3D Printer Simulation and Sampling Study

Zongsheng Liu (s2097920)

```
##
## Attaching package: 'ggExtra'
```

```
## The following object is masked from 'package:shiny':
##
##     runExample
```

1.0 Model Description

The aim is to estimate the parameters of a Bayesian statistical model of material use in a 3D printer. The printer uses rolls of filament that get heated and squeezed through a moving nozzle, gradually building objects. The objects are first designed in a CAD program (Computer Aided Design) that also estimates how much material will be required to print the object. The data contains information about one 3D-printed object per row. The columns are
If the CAD system and printer were both perfect, the CAD_Weight and $Actual\_Weight$ values would be equal for each object. In reality, there are random variations, for example, due to varying humidity and temperature, and systematic deviations due to the CAD system not having perfect information about the properties of the printing materials. When looking at the data (see below) it's clear that the variability of the data is larger for larger values of CAD_Weight. The printer operator has made a simple physics analysis, and settled on a model where the connection between CAD_Weight and $Actual\_Weight$ follows a linear model, and the variance increases with square of CAD_Weight. If we denote the CAD weight for observations i by x_i, and the corresponding actual weight by y_i, the model can be defined by

$$y_i \sim Normal[\beta_1 + \beta_2 x_i, \beta_3 + \beta_4 x_i^2)].$$

To ensure positivity of the variance, the parameterisation

$$\theta = [\theta_1, \theta_2, \theta_3, \theta_4] = [\beta_1, \beta_2, log(\beta_3), log(\beta_4)]$$

is introduced, and the printer operator assigns independent prior distributions as follows:

where $LogExp(a)$ denotes the logarithm of an exponentially distributed random variable with rate parameter a. The

$$\gamma = (\gamma_1, \gamma_2, \gamma_3, \gamma_4)$$

values are positive parameters. The printer operator reasons that due to random fluctuations in the material properties (such as the density) and room temperature should lead to a relative error instead of an additive error, which leads them to the model as an approximation of that. The basic physics assumption is that the error in the CAD software calculation of the weight is proportional to the weight itself.

1.1 Prior Density
Knowing that all the observations and all θ_i are independent. Thus, the joint prior density would be computed as:

$$p(\theta) = \prod_{i=1}^{4} p(\theta_i).$$

By taking the logarithm for both sides, the logarithm of the joint prior density could be reformulated as

$$log\ p(\theta) = log \prod_{i=1}^{4} p(\theta_i) = \sum_{i=1}^{4} log\ p(\theta_i).$$

1.2 Observation Likelihood

Since all the x_i all independent, the observation log-likelihood would simply be computed as:

$$p(y \mid \theta) = log \prod_{i=1}^{86} f(x_i, \theta) = \sum_{i=1}^{86} log\ f(x_i, \theta).$$

1.3 Posterior Density

Firstly, knowing that posterior density is proportional to the prior density times the observation likelihood. In mathematical symbols, it could be written as:

$$p(\theta \mid y) \propto\ p(\theta) * p(y \mid \theta).$$

Thus, in terms of the logarithm, the logarithm of the posterior density would be computed as:

$$log\ p(\theta \mid y) \propto\ log\ p(\theta) + log\ p(y \mid \theta).$$

1.4 Posterior Mode

Finding the posterior mode needs to be done by maximising the posterior distribution with respect to θ. Since the internal command optim in R provide a minimisation function with the default setting. In this case, maximisation was needed. It could be done by either adding an additional argument control=list(fnscale=-1) in optim or manually setting the fn arguments with an additional negative sign. The return value would simply contain par (the posterior mode location) and hessian (the Hessian of the log density at the mode). To find the inverse of the negated Hessian S, it could be done by command solve(-hessian).

1.5 Gaussian Approximation

In this case, setting $\gamma = (1, 1, 1, 1)$ and a start value $\theta = 0$. Then, obtain a multivariate Normal approximation $Normal(\mu,\ S)$ for θ. The mode, Hessian and negated inverse of the Hessian were computed as:

```
#compute mode, Hessian, negated inverse of the Hessian
sol <- posterior_mode(theta_start, x, y, params)
#call out each component from the above list
sol$mod
```

```
## [1] -0.1007081  1.0800213 -2.9817858 -6.7585702
```

```
sol$Hessian
```

```
##              [,1]         [,2]       [,3]       [,4]
## [1,]  -238.750166  -2573.81437   2.865874  -2.742917
## [2,] -2573.814374 -63698.40393 -14.451113  14.701761
## [3,]     2.865874    -14.45111  -1.468185  -3.658789
## [4,]    -2.742917     14.70176  -3.658789 -32.314181
```

```
sol$Negated_Inverse_Hessian
```

```
##               [,1]          [,2]         [,3]          [,4]
## [1,]  0.0083022441 -3.433203e-04  0.030272279 -0.0042885091
## [2,] -0.0003433203  2.995883e-05 -0.001492856  0.0002118015
## [3,]  0.0302722786 -1.492856e-03  1.062908241 -0.1235970950
## [4,] -0.0042885091  2.118015e-04 -0.123597095  0.0454008900
```

1.6 Importance sampling function

Firstly, using a multivariate Normal approximation as the importance sampling distribution. Then, unnormalized importance weight could by obtained by:

$$\tilde{w_k} = \frac{p(\theta)p(y|\theta)}{\tilde{p}(\theta|y)} \mid \theta = x_k.$$

In terms of logarithm, log-weights could be simply computed as

$$log\ p(\theta) + log\ p(y|\theta) - log\ \tilde{p}(\theta|y) \mid \theta = x_k.$$

Due to the lack of unnormalistion, it could not be represented accurately in the computation. Thus, normalized weight could be obtained by

$$\tilde{w_k} = exp[log\ w_k - max_j\ log\ w_j]$$

.

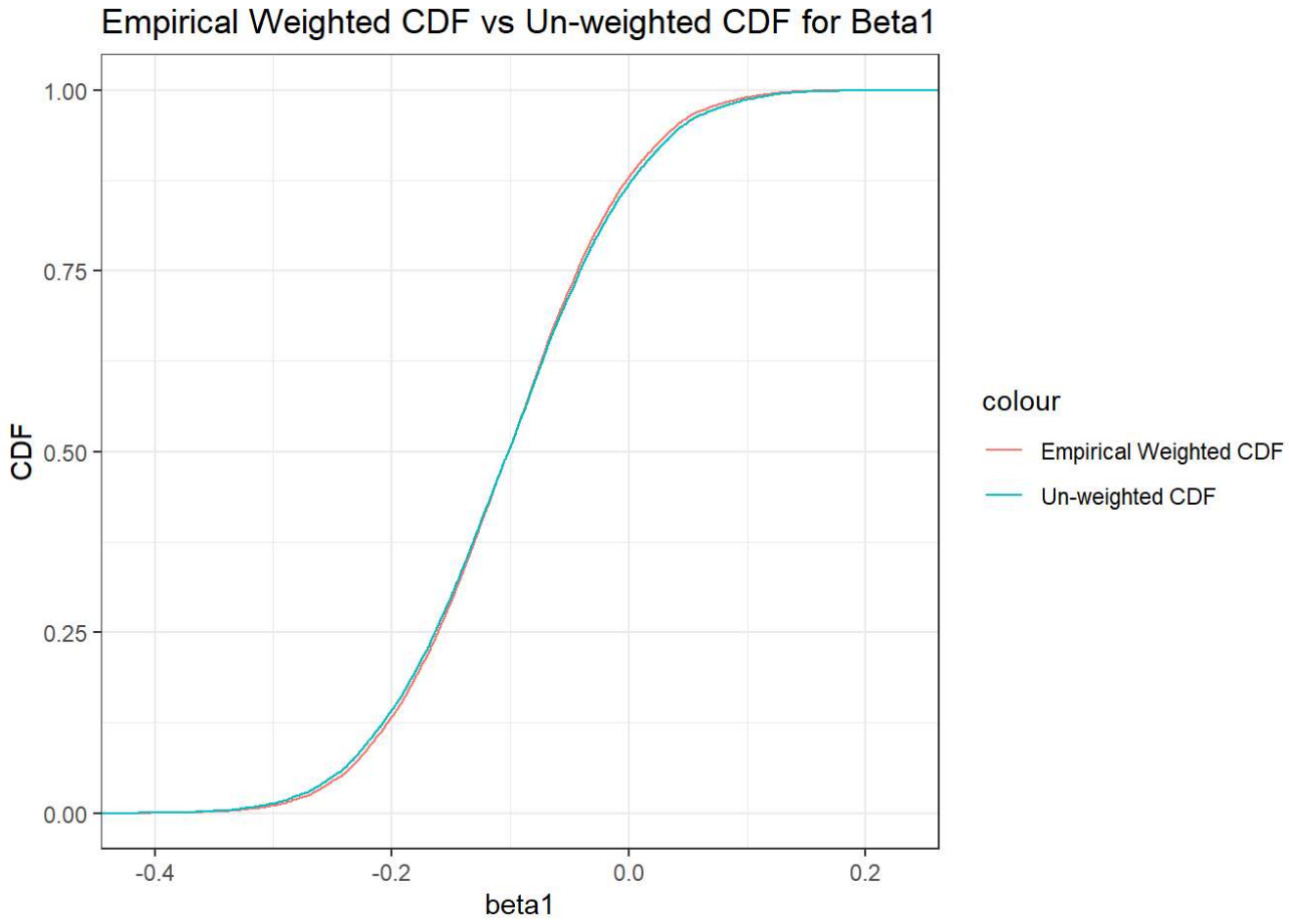
After finishing normalization process, values of β_3 and β_4 should be further computed by exp() since they were defined above by taking logarithm. Finally, put all 10000 samples together into a data frame, it would have column names β_1 to β_4 and the normalized weights.

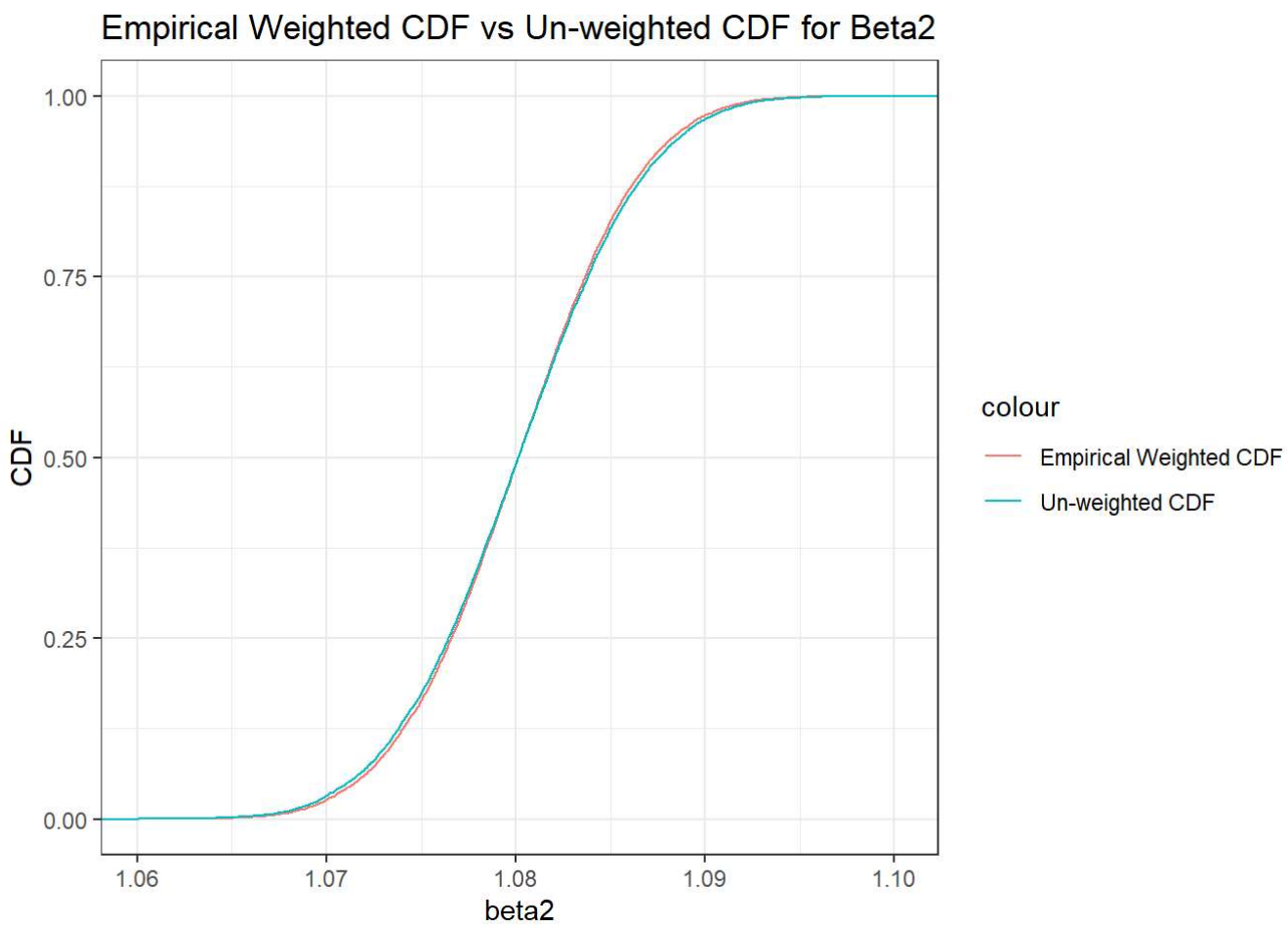
1.7
Since the importance sampling function had already been defined in 2.6, 10000 samples could be computed. If sum up the exponential of all the log normalized weights, it would have a value of 1.

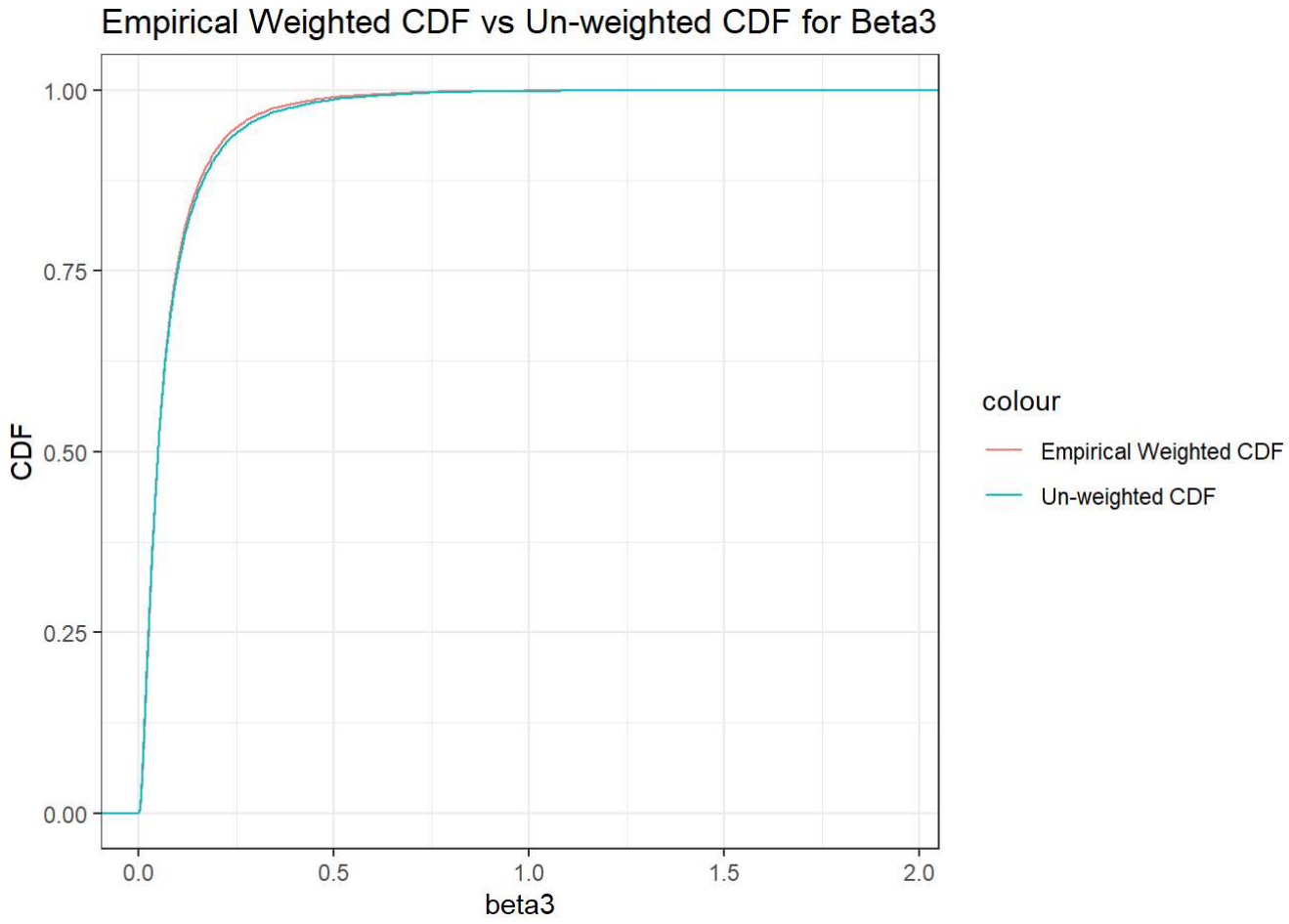
```
sum(exp(data[,5]))
```

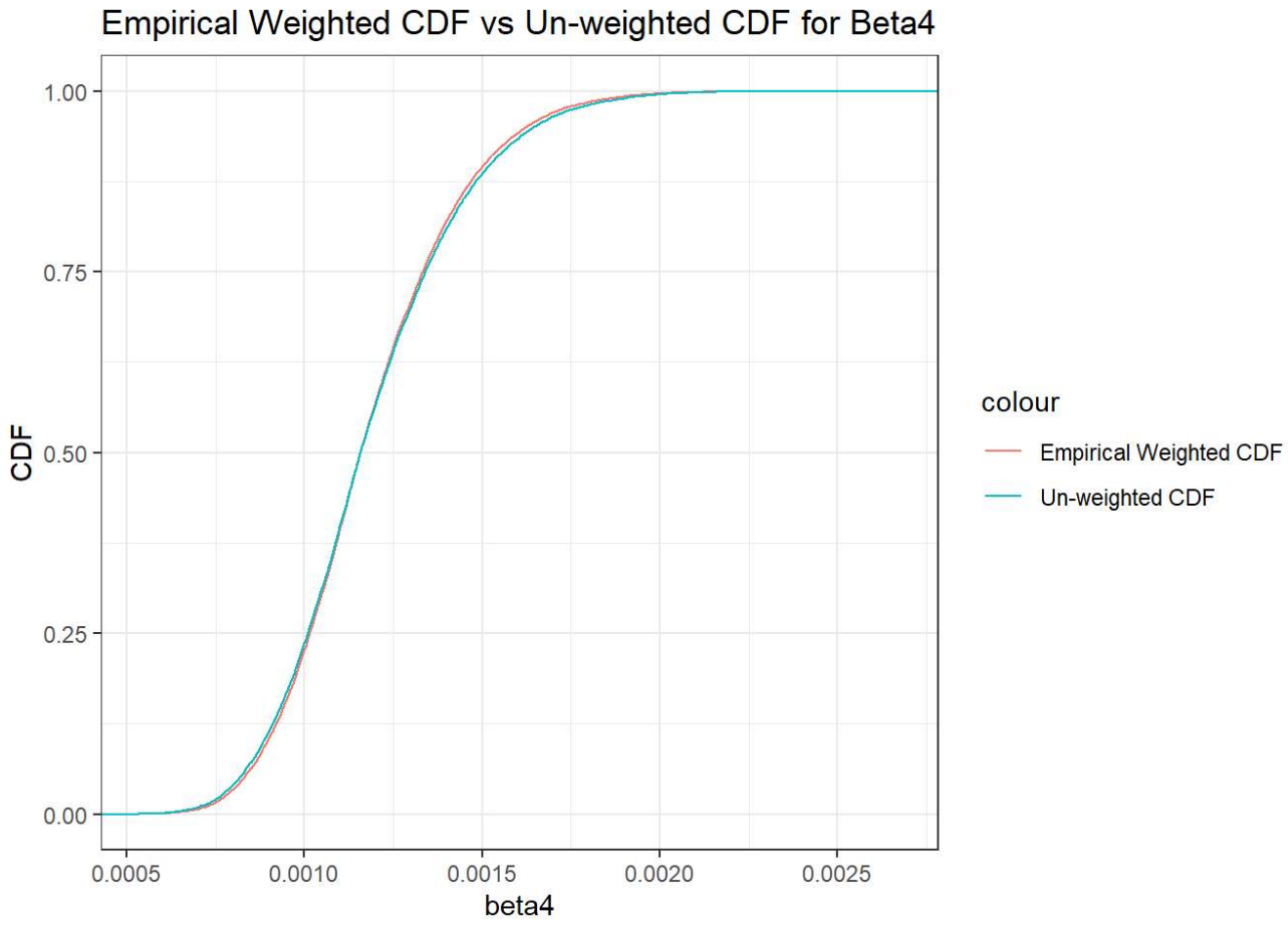
```
## [1] 1
```

Thus, the function defined in 2.6 would make a correct weight. Since there were already a data frame contains 10000 samples. Thus, the weighted CDF VS un-weighted CDF for all β could be plotted as follow:


Empirical Weighted CDF vs Un-weighted CDF for Beta1
1.00
0.75
0.50
0.25
0.00
CDF
-0.4
-0.2
0.0
0.2
beta1
colour
Empirical Weighted CDF
Un-weighted CDF


Empirical Weighted CDF vs Un-weighted CDF for Beta2
1.00
0.75
0.50
0.25
0.00
CDF
1.06
1.07
1.08
1.09
1.10
beta2
colour
Empirical Weighted CDF
Un-weighted CDF





The 90% credible interval for all β could be formed as follow:

```
##    beta   lower_bound upper_bound
## 1 beta1 -0.3505105612 0.124962836
## 2 beta2  1.0683025648 1.091904334
## 3 beta3  0.0024580010 0.446173846
## 4 beta4  0.0007546604 0.002158013
```

From the sampling method point of view, the importance sampling technique was applied in this case. Since it was difficult to sample it directly, the importance sampling method could solve this problem by obtaining samples from an excellent posterior distribution. A multivariate normal distribution was used as the importance sampling distribution in this case. By investigating the above plot, empirical weighted CDF and unweighted CDF were relatively close and virtually indistinguishable for all β. Since it already used a relatively large sample size (N=10000), the conclusion could be drawn that the importance sampling distribution was a good distribution in this case.

From the 3D printer point of view, the generated credible interval may could help to make a more advanced estimation for parameters. Thus, it would lead to a more precise CAD weight estimation. Since the credible intervals involve weight, it may help to identify which parameters are most important and which are less critical. Furthermore, knowing that the credible interval would provide some information on uncertainty about the estimated parameters may help adjust and quantify the uncertainty in the estimation. To sum up, it provides good information to improve the algorithm of the printer.

1 Code appendix

```r
#' s2097920, Zongsheng Liu
#' Add your own function definitions on this file.

#' Log-Exponential density
#'
#' Compute the density or log-density for a Log-Exponential (LogExp)
#' distribution
#'
#' @param x vector of quantiles
#' @param rate vector of rates
#' @param log logical; if TRUE, the log-density is returned

dlogexp <- function(x, rate = 1, log = FALSE) {
  result <- log(rate) + x - rate * exp(x)
  if (!log) {
    exp(result)
  }
  result
}

#' Log-Sum-Exp
#'
#' Convenience function for computing log(sum(exp(x))) in a
#' numerically stable manner
#'
#' @param x numerical vector

log_sum_exp <- function(x) {
  max_x <- max(x, na.rm = TRUE)
  max_x + log(sum(exp(x - max_x)))
}



#1
library(ggplot2)
library(ggExtra)
library(StatCompLab)
library(shiny)
library(mvtnorm)



#2
#Load the required data set
data("filament1", package = "StatCompLab")

#2.1
#compute the joint prior density
log_prior_density <- function(theta, params){
  #Product operator changed into sigma since logarithm was used
  l_prior_d <- (sum(lpd_theta_1 <- dnorm(theta[1], mean = 0, sd = sqrt(params[1]), log = TRU
E),
                    lpd_theta_2 <- dnorm(theta[2], mean = 1, sd = sqrt(params[2]), log = TRU
E),
                    lpd_theta_3 <- dlogexp(theta[3], rate = params[3], log = TRUE),
```

```
                    lpd_theta_4 <- dlogexp(theta[4], rate = params[4], log = TRUE)))
  l_prior_d
}

#2.2
log_like <- function(theta, x, y){
  #sum up all the log likelihood
  LogLikelihood <- (sum(dnorm(y,
                              mean = theta[1]+x*theta[2],
                              sd = sqrt(exp(theta[3])+x^2*exp(theta[4])),
                              log = TRUE)))
  LogLikelihood
}


#2.3
log_posterior_density <- function(theta, x, y, params){
  l_posterior_d <- (log_prior_density(theta, params)+log_like(theta, x, y))
  l_posterior_d
}

#2.4
posterior_mode <- function(theta_start, x, y, params){
  opt <- optim(par = theta_start,  #compute the mode which max the log posterior density
               fn = log_posterior_density,
               x=x,
               y=y,
               params=params,
               hessian = TRUE, # compute Hessian at mode
               control=list(fnscale=-1))  #additional argument to change default minimization
process to maximization

  mode <- opt$par # mode
  hessian <- opt$hessian # Hessian
  S <- solve(-hessian) #negated inverse of the Hessian
  return(list(mode=mode, Hessian=hessian, Negated_Inverse_Hessian=S)) #return them as a list
}

#2.5
#set up all the initial value
params <- c(1,1,1,1)
theta_start <- c(0,0,0,0)
x <- filament1$CAD_Weight # read data from filament1 with column name CAD_Weight
y <- filament1$Actual_Weight # read data from filament1 with column name Actual_Weight
#compute mode, Hessian, negated inverse of the Hessian
sol <- posterior_mode(theta_start, x, y, params)
#call out each component from the above list

#2.6
do_importance <- function(N, mu, S, x, y, params){
  sample <- rmvnorm(N, mean = mu, sigma = S)
  #compute the unnormalized weights
  log_weights <- log_posterior_density(sample, x, y, params)-dmvnorm(sample, mean = mu, sigma
= S, log = TRUE)
  #compute the normalized weight
  log_weights_norm <- log_weights - log_sum_exp(log_weights)
```

```
  log_weight_table <-data.frame(beta1 = sample[,1],
                                beta2 = sample[,2],
                                beta3 = exp(sample[,3]),
                                beta4 = exp(sample[,4]),
                                log_weights = log_weights_norm)
  log_weight_table
}

#2.7
mu <- sol$mode
S <- sol$Negated_Inverse_Hessian
data= do_importance(10000, mu, S, x, y, params)




#using wquantile to construct a 90% credible interval for all Beta
make_CI <- function(x, weights, prob){
  credible_interval <- wquantile(x, probs = c((1-prob)/2, 1-(1-prob)/2), weights = weights)
  credible_interval

}


#make the credible interval for Beta into a table
CI_table <- data.frame(beta=c("beta1","beta2","beta3","beta4"),
                       lower_bound = c(make_CI(data[,"beta1"], exp(data[,"log_weights"]), 0.
9)[1],
                                       make_CI(data[,"beta2"], exp(data[,"log_weights"]), 0.
9)[1],
                                       make_CI(data[,"beta3"], exp(data[,"log_weights"]), 0.
9)[1],
                                       make_CI(data[,"beta4"], exp(data[,"log_weights"]), 0.
9)[1]),
                       upper_bound =c(make_CI(data[,"beta1"], exp(data[,"log_weights"]), 0.9)
[2],
                                      make_CI(data[,"beta2"], exp(data[,"log_weights"]), 0.9)
[2],
                                      make_CI(data[,"beta3"], exp(data[,"log_weights"]), 0.9)
[2],
                                      make_CI(data[,"beta4"], exp(data[,"log_weights"]), 0.9)
[2]) )
```