* 1. 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 be an observed sample.

Initialize ,

Epsilon =

Convergence\_flag = TRUE

While (Convergence\_flag) {

# E-step Compute

# M-step Maximize w.r.t. , set equal to maximizer of

# Check for convergence

If Epsilon {

Convergence\_flag = FALSE

}

}

Return

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)  
 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')  
 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\_identity)  
colnames(test\_results\_steepest\_ascent\_identity) <- c("alpha1\_init", "alpha2\_init", "alpha1", "alpha2", "Iterations", "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", "Iterations", "Backtracks")  
kable(test\_results\_steepest\_ascent\_fisher, format = "latex", caption = "Results for Steepest Ascent with Fisher Matrix")