Assignment 2 Question 2

Sheen Thusoo

Part d)

```
data <- read.csv("EconomicMobility.csv")
```

```
# Prerequisite functions all of which we discussed in class
gradientAscent <- function(theta,
                            rhoFn,
                            gradientFn,
                            lineSearchFn,
                            testConvergenceFn,
                            maxIterations = 100,
                            tolerance = 1E-6,
                            relative = FALSE,
                            lambdaStepsize = 0.01,
                            lambdaMax = 0.5) {

  for (i in 1:maxIterations) {
    g         <- gradientFn(theta) # Unnormalized gradient.
    glength   <- sqrt(sum(g ^ 2))  # Gradient vector length.
    g         <- g / glength       # Unit vector gradient.
    lambda    <- lineSearchFn(theta, rhoFn, g,
                            lambdaStepsize = lambdaStepsize,
                            lambdaMax = lambdaMax)
    thetaNew  <- theta + lambda * g
    converged <- testConvergenceFn(thetaNew, theta,
                                   tolerance = tolerance,
                                   relative = relative)
    theta = thetaNew #Reza added this update
    if (converged) break
  }

  ## Return information about the gradient descent procedure.
  return(list(theta = theta, converged = converged,
              iteration = i, fnValue = rhoFn(theta)))
}

gridLineSearch <- function(theta,
                           rhoFn,
                           g,
                           lambdaStepsize = 0.01,
                           lambdaMax = 1) {
  ## Define equally-spaced grid of lambdas to search over.
  lambdas <- seq(from = 0, by = lambdaStepsize, to = lambdaMax)
  ## Evaluate the objective rho at each such lambda.
  rhoVals <- Map(function(lambda) {rhoFn(theta + lambda * g)}, lambdas)
  ## Return the lambda that gave the minimum objective.
  return(lambdas[which.max(rhoVals)])
}

testConvergence <- function(thetaNew,
                            thetaOld,
                            tolerance = 1E-10,
                            relative = FALSE) {
   sum(abs(thetaNew - thetaOld)) <
    if (relative) tolerance * sum(abs(thetaOld)) else tolerance
}
```

```
rho <- function(x) {
  alpha = x[1]
  beta = x[2]
  loglikelihood <- 0
  P <- data$Commute
  for (y in P) {
    loglikelihood <- loglikelihood + alpha*log(beta) +
      (alpha -1)*log(y) - log(gamma(alpha)) - y*beta
  }
  return(loglikelihood)
}

g <- function(x) {
  alpha = x[1]
  beta = x[2]
  y <- data$Commute
  grad1 <- sum(log(beta) + log(y) - digamma(alpha))
  grad2 <- sum((alpha/beta) - y)
  return(c(grad1 , grad2))
}
```

```
# Starting at (2, 2) as mentioned in Piazza
Optim1 = gradientAscent(rhoFn = rho, gradientFn = g, theta = c(2, 2),
                         lineSearchFn = gridLineSearch,
                         testConvergenceFn = testConvergence,
                        maxIterations = 1000, lambdaMax = 5)
Optim1
```

```
## $theta
## [1] 10.47992 23.09889
##
## $converged
## [1] TRUE
##
## $iteration
## [1] 104
##
## $fnValue
## [1] 422.0633
```