StatComp Project 1: Simulation and sampling

Jack Watts (s2081957, wattsjack11)

1 Confidence interval approximation assessment

We are required to solve the inequality

$$a < \frac{\widehat{\lambda} - \lambda}{\sqrt{\lambda/n}} < b$$

.

We have a given α value for which we will contruct our confidence interval. We consequently have that $a = z_{\alpha/2}$ and $b = z_{1-\alpha/2}$. We shall let $x = \sqrt{\lambda}$ (note consequently that this means $\lambda = x^2$) which changes the inequality to

$$a < \frac{\widehat{\lambda} - \lambda}{x/\sqrt{n}} < b$$

Now we can multiply both sides of our inequality by $x/\sqrt{\lambda}$ to give us

$$\frac{a}{\sqrt{n}}x < \widehat{\lambda} - x^2 < \frac{b}{\sqrt{n}}x$$

To get our confidence interval we will now consdier each side of the inequality on its own. By basic algebraic manipulation we get that:

$$x^2 + \frac{a}{\sqrt{n}}x - \widehat{\lambda} < 0$$

and

$$x^2 + \frac{b}{\sqrt{n}}x - \widehat{\lambda} > 0$$

Now if we utilise the quadratic formula we can generate the following solutions:

$$\frac{-\frac{a}{\sqrt{n}} \pm \sqrt{\frac{a^2}{n} + 4\widehat{\lambda}}}{2} < x$$

and

$$x < \frac{-\frac{b}{\sqrt{n}} \pm \sqrt{\frac{b^2}{n} + 4\widehat{\lambda}}}{2}$$

Because of the $\pm$ nature of the results of the quadratic formula, then get four inequalities involving x. We have $a < 0$ and $b > 0$. Now consider that if the values that an inequality represents is entirely encapsulated within another one, then we can essentially discount it.

It dos not take much effort to notice that we can discount two of the inequalities and note that the region we desire is:

$$\frac{-\frac{b}{\sqrt{n}} + \sqrt{\frac{b^2}{n} + 4\widehat{\lambda}}}{2} < x < \frac{-\frac{a}{\sqrt{n}} + \sqrt{\frac{a^2}{n} + 4\widehat{\lambda}}}{2}$$

as we recall that $a = -b$ and $a < 0$ and also that $x = \sqrt{\lambda}$ then we can rewrite these to have our final confidence interval to be written as

$$\frac{a^2}{2n} - \frac{a}{2\sqrt{n}}\sqrt{\frac{a^2}{n} + 4\widehat{\lambda}} + \widehat{\lambda} < \lambda < \frac{b^2}{2n} - \frac{b}{2\sqrt{n}}\sqrt{\frac{b^2}{n} + 4\widehat{\lambda}} + \widehat{\lambda}$$

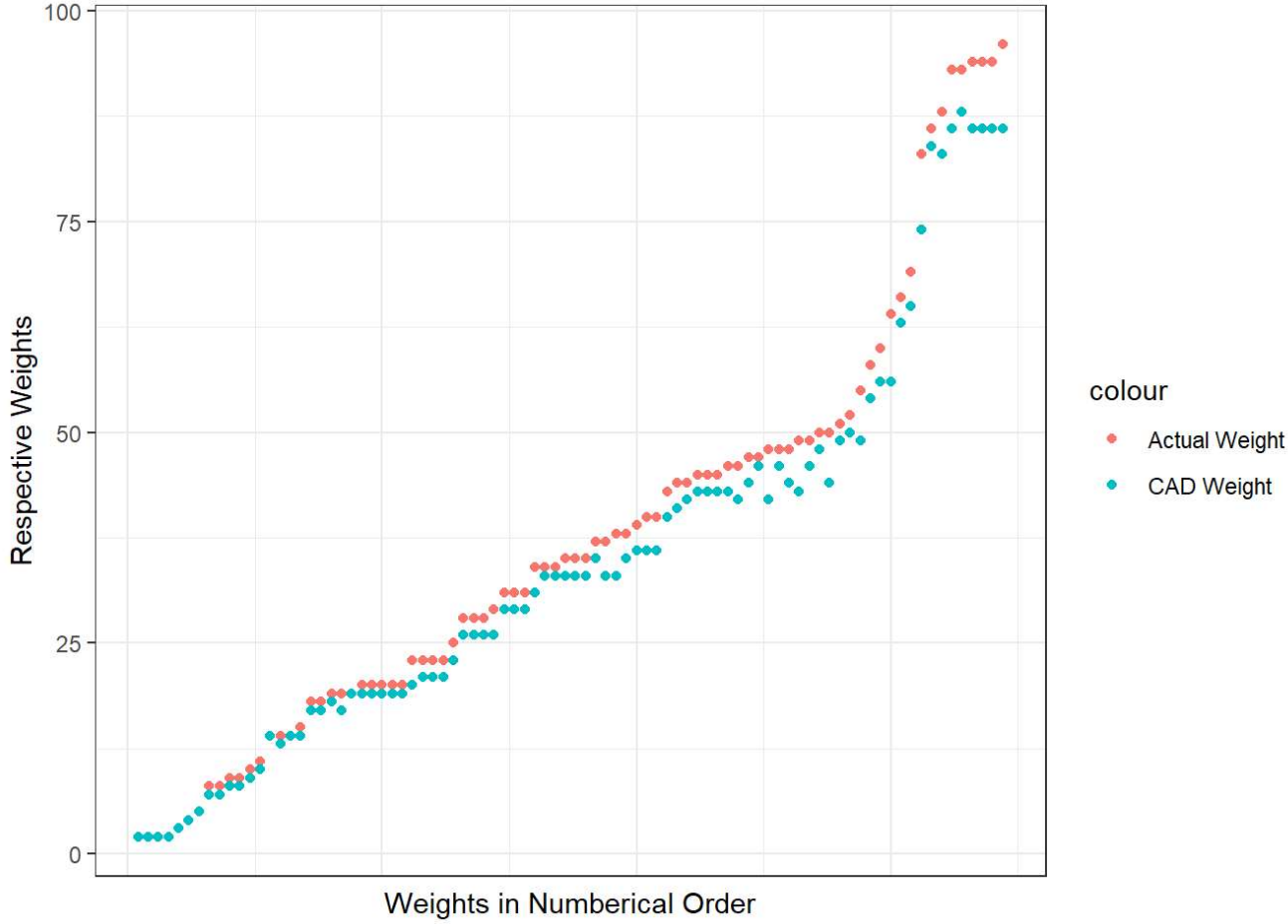
We then estimate the confidence intervals coverage for our two methods (approximated and non-approximated)

	Non-Approximated Method's Coverge	Approximated Method's Coverge
n = 10	0.9162	0.8901
n = 50	0.9087	0.8958
n = 100	0.8986	0.9034
n = 500	0.9020	0.8988
n = 1000	0.8991	0.9005
n = 5000	0.8989	0.9028
n = 10000	0.9039	0.9012

From this table of values we can see that as the size of 'N' increases then the confidence intervals' sizes converge towards what we want them to be, 90%. It is perhaps important to note that the 'Approximated Method, though the difference in coverage is minimal at best. Therefore we can maybe start to consider other factors which means we may want prefer one method over the other. I removed each method once and ran the code to see what the difference in run-time was. I noticed that the run-time was shorter for when we were only running the approximated version. I assume that computationally, this is an easier method to be calculated. This makes sense as the other one had squares and square-roots being churned through it. For such large N, and possibly even larger than we have calculated here, it could be better to run the approximated version.

2 3D printer materials prediction:

We shall graph the data that we have been given for the 'actual' and 'CAD' weights:



#Prior Density: The distributions are independent and this consequently means that we can take the product of the distributions to give the joint prior density. In mathematical terms:

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

We implement the idea of this in the 'log_prior_density' function. To create the log prior density we need to take the log of both sides which can be thought of as:

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

#Observation Likelihood

This considers $p(\boldsymbol{y}|\boldsymbol{\theta})$ We have the analysis of the likelihood of seeing our results $\boldsymbol{y}$, taking into account the set of parameters $\boldsymbol{\theta}$. We do this by making the PDF asssociated with the values $\boldsymbol{y}$ and the θ_i's. This uses the 'log_like()' function.

#Posterior Density:

As seen in the course 'Statistical Methodolgy' we can find the posterior density of our model using

$$p(\boldsymbol{\theta}|\boldsymbol{y}) = \frac{p(\boldsymbol{\theta}) \cdot p(\boldsymbol{y}|\boldsymbol{\theta})}{p(\boldsymbol{y})}$$

Note that the $p(\boldsymbol{y})$ is merely a constant. Now using the basic laws of logarithms, i.e: $log(ab) = log(a) + log(b)$.

$$\begin{aligned} log[p(\boldsymbol{\theta}|\boldsymbol{y})] &= log(p(\boldsymbol{\theta}) \cdot p(\boldsymbol{y}|\boldsymbol{\theta})) \\ &= log(p(\boldsymbol{\theta})) + log(p(\boldsymbol{y}|\boldsymbol{\theta})) \end{aligned}$$

The posterior mode gives us the most common value and so is the 'peak' on the graph, or in other words the maximum. Using the optim function, as we have done previously, to work that out.

```
params <- c(1, 1, 1, 1)
x = filament1 $ CAD_Weight
y = filament1 $ Actual_Weight

fit <- optim(params,
             fn = log_posterior_density,
             x = x,
             y = y,
             params = params,
              hessian = TRUE,
              control = list(fnscale = -1))
```

#Gaussian Estimation:

We note that the Gaussian Estimation is given by the writing below. We have that the posterior mode is $\widehat{\boldsymbol{\theta}}$ and the $[I(\widehat{\boldsymbol{\theta}})]^{-1}$ represents the Hessian matrix:

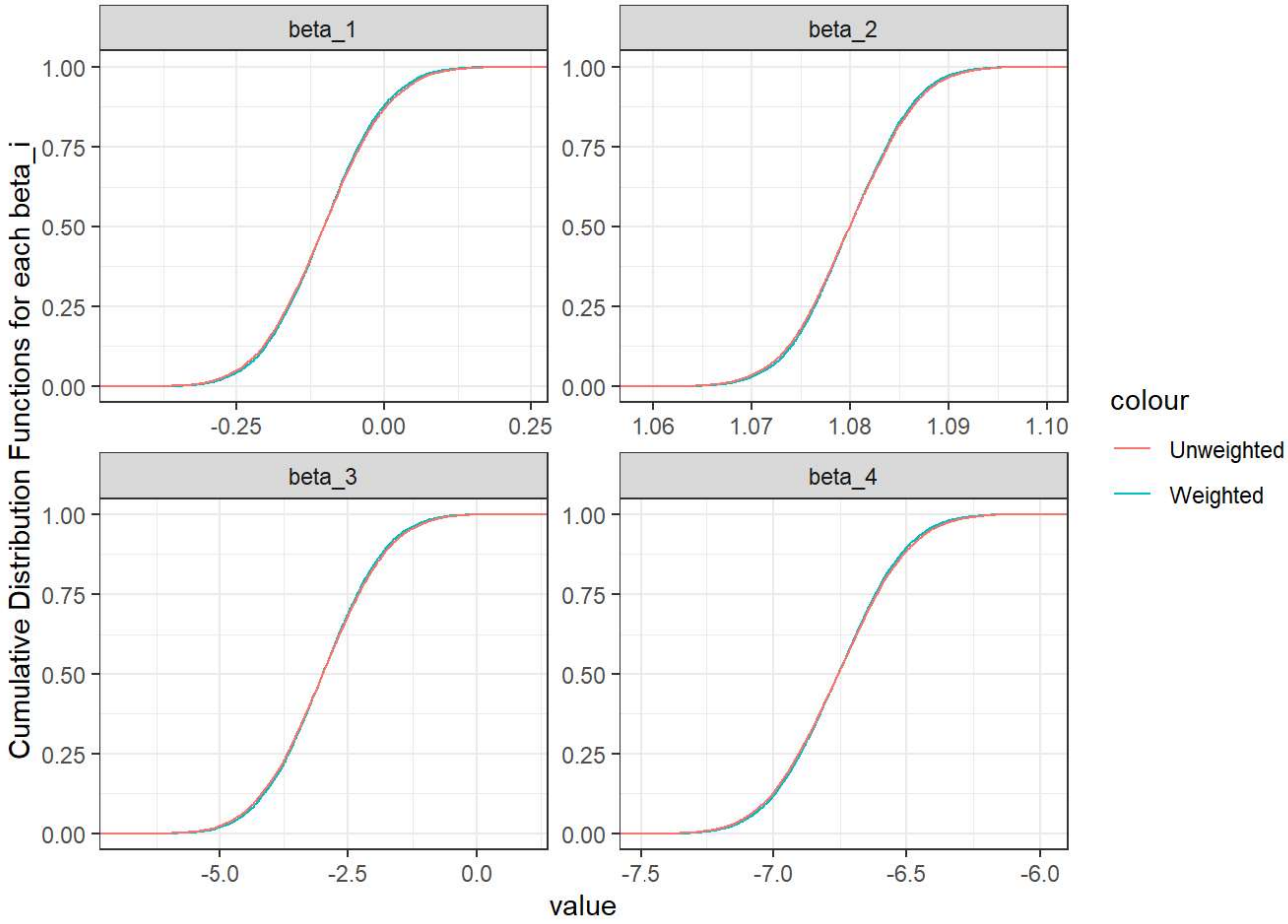
$$p(\boldsymbol{\theta}|\boldsymbol{y}) \approx \mathrm{N}(\widehat{\boldsymbol{\theta}}, [I(\widehat{\boldsymbol{\theta}})]^{-1})$$

#Importance Sampling: Our importance sampling function, 'do_importance' makes use of the Multivariate Normal Approximation. We will now construct the log-importance weights

```
mu <- fit$par
S <- solve(-fit$hessian)
a <- do_importance(10000,
                   mu = fit $ par,
                   S = solve(-fit $ hessian),
                   x = filament1 $ CAD_Weight,
                   y =filament1 $ Actual_Weight,
                   params = params)
data <- pivot_longer(a, starts_with('beta'))
```

Now for each one of the β_i's we can make a graph out of the weighted and un-weighted CDF

```
## Registered S3 method overwritten by 'spatstat.geom':
##   method     from
##   print.boxx cli
```

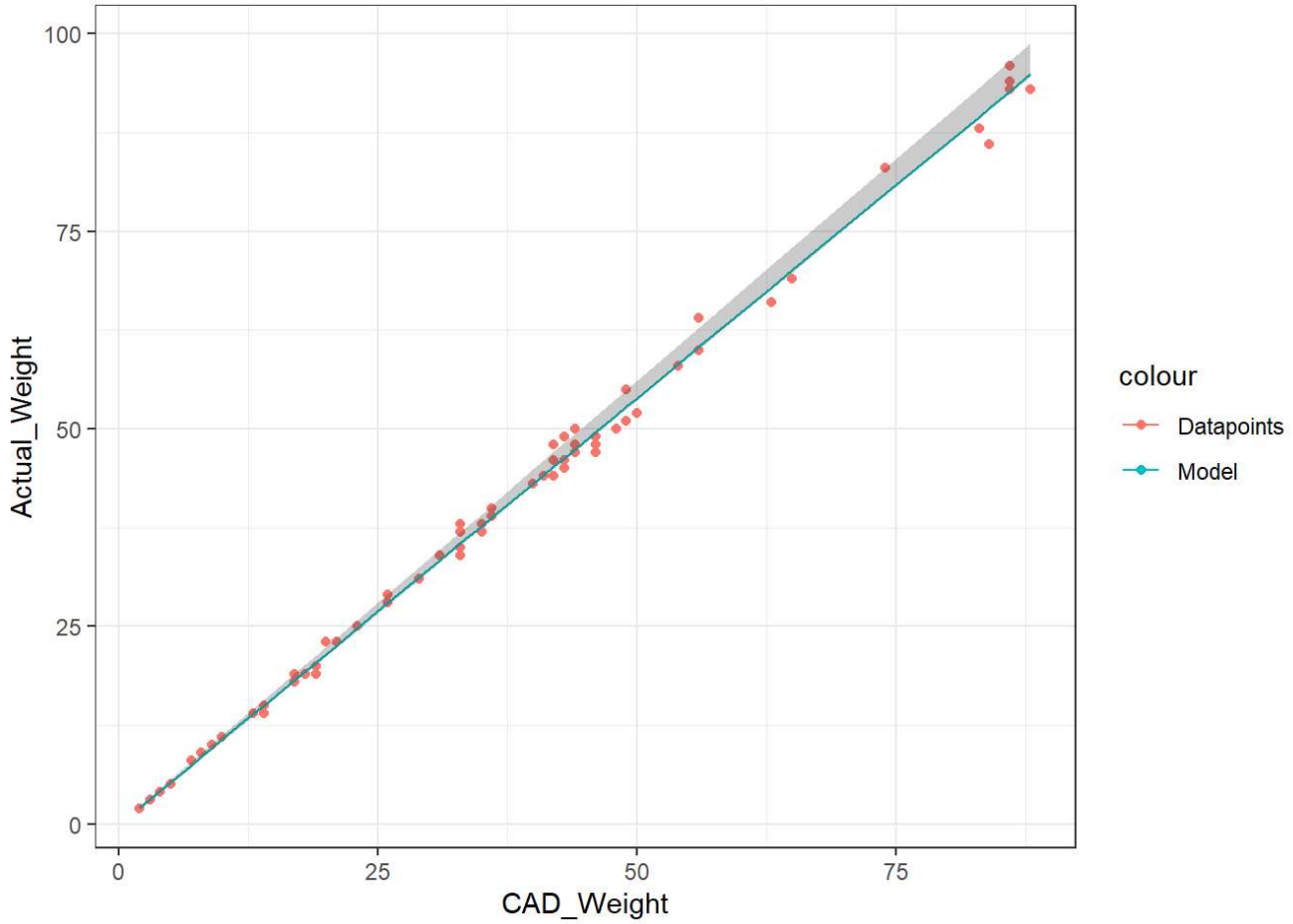


It can be easily seen that our importance sample is a rather impressively close comparison to the unweighted and weighted line of the graph. Consequently we can say that we can use the CAD system to make an accurate and meaningful assessment and prediction for the actual weights.

Then we are able to create a confidence interval for every one of our parameters using 'qnorm' to evaluate the β_i's for a given value of α

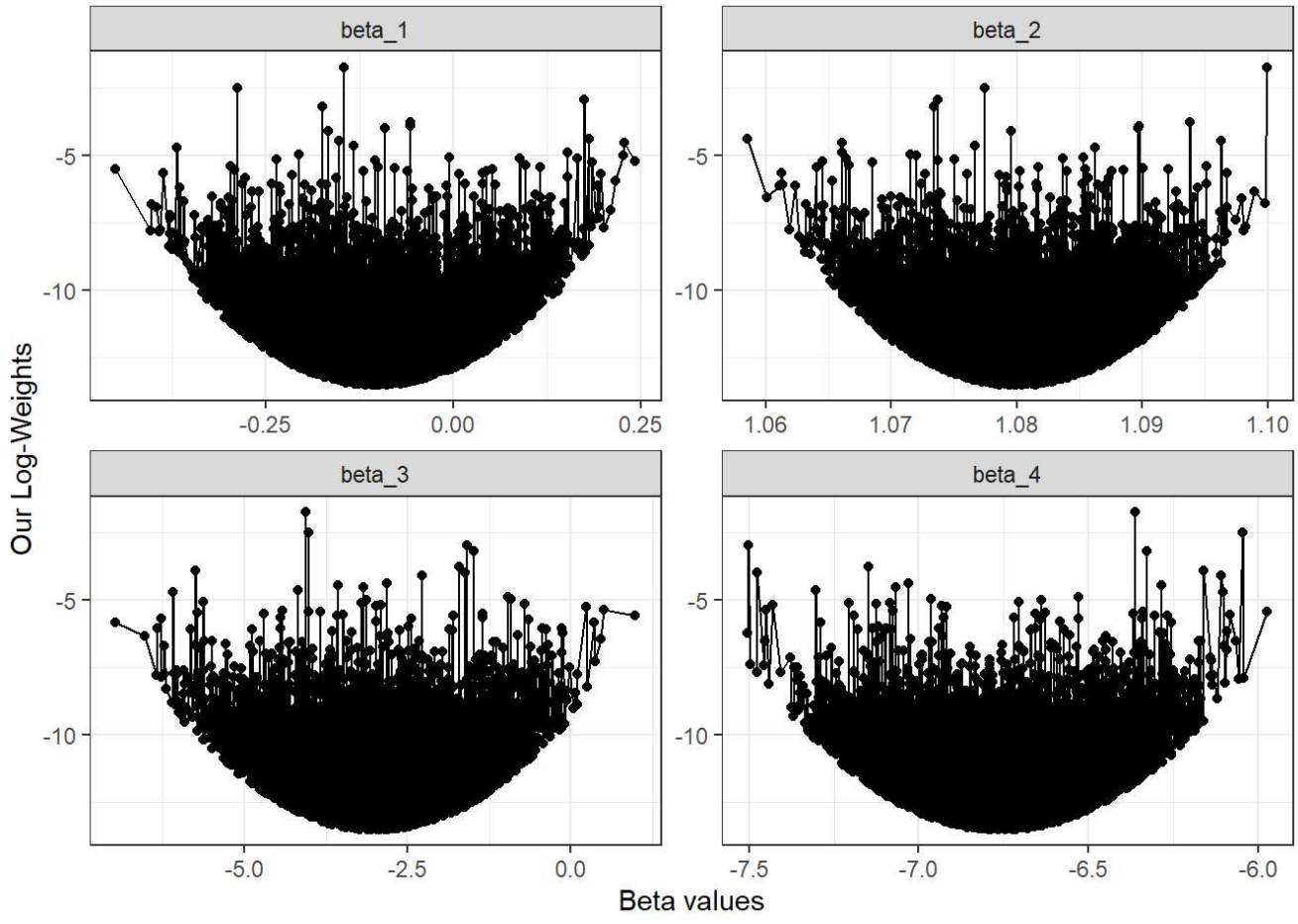
bound	beta_1.Lwr	beta_1.Upr	beta_2.Lwr	beta_2.Upr	beta_3.Lwr	beta_3.Upr	beta_4.Lwr	beta_4.Upr
lower	-0.2967603	0.1775543	1.066081	1.099883	-5.726187	-0.6739418	-7.49738	-6.046048
upper	-0.2967603	0.1775543	1.066081	1.099883	-5.726187	-0.6739418	-7.49738	-6.046048

If we now check for the mode using our prediction interval, then we can do this by inputting the CAD weights into our model and consequently verify these against the genuine weights



Therefore, as we can clearly see using the 90% interval then almost every single weight has fallen within our margin for error. The predictions do however get less accurate as the 'actual weight' increases.

Now we consider plotting the values for each β_i and their log importance weights



Each graph are shaped almost like a soup bowl. There are clearly a great deal of difference as to wat values that the β_i's vary slightly under but they all do follow the same sort of shape. There is definitely a condensation of the points towards the base and middle of the shape created. We also see that there are few bad points which are well away from the place tht we would expect them to be (the centre). These tend to be at the extremes of the results which is not unexpected when using a model as such. It is comparable to the small deviation we see in the Cumulative Distribution Function graphs above.

We note that β_2 is the most consistemt of the β_i's and from the value 1.08 varies less than 2% in either direction. However for this there is clearly quite a large additive error as all of the values deviate quite largely away from the mean which is 1.

The model is then fairly good and it follows our data well.

3 Code appendix

```
#' Jack Watts, s2081957, wattsjack11
#' Add your own function definitions at the top of this file.

#' We shall now define the functions that we shall use in the project.
#'
#'
#'
#'
#' 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)))
}


#' Interval coverage estimation
#'
#' @param CI_method a function object taking arguments x and alpha,
#'   returning a 2-element vector.
#' @param N The number of simulation replications to use for the coverage estimate
#' @param alpha 1-alpha is the intended coverage probability
#' @param n The sample size
#' @param lambda The true lambda values for the Poisson(lambda) model
#'
#' @return A scalar between 0 and 1, estimating the coverage probability
#' for the `CI_method` intervals for the given parameters.

estimate_coverage <- function(CI_method,
                              N = 10000,
                              alpha = 0.1,
                              n = 2,
                              lambda = 3) {
  cover <- 0
```

```
  for (loop in seq_len(N)) {
    y <- rpois(n, lambda)
    ci <- CI_method(y, alpha)
    cover <- cover + ((ci[1] <= lambda) && (lambda <= ci[2]))
  }
  cover / N
}


#' Question 1:
#'
#' 'Resulting Confidence Interval Construction Function'
#' @param y is the obvs
#' @param our key calue for the confidence
#'
non_approximated_v <- function(y, alpha){
  our_y <- mean(y)
  N <- length(y)
  a <- qnorm(alpha / 2)
  lambda_interval <- c( ((a ^ 2)/(2 * N)) + (( a /(2 * N)) * sqrt(a ^ 2 + 4 * our_y * N)) + ou
r_y,
                        ((a ^ 2)/(2 * N)) - (( a /(2 * N)) * sqrt(a ^ 2 + 4 * our_y * N)) + ou
r_y)
  pmax(lambda_interval, 0)
}

approximated_v <- function(y, alpha){
  confidence_int <- qnorm(c((1 - (alpha / 2)), (alpha / 2)))
  N <- length(y)
  our_t <- mean(y)

  lambda1_interval <- (our_t - (sqrt((our_t) / N)) * qnorm(c((1 - (alpha /2)), (alpha /2))))
  pmax(lambda1_interval, 0)
}

estimate_coverage_qn <- function(CI_method,
                                 N = 10000,
                                 alpha = 0.1,
                                 n = 2,
                                 lambda = 3){
  cover_1 <- 0
  cover_2 <- 0
  for (loop in seq_len(N)) {
    y <- rpois(n, lambda)
    ci <- c(CI_method[[1]](y, alpha), CI_method[[2]](y, alpha))
    cover_1 = cover_1 + ((ci[1] <= lambda) && (lambda <= ci[2]))
    cover_2 = cover_2 + ((ci[3] <= lambda) && (lambda <= ci[4]))
  }
  cover = c(cover_1 / N, cover_2 / N)

}


estimate_coverage_qn(c(non_approximated_v, approximated_v))


#' Q2
```

```
#' We now need to load and plot the data
#'
#' Prior Density
#'
#' We will compute the log density, p(theta) of our model
#'
#' @param theta vector of our parameter values
#' @param params vector of our parameter values of gamma
#'
log_prior_density <- function(theta, params){
  value <-
    (dnorm(theta[1],
           sd = params[1],
           mean = 0,
           log = TRUE) +

       dnorm(theta[2],
             sd = params[2],
             mean = 1,
             log = TRUE) +

       dlogexp(theta[3],
               params[3],
               log = TRUE) +

       dlogexp(theta[4],
               params[4],
               log = TRUE))

  value
}


#' Observation Likelihood
#'
#' This will compute our observation likelihood p(y|theta) of our model
#'
#' @param theta a vector with our parameter values
#' @param x a vector with the CAD weights of our obvs
#' @param y a vector with the the actual weights of our obvs
#' @param params a vector of the parameter values for gamma
#'
log_like <- function(theta, x, y){
  result <- sum(dnorm(y,
                      sd = sqrt(exp(theta[3]) + exp(theta[4])* x ** 2),
                      mean = (theta[1] + theta[2] * x),
                      log = TRUE))
  result

}




#Posterior Density

#We will use this to work out the posterior density p(theta|y) of our model
```

```
#' @param theta a vector with our parameter values
#' @param x a vector with the CAD weights of our obvs
#' @param y a vector with the the actual weights of our obvs
#' @param params a vector of the parameter values for gamma
#'
#'
log_posterior_density <- function(theta, x, y, params){
  value <- log_prior_density(theta, params = params) + log_like(theta, x, y)
  value
}



#'Importance Sampling

#'beta_i samples to be created and we calculate the log weights

#' @param N sample number
#' @param mu the mean vector of the importance distribution
#' @param S the co-variance matrix
#'
do_importance <- function(N, mu, S, ...){
  sample_for_x <- rmvnorm(N,
                          sigma = S,
                          mean = mu)
  log_weights1st <- log_posterior_density(sample_for_x, x = x, y = y, params = params) - dmvnor
m(sample_for_x, sigma = S, mean = mu, log = TRUE)
  log_weights_main <- log_weights1st - log_sum_exp(log_weights1st)

  result = data.frame(beta_1 = sample_for_x[,1],
                      beta_2 = sample_for_x[,2],
                      beta_3 = sample_for_x[,3],
                      beta_4 = sample_for_x[,4],
                      log_weights_main = log_weights_main)
  result
}


make_CI <- function(x, weights){
  alpha = 0.1
  ourquantiles <- wquantile(x,
                        probs = c((alpha / 2), (1 - (alpha / 2))),
                        weights = weights)
  result <- data.frame(Lwr = ourquantiles[1],
                       Upr = ourquantiles[2])
  result
}
```