MA40198 Coursework 2023

Luka Philip, Harry Lyness

2023-12-06

Question 1 [4 marks]

Consider the following observed sample:

```
y_sample_q1 <- scan("http://people.bath.ac.uk/kai21/ASI/CW_2023/y_sample_q1.txt")
```

Plot 40 contours of the negative loglikelihood function of the parameter $\boldsymbol{\lambda}$ over the region defined by $-\pi/2 < \lambda_1 < \pi/2$ and $0 < \lambda_2 < 50$. The contours should be sufficiently smooth and cover the entire region. You should indicate a smaller region delimited by a contour that contains the global minimum.

Solution to Question 1

Log Likelihood Function

```
log_likelihood <- function(y, l1, l2, N = 10000) {
  val_of_sumsum <- function(y, l2, N) {
    value <- matrix(nrow = length(y), ncol = N+1)
    for (i in 1:length(y)) {
      for (j in 0:N) {
        value[i, j+1] <- -log(1 + (y[i] / (l2 + 2 * j)) ^ 2)
      }
    }
    return(sum(value))
  }

  k <- length(y)
  exp_value <-
    l1 * sum(y) + k * l2 * log(cos(l1)) + k * l2 * log(2) + k * 2 * log(gamma(l2 /2)) - k *
    log(gamma(l2)) + val_of_sumsum(y, l2, N) - k*2*log(2) - k*log(pi)
  exp_value <-
    return(exp_value)
}
```

Log Likelihood Expression

```
log_likelihood_expr <- expression(l1 * y + l2 * log(cos(l1)) + l2 * log(2) + 2 * log(gamma(l2 /2)) -
    log(gamma(l2)) - 2*log(2) - log(pi))
sumsum_expr <- expression(- log(1+(y/(l2+2*j))^2))
```

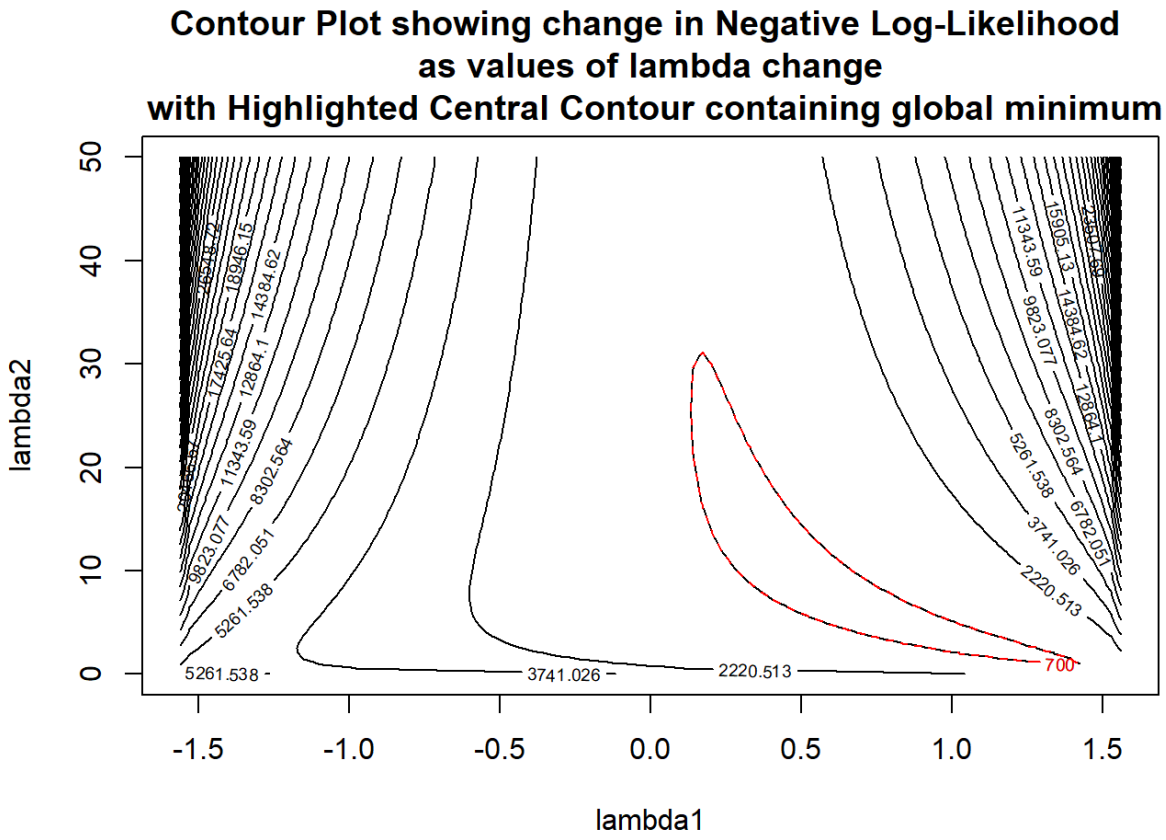
Negative log likelihood Function

```
nll_function <- function(y, lambda, N = 10000) {
  l1 <- lambda[1]
  l2 <- lambda[2]
  -log_likelihood(y, l1, l2, N)
}
```

Calculating data for a contour plot

```
tol <- 0.01
l1_grid <- seq(-pi/2+tol,pi/2-tol,length=100)
l2_grid <- seq(0+tol,50-tol,length=100)
M = matrix(NA, nrow=100,ncol=100)
for (i in 1:100){
  for (j in 1:100){
    M[i,j] = nll_function(lambda = c(l1_grid[i],l2_grid[j]),y = y_sample_q1,N=10000)
  }
}
```

```
contour(x = l1_grid,
        y = l2_grid,
        levels = seq(700,60000,length=40),
        z=M,
        xlab = "lambda1",
        ylab = "lambda2",
        main = "Contour Plot showing change in Negative Log-Likelihood \n as values of lambda change
\n with Highlighted Central Contour containing global minimum")
contour(x = l1_grid,
        y = l2_grid,
        levels = 700,
        z=M,
        add = TRUE,
        col = "red",
        lty = 2)
```




```
thetaM <- seq(-pi/2+tol,pi/2-tol,length=100)
```

This plot shows us the region where the MLE of lambda1 and lambda2 lies, the red boundary highlights the region of the graph that contains the global minimum of the negative loglikelihood function.

Question 2 [6 marks]

Find the maximum likelihood estimate $\widehat{\lambda} = (\hat{\lambda}_1, \hat{\lambda}_2)^T$ by picking the best out of 100 optimisations (using the BFGS algorithm) where each optimisation uses a different initial value. The following data frame gives the list of initial values to be used.

```
L0 <-read.table("http://people.bath.ac.uk/kai21/ASI/CW_2023/starting_vals_q2.txt")
```

Solution to Question 2

We begin by parameterising our expressions to allow for unconstrained optimisation. We take our values of theta such that

$$\lambda_1 = atan(\theta_1)$$

$$\lambda_2 = exp(\theta_2)$$

. We also transform L0 into lists of θ_1 and θ_2

```
log_likelihood_expr_param <- expression(atan(theta1) * y
                            + exp(theta2) * log(cos(atan(theta1)))
                            + exp(theta2) * log(2)
                            + 2 * log(gamma(exp(theta2) /2))
                            - log(gamma(exp(theta2)))
                            - 2*log(2)
                            - log(pi))

sumsum_expr_param <- expression(-log(1+(y/(exp(theta2)+2*j))^2))

theta1_list = tan(L0$lambda1)
theta2_list = log(L0$lambda2)
```

Finding the derivatives by splitting the components of the log-likelihood into two:

```
deriv_log_likelihood_expr_param <- deriv(expr         = log_likelihood_expr_param,
                                         namevec      = c("theta1","theta2"),
                                         function.arg = c("theta1","theta2","y"),
                                         hessian      = T)

deriv_sumsum_expr_param<- deriv(expr         = sumsum_expr_param,
                                namevec      = c("theta2"),
                                function.arg = c("theta2","y","j"),
                                hessian      = T)
```

Parameterising our Log likelihood, Gradient, NLL and gradient of NLL functions:

```
# Parametrised Log likelihood function
fcn_log_likelihood_expr_param=function(theta , y , N=10000){
  total<-0
  for (i in y){
    total = total + deriv_log_likelihood_expr_param(y = i,theta1=theta[1], theta2=theta[2])
  }
  for (i in y){
    total = total + sum(deriv_sumsum_expr_param(y = i, theta2=theta[2],j=0:N))
  }
  return(total)
}
# Parametrised gradient of the log likelihood
grad_log_likelihood_expr_param <- function(theta, y, N = 10000) {
  res1 <- lapply(y, function(i) deriv_log_likelihood_expr_param(y = i, theta1 = theta[1], theta2 = the
ta[2]))
  total_loop_1 <- Reduce(`+`, lapply(res1, function(res) attr(res, "gradient")))

  res2 <- lapply(y, function(i) deriv_sumsum_expr_param(y = i, theta2 = theta[2], j = 0:N))
  total_loop_2 <- Reduce(`+`, lapply(res2, function(res) colSums(attr(res, "gradient"))))

  total_loop_1[2] <- total_loop_1[2] + total_loop_2[1]
  total <- total_loop_1
  return(total)
}
# Parametrised negative log likelihood function
fcn_neg_log_likelihood_expr_param = function(theta,
                                             y,
                                             N = 10000) {
  -1*fcn_log_likelihood_expr_param(theta, y, N)
}
# Parametrised gradient of the negative log likelihood
grad_neg_log_likelihood_expr_param = function(theta,
                                             y,
                                             N = 10000) {
  -1*grad_log_likelihood_expr_param(theta, y, N)
}
```

Using `y_sample_q1` and `L0`, create functions to find the parameters that minimize the negative log likelihood using the provided 100 starting values.

```
# Function to calculate the parameters with the smallest negative log likelihood
min_nll = function(fit, N_samples){
  values = numeric(N_samples)
  for (i in c(1:N_samples)){
    values[i] = (fit[[i]]$value)
  }
  return(fit[[which.min(values)]]$par)
}

min_nll_object = function(fit, N_samples){
  values = numeric(N_samples)
  for (i in c(1:N_samples)){
    values[i] = (fit[[i]]$value)
  }
  return(fit[[which.min(values)]])
}

# Function to calculate the fit for all parameters for the 100 starting samples
fit_optim<- function(par1_list = theta1_list,
                     par2_list = theta2_list,
                     fn ,
                     gr ,
                     method = "BFGS",
                     hessian   = T,
                     y = y_sample_q1,
                     N=10000,
                     N_samples = 100){
  fit <- vector("list",
              length = N_samples)
  for (i in c(1:N_samples)){
    fit[[i]]<-try(
          optim(par = c(par1_list[i], par2_list[i]),
          fn  = fn,
          gr  = gr,
          y =y,
          N=N,
          method  =method,
          hessian = hessian),
                    silent=T)
  }
  return(fit)
}
```

Reparameterise the starting values and save all the ‘fits’ in a object for easier re-use.

```
fit_N_samples = fit_optim(
  par1_list = theta1_list,
  par2_list = theta2_list,
  fn = fcn_neg_log_likelihood_expr_param,
  gr = grad_neg_log_likelihood_expr_param,
  method = "BFGS",
  hessian   = T,
  y = y_sample_q1,
  N = 10000,
  N_samples = 100
)
```

Then calculate the MLE of thetas’ (and thus the MLE of lambda1 and lambda2) by finding the estimate with the minimum NLL value.

```
MLE_theta = min_nll(fit_N_samples, N_samples = 100)
MLE_lambda = c(atan(MLE_theta[1]), exp(MLE_theta[2]))
print(paste0(
  ' MLE of lambda_1 is ',
  MLE_lambda[1],
  '. MLE of lambda_2 is ',
  MLE_lambda[2]
))
```

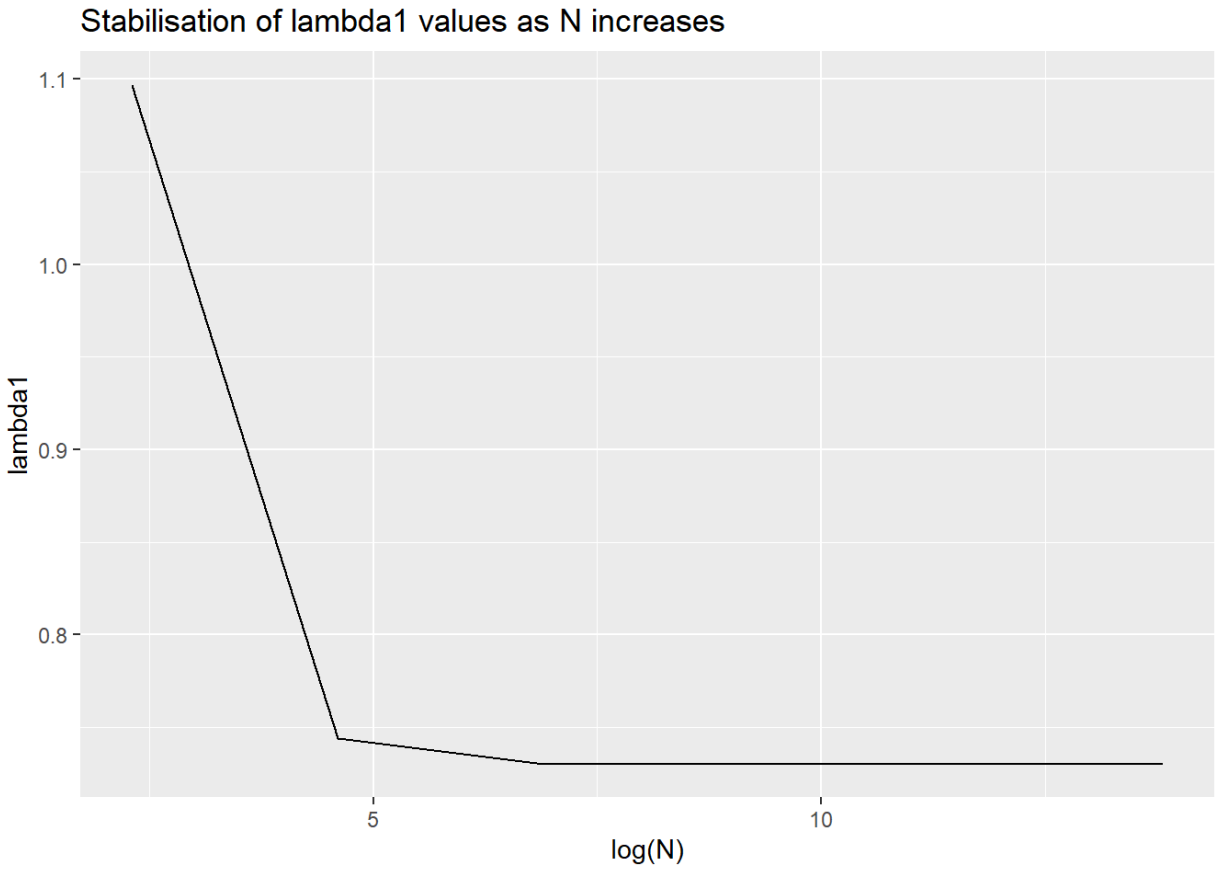
```
## [1] " MLE of lambda_1 is 0.73022040813154. MLE of lambda_2 is 5.88577069731574"
```

Question 3 [4 marks]

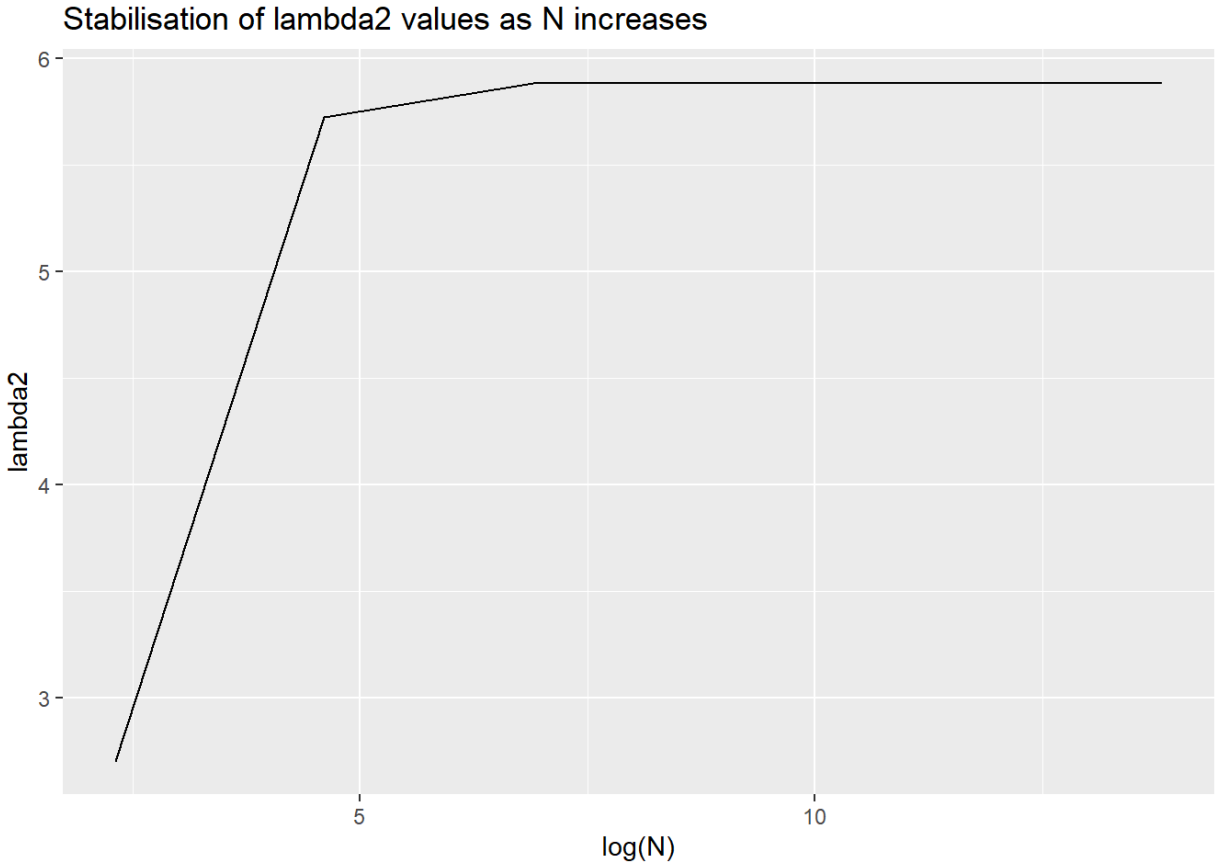
Check the sensitivity of the MLE to the choice of N by plotting (separately) the values of $\hat{\lambda}_1$ and $\hat{\lambda}_2$ as function of $\log_{10}(N)$. You should use the values $10^1, 10^2, 10^3, 10^4, 10^5, 10^6$ for N. What conclusions can you make from these two plots?

Solution to Question 3

```
N_list = c(10,100,1000,10000,100000,1000000)
theta_ests = matrix(0, nrow = length(N_list), ncol = 2)
for (i in c(1:length(N_list))){
  fit = optim(par = c(MLE_theta[1],MLE_theta[2]),
        fn = fcn_neg_log_likelihood_expr_param,
        gr = grad_neg_log_likelihood_expr_param,
        y = y_sample_q1,
        N = N_list[i],
        method="BFGS",
        hessian = T)
  theta_ests[i,1] = fit$par[1]
  theta_ests[i,2] = fit$par[2]
}
theta_ests = data.frame(theta_ests, N_list)
ggplot(theta_ests, aes(x=log(N_list), y = atan(X1))) + geom_line() + ylab('lambda1') + xlab('log(N)')
+ labs(title="Stabilisation of lambda1 values as N increases")
```




```
ggplot(theta_ests, aes(x=log(N_list), y = exp(X2))) + geom_line() + ylab('lambda2') + xlab('log(N)')
+ labs(title="Stabilisation of lambda2 values as N increases")
```



We notice that as N increases the values of lambda1 and lambda2 become less sensitive to change highlighting the need for large enough values of N as our MLE is sensitive to N. It is more sensitive to N = 10 and N = 100 but less so at N = 1000 and after.

Question 4 [4 marks]

Compute the maximum likelihood estimate of the mean parameter

$$\mu(\boldsymbol{\lambda}_*) = E[Y|\boldsymbol{\lambda}_*] = \int_{\mathcal{R}} y\, f(y|\boldsymbol{\lambda}_*) dy\,.$$

Also compute an asymptotic 95% confidence interval for $\mu(\boldsymbol{\lambda}_*)$. State clearly any assumptions you have made.

Solution to Question 4

We attempt to evaluate:

$$\mu(\lambda_*) = \int_R y \exp(\lambda_1 y + \lambda_2 \log(\cos(\lambda_1)) g(y|\lambda_2) dy$$

Taking the derivative with respect to λ_1 gives

$$\frac{\partial}{\partial \lambda_1} \exp(\lambda_1 y + \lambda_2 \log(\cos(\lambda_1)) = (y - \lambda_2 \tan(\lambda_1)) \exp(\lambda_1 y + \lambda_2 \log(\cos(\lambda_1))$$

as

$$\frac{\partial}{\partial \lambda_1} \lambda_2 \log(\cos(\lambda_1)) = -\lambda_2 \tan(\lambda_1)$$

Multiplying through by $g(y|\lambda_2)$ gives

$$\frac{\partial}{\partial \lambda_1} \exp(\lambda_1 y + \lambda_2 \log(\cos(\lambda_1)) g(y|\lambda_2) = (y - \lambda_2 \tan(\lambda_1)) \exp(\lambda_1 y + \lambda_2 \log(\cos(\lambda_1)) g(y|\lambda_2)$$

Then integrating both sides w.r.t. y

$$\frac{\partial}{\partial \lambda_1} \int_{\mathbb{R}} \exp(\lambda_1 y + \lambda_2 \log(\cos(\lambda_1)) g(y|\lambda_2) dy = \int_{\mathbb{R}} (y - \lambda_2 \tan(\lambda_1)) \exp(\lambda_1 y + \lambda_2 \log(\cos(\lambda_1)) g(y|\lambda_2) dy \quad (1)$$

We know that the integral of the distribution

$$\int_R \exp(\lambda_1 y + \lambda_2 \log(\cos(\lambda_1)) g(y|\lambda_2) dy = 1$$

so the LHS of the integral (1) evaluates to 0. Expanding out the RHS of (1) gives

$$0 = \int_{\mathbb{R}} y \exp(\lambda_1 y + \lambda_2 \log(\cos(\lambda_1)) g(y|\lambda_2) dy - \lambda_2 \tan(\lambda_1) \int_{\mathbb{R}} \exp(\lambda_1 y + \lambda_2 \log(\cos(\lambda_1)) g(y|\lambda_2) dy$$

$$0 = \mu(\lambda_*) - \lambda_2 \tan(\lambda_1)$$

so

$$\lambda_2 \tan(\lambda_1) = \mu(\lambda_*)$$

Now finding our Jacobian by taking derivatives of our $\mu(\lambda_*)$ w.r.t. λ_1 and λ_2 gives

$$J(\lambda_1, \lambda_2) = (\lambda_2 \sec^2(\lambda_1), \tan(\lambda_1))$$

Putting this in terms of our earlier parametrisation by θ gives a more useful form,

$$\mu(\theta) = \exp(\theta_2)\theta_1$$

$$J(\theta) = (\exp(\theta_2), \theta_1 \exp(\theta_2))$$

We have made the assumptions of A3, A4 and A5 from the lecture notes ‘Likelihood Theory’ section. These being the assumption of our unknown estimator being within the parameter space, identifibility and the interchangability of integrals. Functions to find the Hessian

```
hess_log_likelihood_expr_param <- function(theta, y, N = 10000) {
  hessian_sum_1 <- matrix(0, nrow = 2, ncol = 2)
  hessian_sum_2 <- 0

  for (i in y) {
    hess_1 <- deriv_log_likelihood_expr_param(y = i, theta1 = theta[1], theta2 = theta[2])
    hessian_1 <- matrix(attr(hess_1, "hessian"), nrow = 2, ncol = 2)
    hessian_sum_1 <- hessian_sum_1 + hessian_1
  }

  for (i in y) {
    for (j in 0:N) {
      hess_2 <- deriv_sumsum_expr_param(y = i, theta2 = theta[2], j = j)
      hessian_2 <- matrix(attr(hess_2, "hessian"), nrow = 1, ncol = 1)
      hessian_sum_2 <- hessian_sum_2 + hessian_2
    }
  }

  hessian_sum_1[2, 2] <- hessian_sum_1[2, 2] + hessian_sum_2
  total_hessian <- hessian_sum_1

  return(total_hessian)
}
hess_neg_log_likelihood_expr_param = function(theta,
                                              y,
                                              N = 10000) {
  -1*hess_log_likelihood_expr_param(theta, y, N)
}
```

First, calculating using our derived mean expression gives

```
MLE_lambda[2]*tan(MLE_lambda[1])
```

```
## [1] 5.269616
```

an estimate for the MLE of $\mu(\lambda_*)$ to be 5.269616

And additionally, computing the value of the integral $\mu(\lambda_*) = \int_R y \exp(\lambda_1 y + \lambda_2 \log(\cos(\lambda_1)) g(y|\lambda_2) dy$ numerically:

```
integrand <- function(x){
  x*exp(log_likelihood(x, l1 = MLE_lambda[1], l2 = MLE_lambda[2]))
}
expected_value <- integrate(Vectorize(integrand), lower = -Inf, upper = Inf)$value
expected_value
```

```
## [1] 5.277949
```

The MLE estimate of the mean with this method is 5.277949, which is very close, so we can be sure that $\mu(\lambda_*) \approx 5.27$.

Now, finding the 95% confidence interval we begin by finding our Hessain and Jacobian using the delta method

```
fit_hessian=hess_neg_log_likelihood_expr_param(theta = MLE_theta, y=y_sample_q1,N=10000)
fit_inverse_hessian = solve(fit_hessian)

jacobian_T = matrix(0, nrow = 2, ncol = 1)
jacobian_T[1,1] = exp(MLE_theta[2])
jacobian_T[2,1] = MLE_theta[1]*exp(MLE_theta[2])

jacobian = matrix(0, nrow = 1, ncol = 2)
jacobian[1,1] = exp(MLE_theta[2])
jacobian[1,2] = MLE_theta[1]*exp(MLE_theta[2])

delta_hess = jacobian  %*% fit_inverse_hessian %*% jacobian_T
```

Giving us a confidence interval for $\mu(\lambda_*)$ of

```
exp(c(MLE_theta[2]-1.96*sqrt(delta_hess),MLE_theta[2]+1.96*sqrt(delta_hess)))
```

```
## [1] 3.930937 8.812732
```

Question 5 [4 marks]

Compute an asymptotic 95% confidence interval for the unknown parameter λ_2^* using:

- the asymptotic normal approximation to the distribution $\hat{\lambda}_2$
- the asymptotic normal approximation to the distribution $\log(\hat{\lambda}_2)$

Solution to Question 5

Compute an asymptotic 95% confidence interval for the unknown parameter λ_2^* using the asymptotic normal approximation to the distribution $\hat{\lambda}_2$

Functions for the derivative of the likelihood that are unparametrised:

```
deriv_log_likelihood_expr <- deriv(expr         = log_likelihood_expr,
                                   namevec      = c("l1","l2"),
                                   function.arg = c("l1","l2","y"),
                                   hessian      = T)

deriv_sumsum_expr<- deriv(expr         = sumsum_expr,
                          namevec      = c("l2"),
                          function.arg = c("l2","y","j"),
                          hessian      = T)
```

Functions for the un-parametrised Hessian

```
hess_log_likelihood_expr <- function(lambda = c(1, 1), y = 1, N = 1) {
  hessian_sum_1 <- matrix(0, nrow = 2, ncol = 2)
  hessian_sum_2 <- 0

  for (i in y) {
    hess_1 <- deriv_log_likelihood_expr(y = i, l1 = lambda[1], l2 = lambda[2])
    hessian_1 <- matrix(attr(hess_1, "hessian"), nrow = 2, ncol = 2)
    hessian_sum_1 <- hessian_sum_1 + hessian_1
  }

  for (i in y) {
    for (j in 0:N) {
      hess_2 <- deriv_sumsum_expr(y = i, l2 = lambda[2], j = j)
      hessian_2 <- matrix(attr(hess_2, "hessian"), nrow = 1, ncol = 1)
      hessian_sum_2 <- hessian_sum_2 + hessian_2
    }
  }

  hessian_sum_1[2, 2] <- hessian_sum_1[2, 2] + hessian_sum_2
  total_hessian <- hessian_sum_1

  return(total_hessian)
}
hess_neg_log_likelihood_expr = function(lambda = c(1, 1),
                                        y = 1,
                                        N = 1) {
  -1*hess_log_likelihood_expr(lambda, y, N)
}
```

To find the 95% confidence intervals for λ_2 using the asymptotic normal approximation to the distribution $\hat{\lambda}_2$:

```
fit_hessian_lam = hess_neg_log_likelihood_expr(lambda = MLE_lambda, y = y_sample_q1, N = 10000)
fit_inverse_hessian_lam = solve(fit_hessian_lam)
c(
  MLE_lambda[2] - 1.96 * sqrt(fit_inverse_hessian_lam[2, 2]),
  MLE_lambda[2] + 1.96 * sqrt(fit_inverse_hessian_lam[2, 2])
)
```

```
## [1] 3.494712 8.276830
```

To find the 95% confidence intervals for λ_2 using the asymptotic normal approximation to the distribution $\log(\hat{\lambda}_2)$:

```
exp(c(
  MLE_theta[2] - 1.96 * sqrt(fit_inverse_hessian[2, 2]),
  MLE_theta[2] + 1.96 * sqrt(fit_inverse_hessian[2, 2])
))
```

```
## [1] 3.920792 8.835535
```

Question 6 [4 marks]

Use the generalised likelihood ratio to test the hypotheses:

$$H_0 : \mu(\boldsymbol{\lambda}_*) = 5 \qquad \text{vs} \qquad H_a : \mu(\boldsymbol{\lambda}_*) \neq 5$$

using a significance level $\alpha = 0.05$.

Separately, also test

$$H_0 : \lambda_2^* = 5 \qquad \text{vs} \qquad H_a : \lambda_2^* \neq 5$$

using a significance level $\alpha = 0.05$.

Solution to Question 6

We start by finding the MLE for λ_1 and λ_2 when $\mu(\lambda_*) = 5$. From Q4 we know that $\mu(\lambda_*) = \lambda_2 \tan(\lambda_1)$ so as $\mu(\lambda_*) = 5$ we can put $exp(\theta_2) = 5/\theta_1$ into our `log_likelihood_expr` and optimise this:

```
log_likelihood_expr_param_H0 <- expression(atan(theta1) * y
                            + (5/theta1) * log(cos(atan(theta1)))
                            + (5/theta1) * log(2)
                            + 2 * log(gamma((5/theta1) /2))
                            - log(gamma((5/theta1)))
                            - 2*log(2)
                            - log(pi))
sumsum_expr_param_H0 <- expression(-log(1+(y/((5/theta1)+2*j))^2))
```

```
deriv_log_likelihood_expr_param_H0 <- deriv(expr         = log_likelihood_expr_param_H0,
                                            namevec      = c("theta1"),
                                            function.arg = c("theta1","y"),
                                            hessian      = T)

deriv_sumsum_expr_param_H0<- deriv(expr         = sumsum_expr_param_H0,
                                   namevec      = c("theta1"),
                                   function.arg = c("theta1","y","j"),
                                   hessian      = T)
```

```
fcn_log_likelihood_expr_param_H0=function(theta , y , N=10000){
  total<-0
  for (i in y){
    total = total + deriv_log_likelihood_expr_param_H0(y = i,theta1=theta)
  }
  for (i in y){
    total = total + sum(deriv_sumsum_expr_param_H0(y = i, theta1=theta,j=0:N))
  }
  return(total)
}

grad_log_likelihood_expr_param_H0 <- function(theta, y, N = 10000) {
  res1 <- lapply(y, function(i) deriv_log_likelihood_expr_param_H0(y = i, theta1 = theta))
  total_loop_1 <- Reduce(`+`, lapply(res1, function(res) attr(res, "gradient")))

  res2 <- lapply(y, function(i) deriv_sumsum_expr_param_H0(y = i, theta1 = theta, j = 0:N))
  total_loop_2 <- Reduce(`+`, lapply(res2, function(res) colSums(attr(res, "gradient"))))

  total_loop_1 <- total_loop_1 + total_loop_2
  total <- total_loop_1
  return(total)
}
# Negative log likelihood function for parameterised version
fcn_neg_log_likelihood_expr_param_H0 = function(theta = 1,
                                        y = 1,
                                        N = 1) {
  -1*fcn_log_likelihood_expr_param_H0(theta, y, N)
}
# Gradient of the negative log likelihood function for parameterised version
grad_neg_log_likelihood_expr_param_H0 = function(theta,
                                        y = 1,
                                        N = 1) {
  -1*grad_log_likelihood_expr_param_H0(theta, y, N)
}
```

We have edited our earlier fit_optim function for q6

```
fit_optim_Q6<- function(par = par,
                     fn ,
                     gr ,
                     sd,
                     method = "BFGS",
                     hessian   = T,
                     y = y_sample_q1,
                     N=10000,
                     N_samples = 100){

  fit <- vector("list",
              length = N_samples)

  for (i in c(1:N_samples)){
    fit[[i]]<-try(
          optim(par = c(rnorm(1,mean = par,sd = sd)),
          fn  = fn,
          gr  = gr,
          y =y,
          N=N,
          method  =method,
          hessian = hessian),
                    silent=T)

    if(inherits(fit[[i]], "try-error")){
      fit[[i]] = NA
    }
    # try statement to surpress error messages
  }
  fit = fit[!is.na(fit)]
  return(fit)
}
```

Optimising

```
N = 10000
N_samples = 10
optim_H0 <- fit_optim_Q6(
    par = 0,
    sd  = 1,
    fn  = fcn_neg_log_likelihood_expr_param_H0,
    gr = grad_neg_log_likelihood_expr_param_H0,
    y   = y_sample_q1,
    N = N,
    method  = "BFGS",
    hessian = TRUE,
    N_samples = N_samples
  )
```

Now finding our MLE values for θ_1 and θ_2 when $\mu(\boldsymbol{\lambda}_*) = 5$

```
MLE_theta1_H0=min_nll(optim_H0, length(optim_H0))
MLE_theta_H0 = c(MLE_theta1_H0, log(5/MLE_theta1_H0))
MLE_lambda_H0 = c(atan(MLE_theta_H0[1]), exp(MLE_theta_H0[2]))
print(MLE_theta_H0)
```

```
## [1] 0.8447052 1.7782056
```

So the MLE values for λ_1 and λ_2 when $\mu(\boldsymbol{\lambda}_*) = 5$ are

```
print(MLE_lambda_H0)
```

```
## [1] 0.7014121 5.9192252
```

GLRT for $H_0 : \mu(\boldsymbol{\lambda}_*) = 5 \qquad \text{vs} \qquad H_a : \mu(\boldsymbol{\lambda}_*) \neq 5$

```
# Likelihood function under H0
likelihood_H0 <- function() {
  return(log_likelihood(y_sample_q1, MLE_lambda_H0[1], MLE_lambda_H0[2], 10000))
}

# Likelihood function under Ha
likelihood_Ha <- function() {
  return(log_likelihood(y_sample_q1, MLE_lambda[1], MLE_lambda[2], 10000))
}

# Test statistic
LR_statistic <- function() {
  LR <- 2 * (likelihood_Ha() - likelihood_H0())
  return(LR)
}

observed_LR <- LR_statistic()
alpha <- 0.05

# Our p-value is
p_value <- 1 - pchisq(observed_LR, df = 1)

if (p_value < alpha) {
  cat("Reject H0: There is evidence that mu(lambda_2) is not equal to 5.\n")
} else {
  cat("Fail to reject H0: There is no evidence to suggest mu(lambda_2) is different from 5.\n")
}
```

```
## Fail to reject H0: There is no evidence to suggest mu(lambda_2) is different from 5.
```

```
# Our test results are thus
cat("Likelihood Ratio Test Statistic:", observed_LR, "\n")
```

```
## Likelihood Ratio Test Statistic: 1.765916
```

```
cat("P-value:", p_value, "\n")
```

```
## P-value: 0.1838884
```

```
likelihood_Ha()
```

```
## [1] -636.2355
```

```
likelihood_H0()
```

```
## [1] -637.1185
```

GLRT for $H_0 : \lambda_2^* = 5 \qquad \text{vs} \qquad H_a : \lambda_2^* \neq 5$

Taking the derivative of the NLL with respect to λ_1 then setting to 0 gives

$$\lambda_1 = atan(\frac{\overline{\mathbf{y}}}{5})$$

where $\overline{\mathbf{y}}$ is the sample mean. This gives us an easier way of finding the value of λ_1 when we have the value of λ_2.

```
# Likelihood function under H0
likelihood_H0 <- function() {
  MLE_lambda1 <- atan(mean(y_sample_q1) / 5)
  return(log_likelihood(y_sample_q1, MLE_lambda1, 5, 10000))
}

# Likelihood function under Ha
likelihood_Ha <- function() {
  return(log_likelihood(y_sample_q1, MLE_lambda[1], MLE_lambda[2], 10000))
}

# Test statistic
LR_statistic <- function() {
  LR <- 2 * (likelihood_Ha() - likelihood_H0())
  return(LR)
}

observed_LR <- LR_statistic()
alpha <- 0.05

# Our p-value is
p_value <- 1 - pchisq(observed_LR, df = 1)

if (p_value < alpha) {
  cat("Reject H0: There is evidence that lambda_2 is not equal to 5.\n")
} else {
  cat("Fail to reject H0: There is no evidence to suggest lambda_2 is different from 5.\n")
}
```

```
## Fail to reject H0: There is no evidence to suggest lambda_2 is different from 5.
```

```
# Our test results are thus
cat("Likelihood Ratio Test Statistic:", observed_LR, "\n")
```

```
## Likelihood Ratio Test Statistic: 0.5835991
```

```
cat("P-value:", p_value, "\n")
```

```
## P-value: 0.444905
```

```
likelihood_Ha()
```

```
## [1] -636.2355
```

```
likelihood_H0()
```

```
## [1] -636.5273
```

Question 7 [10 marks]

Consider the following data frame

```
data_q7 <-read.table("http://people.bath.ac.uk/kai21/ASI/CW_2023/data_q7.txt")
```

that contains a bivariate sample

$$(x_1, y_1),\ (x_2, y_2),\ \ldots,\ (x_n, y_n)$$

of size $n = 300$.

Use the parametric family $\mathcal{F}_1$ defined in Question 1 to find an appropriate model for the unknown conditional distribution of $\mathcal{Y}$ given $\mathcal{X} = x$, that is $f_*(y|x)$. The model should be defined by specifying the mean function $\mu(\boldsymbol{\theta}^{(1)}, x)$ as follows:

$$\mu(\boldsymbol{\theta}^{(1)}, x) = g^{-1}(\theta_1 + \theta_2\, x + \theta_3\, x^2 + \theta_4\, x^3 + \cdots + \theta_{p+1}\, x^p)$$

for some choice of link function g and some choice of integer $p \geq 1$.

From a set of candidate models (that is for different choices of g and p), choose the model with the smallest AIC (Akaike Information Criterion). Only present the results from the maximum likelihood estimation from the best chosen model and simply comment on the other models considered.

Now, repeat the same process above to find an appropriate model for the unknown conditional distribution of $\mathcal{Y}$ given $\mathcal{X} = x$ but now based on the Gamma parametric family:

$$\mathcal{F}_{gamma} = \left\{ f(y|\lambda_1, \lambda_2) = \frac{\lambda_2^{\lambda_1}}{\Gamma(\lambda_1)} y^{\lambda_1 - 1} \exp(-\lambda_2\, y)\ :\ \lambda_1 > 0\,, \lambda_2 > 0, y > 0 \right\}$$

Finally, find an appropriate model for the unknown conditional distribution of $\mathcal{Y}$ given $\mathcal{X} = x$ but now based on the Normal parametric family:

$$\mathcal{F}_{normal} = \left\{ f(y|\lambda_1, \lambda_2) = \frac{1}{\lambda_2\sqrt{2\pi}} \exp\left(-\frac{(y - \lambda_1)^2}{2\lambda_2^2} \right)\ :\ \lambda_1 \in \mathcal{R},\ \lambda_2 > 0, y \in \mathcal{R} \right\}$$

For each of the three chosen models, you should plot the data together with the maximum likelihood estimate of the mean function as well as corresponding asymptotic 95% confidence bands in the range $x \in (-3, 3)$. Comment on the differences between the confidence bands and the mean function estimates. You must select the best model out of the three, based on the Akaike Information Criterion.

Solution to Question 7

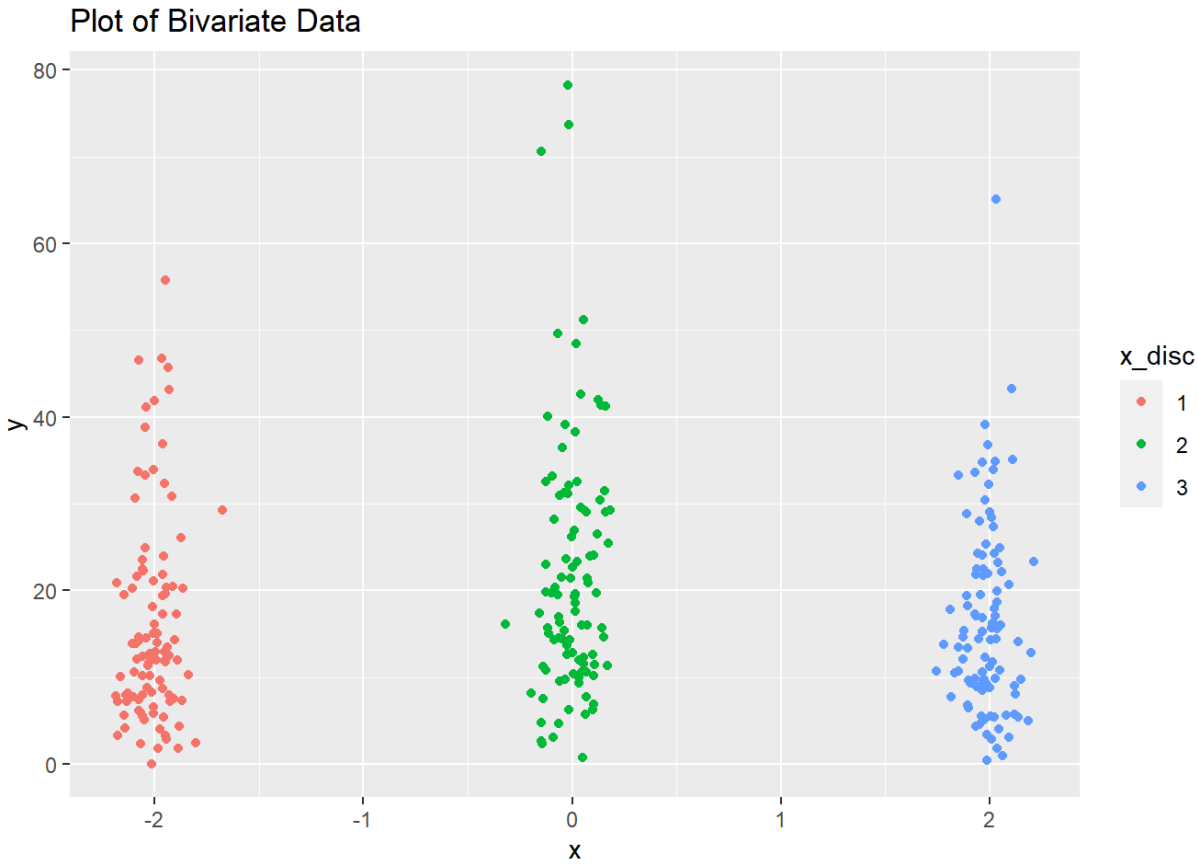
Question 7 will be done in parts. Initially, in order to model the dependencies of the parameters λ_1, λ_2 and μ on the conditional distribution of Y given X. Then, we will guess a suitable mu function and link function g, and compute the AIC by re-parameterising by the mean function. We will then repeat this process for the normal distribution and gamma distribution, and then make some 95% confidence intervals.

Here is an adjusted `fit_optim_Q7` function, and a new `min_nll_object` function

```
fit_optim_Q7<- function(par = par,
                     fn ,
                     gr ,
                     sd,
                     method = "BFGS",
                     hessian   = T,
                     y = y_sample_q1,
                     N=10000,
                     N_samples = 100){

  fit <- vector("list",
              length = N_samples)

  for (i in c(1:N_samples)){
    fit[[i]]<-try(
          optim(par = c(rnorm(1,mean = par[1],sd = sd[1]),
                        rnorm(1,mean = par[2],sd = sd[2])),
          fn  = fn,
          gr  = gr,
          y =y,
          N=N,
          method  =method,
          hessian = hessian),
                    silent=T) # this suppresses the error messages
    if(inherits(fit[[i]], "try-error")){
      fit[[i]] = NA
    }
    # this suppresses the error messages
  }
  fit = fit[!is.na(fit)]
  return(fit)
}
# function that returns the optim object containing the MLE parameters
min_nll_object = function(fit, N_samples){
  values = numeric(N_samples)
  for (i in c(1:N_samples)){
    values[i] = (fit[[i]]$value)
  }
  return(fit[[which.min(values)]])
}
```

Segregating the data into 3 bins and plotting:

```
n_bins<-3

dat.matrix<-data_q7
x<-dat.matrix$x
y<-dat.matrix$y
n<-length(x)

breaks = c(-4,-1,1,4)
dat.matrix = dat.matrix %>% mutate(x_disc = cut(x, breaks = breaks,labels=c(1,2,3)))
ggplot(dat.matrix, aes(x,y,color = x_disc), color = x_disc) + geom_point() + labs(title="Plot of Bivar
iate Data")
```



F_1 Paramater Family:

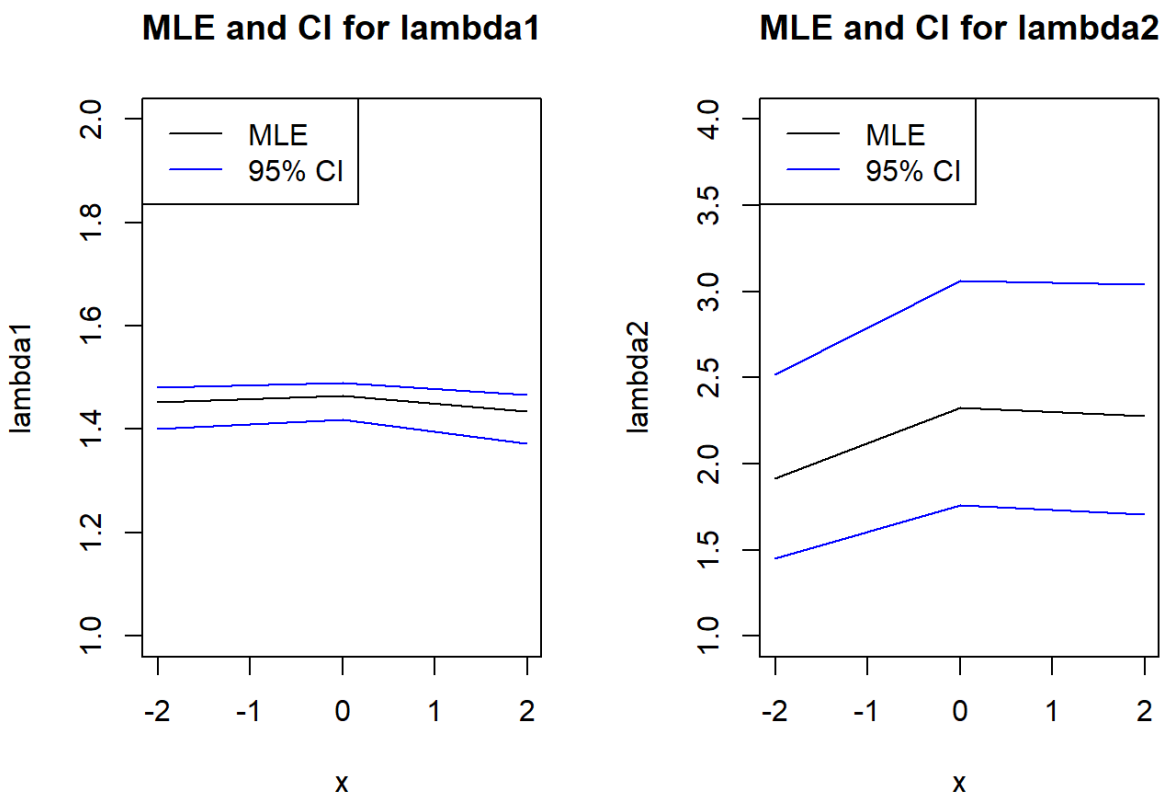
Calculating the confidence intervals and MLEs for lambda parameters and Mu function from bivariate data

```
lambda1_disc <-rep(NA,n_bins)
lambda2_disc <-rep(NA,n_bins)
mu_disc <-rep(NA,n_bins)


lambda1_disc_up    <-rep(NA,n_bins)
lambda2_disc_up <-rep(NA,n_bins)
mu_disc_up<-rep(NA,n_bins)

lambda1_disc_low  <-rep(NA,n_bins)
lambda2_disc_low <-rep(NA,n_bins)
mu_disc_low <-rep(NA,n_bins)

xx <-rep(NA,n_bins)

N_samples=10
N=100
for (i in 1:n_bins) {
  ind  <- which(dat.matrix$x_disc == i)
  samp  <- dat.matrix$y[ind]
  xx[i] <- median(dat.matrix$x[ind])

  if (i == 1) {
    center <- rep(0, 2)
  } else{
    center <- MLE
  }
  optim1 <- fit_optim_Q7(
    par = c(center),
    sd  = c(1, 1),
    fn  = fcn_neg_log_likelihood_expr_param,
    gr = grad_neg_log_likelihood_expr_param,
    y   = samp,
    N = N,
    method  = "BFGS",
    hessian = TRUE,
    N_samples = N_samples
  )
  MLE = min_nll(optim1, length(optim1))
  fit_inverse_hessian = solve(hess_neg_log_likelihood_expr_param(theta = c(MLE[1],MLE[2]),y=samp, N =
N))
  std.err <- sqrt(diag(fit_inverse_hessian ))

  lambda1_disc[i] <- atan(MLE[1])
  lambda2_disc[i] <- exp(MLE[2])

  lambda1_disc_low[i] <- atan(MLE[1] - 1.96 * std.err[1])
  lambda2_disc_low[i] <- exp(MLE[2] - 1.96 * std.err[2])

  lambda1_disc_up[i] <- atan(MLE[1] + 1.96 * std.err[1])
  lambda2_disc_up[i] <- exp(MLE[2] + 1.96 * std.err[2])

  jacobian_T = matrix(0, nrow = 2, ncol = 1)
  jacobian_T[1,1] = exp(MLE[2])
  jacobian_T[2,1] = MLE[1]*exp(MLE[2])

  jacobian = matrix(0, nrow = 1, ncol = 2)
  jacobian[1,1] = exp(MLE[2])
  jacobian[1,2] = MLE[1]*exp(MLE[2])
```

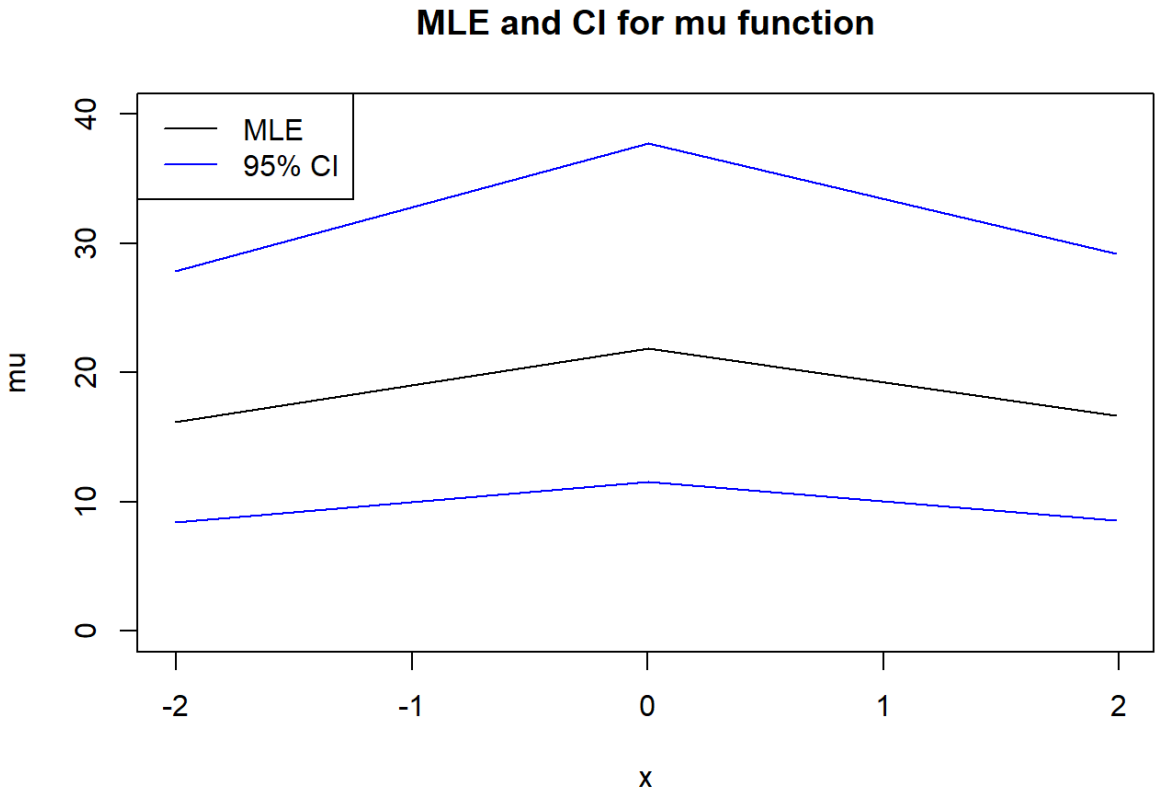
```
  delta_hess = jacobian  %*% fit_inverse_hessian %*% jacobian_T

  mu_disc[i] = lambda2_disc[i]*tan(lambda1_disc[i])
  mu_disc_up[i] = lambda2_disc_up[i]*tan(lambda1_disc_up[i])
  mu_disc_low[i]= lambda2_disc_low[i]*tan(lambda1_disc_low[i])
}
```

Plotting the calculated confidence intervals and MLE for F_1 Parameter Family

```
par(mfrow=c(1,2))
plot(xx,lambda1_disc,type="l",ylab="lambda1",xlab="x", ylim = c(1, 2))
lines(xx,lambda1_disc_low,col="blue")
lines(xx,lambda1_disc_up,col="blue")
legend("topleft",legend=c("MLE","95% CI"),col=c("black","blue"),lty=1)
title(main = "MLE and CI for lambda1")

plot(xx,lambda2_disc,type="l",ylab="lambda2",xlab="x", ylim = c(1, 4))
lines(xx,lambda2_disc_low,col="blue")
lines(xx,lambda2_disc_up,col="blue")
legend("topleft",legend=c("MLE","95% CI"),col=c("black","blue"),lty=1)
title(main = "MLE and CI for lambda2")
```




```
par(mfrow=c(1,1))
plot(xx,mu_disc,type="l",ylab="mu",xlab="x", ylim = c(0, 40))
lines(xx,mu_disc_low,col="blue")
lines(xx,mu_disc_up,col="blue")
legend("topleft",legend=c("MLE","95% CI"),col=c("black","blue"),lty=1)
title(main = "MLE and CI for mu function")
```



Already just

looking at this plot we could see that an appropriate model could be a constant, linear or quadratic model.

During this investigation we tried the following μ functions:

- $\mu_a + \mu_b * x$
- $log(\mu_a + \mu_b * x)$
- $exp(\mu_a + \mu_b * x)$
- $1/(\mu_a + \mu_b * x)$
- $\mu_a + \mu_b * x + \mu_c * x^2$
- $1/(\mu_a + \mu_b * x + \mu_c * x^2)$
- $asin(\mu_a + \mu_b * x + \mu_c * x^2)$
- $acos(\mu_a + \mu_b * x + \mu_c * x^2)$
- $acos(\mu_a + \mu_b * x)$
- $asin(\mu_a + \mu_b * x)$

However, we found that the lowest AIC was for $\mu_a + \mu_b * x + \mu_c * x^2$, with g being the identity. Using this to parametrise our mu function gave us

```
mu = expr(mu_a + mu_b*x+mu_c*x^2)

log_likelihood_expr_param_mu <- expr(atan((((!!mu)/(exp(theta2)))) * y + exp(theta2) * log(cos(atan
((((!!mu)/(exp(theta2)))))) + exp(theta2) * log(2) + 2 * log(gamma(exp(theta2) /2)) -
    log(gamma(exp(theta2))) - 2*log(2) - log(pi))
sumsum_expr_param_mu <- expr(- log(1+(y/(exp(theta2)+2*j))^2))

deriv_log_likelihood_expr_param_mu <- deriv(expr         = log_likelihood_expr_param_mu,
                                            namevec      = c("mu_a","mu_b","mu_c","theta2"),
                                            function.arg = c("mu_a","mu_b","mu_c", "theta2","y","x"),
                                           hessian      = T)

deriv_sumsum_expr_param_mu<- deriv(expr         = sumsum_expr_param_mu,
                                   namevec      = c("mu_a","mu_b","mu_c","theta2"),
                                   function.arg = c("mu_a","mu_b","mu_c","theta2","y","j","x"),
                                   hessian      = T)

neg_fcn_log_likelihood_expr_param_mu=function(theta , y , x , N=10000){
  total<-0
  for (i in c(1:length(y))){
    total = total + deriv_log_likelihood_expr_param_mu(y = y[i], x = x[i],mu_a=theta[1], mu_b=theta
[2],mu_c=theta[3], theta2=theta[4])
  }
  for (i in c(1:length(y))){
    total = total + sum(deriv_sumsum_expr_param_mu(y = y[i],x=x[i], mu_a=theta[1], mu_b=theta[2], mu_c
=theta[3], theta2=theta[4],j=0:N))
  }
  return(-total)
}

neg_grad_log_likelihood_expr_param_mu=function(theta, y,x, N=10000){

  total_loop_1<-0
  total_loop_2<-0
  for (i in c(1:length(y))){
    res1 = deriv_log_likelihood_expr_param_mu(y = y[i],x=x[i],
                                                                  mu_a=theta[1],
                                                                  mu_b=theta[2],
                                              mu_c=theta[3],
                                                                  theta2=theta[4])
    total_loop_1 = total_loop_1 + apply(attr(res1,"gradient"),2,sum)
  }
  for (i in c(1:length(y))){
    res2 = deriv_sumsum_expr_param_mu(y = y[i],x=x[i], mu_a=theta[1],
                                                          mu_b=theta[2],
                                       mu_c=theta[3],
                                                          theta2=theta[4],
                                                          j = 0:N)
    total_loop_2 = total_loop_2 + apply(attr(res2,"gradient"),2,sum)
  }

  total = total_loop_1 + total_loop_2
  return(-total)
}
```

```
fit_optim_Q7_F1_mu<- function(par = par,
                     fn ,
                     gr ,
                     sd,
                     method = "BFGS",
                     hessian   = T,
                     data,
                     N_samples = 100,
                     N){
  fit <- vector("list",
              length = N_samples)
  for (i in c(1:N_samples)){
    fit[[i]]<-try(
          optim(par = c(rnorm(1,mean = par[1],sd = sd[1]),
                        rnorm(1,mean = par[2],sd = sd[2]),
                        rnorm(1,mean = par[3],sd = sd[3]),
                        rnorm(1,mean = par[4],sd = sd[4])),
          fn  = fn,
          gr  = gr,
          y = data$y,
          x = data$x,
          method  =method,
          hessian = hessian,
          N),
                    silent=T)
    #print(fit[[i]])
    if(inherits(fit[[i]], "try-error")){
      fit[[i]] = NA
    }  # this supresses the red error messages
  }
  fit = fit[!is.na(fit)]
  return(fit)
}
```

Calculating the AIC for this model on F_1

```
N_samples = 100
optim_F1_mu <- fit_optim_Q7_F1_mu(
    par = c(0,0,0,0),
    sd  = c(1,1,1,1),
    fn  = neg_fcn_log_likelihood_expr_param_mu,
    gr = neg_grad_log_likelihood_expr_param_mu,
    data = data_q7,
    method  = "BFGS",
    hessian = TRUE,
    N_samples = N_samples,
    N = 10000)
optim_F1_mu_min = min_nll_object(optim_F1_mu, length(optim_F1_mu))
AIC = 2*(optim_F1_mu_min$value + 4)
```

The AIC is for this is

```
print(AIC)
```

```
## [1] 2257.687
```

```
print(paste0('MLE mu_a: ',optim_F1_mu_min$par[1],
             ' ,MLE mu_b: ',optim_F1_mu_min$par[2] ,
             ' ,MLE mu_c: ',optim_F1_mu_min$par[3],
             ' ,MLE lambda_2: ',exp(optim_F1_mu_min$par[3])))
```

```
## [1] "MLE mu_a: 22.0159514502034 ,MLE mu_b: 0.145001139948722 ,MLE mu_c: -1.41807692596122 ,MLE lamb
da_2: 0.242179298089439"
```

Gamma distribution:

```
gamma_log_likelihood_expr_param = expression(
  log( (exp(theta2)^exp(theta1))/gamma(exp(theta1)) * y^(exp(theta1)-1)*exp(-exp(theta2)*y))
)
deriv_gamma_log_likelihood_expr_param <- deriv(expr         = gamma_log_likelihood_expr_param,
                                               namevec      = c("theta1","theta2"),
                                               function.arg = c("theta1","theta2","y"),
                                               hessian      = T)

fcn_gamma_log_likelihood_expr_param = function(theta = c(1,1), y=1){
  res = deriv_gamma_log_likelihood_expr_param(theta1=theta[1],theta2=theta[2],y=y)
  return(sum(res))
}
grad_gamma_log_likelihood_expr_param = function(theta = c(1,1), y=1){
  res = deriv_gamma_log_likelihood_expr_param(theta1=theta[1],theta2=theta[2],y=y)
  grad = apply(attr(res,"gradient"),2,sum)
  return(grad)
}
neg_fcn_gamma_log_likelihood_expr_param = function(theta = c(1,1), y=1){
  res = -1*fcn_gamma_log_likelihood_expr_param(theta,y)
  return(res)
}
neg_grad_gamma_log_likelihood_expr_param = function(theta = c(1,1), y=1){
  res = -1*grad_gamma_log_likelihood_expr_param(theta,y)
  return(res)
}
```

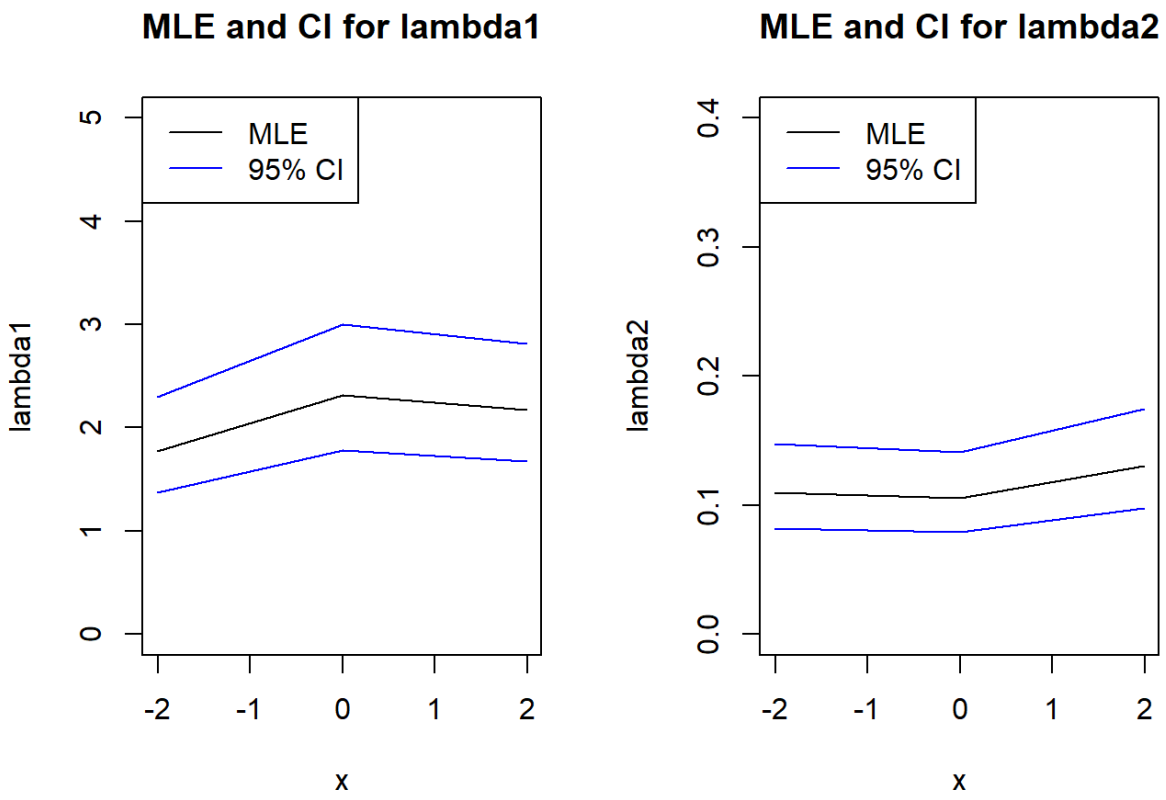
```
fit_optim_Q7_gamma<- function(par = par,
                     fn ,
                     gr ,
                     sd,
                     method = "BFGS",
                     hessian   = T,
                     y = y_sample_q1,
                     N_samples = 100){

  fit <- vector("list",
              length = N_samples)

  for (i in c(1:N_samples)){
    fit[[i]]<-try(
          optim(par = c(rnorm(1,mean = par[1],sd = sd[1]),
                        rnorm(1,mean = par[2],sd = sd[2])),
          fn  = fn,
          gr  = gr,
          y =y,
          method  =method,
          hessian = hessian),
                    silent=T) # this suppresses the error messages
  }
  return(fit)
}
```

Calculating the MLE and the confidence Intervals

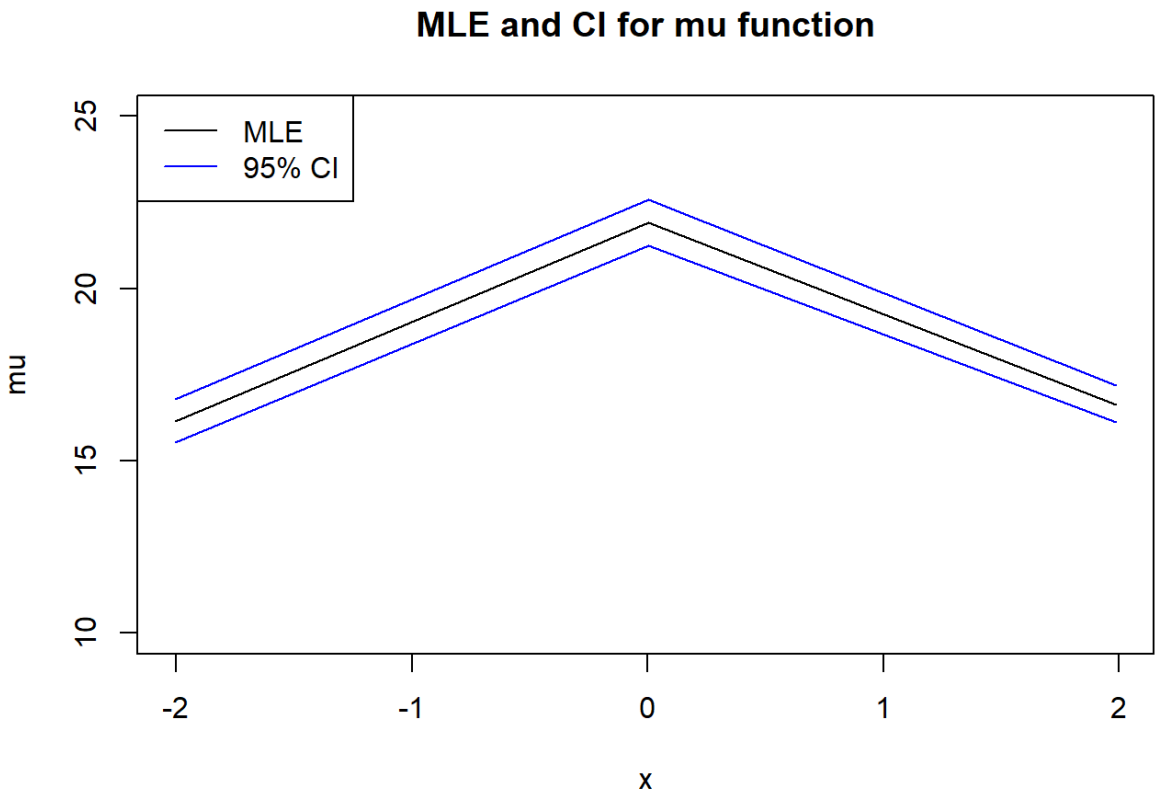
```
lambda1_disc <-rep(NA,n_bins)
lambda2_disc <-rep(NA,n_bins)
mu_disc <-rep(NA,n_bins)

lambda1_disc_up    <-rep(NA,n_bins)
lambda2_disc_up <-rep(NA,n_bins)
mu_disc_up<-rep(NA,n_bins)

lambda1_disc_low  <-rep(NA,n_bins)
lambda2_disc_low <-rep(NA,n_bins)
mu_disc_low <-rep(NA,n_bins)

xx <-rep(NA,n_bins)

N_samples=10
for (i in 1:n_bins) {
  ind  <- which(dat.matrix$x_disc == i)
  samp  <- dat.matrix$y[ind]
  xx[i] <- median(dat.matrix$x[ind])

  if (i == 1) {
    center <- rep(0, 2)
  } else{
    center <- MLE
  }
  optim_gamma <- fit_optim_Q7_gamma(
    par = c(center),
    sd  = c(1, 1),
    fn  = neg_fcn_gamma_log_likelihood_expr_param,
    gr = neg_grad_gamma_log_likelihood_expr_param,
    y   = samp,
    method  = "BFGS",
    hessian = TRUE,
    N_samples = N_samples
  )
  MLE = min_nll(optim_gamma, N_samples)
  fit_inverse_hessian = solve(min_nll_object(optim_gamma, N_samples)$hessian)
  std.err <- sqrt(diag(fit_inverse_hessian ))

  lambda1_disc[i] <- exp(MLE[1])
  lambda2_disc[i] <- exp(MLE[2])

  lambda1_disc_low[i] <- exp(MLE[1] - 1.96 * std.err[1])
  lambda2_disc_low[i] <- exp(MLE[2] - 1.96 * std.err[2])

  lambda1_disc_up[i] <- exp(MLE[1] + 1.96 * std.err[1])
  lambda2_disc_up[i] <- exp(MLE[2] + 1.96 * std.err[2])

  mu_disc[i] = lambda1_disc[i]/lambda2_disc[i]
  mu_disc_up[i] = lambda1_disc_up[i]/lambda2_disc_up[i]
  mu_disc_low[i]= lambda1_disc_low[i]/lambda2_disc_low[i]
}
```

Plotting these

```
par(mfrow=c(1,2))
plot(xx,lambda1_disc,type="l",ylab="lambda1",xlab="x", ylim=c(0,5))
lines(xx,lambda1_disc_low,col="blue")
lines(xx,lambda1_disc_up,col="blue")
legend("topleft",legend=c("MLE","95% CI"),col=c("black","blue"),lty=1)
title(main = "MLE and CI for lambda1")

plot(xx,lambda2_disc,type="l",ylab="lambda2",xlab="x",ylim=c(0,0.4))
lines(xx,lambda2_disc_low,col="blue")
lines(xx,lambda2_disc_up,col="blue")
legend("topleft",legend=c("MLE","95% CI"),col=c("black","blue"),lty=1)
title(main = "MLE and CI for lambda2")
```




```
par(mfrow=c(1,1))
plot(xx,mu_disc,type="l",ylab="mu",xlab="x", ylim =c(10,25))
lines(xx,mu_disc_low,col="blue")
lines(xx,mu_disc_up,col="blue")
legend("topleft",legend=c("MLE","95% CI"),col=c("black","blue"),lty=1)
title(main = "MLE and CI for mu function")
```




```
fit_optim_Q7_gamma_mu<- function(par = par,
                     fn ,
                     gr ,
                     sd,
                     method = "BFGS",
                     hessian   = T,
                     data,
                     N_samples = 100){

  fit <- vector("list",
              length = N_samples)

  for (i in c(1:N_samples)){
    fit[[i]]<-try(
          optim(par = c(rnorm(1,mean = par[1],sd = sd[1]),
                        rnorm(1,mean = par[2],sd = sd[2]),
                        rnorm(1,mean = par[3],sd = sd[3]),
                        rnorm(1,mean = par[4],sd = sd[4])),
          fn  = fn,
          gr  = gr,
          data,
          method  =method,
          hessian = hessian),
                    silent=T)
    if(inherits(fit[[i]], "try-error")){
      fit[[i]] = NA
    }  # this supresses the error messages
  }
  fit = fit[!is.na(fit)]
  return(fit)
}
```

During this investigation we tried the following μ functions:

- $\mu_a + \mu_b * x$
- $log(\mu_a + \mu_b * x)$
- $exp(\mu_a + \mu_b * x)$
- $1/(\mu_a + \mu_b * x)$
- $\mu_a + \mu_b * x + \mu_c * x^2$
- $1/(\mu_a + \mu_b * x + \mu_c * x^2)$

We calculated the AIC for each but the one with the lowest was $\mu_a + \mu_b * x + \mu_c * x^2$, with g being the identity.

Note that we re-parameterise by the mean by eliminating `theta1` .

```
mu = expr((mu_a + mu_b*x + mu_c*x^2))

gamma_log_likelihood_expr_param_mu = expr(
  log( (exp(theta2)^((!!mu)*exp(theta2)))/gamma(((!!mu)*exp(theta2))) * y^(((!!mu)*exp(theta2))-1)*exp
(-exp(theta2)*y))
)
deriv_gamma_log_likelihood_expr_param_mu <- deriv(expr         = gamma_log_likelihood_expr_param_mu,
                                        namevec      = c("mu_a","mu_b","mu_c", "theta2"),
                                        function.arg = c("theta2","y", "mu_a","mu_b","mu_c", "x"),
                                        hessian      = T)

fnc_neg_gamma_log_likelihood_expr_param_mu = function(theta = c(1,1,1,1), data = 1){
  aux <- deriv_gamma_log_likelihood_expr_param_mu(mu_a = theta[1],
                                                  mu_b = theta[2],
                                                  mu_c = theta[3],
                                                  theta2 = theta[4],
                                                  y  = data$y,
                                                  x = data$x)
  fn<-sum(as.numeric(aux))
  return(-fn)
}

grad_neg_gamma_log_likelihood_expr_param_mu = function(theta = c(1,1,1,1), data = 1){
  aux <- deriv_gamma_log_likelihood_expr_param_mu(mu_a = theta[1],
                                                  mu_b = theta[2],
                                                 mu_c = theta[3],
                                                  theta2 = theta[4],
                                                  y  = data$y,
                                                  x = data$x)
  grad <- apply(attr(aux,"gradient"),2,sum)
  return(-grad)
}
```

```
N_samples = 100
optim_gamma_mu <- fit_optim_Q7_gamma_mu(
    par =  c(1,1,1,1),
    sd  = c(1, 1,1,1),
    fn  = fnc_neg_gamma_log_likelihood_expr_param_mu,
    gr = grad_neg_gamma_log_likelihood_expr_param_mu,
    data = data_q7,
    method  = "BFGS",
    hessian = TRUE,
    N_samples = N_samples)
optim_gamma_mu_min = min_nll_object(optim_gamma_mu, length(optim_gamma_mu))
AIC = 2*(optim_gamma_mu_min$value + 4)
```

Thus we find the AIC to be

```
print(AIC)
```

```
## [1] 2267.177
```

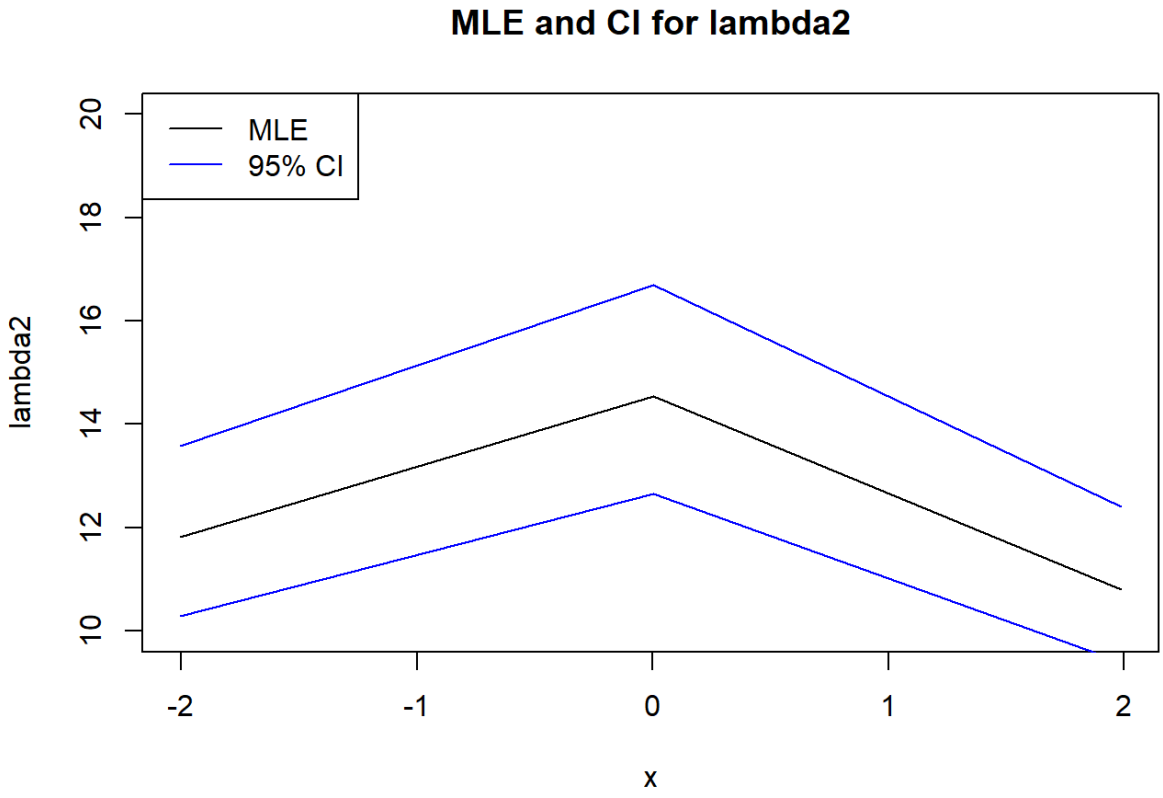
```
print(paste0('MLE mu_a: ',optim_gamma_mu_min$par[1],
             ' ,MLE mu_b: ',optim_gamma_mu_min$par[2] ,
             ' ,MLE mu_c: ',optim_gamma_mu_min$par[3] ,
             ' ,MLE lambda_2: ',exp(optim_gamma_mu_min$par[4])))
```

```
## [1] "MLE mu_a: 21.642593496374 ,MLE mu_b: 0.272251192097534 ,MLE mu_c: -1.27915111001465 ,MLE lambd
a_2: 0.114829786968214"
```

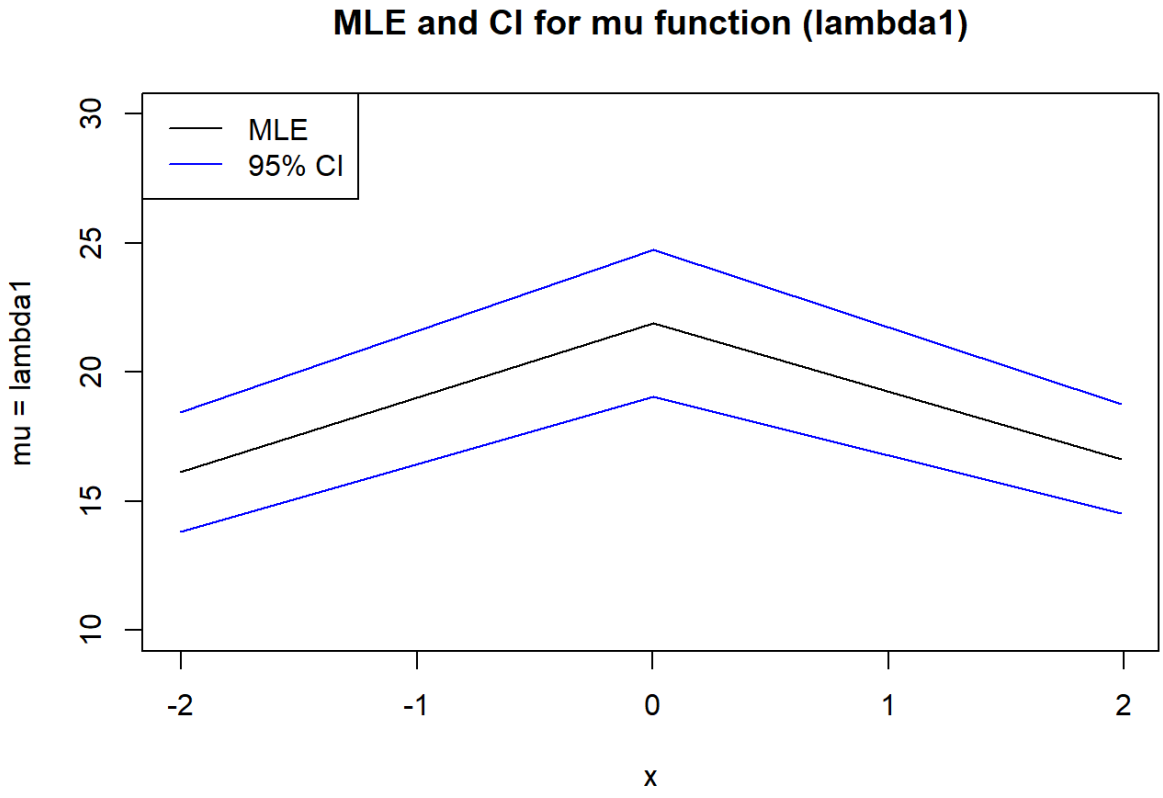
Normal distribution

```
normal_log_likelihood_expr_param = expression(
  log(1/(exp(theta2)*sqrt(2*pi))*exp(-(y-l1)^2/(2*exp(theta2)^2))))

deriv_normal_log_likelihood_expr_param <- deriv(expr         = normal_log_likelihood_expr_param,
                                        namevec      = c("l1","theta2"),
                                        function.arg = c("l1","theta2","y"),
                                        hessian      = T)

fcn_normal_log_likelihood_expr_param = function(params = c(1,1), y=1){
  res = deriv_normal_log_likelihood_expr_param(l1=params[1],theta2=params[2],y=y)
  return(sum(res))
}
grad_normal_log_likelihood_expr_param = function(params = c(1,1), y=1){
  res = deriv_normal_log_likelihood_expr_param(l1=params[1],theta2=params[2],y=y)
  grad = apply(attr(res,"gradient"),2,sum)
  return(grad)
}
fnc_neg_normal_log_likelihood_expr_param = function(params = c(1,1), y=1){
  res = -1*fcn_normal_log_likelihood_expr_param(params,y)
  return(res)
}
grad_neg_normal_log_likelihood_expr_param = function(params = c(1,1), y=1){
  res = -1*grad_normal_log_likelihood_expr_param(params,y)
  return(res)
}
fit_optim_Q7_Normal<- function(par = par,
                     fn ,
                     gr ,
                     sd,
                     method = "BFGS",
                     hessian   = T,
                     y = y_sample_q1,
                     N_samples = 100){

  fit <- vector("list",
              length = N_samples)

  for (i in c(1:N_samples)){
    fit[[i]]<-try(
          optim(par = c(rnorm(1,mean = par[1],sd = sd[1]),
                        rnorm(1,mean = par[2],sd = sd[2])),
          fn  = fn,
          gr  = gr,
          y =y,
          method  =method,
          hessian = hessian),
                    silent=T)
    if(inherits(fit[[i]], "try-error")){
      fit[[i]] = NA
    }  # this supresses the error messages
  }
  fit = fit[!is.na(fit)]
  return(fit)
}
```

```
lambda2_disc <-rep(NA,n_bins)
mu_disc <-rep(NA,n_bins)

lambda2_disc_up <-rep(NA,n_bins)
mu_disc_up<-rep(NA,n_bins)

N_samples=10
for (i in 1:n_bins) {
  ind  <- which(dat.matrix$x_disc == i)
  samp  <- dat.matrix$y[ind]
  xx[i] <- median(dat.matrix$x[ind])

  if (i == 1) {
    center <- rep(0, 2)
  } else{
    center <- MLE
  }
  optim_normal <- fit_optim_Q7_Normal(
    par = c(center),
    sd  = c(1, 1),
    fn  = fnc_neg_normal_log_likelihood_expr_param,
    gr = grad_neg_normal_log_likelihood_expr_param ,
    y   = samp,
    method  = "BFGS",
    hessian = TRUE,
    N_samples = N_samples
  )
  MLE = min_nll(optim_normal, length(optim_normal))
  fit_inverse_hessian = solve(min_nll_object(optim_normal, length(optim_normal))$hessian)
  std.err <- sqrt(diag(fit_inverse_hessian ))

  mu_disc[i] <- (MLE[1])
  lambda2_disc[i] <- exp(MLE[2])

  mu_disc_low[i] <- (MLE[1] - 1.96 * std.err[1])
  lambda2_disc_low[i] <- exp(MLE[2] - 1.96 * std.err[2])

  mu_disc_up[i] <- (MLE[1] + 1.96 * std.err[1])
  lambda2_disc_up[i] <- exp(MLE[2] + 1.96 * std.err[2])
}

plot(xx,lambda2_disc,type="l",ylab="lambda2",xlab="x", ylim =c(10,20))
lines(xx,lambda2_disc_low,col="blue")
lines(xx,lambda2_disc_up,col="blue")
legend("topleft",legend=c("MLE","95% CI"),col=c("black","blue"),lty=1)
title(main = "MLE and CI for lambda2")
```




```
plot(xx,mu_disc,type="l",ylab="mu = lambda1",xlab="x", ylim =c(10,30))
lines(xx,mu_disc_low,col="blue")
lines(xx,mu_disc_up,col="blue")
legend("topleft",legend=c("MLE","95% CI"),col=c("black","blue"),lty=1)
title(main = "MLE and CI for mu function (lambda1)")
```



During this investigation we tried the following μ functions:

- $\mu_a + \mu_b * x$
- $log(\mu_a + \mu_b * x)$
- $exp(\mu_a + \mu_b * x)$
- $1/(\mu_a + \mu_b * x)$
- $\mu_a + \mu_b * x + \mu_c * x^2$
- $1/(\mu_a + \mu_b * x + \mu_c * x^2)$

We calculated the AIC for each but the one with the lowest was $\mu_a + \mu_b * x + \mu_c * x^2$, with g being the identity.

Note that λ_1 here is replaced by the mean function.

```
l1 = expr(mu_a + mu_b*x + mu_c*x^2)

normal_log_likelihood_expr_param_mu = expr(
  log(1/(exp(theta2)*sqrt(2*pi))*exp(-(y-!!l1)^2/(2*exp(theta2)^2))))

deriv_normal_log_likelihood_expr_param_mu <- deriv(expr         = normal_log_likelihood_expr_param_mu,
                                          namevec      = c("mu_a","mu_b","mu_c", "theta2"),
                                          function.arg = c("mu_a","mu_b","theta2","y","x", "mu_c"),
                                          hessian      = T)

fnc_neg_normal_log_likelihood_expr_param_mu = function(theta = c(1,1,1,1), data = 1){
  aux <- deriv_normal_log_likelihood_expr_param_mu(mu_a = theta[1],
                                                   mu_b = theta[2],
                                                   theta2 = theta[4],
                                                   mu_c = theta[3],
                                                   y  = data$y,
                                                   x = data$x)
  fn<-sum(as.numeric(aux))
  return(-fn)
}

grad_neg_normal_log_likelihood_expr_param_mu = function(theta = c(1,1,1,1), data = 1){
  aux <- deriv_normal_log_likelihood_expr_param_mu(mu_a = theta[1],
                                                   mu_b = theta[2],
                                                   theta2 = theta[4],
                                                   mu_c = theta[3],
                                                   y  = data$y,
                                                   x = data$x)
  grad <- apply(attr(aux,"gradient"),2,sum)
  return(-grad)
}
fit_optim_Q7_Normal_mu<- function(par = par,
                     fn ,
                     gr ,
                     sd,
                     method = "BFGS",
                     hessian   = T,
                     data,
                     N_samples = 100){

  fit <- vector("list",
              length = N_samples)

  for (i in c(1:N_samples)){
    fit[[i]]<-try(
          optim(par = c(rnorm(1,mean = par[1],sd = sd[1]),
                        rnorm(1,mean = par[2],sd = sd[2]),
                        rnorm(1,mean = par[3],sd = sd[3]),
                        rnorm(1,mean = par[4],sd = sd[4])),
          fn  = fn,
          gr  = gr,
          data,
          method  =method,
          hessian = hessian),
                    silent=T)
    if(inherits(fit[[i]], "try-error")){
      fit[[i]] = NA
    }  # this supresses the error messages
  }
```

```
  fit = fit[!is.na(fit)]
  return(fit)
}
```

```
N_samples = 100
optim_normal_mu <- fit_optim_Q7_Normal_mu(
    par = c(0,0,0,0),
    sd  = c(1, 1,1,1),
    fn  = fnc_neg_normal_log_likelihood_expr_param_mu,
    gr = grad_neg_normal_log_likelihood_expr_param_mu,
    data = data_q7,
    method  = "BFGS",
    hessian = TRUE,
    N_samples = N_samples)
optim_normal_mu_min = min_nll_object(optim_normal_mu, length(optim_normal_mu))
AIC = 2*(optim_normal_mu_min$value + 4)
```

Thus the AIC is

```
print(AIC)
```

```
## [1] 2373.666
```

```
print(paste0('MLE mu_a: ',optim_normal_mu_min$par[1],
             ' ,MLE mu_b: ',optim_normal_mu_min$par[2] ,
             ' ,MLE mu_c: ',optim_normal_mu_min$par[3] ,
             ' ,MLE lambda_2: ',exp(optim_normal_mu_min$par[4]) ))
```

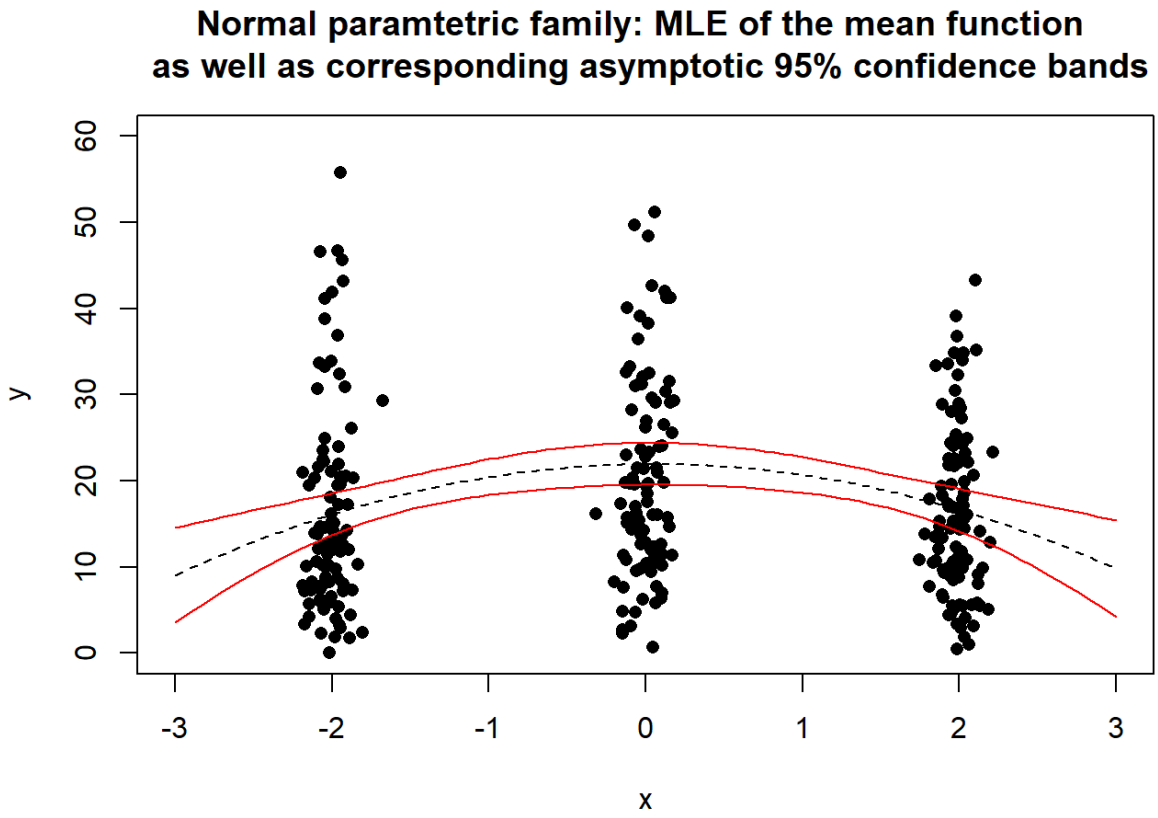
```
## [1] "MLE mu_a: 21.9508698897094 ,MLE mu_b: 0.126825961672702 ,MLE mu_c: -1.39514742835232 ,MLE lamb
da_2: 12.476396767393"
```

Confidence intervals

Proceed to compute the confidence intervals using the delta method and taking the $J = (1, x, x^2, 0)$

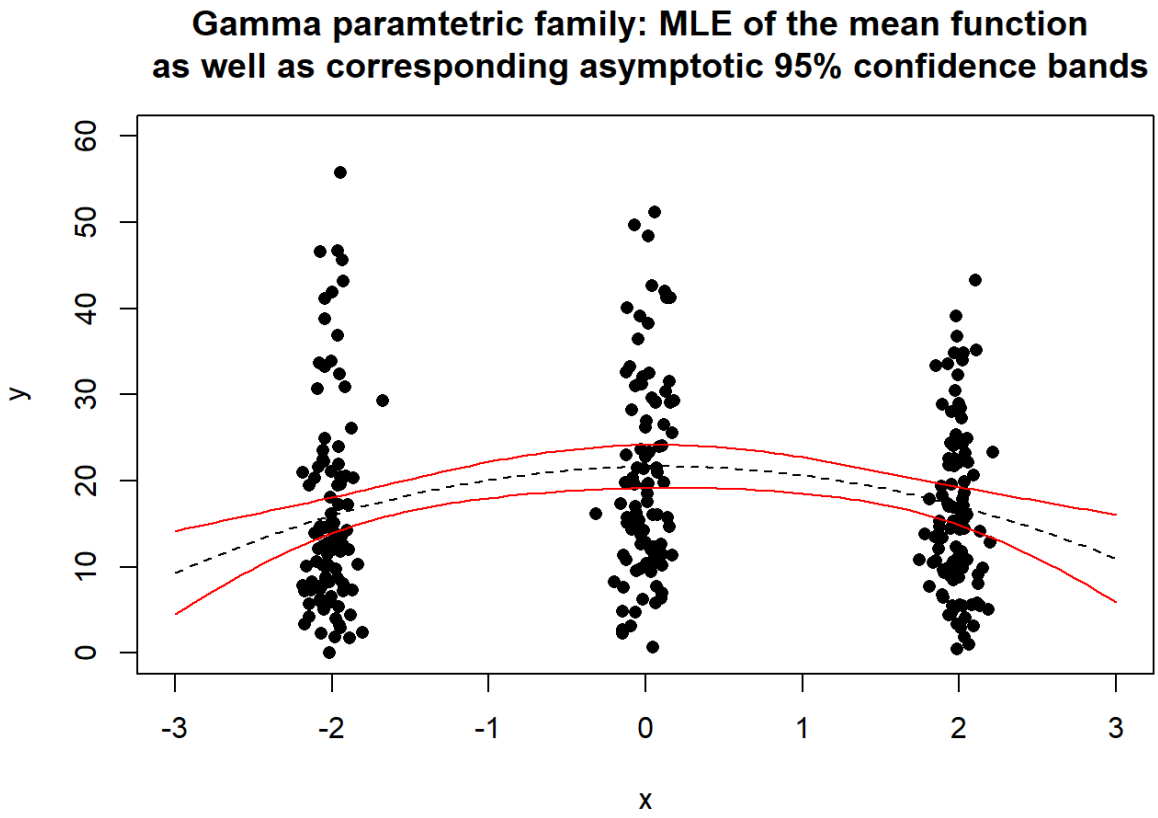
For normal:

```
n_grid <- 100
x <- seq(-3, 3, length = n_grid)
mean <- rep(NA, n_grid)
ci <- matrix(NA, nrow = n_grid, ncol = 2)
for (i in 1:n_grid) {
  mean[i] <- optim_normal_mu_min$par[1] + optim_normal_mu_min$par[2] * x[i] + optim_normal_mu_min$par
[3]*(x[i]^2)
  J <- c(1, x[i], (x[i])^2, 0)
  se <- sqrt(J %*% solve(optim_normal_mu_min$hessian) %*% J)
  ci[i, 1:2] <- c(mean[i] - 1.96 * se, mean[i] + 1.96 * se)
}
plot(
  data_q7$x,
  data_q7$y,
  ylab = "y",
  xlab = "x",
  pch = 16,
  xlim = c(-3, 3),
  ylim = c(0, 60)
)
lines(x, mean, lty = 2)
lines(x, ci[, 1], col = "red")
lines(x, ci[, 2], col = "red")
title(main = 'Normal paramtetric family: MLE of the mean function \n as well as corresponding asymptot
ic 95% confidence bands')
```



For Gamma:

```
n_grid <- 100
x <- seq(-3, 3, length = n_grid)
mean <- rep(NA, n_grid)
ci <- matrix(NA, nrow = n_grid, ncol = 2)
for (i in 1:n_grid) {
  mean[i] <- optim_gamma_mu_min$par[1] + optim_gamma_mu_min$par[2] * x[i] + optim_gamma_mu_min$par[3]
* (x[i])^2
  J <- c(1, x[i],(x[i]^2), 0)
  se <- sqrt(J %*% solve(optim_gamma_mu_min$hessian) %*% J)
  ci[i, 1:2] <- c(mean[i] - 1.96 * se, mean[i] + 1.96 * se)
}
plot(
  data_q7$x,
  data_q7$y,
  ylab = "y",
  xlab = "x",
  pch = 16,
  xlim = c(-3, 3),
  ylim = c(0, 60)
)
lines(x, mean, lty = 2)
lines(x, ci[, 1], col = "red")
lines(x, ci[, 2], col = "red")
title(main = 'Gamma paramtetric family: MLE of the mean function \n as well as corresponding asymptoti
c 95% confidence bands')
```

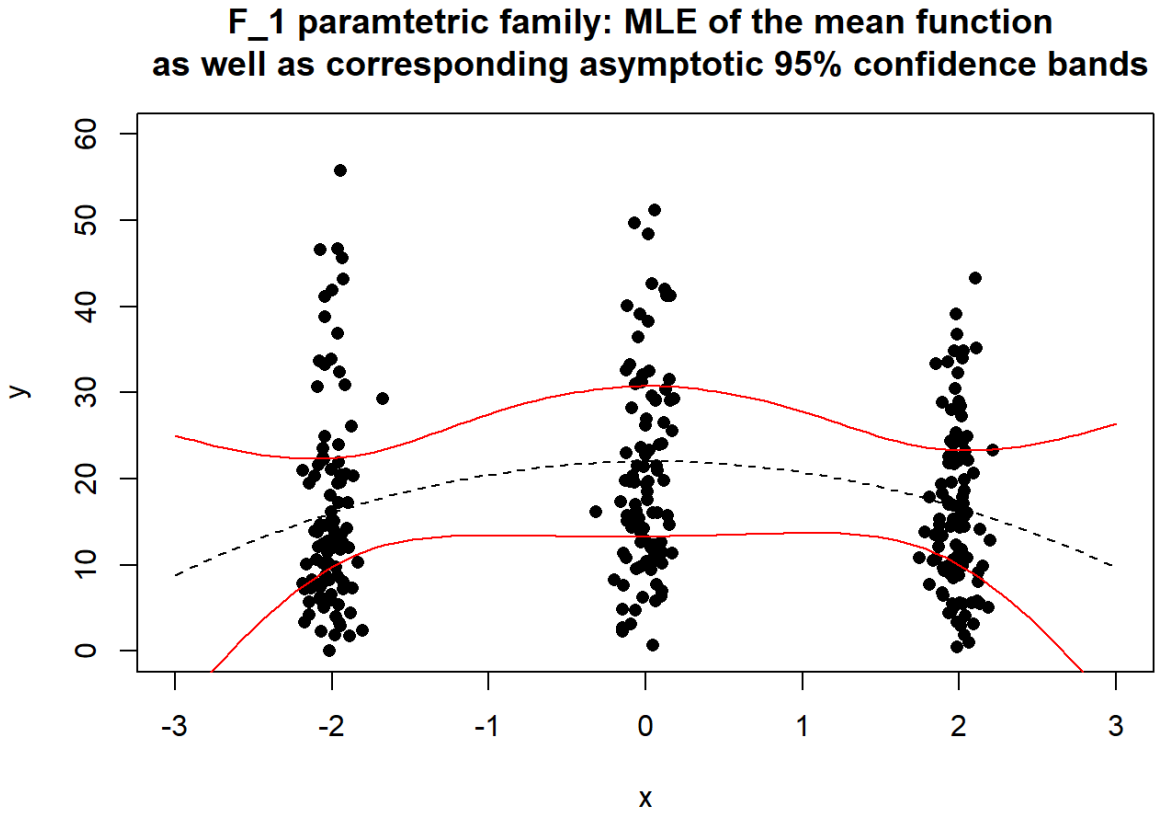


F_1 parameter value:

```
neg_hess_log_likelihood_expr_mu <- function(theta, y = 1,x = 1, N = 1) {
  hessian_sum_1 <- matrix(0, nrow = 4, ncol = 4)
  hessian_sum_2 <- matrix(0, nrow = 4, ncol = 4)

  for (i in c(1:length(y))) {
    hess_1 <- deriv_log_likelihood_expr_param_mu(y = y[i], x = x[i], mu_a=theta[1], mu_b=theta[2],mu_c
=theta[3],  theta2=theta[3])
    hessian_1 <- matrix(attr(hess_1, "hessian"), nrow = 4, ncol = 4)
    hessian_sum_1 <- hessian_sum_1 + hessian_1
  }

  for (i in c(1:length(y))) {
    for (j in 0:N) {
      hess_2 <- deriv_sumsum_expr_param_mu(y = y[i], x = x[i],mu_a=theta[1], mu_b=theta[2],mu_c=theta
[3], theta2=theta[4], j=j)
      hessian_2 <- matrix(attr(hess_2, "hessian"), nrow = 4, ncol = 4)
      hessian_sum_2 <- hessian_sum_2 + hessian_2
    }
  }
  total_hessian <- hessian_sum_1 + hessian_sum_2

  return(-total_hessian)
}
```

Now we calculate the inverse hessian below

```
n_grid <- 100
x <- seq(-3, 3, length = n_grid)
mean <- rep(NA, n_grid)
ci <- matrix(NA, nrow = n_grid, ncol = 2)
H = neg_hess_log_likelihood_expr_mu(theta = optim_F1_mu_min$par, y = data_q7$y, x = data_q7$x, N = 100
00)
for (i in 1:n_grid) {
  mean[i] <- optim_F1_mu_min$par[1] + optim_F1_mu_min$par[2] * x[i] + optim_F1_mu_min$par[3] * x[i]^2
  J <- c(1, x[i],x[i]^2, 0)
  se <- sqrt(J %*% solve(H) %*% J)
  ci[i, 1:2] <- c(mean[i] - 1.96 * se, mean[i] + 1.96 * se)
}
plot(
  data_q7$x,
  data_q7$y,
  ylab = "y",
  xlab = "x",
  pch = 16,
  xlim = c(-3, 3),
  ylim = c(0, 60)
)
lines(x, mean, lty = 2)
lines(x, ci[, 1], col = "red")
lines(x, ci[, 2], col = "red")
title(main = 'F_1 paramtetric family: MLE of the mean function \n as well as corresponding asymptotic
95% confidence bands')
```



Interestingly, $\mathcal{F}_1$ had the lowest AIC which implies it was was the best out of the three distributions. The confidence interval for $\mathcal{F}_1$ shows the mean being significantly different compared to the gamma and normal. The confidence intervals for the normal and gamma look near identical. However, there are differences between the normal and gamma compared to the $\mathcal{F}_1$.

Question 8 [4 marks]

Use the data in Question 7 to compute 95% confidence intervals for the least worse value of the mean function at each x, that is $\mu(\boldsymbol{\theta}_\dagger^{(1)}, x)$ for each of the three parametric families: $\mathcal{F}_1$, the Gamma and the Normal. Plot the computed confidence bands in the range $x \in (-3, 3)$ for each parametric family and comment on the differences obtained.

Solution to Question 8

First lets create functions to compute the $\mathcal{K}$ matrices for each distribution, we can use the section of lecture notes under "Asymptotic distribution of the MLE for misspecified models" and more specifically equations 3.14 and 3.15 that

$$\widehat{\mathbf{K}(\theta^\dagger)} = \frac{1}{n}\sum_{i=1}^{n}\left[\nabla_\theta \log f(y_i|\hat{\theta})\right]\left[\nabla_\theta \log f(y_i|\hat{\theta})\right]^T \tag{3.14}$$

$$\widehat{\mathbf{J}(\theta^\dagger)} = -\nabla_\theta^2 \ell(\hat{\theta}|\mathbf{y}) = \nabla_{\boldsymbol{\theta}}^{\mathbf{2}} \phi(\hat{\theta}|\mathbf{y}) \tag{3.15}$$

We then use the delta method to calculate the confidence intervals with $J = (1, x, x^2, 0)$. Note that the asymptotic variance is

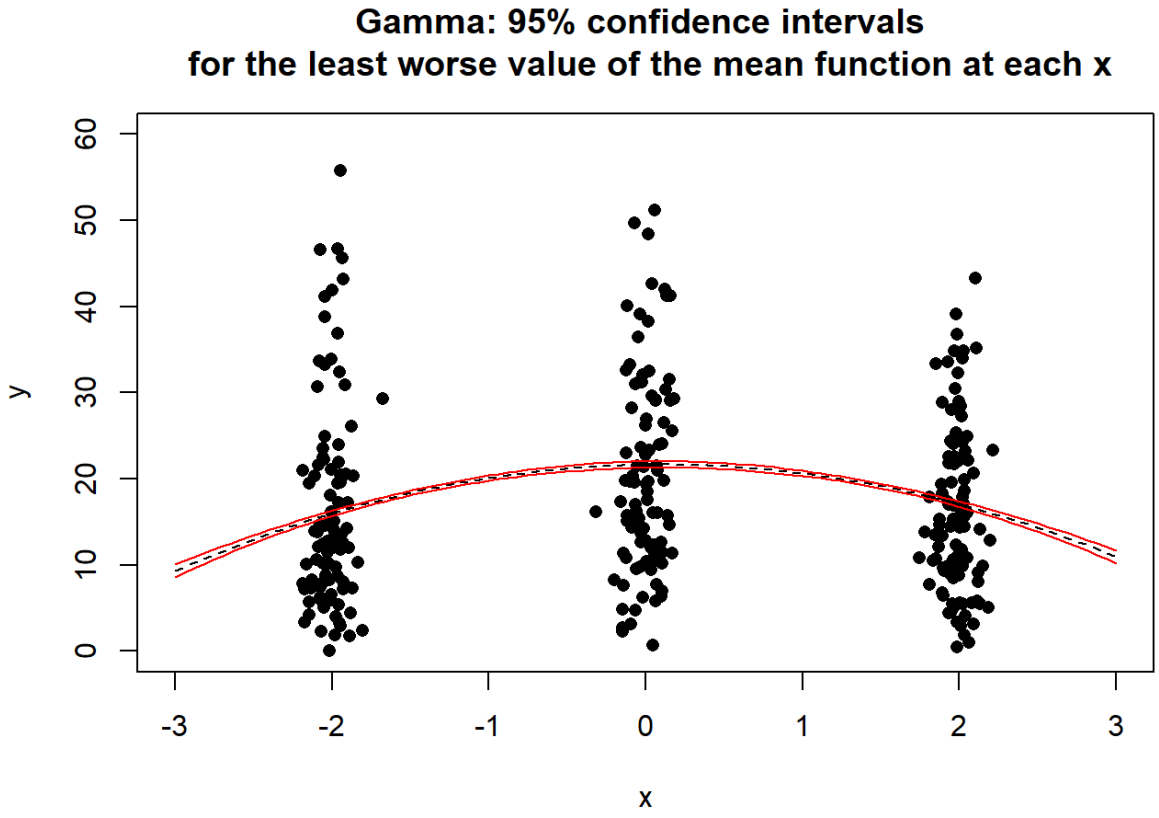
$$\left(J\widehat{\mathbf{J}(\theta^\dagger)}^{-1}\widehat{\mathbf{K}(\theta^\dagger)}\widehat{\mathbf{J}(\theta^\dagger)}^{-1}J^\top\right)^{1/2}$$

Here are the $\mathcal{K}$ for gamma and normal

```
K_mat_gamma_q8 = function(theta = c(1,1,1,1), data){
  y = data$y
  x = data$x
  K_mat = matrix(0, nrow = length(theta), ncol = length(theta))
  for (i in c(1:length(y))){
    aux <- deriv_gamma_log_likelihood_expr_param_mu(mu_a = theta[1],
                                                    mu_b = theta[2],
                                                    mu_c = theta[3],
                                                    theta2 = theta[3],
                                                    y= y[i],
                                                    x= x[i])
    grad <- matrix(apply(attr(aux,"gradient"),2,sum))
    K_mat = K_mat + grad %*% t(grad)
  }
  return(1/length(y)*K_mat)
}
K_mat_normal_q8 = function(theta = c(1,1,1,1), data){
  y = data$y
  x = data$x
  K_mat = matrix(0, nrow = length(theta), ncol = length(theta))
  for (i in c(1:length(y))){
    aux <- deriv_normal_log_likelihood_expr_param_mu(mu_a = theta[1],
                                                    mu_b = theta[2],
                                                    mu_c = theta[3],
                                                    theta2 = theta[4],
                                                    y= y[i],
                                                    x= x[i])
    grad <- matrix(apply(attr(aux,"gradient"),2,sum))
    K_mat = K_mat + grad %*% t(grad)
  }
  return(1/length(y)*K_mat)
}
```

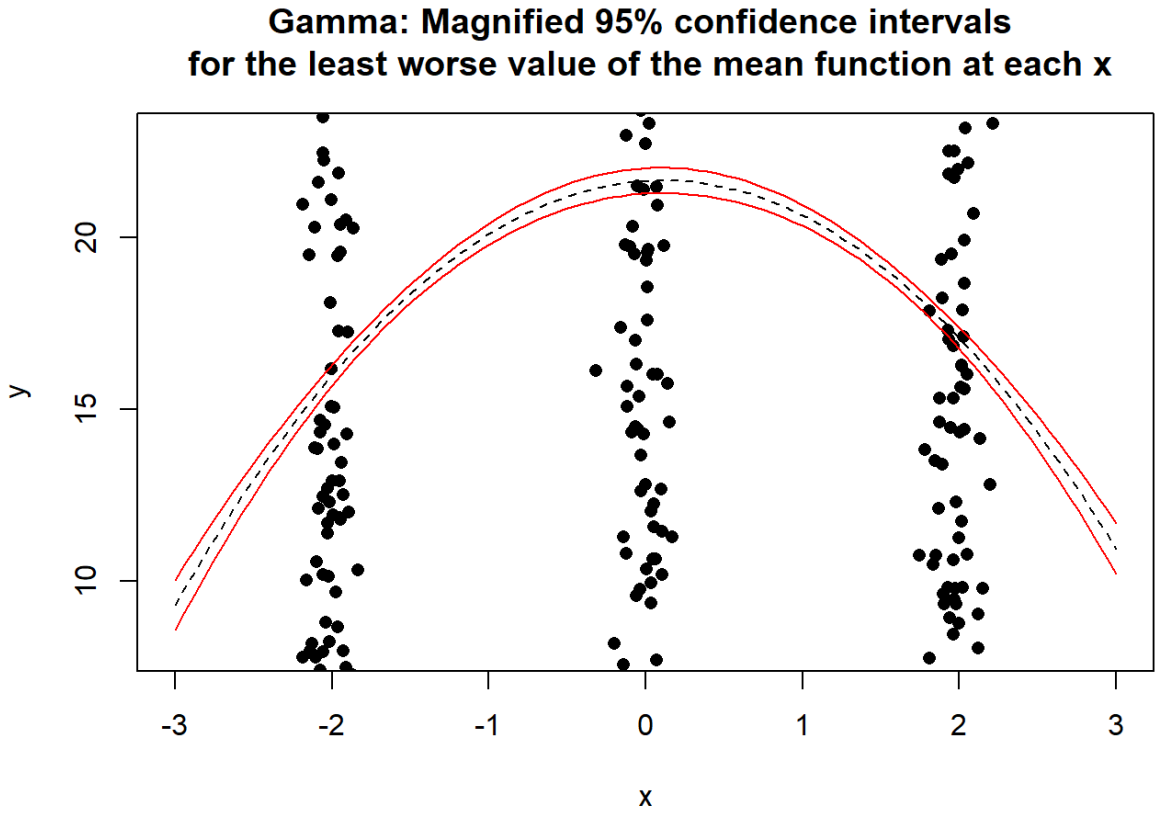
For the gamma distribution we have that

```
n_grid <- 100
x <- seq(-3, 3, length = n_grid)
mean_g <- rep(NA, n_grid)
ci_g <- matrix(NA, nrow = n_grid, ncol = 2)
K = K_mat_gamma_q8(theta = c(optim_gamma_mu_min$par), data = data_q7)
for (i in 1:n_grid) {
  mean_g[i] <- optim_gamma_mu_min$par[1] + optim_gamma_mu_min$par[2] * x[i] + optim_gamma_mu_min$par
[3] * (x[i])^2
  J <- c(1, x[i],x[i]^2, 0)
  KH = solve(optim_gamma_mu_min$hessian) %*% K %*% solve(optim_gamma_mu_min$hessian)
  se <- sqrt(J %*% KH %*% J)
  ci_g[i, 1:2] <- c(mean_g[i] - 1.96 * se, mean_g[i] + 1.96 * se)
}
plot(
  data_q7$x,
  data_q7$y,
  ylab = "y",
  xlab = "x",
  pch = 16,
  xlim = c(-3, 3),
  ylim = c(0, 60)
)
lines(x, mean_g, lty = 2)
lines(x, ci_g[, 1], col = "red")
lines(x, ci_g[, 2], col = "red")
title(main = "Gamma: 95% confidence intervals \n for the least worse value of the mean function at eac
h x")
```



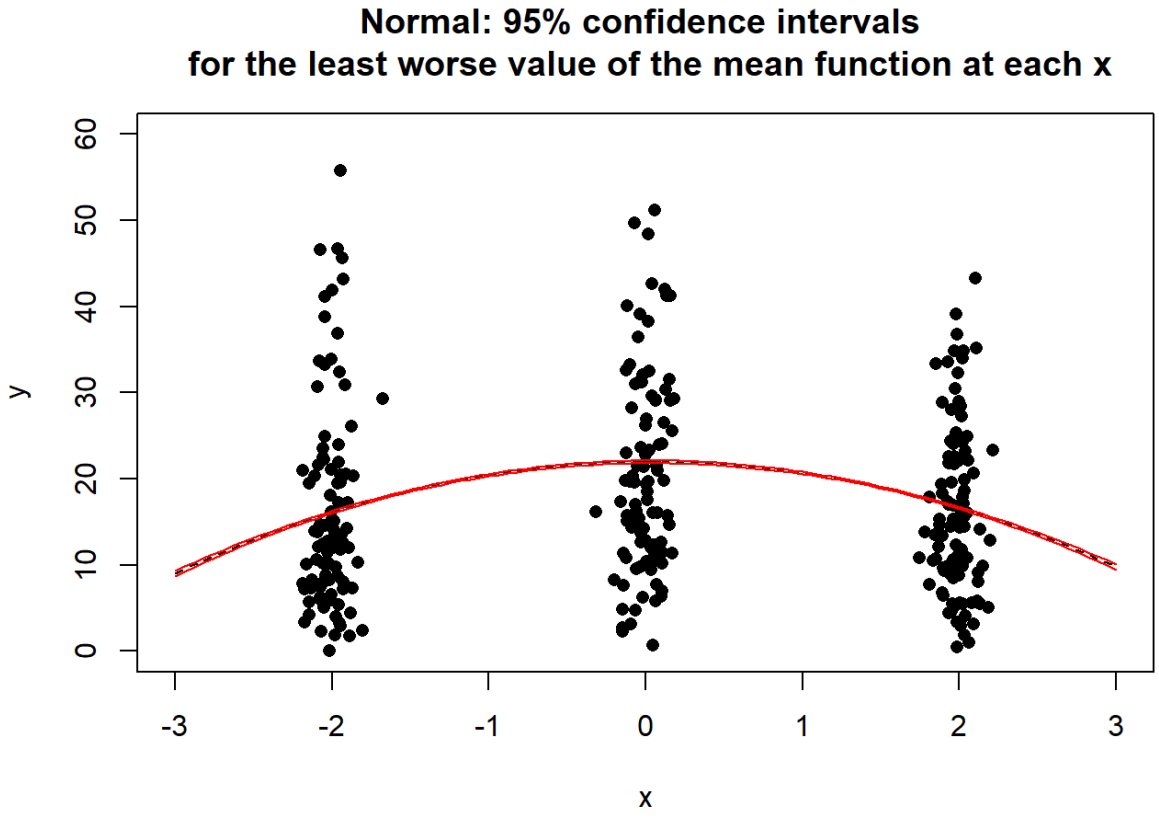
Zooming onto this plot

```
plot(
  data_q7$x,
  data_q7$y,
  ylab = "y",
  xlab = "x",
  pch = 16,
  xlim = c(-3, 3),
  ylim = c(8, 23)
)
lines(x, mean_g, lty = 2)
lines(x, ci_g[, 1], col = "red")
lines(x, ci_g[, 2], col = "red")
title(main = "Gamma: Magnified 95% confidence intervals \n for the least worse value of the mean funct
ion at each x")
```



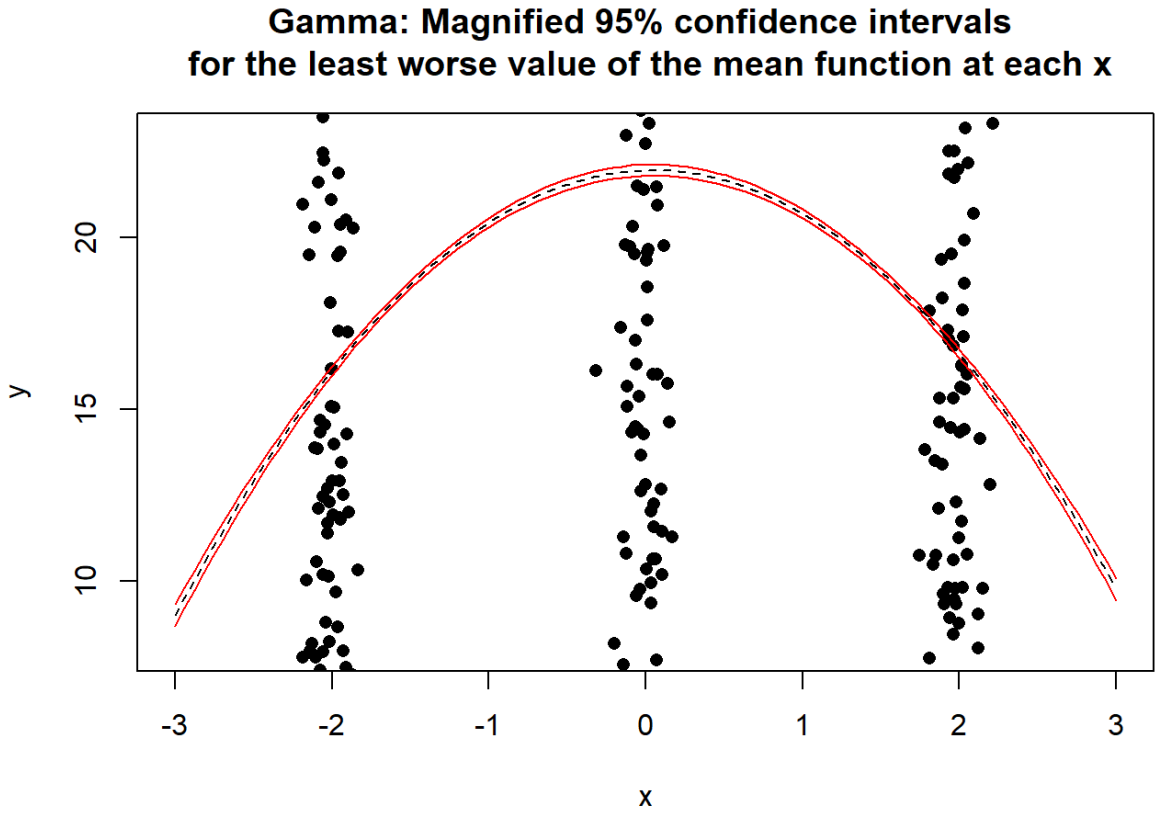
For the normal distribution we have that

```
n_grid <- 100
x <- seq(-3, 3, length = n_grid)
mean_n <- rep(NA, n_grid)
ci_n <- matrix(NA, nrow = n_grid, ncol = 2)
K = K_mat_normal_q8(theta = c(optim_normal_mu_min$par), data = data_q7)
for (i in 1:n_grid) {
  mean_n[i] <- optim_normal_mu_min$par[1] + optim_normal_mu_min$par[2] * x[i] + optim_normal_mu_min$pa
r[3]*(x[i]^2)
  J <- c(1, x[i], (x[i])^2, 0)
  KH = solve(optim_normal_mu_min$hessian) %*% K %*% solve(optim_normal_mu_min$hessian)
  se <- sqrt(J %*% KH %*% J)
  ci_n[i, 1:2] <- c(mean_n[i] - 1.96 * se, mean_n[i] + 1.96 * se)
}
plot(
  data_q7$x,
  data_q7$y,
  ylab = "y",
  xlab = "x",
  pch = 16,
  xlim = c(-3, 3),
  ylim = c(0, 60)
)
lines(x, mean_n, lty = 2)
lines(x, ci_n[, 1], col = "red")
lines(x, ci_n[, 2], col = "red")
title(main = "Normal: 95% confidence intervals \n for the least worse value of the mean function at ea
ch x")
```



Zooming onto this plot

```
plot(
  data_q7$x,
  data_q7$y,
  ylab = "y",
  xlab = "x",
  pch = 16,
  xlim = c(-3, 3),
  ylim = c(8, 23)
)
lines(x, mean_n, lty = 2)
lines(x, ci_n[, 1], col = "red")
lines(x, ci_n[, 2], col = "red")
title(main = "Gamma: Magnified 95% confidence intervals \n for the least worse value of the mean funct
ion at each x")
```

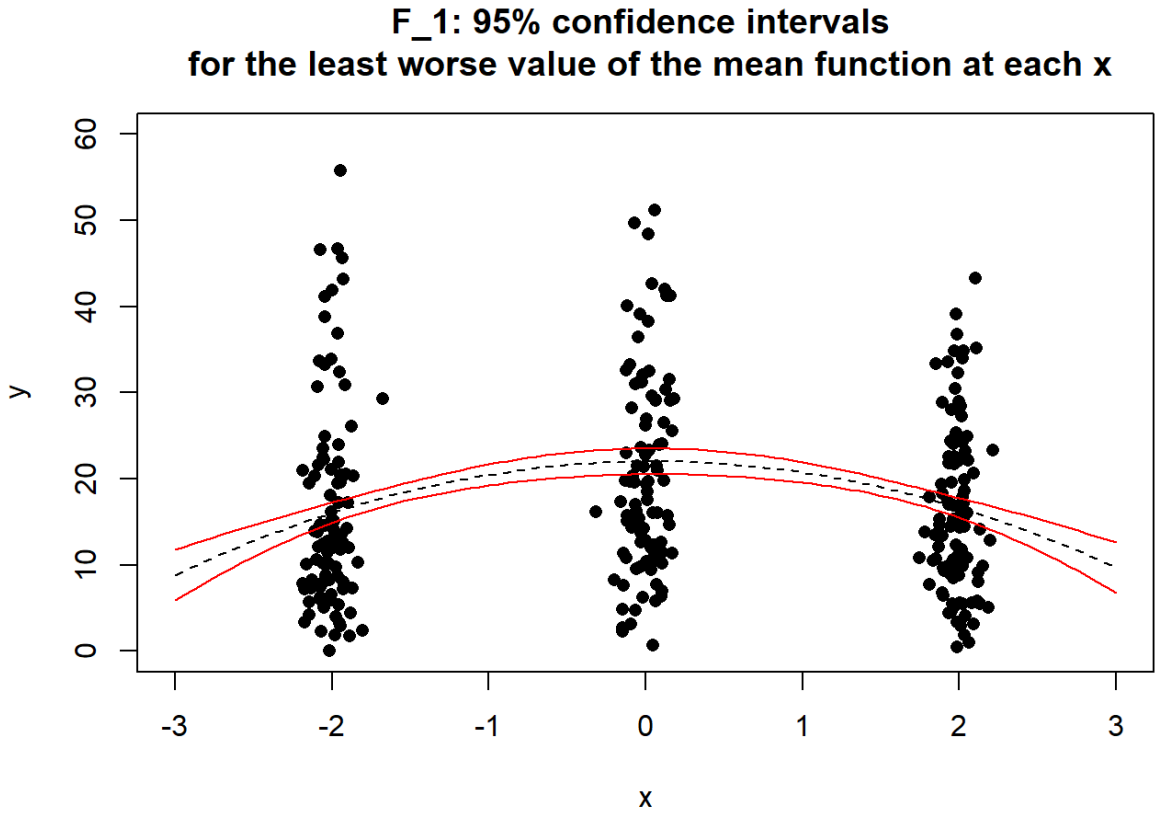


This is for the $\mathcal{F}_1$ distribution above. Compute the $\mathcal{K}$

```
n = length(data_q7$y)
K_mat_1 = matrix(0, nrow = 4, ncol = 4)
xdat = data_q7$x
ydat = data_q7$y
for (i in c(1:n)){
  temp = matrix(neg_grad_log_likelihood_expr_param_mu(theta = c(optim_F1_mu_min$par), x = xdat[i], y =
ydat[i], N=10000))
  K_mat_1 = temp %*% t(temp) + K_mat_1
}
K = 1/n*K_mat_1
```

Here are the confidence intervals for $\mathcal{F}_1$ distribution.

```
n_grid <- 100
x <- seq(-3, 3, length = n_grid)
mean <- rep(NA, n_grid)
ci <- matrix(NA, nrow = n_grid, ncol = 2)
H = neg_hess_log_likelihood_expr_mu(theta = optim_F1_mu_min$par, y = data_q7$y, x = data_q7$x, N = 100
00)
for (i in 1:n_grid) {
  mean[i] <- optim_F1_mu_min$par[1] + optim_F1_mu_min$par[2] * x[i] + optim_F1_mu_min$par[3] * x[i]^2
  J <- c(1, x[i], x[i]^2, 0)
  KH = solve(H) %*% K %*% solve(H)
  se <- sqrt(J %*% KH %*% J)
  ci[i, 1:2] <- c(mean[i] - 1.96 * se, mean[i] + 1.96 * se)
}
plot(
  data_q7$x,
  data_q7$y,
  ylab = "y",
  xlab = "x",
  pch = 16,
  xlim = c(-3, 3),
  ylim = c(0, 60)
)
lines(x, mean, lty = 2)
lines(x, ci[, 1], col = "red")
lines(x, ci[, 2], col = "red")
title(main = "F_1: 95% confidence intervals \n for the least worse value of the mean function at each
x")
```



Our plots of confidence intervals for the least worst values of θ highlight the certainty which we have in our models drawn from the gamma and normal distributions. We can be less certain we have chosen the correct model for the F_1 parametric family.