,c,y,x,mu,Sigma,brnin){
  # First we buld our data frame of samples
  sample <- data.frame(matrix(0,nrow = nDraws - brnin, ncol = ncol(x)))
  colnames(sample) <- colnames(x)
  # The inital sample value c here represent a tuning parameter
  # sample[1,] <- mvrnorm(1, posteriorMode, c*postCov)
  # Now we implement the Metropolis-Hastings
  #in which we generate samples from the proposal distribution in this case
```

```
  # We look at the results of the first sample
  #as theta_i-1 pluged in mvnorm to get theta_i and the we use the values in
  # our proposed logPostFunc

  theta_old<-as.numeric(sample[1,])
  for (i in 1:nDraws) {

    theta_new<-mvrnorm(1,theta_old,c*postCov)
    # We define th our accept/reject threshold
    th<-runif(1,0,1)
    # now we find the value of the target/proposed distribution
    proposed<- logPostFunc(theta_new,y = y,
                           x = x,
                           mu = posteriorMode,
                           Sigma = postCov)
    target<- logPostFunc(theta_old,y = y,
                         x = x,
                         mu = posteriorMode,
                         Sigma = postCov)
    # the ratio of posterior densities in the Metropolis acceptance probability
    if (th<exp(proposed-target)) {
      theta_old<-theta_new
    }
    if (i> brnin) {
      sample[i-brnin,]<-theta_old
    }
  }
  return(sample)
}

nDraws=10000
c=.5
brnin=100 ##adding this new parameter for buirnin interval
posteriorMode=OptimRes$par
postCov=solve(-OptimRes$hessian)
res <- RWMSampler(logPostFunc = logPossion,
                                nDraws = nDraws,
                                  c=c,
                                  y = y,
                                  x = x,
                                  mu = posteriorMode,
                                  Sigma = postCov,
                                  brnin = brnin )
### Changing the plot to be trace plot instead of density
names<-colnames(res)
p_fun<- function(coln){
  plt <- ggplot(res,aes_string(1:(nDraws-brnin),y = coln)) +
    geom_line( color='#FCA311', size=.8)+
      labs(x= 'nDraws', y=coln)
  # We look
  plt
}
```

```
plot(arrangeGrob(grobs = lapply(names, p_fun)))
#First we estimate the betas from our RWMSampler function
betas<- as.matrix(res)
# Input data
x_new <- as.matrix(c(1,1,0,1,0,1,0,1.2,0.8))
##### This code has been modified
# calculating the bet for Poisson since we have our y follow poisson
beta_pois<-exp(betas  %*% x_new)
# Finding number of bidders using poisson function
nbiders<-data.frame(x=rpois(length(beta_pois),beta_pois))
ggplot(nbiders,aes(x = x)) +
  geom_histogram(aes(y=..density..),linetype=1,fill='#14213D',bins = 20)+
    labs(x = 'Number of bidders', y = 'density',
           title ='Plot of the predictive distribution for Number of bidders')
ar_process<- function(phi,mu, sigma,t){
  x_t<-c()
  x_t[1]<-mu
  for(i in 2:t){
    e<- rnorm(1,0,sqrt(3))
    x_t[i]<-mu+(phi*(x_t[i-1]-mu))+e
  }
  return(x_t)
}
phi<-seq(-1,1,by=.5)
res<- list()
for (i in phi) {
  res1 <- ar_process(i, 13, 3, 300)
  res[[paste0("Phi_", i)]] <- res1
}

res<- data.frame(res)
names<-colnames(res)
p_fun<- function(coln){
  plt <- ggplot(res,aes(x = 1:300)) +
    geom_line(aes_string(y = coln),color='#FCA311', size=.8)+
  labs(title = coln,
       x= 'Simulations', y='Phi Value')+ theme_classic()
  plt
}

plot(arrangeGrob(grobs = lapply(names, p_fun)))
data {
  int<lower=0> N;
  vector[N] y;
}
parameters {
  real alpha;
  real beta;
  real<lower=0> sigma;
}
model {
  for (n in 2:N) {
    y[n] ~ normal(alpha + beta * y[n-1], sigma);
```

```
  }
}
data {
  int<lower=0> N;
  vector[N] y;
}
parameters {
  real alpha;
  real beta;
  real<lower=0> sigma;
}
model {
   y[2:N] ~ normal(alpha + beta * y[1:(N - 1)], sigma);
}
##### Example from notes #####
# library(rstan)
# y=c(4,5,6,4,0,2,5,3,8,6,10,8)
# N=length(y)
#
# StanModel = '
# data {
#   int<lower=0> N; // Number of observations
#   int<lower=0> y[N]; // Number of flowers
# }
# parameters {
#   real mu;
#   real<lower=0> sigma2;
# }
# model {
#   mu ~ normal(0,100); // Normal with mean 0, st.dev. 100
#   sigma2 ~ scaled_inv_chi_square(1,2); // Scaled-inv-chi2 with nu 1,sigma 2
#   for(i in 1:N){
#     y[i] ~ normal(mu,sqrt(sigma2));
#   }
# }'
#
# data <- list(N=N, y=y)
# warmup <- 1000
# niter <- 2000
# fit <- stan(model_code=StanModel,data=data, warmup=warmup,iter=niter,chains=4)
# # Print the fitted model
# print(fit,digits_summary=3)
# # Extract posterior samples
# postDraws <- extract(fit)
# # Do traceplots of the first chain
# par(mfrow = c(1,1))
# plot(postDraws$mu[1:(niter-warmup)],type="l",ylab="mu",main="Traceplot")
# # Do automatic traceplots of all chains
# traceplot(fit)
# # Bivariate posterior plots
# pairs(fit)
# ##############################
StanModel = '
```

```
data {
  int<lower=0> N; // Number of observations
  vector[N] y;
}
parameters {
  real mu;
  real<lower=0> sigma2;
  real<lower=-1, upper=1> phi;
  //To enforce the estimation of a
  //stationary AR(1) process, the slope
  //coefficient beta may be constrained with bounds as follows.
}
model {
  mu ~ normal(0,100); // Normal with mean 0, st.dev. 100
  sigma2 ~ scaled_inv_chi_square(1,2); // Scaled-inv-chi2 with nu 1,sigma 2
  // Changing the model to be y_i-mu
  y[2:N] ~ normal(mu + phi * (y[1:(N - 1)]-mu), sqrt(sigma2^2));
}'

# Simulate of x
ar_x<-ar_process(.2, 13, 3, 300)

#From lec notes we have the stanmodel function defined as
y=ar_x
N=length(y)

data <- list(N=N, y=y)
warmup <- 1000
niter <- 2000
fitx <- stan(model_code=StanModel,data=data,warmup=warmup,iter=niter,chains=4)
# Print the fitted model
print(fitx,digits_summary=3)
samples_x <-extract(fitx, pars = c("mu", "sigma2", "phi"))
# Compute the mean
mean_x <- sapply(samples_x, mean)
# 95% credible intervals
intv_x <- sapply(samples_x,function(samples) quantile(samples,c(0.025, 0.975)))
# effective posterior samples
eff_samp_x <- summary(fitx)$summary[1:3,"n_eff"]

result <- data.frame( Mean = mean_x,
                      Credible_Interval_Lower = intv_x[1, ],
                      Credible_Interval_Upper = intv_x[2, ],
                      Effective_Samples = eff_samp_x)
print(result)
# Simulate of y
ar_y<-ar_process(.95, 13, 3, 300)
#From lec notes we have the stanmodel function defined as
#y=ar_y
N=length(y)

data <- list(N=N, y=y)
warmup <- 1000
```

```
niter <- 2000
fity <- stan(model_code=StanModel,data=data, warmup=warmup,iter=niter,chains=4)
# Print the fitted model
print(fity,digits_summary=3)
samples_y <-extract(fity, pars = c("mu", "sigma2", "phi"))
# Compute the mean
mean_y <- sapply(samples_y, mean)
# 95% credible intervals
intv_y <- sapply(samples_y,function(samples) quantile(samples,c(0.025, 0.975)))
# effective posterior samples
eff_samp_y <- summary(fitx)$summary[1:3,"n_eff"]

result <- data.frame( Mean = mean_y,
                      Credible_Interval_Lower = intv_y[1, ],
                      Credible_Interval_Upper = intv_y[2, ],
                      Effective_Samples = eff_samp_y)
print(result)
df_x <-as.data.frame(extract(fitx, pars = c("mu", "phi")))
names<-colnames(df_x)
ln<-length(df_x[,1])
p_fun<- function(coln){
  plt <- ggplot(df_x,aes(x = 1:ln)) +
    geom_line(aes_string(y = coln),color='#FCA311', size=.8)+
    labs(title = coln,
         x= 'Simulations', y='Phi Value')+ theme_classic()
  plt
}

plot(arrangeGrob(grobs = lapply(names, p_fun)))
df_y <-as.data.frame(extract(fity, pars = c("mu", "phi")))
names<-colnames(df_y)
ln<-length(df_y[,1])
p_fun<- function(coln){
  plt <- ggplot(df_y,aes(x = 1:ln)) +
    geom_line(aes_string(y = coln),color='#FCA311', size=.8)+
    labs(title = coln,
         x= 'Simulations', y='Phi Value')+ theme_classic()
  plt
}

plot(arrangeGrob(grobs = lapply(names, p_fun)))
#### In case of normal
# library(rstan)
# y=c(4,5,6,4,0,2,5,3,8,6,10,8)
# N=length(y)
# StanModel = '
# data {
# int<lower=0> N; // Number of observations
# int<lower=0> y[N]; // Number of flowers
# }
# parameters {
# real mu;
# real<lower=0> sigma2;
```

```
# }
# model {
# mu ~ normal(0,100); // Normal with mean 0, st.dev. 100
# sigma2 ~ scaled_inv_chi_square(1,2); // Scaled-inv-chi2 with nu 1,sigma 2
# for(i in 1:N){
# y[i] ~ normal(mu,sqrt(sigma2));
# }
# }'
########### Multilevel normal
# StanModel <- '
# data {
# int<lower=0> N; // Number of observations
# int<lower=0> y[N]; // Number of flowers
# int<lower=0> P; // Number of plants
# }
# transformed data {
# int<lower=0> M; // Number of months
# M = N / P;
# }
# parameters {
# real mu;
# real<lower=0> sigma2;
# real mup[P];
# real sigmap2[P];
# }
# model {
# mu ~ normal(0,100); // Normal with mean 0, st.dev. 100
# sigma2 ~ scaled_inv_chi_square(1,2); // Scaled-inv-chi2 with nu 1, sigma 2
# for(p in 1:P){
# mup[p] ~ normal(mu,sqrt(sigma2));
# for(m in 1:M) {
# y[M*(p-1)+m] ~ normal(mup[p],sqrt(sigmap2[p]));
# }
# }
# }'
##############Possion
# StanModel <- '
# data {
# int<lower=0> N; // Number of observations
# int<lower=0> y[N]; // Number of flowers
# int<lower=0> P; // Number of plants
# }
# transformed data {
# int<lower=0> M; // Number of months
# M = N / P;
# }
# parameters {
# real mu;
# real<lower=0> sigma2;
# real mup[P];
# }
# model {
# mu ~ normal(0,100); // Normal with mean 0, st.dev. 100
```

```
# sigma2 ~ scaled_inv_chi_square(1,2); // Scaled-inv-chi2 with nu 1, sigma 2
# for(p in 1:P){
# mup[p] ~ lognormal(mu,sqrt(sigma2)); // Log-normal
# for(m in 1:M) {
# y[M*(p-1)+m] ~ poisson(mup[p]); // Poisson
# }
# }
# }'
### The original Code:

# Direct vs Gibbs sampling for bivariate normal distribution

# Initial setting
mu1 <- 1
mu2 <- 1
rho <- 0.9
mu <- c(mu1,mu2)
Sigma = matrix(c(1,rho,rho,1),2,2)
nDraws <- 500 # Number of draws

library(MASS) # To access the mvrnorm() function

# Direct sampling from bivariate normal distribution
directDraws <- mvrnorm(nDraws, mu, Sigma)

# Gibbs sampling
gibbsDraws <- matrix(0,nDraws,2)
theta2 <- 0 # Initial value for theta2
for (i in 1:nDraws){

  # Update theta1 given theta2
  theta1 <- rnorm(1, mean = mu1 + rho*(theta2-mu2), sd = sqrt(1-rho**2))
  gibbsDraws[i,1] <- theta1

  # Update theta2 given theta1
  theta2 <- rnorm(1, mean = mu2 + rho*(theta1-mu1), sd = sqrt(1-rho**2))
  gibbsDraws[i,2] <- theta2

}

a_Direct <- acf(directDraws[,1])
a_Gibbs <- acf(gibbsDraws[,1])

IF_Gibbs <- 1+2*sum(a_Gibbs$acf[-1])

par(mfrow=c(2,4))

# DIRECT SAMPLING
plot(1:nDraws, directDraws[,1], type = "l",col="blue") # traceplot of direct draws

hist(directDraws[,1],col="blue") # histogram of direct draws
) # Cumulative mean value of theta1, direct draws
cusumData <- cumsum(directDraws[,1])/seq(1,nDraws
```

```r
plot(1:nDraws, cusumData, type = "l", col="blue")

barplot(height = a_Direct$acf[-1],col="blue") # acf for direct draws

# GIBBS SAMPLING
plot(1:nDraws, gibbsDraws[,1], type = "l",col="red") # traceplot of Gibbs draws

hist(gibbsDraws[,1],col="red") # histogram of Gibbs draws
# Cumulative mean value of theta1, Gibbs draws
cusumData =  cumsum(gibbsDraws[,1])/seq(1,nDraws)
plot(1:nDraws, cusumData, type = "l", col="red")

barplot(height = a_Gibbs$acf[-1],col="red") # acf for Gibbs draws

# Plotting the cumulative path of estimates of Pr(theta1>0, theta2>0)
par(mfrow=c(2,1))
plot(cumsum(directDraws[,1]>0 & directDraws[,2]>0)/seq(1,nDraws),type="l",
     main='Direct draws', xlab='Iteration number', ylab='', ylim = c(0,1))
plot(cumsum(gibbsDraws[,1]>0 & gibbsDraws[,2]>0)/seq(1,nDraws),type="l",
     main='Gibbs draws', xlab='Iteration number', ylab='', ylim = c(0,1))


#######################################################
```