StatComp Project 1: 3D printer materials estimation

Your Name (s0000000)

```
## NULL
```

report_files/figure-latex/unnamed-chunk-1-1.pdf

```
## $parameters
## [1] -0.08932974  1.07920922 -1.90378021  0.05569486
##
## $hessian
##              [,1]          [,2]         [,3]          [,4]
## [1,] 1.497239e+02  2506.6240441 1.594504e-02     0.4571632
## [2,] 2.506624e+03 70498.9820956 4.592262e-01  -237.9179862
## [3,] 1.594504e-02     0.4592262 4.301321e+01  1493.0697185
## [4,] 4.571632e-01  -237.9179862 1.493070e+03 64401.1139825
```

```
## $parameters
## [1]  -0.1612845   1.0828591 -13.5014797  -6.6136018
##
## $hessian
##               [,1]          [,2]          [,3]          [,4]
## [1,]  1.046323e+03  4.390261e+03 -1.891388e-03  1.667907e-03
## [2,]  4.390261e+03  6.408287e+04 -3.596977e-03  2.941629e-04
## [3,] -1.891388e-03 -3.596977e-03  3.785878e-04 -4.111378e-05
## [4,]  1.667907e-03  2.941629e-04 -4.111378e-05  4.299943e+01
```

```
## [1] -4.250559
```

```
## [1] -182.7504
```

```
## [1] -187.0009
```

```
## $mode
## [1] -0.08571000  0.01445975 -0.41467604 -1.80060467
##
## $hessian
##              [,1]       [,2]       [,3]       [,4]
## [1,] -127.7348578  -8.367970  -0.143434  0.2198937
## [2,]   -8.3679700 -99.576600   2.898706 -1.9042061
## [3,]   -0.1434340   2.898706 -37.915838 -4.2745728
```

```
## [4,]    0.2198937  -1.904206  -4.274573 -3.2018876
##
## $S
##               [,1]          [,2]         [,3]         [,4]
## [1,]  0.0078764786 -0.0006923513 -0.000223817  0.001251477
## [2,] -0.0006923513  0.0103146330  0.001751764 -0.008520435
## [3,] -0.0002238170  0.0017517643  0.031345967 -0.042904552
## [4,]  0.0012514774 -0.0085204349 -0.042904552  0.374747221

##            beta1        beta2      beta3     beta4   log_weights
## 1   -2.000929238  0.333777197  3.2262650 7.8423556   -88.4374544
## 2   -1.376861598 -1.150855566  0.4937029 0.3485214  -174.8743558
## 3   -0.645743723 -0.185377968  0.3008264 7.6673585   -47.4262442
## 4    0.107774745 -0.084108101  1.6415151 1.0381239    -9.0526868
## 5   -0.132088037  1.476787424  0.8049057 0.2770376   -63.6871075
## 6    0.385667890 -0.351512874  0.5934537 0.3436501    -2.8440404
## 7    0.428365903 -0.174018234  1.6747564 0.7910728   -12.3049519
## 8   -0.658503426  1.250236604  0.7620343 2.5804195   -34.4545462
## 9   -1.201582430 -0.466116096  0.7638748 0.6764036   -53.8690321
## 10   1.348707012 -0.022764701  1.2766326 0.3897025   -60.4334268
## 11  -0.729217277  0.998068909  3.5200728 3.4863791   -52.1209175
## 12  -1.380637050  2.049960694  2.7645361 0.9736363   -82.7604295
## 13   0.703607779 -0.971385229  0.3341530 1.0502733   -54.4533089
## 14  -1.198495857  0.190018999  3.6608886 0.3556267   -41.4963344
## 15  -0.738440754  0.046563939  0.3614628 0.6816193   -23.6584221
## 16   0.872755412  0.969545014  1.4679203 0.1569927   -50.5153871
## 17  -0.053996737  1.064773214  2.2551016 0.8262842   -27.7860741
## 18  -2.699929809  0.060966639  1.7749135 1.0468688  -149.9692628
## 19   0.157412540  0.431565373  0.6726368 3.7060930   -20.3742449
## 20   0.470393400 -1.242670271  3.9811688 3.3349542   -57.7914781
## 21   0.824073964 -1.662629402  0.5659179 1.8879920   -67.1556659
## 22   0.043722008  0.348012304 11.7000551 0.4411456   -79.9411080
## 23  -2.113200115  0.273695272  0.5027829 1.5621156  -187.0346407
## 24  -0.812384724  2.212055480  0.8836396 0.6204343  -122.5127947
## 25  -0.166261491  0.862563384  1.1022356 0.1967903   -12.0146975
## 26  -0.004620768  0.760242168  1.0397610 2.0856324   -16.3879075
## 27  -0.146472627 -0.057887335  1.6199082 2.6991682   -21.5308551
## 28  -1.246395498 -0.033487525  0.9314971 0.4681715   -45.4241018
## 29  -1.034359361 -0.630731954  1.7982388 0.6594675   -34.8297398
## 30  -0.784887810  0.163416319  0.2903366 2.8458841   -35.3167582
## 31  -0.484595416  0.189128812  1.0523296 0.9997594    -4.4689608
## 32   1.809382042 -0.825327957  3.1439201 1.0320769   -76.9727586
## 33  -0.835205805 -0.068763649  2.1101767 0.6534307   -21.8013842
## 34  -0.772082235  0.152764107  2.6874609 0.9291749   -27.5241732
## 35  -1.387026554 -1.306675904  0.4637567 0.5903096  -168.6112335
## 36  -0.021427065  0.670498071  0.6475126 0.3282828    -3.7646044
## 37   0.607105995  0.275456969  3.1814818 0.1859122   -26.8968794
## 38   0.087319089  1.353361894  2.0630259 0.4349375   -37.6193357
## 39   0.732528487 -0.871926870  0.6354655 3.2789862   -38.6866047
## 40  -0.290145312  0.828546145  0.7473455 0.2067257   -13.9567246
## 41  -0.848815697 -1.088519862  0.6161341 0.7144007   -55.2541819
## 42  -0.153357891 -0.243247229  6.6339609 0.2500740   -51.2010840
## 43  -0.414824301  0.349081528  5.0959304 1.0925582   -45.6951832
## 44   1.239150708 -1.644555536  4.2476102 0.5012952   -76.9701991
```

```
## 45  -0.276431085 -1.109418760  1.1432434 5.9616009   -42.0863941
## 46   2.422163355 -1.076828902  1.6257043 4.0089194  -129.3161340
## 47  -0.195656817 -0.218174798  0.7372871 1.8181740    -4.0616842
## 48   1.397429411  0.687619761  1.3773867 0.7394342   -71.8717440
## 49   0.498348686 -0.549536918  0.7563459 2.9937101   -24.2278874
## 50   0.442013088  0.241016294  0.7744458 2.5371284   -18.9528138
## 51   1.334912585 -0.869271764  1.0570552 1.0502906   -68.9788394
## 52  -0.578355728 -0.998738656  0.9975702 1.9261282   -27.0870640
## 53   1.476842279 -1.909152788  0.4953754 0.7323987  -211.9864020
## 54  -1.663157031 -0.750533442  0.4596216 0.4855031  -191.6263411
## 55  -2.188834599  0.213418550  0.5315686 4.5744716  -152.2101796
## 56   0.795955949 -1.453529565  1.1033990 0.5522409   -62.9767707
## 57   0.888281169  0.053070415  0.5729117 1.5502203   -39.6293862
## 58   0.152608159 -0.164617582  7.5375004 0.5889666   -59.2337278
## 59  -0.470786973 -1.545936924  0.9602835 2.4359975   -40.8358392
## 60  -2.071387851 -0.250065120  0.3067720 4.2288803  -188.3672997
## 61   1.357895539  0.334502847  4.1759341 0.4200767   -56.0461193
## 62   0.950651725 -0.585011509  1.3784470 0.7412658   -33.8561514
## 63  -0.278543083  0.546115158  0.2714924 0.7780889   -10.1965395
## 64   0.171007374 -0.403467479  1.1103324 0.7269622    -1.6913893
## 65   1.618343936  0.714188601 19.4114941 0.4515462  -117.0294718
## 66   0.814365915  2.098030810  1.3511824 0.3385528  -131.6625759
## 67  -1.006322502 -0.035414565  3.7029299 2.1178480   -47.4979122
## 68  -2.138368328 -0.700354109  0.9909844 0.2326702  -193.0616472
## 69   0.694529646 -2.461335475  1.1540642 0.6762299  -129.6866601
## 70  -0.491164086 -0.283647452  1.3699782 1.4863546   -13.8118424
## 71  -0.225603711 -1.924950430  0.2371104 0.2300042  -282.9222315
## 72   0.761863447 -0.243614982  1.3095409 0.2103616   -18.1388263
## 73  -0.535588007  0.562451973  0.8366695 0.8912449    -8.1063461
## 74  -0.072061472  1.210909807  0.5406966 1.9662466   -24.6926804
## 75   0.898599606 -1.189317904  1.1289171 0.9888410   -47.6834663
## 76   1.029140719  0.914774868  0.9975467 1.1456928   -57.0282476
## 77  -0.720153545 -0.198124330  0.3572895 0.3802388   -29.1607005
## 78  -1.220813089  0.836207704  3.0494822 0.6616988   -45.5030315
## 79  -1.128977398 -0.087931332  9.4112303 7.7007149   -95.8199223
## 80  -1.719800337 -0.356906656  4.6333516 0.9624780   -66.6802396
## 81   1.597413242 -0.333585393  1.8312264 1.2513729   -67.3088879
## 82   3.229069495  0.920452567  0.2992311 0.5466857 -1075.9090881
## 83   0.370235285 -1.901000647  0.1645715 0.3256857  -277.5497718
## 84  -0.347929607  1.238902149  0.7601819 1.1763080   -26.7847838
## 85  -0.064606910 -0.705237097  3.9047637 0.3340338   -34.8976809
## 86  -0.228433519 -0.347828081  1.7025525 4.9889954   -36.6982217
## 87   0.513814526  1.382373161  2.1449088 0.5354839   -46.4730749
## 88   0.081543800  1.376079111  0.2098703 1.3837133   -55.0769327
## 89  -0.156790317  0.877783861  2.1173527 1.3512815   -24.9180382
## 90   1.492811117 -1.525493800  2.4861058 0.2060715   -95.0164200
## 91   0.587716257  0.089642296  2.6307307 1.0820013   -28.0468004
## 92  -1.568699836 -2.007823184  1.7176782 0.9292508  -121.7759197
## 93  -0.571018394 -0.311068460  0.5109850 0.8544131    -7.5469796
## 94  -0.931305071 -1.983009479  0.8028400 2.8441830   -74.3766385
## 95   1.877329566  0.002606196  0.9224981 2.6194969  -112.1541947
## 96   0.053571017 -0.434898409  0.1759954 0.2826077   -29.3729428
## 97   0.406308512 -1.459653968  2.8532462 0.2601673   -52.4335459
## 98  -0.193570558 -0.002335957  0.9872522 1.1640989    -0.3499919
```

```
## 99   0.598511131 -0.126212437  0.7799421 1.1738951   -14.4878548
## 100 -0.433641942  1.537412419  0.1141495 2.7926881   -67.1415035
```

1 Classical estimation

2 Bayesian estimation

3 Code appendix

```r
#' Orlagh Keane, S2084384
#' 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)))
}


#' wquantile
#'
#' Calculates empirical sample quantiles with optional weights, for given probabilities.
#' Like in quantile(), the smallest observation corresponds to a probability of 0 and the largest to a p
#' Interpolation between discrete values is done when type=7, as in quantile().
#' Use type=1 to only generate quantile values from the raw input samples.
#'
#' @param x numeric vector whose sample quantiles are wanted
```

```r
#' NA and NaN values are not allowed in numeric vectors unless na.rm is TRUE
#' @param probs numeric vector of probabilities with values in [0,1]
#' @param na.rm logical; if true, any NA and NaN's are removed from x before the quantiles are computed
#' @param type numeric, 1 for no interpolation, or 7, for interpolated quantiles. Default is 7
#' @param weights    numeric vector of non-negative weights, the same length as x, or NULL.
#' The weights are normalised to sum to 1.
#' If NULL, then wquantile(x) behaves the same as quantile(x), with equal weight for each sample value

wquantile <- function (x, probs = seq(0, 1, 0.25), na.rm = FALSE, type = 7,
                       weights = NULL, ...)
{
  if (is.null(weights) || (length(weights) == 1)) {
    weights <- rep(1, length(x))
  }
  stopifnot(all(weights >= 0))
  stopifnot(length(weights) == length(x))
  if (length(x) == 1) {
    return(rep(x, length(probs)))
  }
  n <- length(x)
  q <- numeric(length(probs))
  reorder <- order(x)
  weights <- weights[reorder]
  x <- x[reorder]
  wecdf <- pmin(1, cumsum(weights)/sum(weights))
  if (type == 1) {
  }
  else {
    weights2 <- (weights[-n] + weights[-1])/2
    wecdf2 <- pmin(1, cumsum(weights2)/sum(weights2))
  }
  for (pr_idx in seq_along(probs)) {
    pr <- probs[pr_idx]
    if (pr <= 0) {
      q[pr_idx] <- x[1]
    }
    else if (pr >= 1) {
      q[pr_idx] <- x[n]
    }
    else {
      if (type == 1) {
        j <- 1 + pmax(0, pmin(n - 1, sum(wecdf <= pr)))
        q[pr_idx] <- x[j]
      }
      else {
        j <- 1 + pmax(0, pmin(n - 2, sum(wecdf2 <= pr)))
        g <- (pr - c(0, wecdf2)[j])/(wecdf2[j] - c(0,
                                                   wecdf2)[j])
        q[pr_idx] <- (1 - g) * x[j] + g * x[j + 1]
      }
    }
  }
  q
```

```r
}

#' Compute empirical weighted cumulative distribution
#'
#' Version of `ggplot2::stat_ecdf` that adds a `weights` property for each
#' observation, to produce an empirical weighted cumulative distribution function.
#' The empirical cumulative distribution function (ECDF) provides an alternative
#' visualisation of distribution. Compared to other visualisations that rely on
#' density (like [geom_histogram()]), the ECDF doesn't require any
#' tuning parameters and handles both continuous and discrete variables.
#' The downside is that it requires more training to accurately interpret,
#' and the underlying visual tasks are somewhat more challenging.
#'
# @inheritParams layer
# @inheritParams geom_point
#' @param na.rm If `FALSE` (the default), removes missing values with
#'    a warning.  If `TRUE` silently removes missing values.
#' @param n if NULL, do not interpolate. If not NULL, this is the number
#'   of points to interpolate with.
#' @param pad If `TRUE`, pad the ecdf with additional points (-Inf, 0)
#'   and (Inf, 1)
#' @section Computed variables:
#' \describe{
#'   \item{x}{x in data}
#'   \item{y}{cumulative density corresponding x}
#' }
#' @seealso wquantile
#' @export
#' @examples
#' library(ggplot2)
#'
#' n <- 100
#' df <- data.frame(
#'   x = c(rnorm(n, 0, 10), rnorm(n, 0, 10)),
#'   g = gl(2, n),
#'   w = c(rep(1/n, n), sort(runif(n))^sqrt(n))
#' )
#' ggplot(df, aes(x, weights = w)) + stat_ewcdf(geom = "step")
#'
#' # Don't go to positive/negative infinity
#' ggplot(df, aes(x, weights = w)) + stat_ewcdf(geom = "step", pad = FALSE)
#'
#' # Multiple ECDFs
#' ggplot(df, aes(x, colour = g, weights = w)) + stat_ewcdf()
#' ggplot(df, aes(x, colour = g, weights = w)) +
#'   stat_ewcdf() +
#'   facet_wrap(vars(g), ncol = 1)

stat_ewcdf <- function(mapping = NULL, data = NULL,
                       geom = "step", position = "identity",
                       ...,
                       n = NULL,
                       pad = TRUE,
```

```r
                        na.rm = FALSE,
                        show.legend = NA,
                        inherit.aes = TRUE) {
  ggplot2::layer(
    data = data,
    mapping = mapping,
    stat = StatEwcdf,
    geom = geom,
    position = position,
    show.legend = show.legend,
    inherit.aes = inherit.aes,
    params = list(
      n = n,
      pad = pad,
      na.rm = na.rm,
      ...
    )
  )
}


#' @title StatEwcdf ggproto object
#' @name StatEwcdf
#' @rdname StatEwcdf
#' @aliases StatEwcdf
#' @format NULL
#' @usage NULL
#' @export
#' @importFrom ggplot2 aes after_stat has_flipped_aes Stat
NULL

StatEwcdf <- ggplot2::ggproto(
  "StatEwcdf", ggplot2::Stat,
  required_aes = c("x|y", "weights"),
  dropped_aes = c("weights"),

  default_aes = ggplot2::aes(y = ggplot2::after_stat(y)),

  setup_params = function(data, params) {
    params$flipped_aes <-
      ggplot2::has_flipped_aes(data,
                               params,
                               main_is_orthogonal = FALSE,
                               main_is_continuous = TRUE)

    has_x <- !(is.null(data$x) && is.null(params$x))
    has_y <- !(is.null(data$y) && is.null(params$y))
    if (!has_x && !has_y) {
      rlang::abort("stat_ewcdf() requires an x or y aesthetic.")
    }
    has_weights <- !(is.null(data$weights) && is.null(params$weights))
    #    if (!has_weights) {
    #      rlang::abort("stat_ewcdf() requires a weights aesthetic.")
```

```
  #    }

  params
},

compute_group = function(data, scales, n = NULL, pad = TRUE, flipped_aes = FALSE) {
  data <- flip_data(data, flipped_aes)
  # If n is NULL, use raw values; otherwise interpolate
  if (is.null(n)) {
    x <- unique(data$x)
  } else {
    x <- seq(min(data$x), max(data$x), length.out = n)
  }

  if (pad) {
    x <- c(-Inf, x, Inf)
  }
  if (is.null(data$weights)) {
    data_ecdf <- ecdf(data$x)(x)
  } else {
    data_ecdf <-
      spatstat.geom::ewcdf(
        data$x,
        weights = data$weights / sum(abs(data$weights))
      )(x)
  }

  df_ecdf <- vctrs::new_data_frame(list(x = x, y = data_ecdf), n = length(x))
  df_ecdf$flipped_aes <- flipped_aes
  ggplot2::flip_data(df_ecdf, flipped_aes)
}
)



# MY ANSWERS

# QUESTION 1

# Load the data
load("/Users/orlagh/project01/filament1.rda")

# Plot the data
library(ggplot2)

ggplot(data = filament1, aes(x = CAD_Weight, y = Actual_Weight, color = Material)) +
  geom_point() +
  labs(x = "CAD_Weight (g)", y = "Actual_Weight (g)") +
  ggtitle("Relationship between CAD_Weight and Actual_Weight") +
  theme_minimal()

# QUESTION 2
```

```r
# Define function neg_log_like
neg_log_like <- function(beta, data, model) {
  xi <- data$CAD_Weight
  yi <- data$Actual_Weight
  if(model == "A") {
    sigmasq <- exp(beta[3] + beta[4] * xi)
  } else if(model == "B") {
    sigmasq <- exp(beta[3]) + exp(beta[4]) * xi^2
  }

  nll <- -sum(dnorm(yi, mean = beta[1] + beta[2] * xi, sd = sqrt(sigmasq), log = TRUE))
  return(nll)
}

# Define function filament1_estimate
filament1_estimate <- function(data, model) {
  # Define initial values for beta
  if(model == "A") {
    init_beta <- c(-0.1, 1.07, -2, 0.05)
  } else if(model == "B") {
    init_beta <- c(-0.15, 1.07, -13.5, -6.5)
  }

  # Run optimization
  fit <- optim(par = init_beta, fn = neg_log_like, data = filament1, model = model, method = "BFGS", hes

  # Return best set of parameters found and estimate of the Hessian
  return(list(parameters = fit$par, hessian = fit$hessian))
}

# Estimate Model A
fit_A <- filament1_estimate(filament1, "A")

# Estimate Model B
fit_B <- filament1_estimate(filament1, "B")

# Print the estimated parameters
print(fit_A)
print(fit_B)







# Question 3

library(mvtnorm)

# Poisson parameter confidence interval A

CI_A <- function(y, alpha = 0.05) {
  n <- length(y)
```

```r
  lambda_hat <- mean(y)
  z <- qnorm(c(1 - alpha / 2, alpha / 2))
  lambda_hat - z * sqrt(lambda_hat / n)
}

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
}

# 3.1 Log-density for prior distribution

log_prior_density <- function(theta, params) {
    dnorm(theta[1], sd = sqrt(params[1]), log = TRUE) +
    dnorm(theta[2], mean = 1, sd = sqrt(params[2]), log = TRUE) +
    dlogexp(theta[3], rate = params[3], log = TRUE) +
    dlogexp(theta[4], rate = params[4], log = TRUE)
}

# 3.2 Observation log-likelihood

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

# 3.3 Log-density for the posterior distribution

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

# 3.4 Posterior mode

posterior_mode <- function(theta_start, x, y, params) {
  opt <- optim(theta_start, log_posterior_density,
      x = x, y = y, params = params,
      control = list(fnscale = -1),
      hessian = TRUE
)
if (opt$convergence != 0) {
  warning(paste0(
    "Optimisation may not have been successful; 'convergence' = ",
```

```r
    opt$convergence
  ))
}
if (any(eigen(opt$hessian)$values > 0)) {
  warning(paste0("Positive eigenvalues detected in the hessian; result isn't a local maximum"))
  }
  list(
    mode = opt$par,
    hessian = opt$hessian,
    S = solve(-opt$hessian)
) }

# 3.6 Importance sampler

do_importance <- function(N, mu, S, x, y, params = c(1, 1, 1, 1)) {
  samples <- mvtnorm::rmvnorm(N, mean = mu, sigma = S)
log_weights <- numeric(N)
for (k in seq_len(N)) {
    log_weights[k] <-
      log_posterior_density(samples[k, ], x = x, y = y, params = params) -
      mvtnorm::dmvnorm(samples[k, ], mean = mu, sigma = S, log = TRUE)
}
  # Normalise the weights
  log_weights <- log_weights - log_sum_exp(log_weights)
  # Convert theta to beta-values
  samples[, 3:4] <- exp(samples[, 3:4])
  colnames(samples) <- paste0("beta", seq_len(ncol(samples)))
  cbind(as.data.frame(samples),
        data.frame(log_weights = log_weights))
}

# Log-Exponential density

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

# Log-Sum-Exp

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


library(mvtnorm)

# Define some example data
set.seed(42)
theta_example <- c(0.5, 1, log(0.5), log(1.5))
x_example <- rnorm(100)
```

```r
y_example <- rnorm(100)

# Define the parameters for the prior distribution
params_example <- c(1, 1, 1, 1)

# Run the log_prior_density function with the example data
log_prior_density(theta_example, params_example)

# Run the log_like function with the example data
log_like(theta_example, x_example, y_example)

# Run the log_posterior_density function with the example data
log_posterior_density(theta_example, x_example, y_example, params_example)

# Run the posterior_mode function with the example data
posterior_mode(theta_example, x_example, y_example, params_example)

# Define the mean vector and covariance matrix for the importance distribution
mu_example <- c(0, 0, 0, 0)
S_example <- diag(4)

# Run the do_importance function with the example data
do_importance(100, mu_example, S_example, x_example, y_example, params_example)
```