f. Below is the pseudocode for the EM-algorithm. It was asked in an email thread with the professor if such a format is acceptable to represent pseudocode and it was said that this is okay.

Let $\vec{x} = (x_1, x_2, \cdots, x_n)$ be an observed sample.

Initialize $k = 0$, $\vec{\theta}^{(0)} = \left(\pi^{(0)}, \mu_1^{(0)}, \mu_2^{(0)}, {\sigma_1^2}^{(0)}, {\sigma_2^2}^{(0)}\right)$

Epsilon = $1e^{-10}$

Convergence_flag = TRUE

While (Convergence_flag) {

E-step Compute $Q\left(\vec{\theta}|\vec{\theta}^{(k)}\right)$

$Q\left(\vec{\theta}|\vec{\theta}^{(k)}\right) = \log \pi^{(k)} \sum_{i=1}^{n} E\left[Z_i|x_i, \theta^{(k)}\right] + \log \phi\left(x_i; \mu_1^{(k)}, (\sigma_1^2)^{(k)}\right) \sum_{i=1}^{n} E\left[Z_i|x_i, \theta^{(k)}\right] +$
$\log\left(1 - \pi^{(k)}\right)\left(n - \sum_{i=1}^{n} E\left[Z_i|x_i, \theta^{(k)}\right]\right) + \log \phi\left(x_i; \mu_2^{(k)}, (\sigma_2^2)^{(k)}\right)\left(n - \sum_{i=1}^{n} E\left[Z_i|x_i, \theta^{(k)}\right]\right)$

M-step Maximize $Q\left(\vec{\theta}|\vec{\theta}^{(k)}\right)$ w.r.t. $\vec{\theta}$, set $\vec{\theta}^{(k+1)}$ equal to maximizer of Q

$\pi^{(k+1)} = \frac{\eta^{(k)}}{n} = \frac{1}{n}\sum_{i=1}^{n} E\left[Z_i|x_i, \theta^{(k)}\right]$

$\mu_1^{(k+1)} = \frac{1}{\eta^{(k)}}\sum_{i=1}^{n} \eta_i^{(k)} x_i$

$(\sigma_1^2)^{(k+1)} = \frac{1}{\eta^{(k)}}\sum_{i=1}^{n} \eta_i^{(k)}\left(x_i - \mu_1^{(k+1)}\right)^2$

$\mu_2^{(k+1)} = \frac{1}{n-\eta^{(k)}}\sum_{i=1}^{n}\left(1 - \eta_i^{(k)}\right)x_i$

$(\sigma_2^2)^{(k+1)} = \frac{1}{n-\eta^{(k)}}\sum_{i=1}^{n}\left(1 - \eta_i^{(k)}\right)\left(x_i - \mu_2^{(k+1)}\right)^2$

Check for convergence

If $\frac{\left|\left|\vec{\theta}^{(k)} - \vec{\theta}^{(k-1)}\right|\right|}{||\vec{\theta}^{(k-1)}||} <$ Epsilon {

Convergence_flag = FALSE

}

}

Return $\vec{\theta}^{(k+1)}$

Code Appendix (*NOTE: Towards the end in the steepest ascent algorithm there are two lines commented to allow for switching of updating methods. They alternate between utilizing the identity matrix and the Fisher Information matrix.*)

```
library(plot3D); library(knitr) # Libraries

### Problem 2.5
oilspills <- read.table(file = "oilspills.dat", header = TRUE)

### part a
# log-likelihood of Poisson
g0 <- function(x = oilspills, alpha1, alpha2) {
  n <- x[,2]; b1 <- x[,3]; b2 <- x[,4]
  sum(n * log(alpha1 * b1 + alpha2 * b2)) +
    sum(-alpha1 * b1 - alpha2 * b2) -
    sum(log(factorial(n)))
}

# dg/dalpha1
g1 <- function(x = oilspills, alpha1, alpha2) {
  n <- x[,2]; b1 <- x[,3]; b2 <- x[,4]
  sum((n * b1) / (alpha1 * b1 + alpha2 * b2)) - sum(b1)
}

# dg/dalpha2
g2 <- function(x = oilspills, alpha1, alpha2) {
  n <- x[,2]; b1 <- x[,3]; b2 <- x[,4]
  sum((n * b2) / (alpha1 * b1 + alpha2 * b2)) - sum(b2)
}

# gradient(alpha1, alpha2)
g3 <- function(x = oilspills, alpha1, alpha2) {
  return(c(g1(x, alpha1, alpha2), g2(x, alpha1, alpha2)))
}

# d^2g/dalpha1^2
g4 <- function(x = oilspills, alpha1, alpha2) {
  n <- x[,2]; b1 <- x[,3]; b2 <- x[,4]
  -sum((n * b1^2) / (alpha1 * b1 + alpha2 * b2)^2)
}

# d^2g/dalpha2^2
g5 <- function(x = oilspills, alpha1, alpha2) {
  n <- x[,2]; b1 <- x[,3]; b2 <- x[,4]
  -sum((n * b2^2) / (alpha1 * b1 + alpha2 * b2)^2)
}

# d^2g/(dalpha1 * dalpha2)
g6 <- function(x = oilspills, alpha1, alpha2) {
  n <- x[,2]; b1 <- x[,3]; b2 <- x[,4]
  -sum((n * b1 * b2) / (alpha1 * b1 + alpha2 * b2)^2)
}

# 2x2 Hessian
g7 <- function(x = oilspills, alpha1, alpha2) {
  return(matrix(c(
    g4(x, alpha1, alpha2), g6(x, alpha1, alpha2),
    g6(x, alpha1, alpha2), g5(x, alpha1, alpha2)), ncol = 2))
}

h <- function(x = oilspills, alpha1, alpha2) {  # h function
  return(-solve(g7(x, alpha1, alpha2)) %*% g3(x, alpha1, alpha2))
}

relative_convergence <- function(alpha1_old, alpha1_new,  # Test for convergence
                                 alpha2_old, alpha2_new, epsilon) {
  old_vec <- c(alpha1_old, alpha2_old)
```

```r
  new_vec <- c(alpha1_new, alpha2_new)
  relative_convergence_criterion <- sqrt(sum((new_vec - old_vec)^2)) /
    (sqrt(sum(old_vec^2)) + epsilon)
  bool_flag <- ifelse(relative_convergence_criterion < epsilon, FALSE, TRUE)
  return(c(relative_convergence_criterion, bool_flag))
}

# Plot 3D graph of the log-likelihood function
# Reference: https://www.datamentor.io/r-programming/3d-plot/
xs <- seq(-5, 10, length.out = 100)
ys <- seq(-5, 10, length.out = 100)

# Reference: https://stackoverflow.com/questions/52317124/dims-product-xx-do-not-match-the-length-of-object-xx-error-in-using-r-func
g0_vectorize <- Vectorize(g0, vectorize.args = c("alpha1", "alpha2"))
z <- outer(xs, ys, function(xs, ys) g0_vectorize(alpha1 = xs, alpha2 = ys))

# Reference: https://stackoverflow.com/questions/17606906/find-row-and-column-index-of-maximum-value-in-a-matrix
max_x_y <- which(z == max(z, na.rm = TRUE), arr.ind = TRUE) # 41,40
# xs[max_x_y[1]]; ys[max_x_y[2]] # 1.060606, 0.9090909

persp3D(xs, ys, z,
      main = "log-likelihood of Poisson distribution",
      zlab = "log-likelihood", xlab = "alpha1", ylab = "alpha2",
      theta = 45, phi = 35, ticktype = "detailed")

par(mfrow = c(2,2))
persp(xs, ys, z,
      main = "log-likelihood of Poisson distribution",
      zlab = "log-likelihood", xlab = "alpha1", ylab = "alpha2",
      theta = 15, phi = 10, col = "springgreen", shade = 0.3,
      ticktype = "detailed")

persp(xs, ys, z,
      main = "log-likelihood of Poisson distribution",
      zlab = "log-likelihood", xlab = "alpha1", ylab = "alpha2",
      theta = 90, phi = 10, col = "springgreen", shade = 0.3,
      ticktype = "detailed")

persp(xs, ys, z,
      main = "log-likelihood of Poisson distribution",
      zlab = "log-likelihood", xlab = "alpha1", ylab = "alpha2",
      theta = -40, phi = 10, col = "springgreen", shade = 0.3,
      ticktype = "detailed")

persp(xs, ys, z,
      main = "log-likelihood of Poisson distribution",
      zlab = "log-likelihood", xlab = "alpha1", ylab = "alpha2",
      theta = -180, phi = 10, col = "springgreen", shade = 0.3,
      ticktype = "detailed")
dev.off()

# Newton-Raphson
nr <- function(x = oilspills, alpha1_init, alpha2_init, epsilon = 1e-5) {
  alpha1_new <- alpha1_init; alpha2_new <- alpha2_init  # Initialize variables
  convergence_flag <- TRUE; count <- 0

  while(convergence_flag) {  # Loop through algorithm
    alpha1_old <- alpha1_new; alpha2_old <- alpha2_new

    alpha_vec <- c(alpha1_old, alpha2_old) + h(x, alpha1_old, alpha2_old)
    alpha1_new <- alpha_vec[1]; alpha2_new <- alpha_vec[2]; count <- count + 1

    convergence_vec <- relative_convergence(alpha1_old, alpha1_new, alpha2_old,
                                            alpha2_new, epsilon)
    convergence_value <- convergence_vec[1]; convergence_flag <- convergence_vec[2]
  }
  cat('Iterations: ', count, '\n')
```

```r
  return(c(alpha_vec, count))
}

# Test values for Newton-Raphson
test_initializations <- expand.grid(-1:3, -1:3)
# Reference: https://stackoverflow.com/questions/4227223/convert-a-list-to-a-data-frame
test_initializations <- matrix(unlist(test_initializations),
                               ncol = length(test_initializations))

test_results <- mapply(function(a1, a2) {
    tryCatch(nr(alpha1_init = a1, alpha2_init = a2),
             error = function(e) NA,
             warning = function(w) NA,
             message = function(c) NA
    )
  }, a1 = test_initializations[,1], a2 = test_initializations[,2]
)

test_results <- do.call(rbind, test_results)
test_results <- as.data.frame(test_results)
test_results <- cbind(test_initializations, test_results)
colnames(test_results) <- c("alpha1_init", "alpha2_init", "alpha1", "alpha2", "Iterations")
kable(test_results, format = "latex", caption = "Results for Newton-Raphson")

nr(alpha1_init = -1, alpha2_init = -1)
nr_fisher(alpha1_init = 1, alpha2_init = -1)

# part b
fisher_information <- function(x = oilspills,
                               alpha1, alpha2) {  # Fisher Information matrix
  n <- x[,2]; b1 <- x[,3]; b2 <- x[,4]
  lambda_i <- alpha1 * b1 + alpha2 * b2
  top_left <- -sum(b1^2 / lambda_i)
  bottom_left <- top_right <- -sum(b1 * b2 / lambda_i)
  bottom_right <- -sum(b2^2 / lambda_i)
  matrix(c(top_left, bottom_left, top_right, bottom_right), ncol = 2)
}

h_fisher <- function(x, alpha1, alpha2) {  # h function with Fisher scoring
  return(-solve(fisher_information(x, alpha1, alpha2)) %*%
           g3(x, alpha1, alpha2))
}

nr_fisher <- function(x = oilspills, alpha1_init, alpha2_init, epsilon = 1e-5) {
  alpha1_new <- alpha1_init; alpha2_new <- alpha2_init  # Initialize variables
  convergence_flag <- TRUE; count <- 0

  while(convergence_flag) {  # Loop through algorithm
    alpha1_old <- alpha1_new; alpha2_old <- alpha2_new  # Update old variables

    # Update new variables
    alpha_vec <- c(alpha1_old, alpha2_old) + h_fisher(x, alpha1_old, alpha2_old)
    alpha1_new <- alpha_vec[1]; alpha2_new <- alpha_vec[2]; count <- count + 1

    # Check for convergence
    convergence_vec <- relative_convergence(alpha1_old, alpha1_new, alpha2_old,
                                            alpha2_new, epsilon)
    convergence_value <- convergence_vec[1]; convergence_flag <- convergence_vec[2]
  }
  cat('Iterations: ', count, '\n')
  return(c(alpha_vec, count))
}

test_results_fisher <- mapply(function(a1, a2) {
  tryCatch(nr_fisher(alpha1_init = a1, alpha2_init = a2),
           error = function(e) NA,
           warning = function(w) NA,
           message = function(c) NA
  )
```

```
  }, a1 = test_initializations[,1], a2 = test_initializations[,2]
)

test_results_fisher <- do.call(rbind, test_results_fisher)
test_results_fisher <- as.data.frame(test_results_fisher)
test_results_fisher <- cbind(test_initializations, test_results_fisher)
colnames(test_results_fisher) <- c("alpha1_init", "alpha2_init", "alpha1", "alpha2", "Iterations")
kable(test_results_fisher, format = "latex", caption = "Results for Newton-Raphson with Fisher Scoring")

# part e
backtrack_check <- function(x = oilspills,
                            alpha1_old_btc,
                            alpha1_test,
                            alpha2_old_btc,
                            alpha2_test,
                            alpha_btc,
                            backtracks_btc,
                            backtrack_flag_btc) {
  step_old <- g0(x = x,  # Calculate steps
                 alpha1 = alpha1_old_btc,
                 alpha2 = alpha2_old_btc)
  step_test <- g0(x = x,
                  alpha1 = alpha1_test,
                  alpha2 = alpha2_test)

  if(step_test < step_old) {  # Check if "downhill" or sideways
    alpha_btc <- alpha_btc / 2
    backtracks_btc <- backtracks_btc + 1
  } else {  # Stop tuning alpha
    backtrack_flag_btc <- FALSE
  }
  return(c(alpha_btc, backtracks_btc, backtrack_flag_btc))
}

alpha_tuner <- function(x = oilspills,
                        alpha1_old_at,
                        alpha2_old_at,
                        alpha_at,
                        backtracks_at) {
  backtrack_flag <- TRUE  # (Re-)Initialize blacktracking variables

  while (backtrack_flag) {  # Loop until a suitable alpha is found
    # theta_test <- c(alpha1_old_at, alpha2_old_at) + # trying to find first update
      # alpha_at * h_fisher(x, alpha1_old_at, alpha2_old_at)
    theta_test <- c(alpha1_old_at, alpha2_old_at) + # trying to find first update
      alpha_at * g3(x, alpha1_old_at, alpha2_old_at)
    backtrack_vec <- backtrack_check(x = x,
                                     alpha1_old_btc = alpha1_old_at,
                                     alpha1_test = theta_test[1],
                                     alpha2_old_btc = alpha2_old_at,
                                     alpha2_test = theta_test[2],
                                     alpha_btc = alpha_at,
                                     backtracks_btc = backtracks_at,
                                     backtrack_flag_btc = backtrack_flag)
    alpha_at <- backtrack_vec[1]; backtracks_at <- backtrack_vec[2]
    backtrack_flag <- backtrack_vec[3]
  }
  return(c(alpha_at, backtracks_at))
}

steepest_ascent <- function(x = oilspills,
                            alpha1_init,
                            alpha2_init,
                            epsilon = 1e-10) {
  alpha1_new <- alpha1_old <- alpha1_init  # Initialize variables
  alpha2_new <- alpha2_old <- alpha2_init
  convergence_flag <- TRUE; count <- 0; backtracks <- 0; alpha <- 1

  while(convergence_flag) {  # Loop through algorithm
```

```
    alpha1_old <- alpha1_new; alpha2_old <- alpha2_new; alpha <- 1

    # Tune alpha
    alpha_tune_vec <- alpha_tuner(x = x,
                                  alpha1_old_at = alpha1_old,
                                  alpha2_old_at = alpha2_old,
                                  alpha_at = alpha,
                                  backtracks_at = backtracks)
    alpha <- alpha_tune_vec[1]; backtracks <- alpha_tune_vec[2]

    # Update old / new variables
    # theta_vec <- c(alpha1_old, alpha2_old) + alpha * h_fisher(x, alpha1_old, alpha2_old)
    theta_vec <- c(alpha1_old, alpha2_old) + alpha * g3(x, alpha1_old, alpha2_old)
    alpha1_new <- theta_vec[1]; alpha2_new <- theta_vec[2]; count <- count + 1

    # Check for convergence
    convergence_vec <- relative_convergence(alpha1_old, alpha1_new, alpha2_old,
                                            alpha2_new, epsilon)
    convergence_flag <- convergence_vec[1]; convergence_value <- convergence_vec[2]
  }
  cat('Iterations: ', count, '\n')
  cat('Backtracks: ', backtracks, '\n')

  return(c(theta_vec, count, backtracks))
}

test_results_steepest_ascent_identity <- mapply(function(a1, a2) {
  tryCatch(steepest_ascent(alpha1_init = a1, alpha2_init = a2),
           error = function(e) NA,
           warning = function(w) NA,
           message = function(c) NA
  )
  }, a1 = test_initializations[,1], a2 = test_initializations[,2]
)

test_results_steepest_ascent_identity <- do.call(rbind, test_results_steepest_ascent_identity)
test_results_steepest_ascent_identity <- as.data.frame(test_results_steepest_ascent_identity)
test_results_steepest_ascent_identity <- cbind(test_initializations, test_results_steepest_ascent_identit
y)
colnames(test_results_steepest_ascent_identity) <- c("alpha1_init", "alpha2_init", "alpha1", "alpha2", "It
erations", "Backtracks")
kable(test_results_steepest_ascent_identity, format = "latex", caption = "Results for Steepest Ascent with
 Identity Matrix")

test_results_steepest_ascent_fisher <- mapply(function(a1, a2) {
  tryCatch(steepest_ascent(alpha1_init = a1, alpha2_init = a2),
           error = function(e) NA,
           warning = function(w) NA,
           message = function(c) NA
  )
  }, a1 = test_initializations[,1], a2 = test_initializations[,2]
)

test_results_steepest_ascent_fisher <- do.call(rbind, test_results_steepest_ascent_fisher)
test_results_steepest_ascent_fisher <- as.data.frame(test_results_steepest_ascent_fisher)
test_results_steepest_ascent_fisher <- cbind(test_initializations, test_results_steepest_ascent_fisher)
colnames(test_results_steepest_ascent_fisher) <- c("alpha1_init", "alpha2_init", "alpha1", "alpha2", "Iter
ations", "Backtracks")
kable(test_results_steepest_ascent_fisher, format = "latex", caption = "Results for Steepest Ascent with F
isher Matrix")
```