Assignment 2 Question 2

Sheen Thusoo

Part e)

```
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 on Piazza
# lambda=0,1,2,3,4,5
Optim1 = gradientAscent(rhoFn = rho, gradientFn = g, theta = c(2, 2),
                        lineSearchFn = gridLineSearch,
                        testConvergenceFn = testConvergence,
                        maxIterations = 1000, lambdaStepsize = 1, lambdaMax = 5)
Optim1
```

```
## $theta
## [1] 1.278860 2.692789
##
## $converged
## [1] TRUE
##
## $iteration
## [1] 2
##
```

```
## $fnValue
## [1] -61.29835
```

```
# Starting at (2, 2) as mentioned on Piazza
# lambda=0.0001,0.0002,...,0.01
Optim2 = gradientAscent(rhoFn = rho, gradientFn = g, theta = c(2, 2),
                        lineSearchFn = gridLineSearch,
                        testConvergenceFn = testConvergence,
                        maxIterations = 1000, lambdaStepsize = 0.0001, lambdaMax = 0.01)
Optim2
```

```
## $theta
## [1]  5.244547 11.094237
##
## $converged
## [1] FALSE
##
## $iteration
## [1] 1000
##
## $fnValue
## [1] 345.7479
```

What do you observe? Justify your observation.
For the first lambda sequence, we converge within 2 iterations to a value that is much different than that computed in part d). This seems to be incorrect as the value computed in part d) was much lower. Thus, we may have converged to a local minimum using this sequence. This may be because the step size was much larger (1 rather than 0.01 in d). Thus, we may have jumped over the solution and hence we computed a local minimum value.

The second lambda sequence does not converge and stops when it reaches the maximum iteration of 1000. Thus, for this sequence, the gradient descent algorithm was unable to find a solution. This is because the step size was very small (0.0001) and so the algorithm took a long time to compute and could not converge within the maximum number of iterations.