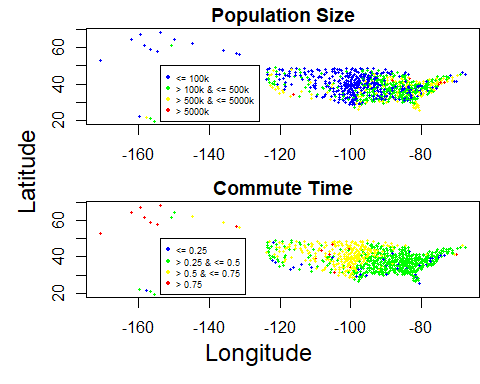
Assignment 2

Fenglin Chen

26/10/2021

### Question 1 a)

econ <- read.csv("EconomicMobility.csv", header=TRUE)  
colours <- c('blue', 'green', 'yellow', 'red')  
  
# Divide communities by population size  
econ$PopCategory[econ$Population <= 100000] = 1  
econ$PopCategory[econ$Population > 100000 & econ$Population <= 500000] = 2  
econ$PopCategory[econ$Population > 500000 & econ$Population <= 5000000] = 3  
econ$PopCategory[econ$Population > 5000000] = 4  
  
# Divide communities by commute time  
econ$CommCategory[econ$Commute <= 0.25] = 1  
econ$CommCategory[econ$Commute > 0.25 & econ$Commute <= 0.5] = 2  
econ$CommCategory[econ$Commute > 0.5 & econ$Commute <= 0.75] = 3  
econ$CommCategory[econ$Commute > 0.75] = 4  
  
# Location vs. population size  
par(mfrow=c(2,1), mar=c(2.5, 2.5, 1.5, 0), oma=c(2, 2, 0, 0))  
plot(Latitude ~ Longitude, data=econ, pch=19, col=colours[econ$PopCategory],  
 main="Population Size", cex=0.4)  
legend(-154, 50, col=colours, cex=0.585, pch=19,   
 legend=c("<= 100k", "> 100k & <= 500k", "> 500k & <= 5000k", "> 5000k"))  
  
# Location vs. commute time  
plot(Latitude ~ Longitude, data=econ, pch=19, col=colours[econ$CommCategory],  
 main="Commute Time", cex=0.4)  
legend(-154, 50, col=colours, cex=0.585, pch=19,   
 legend=c("<= 0.25", "> 0.25 & <= 0.5", "> 0.5 & <= 0.75", "> 0.75"))  
  
# Axis labels  
mtext("Longitude", side=1, line=0, outer=TRUE, cex=1.5)  
mtext("Latitude", side=2, line=0, outer=TRUE, cex=1.5, las=0)



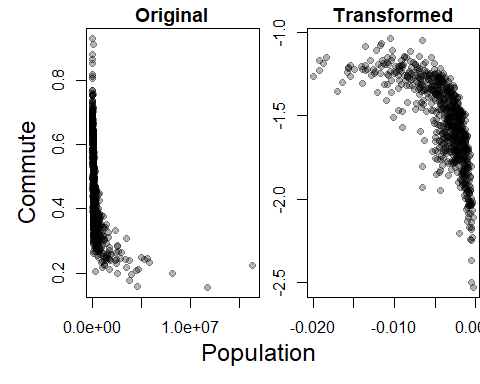
The plot of population size shows that bigger communities are mostly on the East coast and in California, with most smaller communities (<= 100,000) spread out in the Western and Central regions. There is a gradual reduction in community size as longtitude decreases.

The plot of commute time shows a higher proportion of communities with short travel times on the East side than the West.

The two variables appear to be highly correlated: smaller communities have higher fractions of workers commuting less than 15 minutes and vice versa. Almost every red dot in the Commute plot (short commute time) corresponds to a blue dot in the Population plot (small population size).

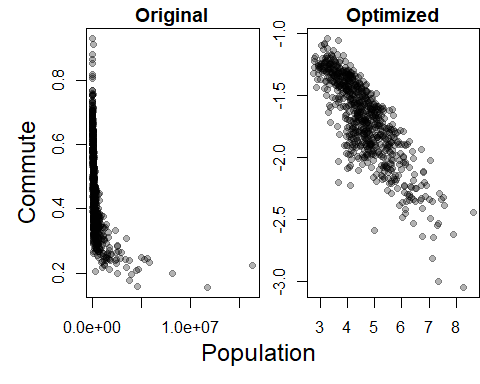
### Question 1 b)

# Using the power function from class  
powerfun <- function(x, alpha) {  
 if(sum(x <= 0) > 1) stop("x must be positive")  
 if (alpha == 0)  
 ln(x)  
 else if (alpha > 0) {  
 x^alpha  
 } else -x^alpha  
}  
  
# Define the multi-variate power transformation  
power.transform.factory <- function(pop1, pop2) {  
 function(alpha\_x, alpha\_y) {  
 return(list(x = powerfun(pop1, alpha\_x), y = powerfun(pop2, alpha\_y)))  
 }  
}  
  
# Transform the data (contains $x=population and $y=commute)  
power.transform <- power.transform.factory(econ$Population, econ$Commute)  
transformed <- power.transform(-1/2, -1/2)  
  
# Generate two scatterplots  
par(mfrow=c(1,2), mar=c(2.5, 2.5, 1.5, 0), oma=c(2, 2, 0, 0))  
plot(Commute ~ Population, data=econ, pch=19, main="Original",   
 col=adjustcolor(col="black", alpha=0.3))  
plot(y ~ x, data=transformed, pch=19, main="Transformed",   
 col=adjustcolor(col="black", alpha=0.3))  
  
# Axis labels  
mtext("Population", side=1, line=0, outer=TRUE, cex=1.5)  
mtext("Commute", side=2, line=0, outer=TRUE, cex=1.5, las=0)



### Question 1 c)

# Calculate the strength of the linear relationship  
L <- function(alphas) {  
 transformed <- power.transform(alphas[1], alphas[2])  
 return(1 - cor(transformed$x, transformed$y)\*\*2)  
}  
  
# Optimize L with initial values (1, 1)  
res <- nlminb(start=c(1, 1), objective=L)  
optim.transformed = power.transform(res$par[1], res$par[2])  
  
# Scatterplot the original vs. power-transformed data  
par(mfrow=c(1,2), mar=c(2.5, 2.5, 1.5, 0), oma=c(2, 2, 0, 0))  
plot(Commute ~ Population, data=econ, pch=19, main="Original",   
 col=adjustcolor(col="black", alpha=0.3))  
plot(y ~ x, data=optim.transformed, pch=19, main="Optimized",   
 col=adjustcolor(col="black", alpha=0.3))  
  
# Axis labels  
mtext("Population", side=1, line=0, outer=TRUE, cex=1.5)  
mtext("Commute", side=2, line=0, outer=TRUE, cex=1.5, las=0)



Through optimization, the best parameters are found to be , , converging after 12 iterations.

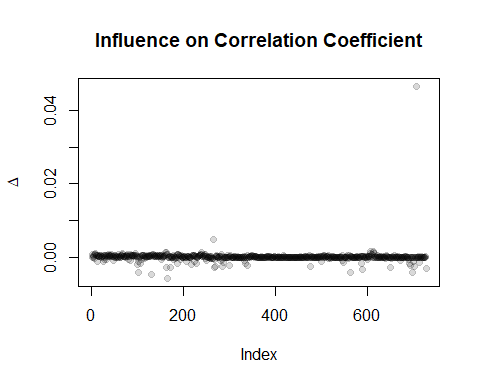
# Calculate the correlation coefficient for each scatter plot  
c(cor(econ$Population, econ$Commute), cor(optim.transformed$x, optim.transformed$y))

## [1] -0.4141503 -0.8274049

The coefficient of correlation for the original data is , and for the transformed data. Therefore, the transformation resulted in an objectively more linear relationship.

### Question 1 d)

# Baseline correlation  
correlation <- cor(econ$Population, econ$Commute)  
  
# Calculate the influence of (pop\_i, commute\_i), i = 1, ..., 729 on the correlation coefficient Pxy  
delta <- rep(0, length(econ$Population))  
for (i in 1:length(econ$Population)) {  
 delta[i] = correlation - cor(econ$Population[-i], econ$Commute[-i])  
}  
  
# Plot the influence of each observation on the correlation coefficient  
plot(delta, main="Influence on Correlation Coefficient", pch=19,  
 col=adjustcolor("black", alpha=0.15),  
 xlab="Index", ylab=bquote(Delta))



Yes, there are some points with much more influence than others. For example, the pair at has an influence of , which is ~25 standard deviations away from the mean. Most influences are below average.

library(knitr)  
pop.special <- table(econ$PopCategory[delta < mean(delta)-sd(delta) | delta > mean(delta)+sd(delta)])  
comm.special <- table(econ$CommCategory[delta < mean(delta)-sd(delta) | delta > mean(delta)+sd(delta)])  
kable(pop.special, col.names=c("Population", "Frequency"), align="cc")

|  |  |
| --- | --- |
| Population | Frequency |
| 3 | 17 |
| 4 | 5 |

kable(comm.special, col.names=c("Commute Time", "Frequency"), align="cc")

|  |  |
| --- | --- |
| Commute Time | Frequency |
| 1 | 18 |
| 2 | 4 |

The outliers, defined by being more than 1 standard deviation away from the mean, are special because they only belong to population groups 3 and 4, and commute groups 1 and 2. This means that communities with higher populations and longer commute times are more influential than others.

### Question 2 a)

The log-likelihood of is as follows:

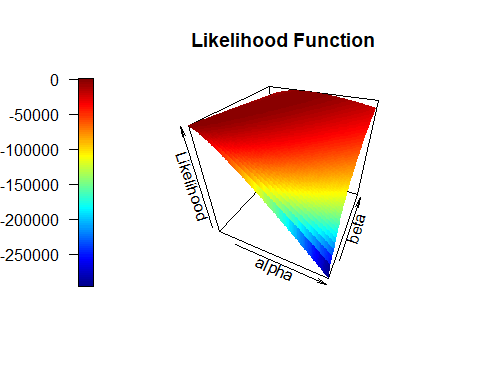
### Question 2 b)

# Return a function that calculates the log-likelihood given (a, b) as input  
# Make sure alpha and beta never go below 0  
CreateLogLikeFunction <- function(obs) {  
 function(alpha, beta) {  
 alpha = max(alpha, 1e-06)  
 beta = max(beta, 1e-06)  
 sum(alpha\*log(beta) + (alpha-1)\*log(obs) - log(gamma(alpha)) - obs\*beta)  
 }  
}  
like.function <- CreateLogLikeFunction(econ$Commute)  
like.function(2, 2)

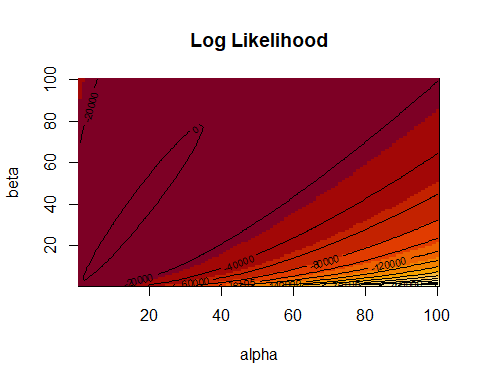
## [1] -262.4003

### Question 2 c)

# 3D plot of the log-likelihood function  
alpha <- seq(1, 100)  
beta <- seq(1, 100)  
mat <- outer(alpha, beta, FUN=Vectorize(like.function))  
  
library(plot3D)  
colours <- colorRampPalette(c("blue", "red"))(100)  
persp3D(x=alpha, y=beta, z=mat, xlab="alpha", ylab="beta", zlab="Likelihood",  
 main="Likelihood Function", theta=25, phi=35, ticktype="simple",   
 col.palette = heat.colors, colkey=list(side=2))



# Heatmap with superimposed contour plot  
image(x=alpha, y=beta, z=mat, xlab="alpha", ylab="beta",   
 main="Log Likelihood")  
contour(x=alpha, y=beta, z=mat, nlevels=20, add=TRUE)



### Question 2 d)

The gradient functions are:

gradientDescent <- function(alpha=2, beta=2, rhoFn, alphaPrime, betaPrime, lineSearchFn, testConvergenceFn, maxIterations = 100, tolerance = 1e-06, relative = FALSE, lambdaStepsize = 0.01, lambdaMax = 5) {  
 # Save paths  
 alphas.all <- c(alpha, rep(NA, maxIterations-1))  
 betas.all <- c(beta, rep(NA, maxIterations-1))  
 loglike.all <- c(rhoFn(alpha, beta), rep(NA, maxIterations-1))  
   
 converged <- FALSE  
 i <- 0  
 while (!converged & i <= maxIterations) {  
 # Gradient/derivative of alpha and beta  
 g.alpha <- alphaPrime(alpha, beta)  
 g.beta <- betaPrime(alpha, beta)  
   
 # Gradient direction of alpha and beta (divide by length)  
 glength.alpha <- sqrt(sum(g.alpha^2))  
 glength.beta <- sqrt(sum(g.beta^2))  
 if (glength.alpha > 0) {g.alpha <- g.alpha/glength.alpha}  
 if (glength.beta > 0) {g.beta <- g.beta/glength.beta}  
   
 # Search for best step size in the direction of the gradients (ascent)  
 lambda <- lineSearchFn(alpha, beta, rhoFn, g.alpha, g.beta,   
 lambdaStepsize = lambdaStepsize, lambdaMax = lambdaMax)  
 alphaNew <- alpha + lambda[1] \* g.alpha  
 betaNew <- beta + lambda[2] \* g.beta  
   
 # Test whether the change is small enough  
 converged <- testConvergenceFn(c(alphaNew, betaNew), c(alpha, beta),   
 tolerance = tolerance, relative = relative)  
 alpha <- alphaNew  
 beta <- betaNew  
 alphas.all[i+1] <- alpha  
 betas.all[i+1] <- beta  
 loglike.all[i+1] <- rhoFn(alpha, beta)  
 i <- i + 1  
 }  
 list(params = list(alpha=alpha, beta=beta, converged = converged, iteration = i,   
 fnValue = rhoFn(alpha, beta)),   
 a = alphas.all, b = betas.all, l = loglike.all)  
}  
  
gridLineSearch <- function(alpha, beta, rhoFn, g.alpha, g.beta, lambdaStepsize = 0.01, lambdaMax = 1) {  
 lambdas <- seq(from = 0, by = lambdaStepsize, to = lambdaMax)  
 rhoVals <- outer(alpha + lambdas \* g.alpha,   
 beta + lambdas \* g.beta,   
 Vectorize(rhoFn))  
 lambdas[arrayInd(which.max(rhoVals), dim(rhoVals))]  
}  
  
testConvergence <- function(thetaNew, thetaOld, tolerance = 1e-10,   
 relative = FALSE) {  
 sum(abs(thetaNew - thetaOld)) < if (relative)   
 tolerance \* sum(abs(thetaOld)) else tolerance  
}  
  
alphaPrime <- function(alpha, beta) {  
 sum(log(beta) + log(econ$Commute) - digamma(alpha))  
}  
betaPrime <- function(alpha, beta) {  
 sum(alpha/beta - econ$Commute)  
}

# Minimize the negative of the likelihood function  
res1 <- gradientDescent(alpha=2, beta=2, rhoFn = like.function, alphaPrime = alphaPrime,   
 betaPrime = betaPrime, lineSearchFn = gridLineSearch,   
 testConvergenceFn = testConvergence)  
res1$params

## $alpha  
## [1] 10.48  
##   
## $beta  
## [1] 23.11  
##   
## $converged  
## [1] TRUE  
##   
## $iteration  
## [1] 10  
##   
## $fnValue  
## [1] 422.0639

### Question 2 e)

res2 <- gradientDescent(alpha=2, beta=2, rhoFn = like.function, alphaPrime = alphaPrime,   
 betaPrime = betaPrime, lineSearchFn = gridLineSearch,   
 testConvergenceFn = testConvergence,  
 lambdaStepsize = 1, lambdaMax = 5, maxIterations = 1000)  
res2$params

## $alpha  
## [1] 4  
##   
## $beta  
## [1] 9  
##   
## $converged  
## [1] TRUE  
##   
## $iteration  
## [1] 3  
##   
## $fnValue  
## [1] 290.0545

res3 <- gradientDescent(alpha=2, beta=2, rhoFn = like.function, alphaPrime = alphaPrime,   
 betaPrime = betaPrime, lineSearchFn = gridLineSearch,   
 testConvergenceFn = testConvergence,  
 lambdaStepsize = 0.0001, lambdaMax = 0.01, maxIterations = 1000)  
res3$params

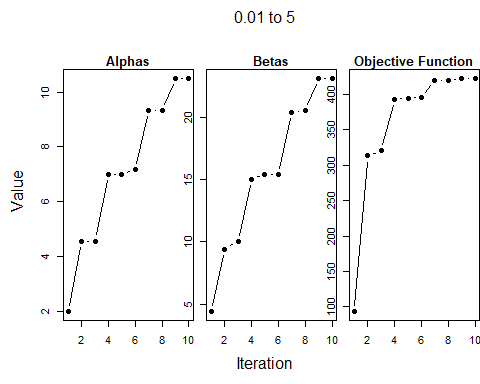
## $alpha  
## [1] 5.6809  
##   
## $beta  
## [1] 12.01  
##   
## $converged  
## [1] FALSE  
##   
## $iteration  
## [1] 1001  
##   
## $fnValue  
## [1] 359.9704

Comparing these two approaches to the one in Q2 d), it is clear that the new algorithms don’t perform as well. That is, they were unable to achieve a higher objective function score.

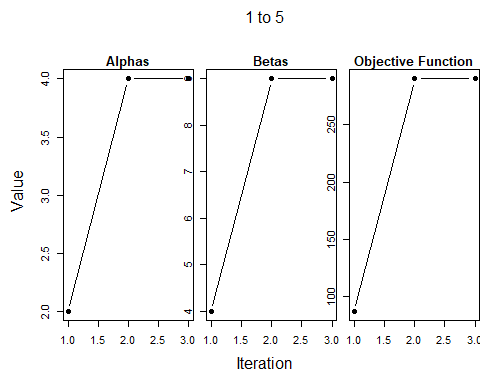
For , the step sizes are too large to hit the top of the curve, resulting in a trajectory that bounces back and forth. Hence, it quickly converges on step 3 because it is unable to make finer adjustments. On the other hand, doesn’t take enough of a step towards the center, and was unable to converge even after 1000 steps.

### Question 2 f)

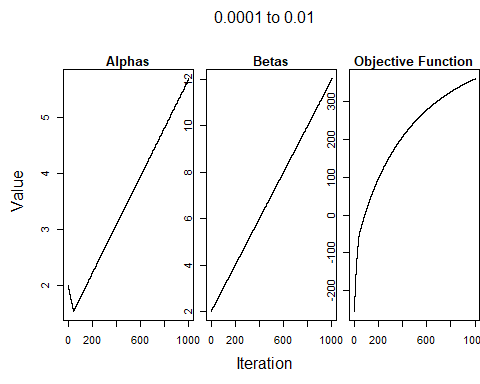
par(mfrow=c(1,3), mar=c(1, 1, 1.5, 0), oma=c(4, 4, 4, 0))  
plot(res1$a[!is.na(res1$a)], pch=19, type='b', main="Alphas")  
plot(res1$b[!is.na(res1$b)], pch=19, type='b', main="Betas")  
plot(res1$l[!is.na(res1$l)], pch=19, type='b', main="Objective Function")  
mtext('Iteration', side=1, outer=TRUE, line=2)  
mtext('Value', side=2, outer=TRUE, line=2)  
mtext('0.01 to 5', side=3, outer=TRUE, line=2)



par(mfrow=c(1,3), mar=c(1, 1, 1.5, 0), oma=c(4, 4, 4, 0))  
plot(res2$a[!is.na(res2$a)], pch=19, type='b', main="Alphas")  
plot(res2$b[!is.na(res2$b)], pch=19, type='b', main="Betas")  
plot(res2$l[!is.na(res2$l)], pch=19, type='b', main="Objective Function")  
mtext('Iteration', side=1, outer=TRUE, line=2)  
mtext('Value', side=2, outer=TRUE, line=2)  
mtext('1 to 5', side=3, outer=TRUE, line=2)

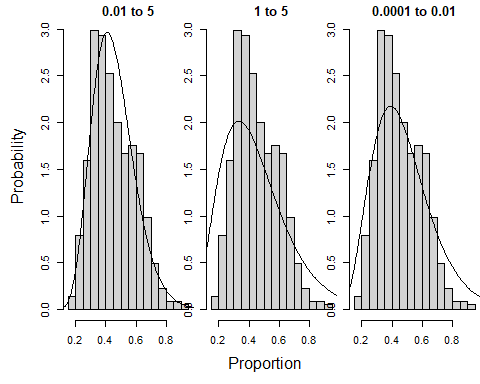


par(mfrow=c(1,3), mar=c(1, 1, 1.5, 0), oma=c(4, 4, 4, 0))  
plot(res3$a[!is.na(res3$a)], pch=19, type='S', main="Alphas")  
plot(res3$b[!is.na(res3$b)], pch=19, type='S', main="Betas")  
plot(res3$l[!is.na(res3$l)], pch=19, type='S', main="Objective Function")  
mtext('Iteration', side=1, outer=TRUE, line=2)  
mtext('Value', side=2, outer=TRUE, line=2)  
mtext('0.0001 to 0.01', side=3, outer=TRUE, line=2)



### Question 2 g)

gamma.pdf <- function(alpha, beta) {  
 function(y) {  
 beta\*\*alpha \* y\*\*(alpha-1) / gamma(alpha) \* exp(-y \* beta)  
 }  
}  
pdf1 <- gamma.pdf(res1$params$alpha, res1$params$beta)  
pdf2 <- gamma.pdf(res2$params$alpha, res2$params$beta)  
pdf3 <- gamma.pdf(res3$params$alpha, res3$params$beta)  
  
par(mfrow=c(1,3), mar=c(1, 1, 1.5, 0), oma=c(4, 4, 0, 0))  
hist(econ$Commute, probability=TRUE, breaks=20, main="0.01 to 5")  
plot(pdf1, add=TRUE)  
hist(econ$Commute, probability=TRUE, breaks=20, main="1 to 5")  
plot(pdf2, add=TRUE)  
hist(econ$Commute, probability=TRUE, breaks=20, main="0.0001 to 0.01")  
plot(pdf3, add=TRUE)  
  
mtext('Proportion', side=1, outer=TRUE, line=2)  
mtext('Probability', side=2, outer=TRUE, line=2)



The plots show that algorithm 1, which converged best to the objective function, fits the probability histogram of Commute the best. This also proves that the objective function was good at approximating the shape of the data.

In plot 2, the estimation curve has a poor fit and is also skewed to the side. The skew is expected because it was the worst fit to the objective function. The lack of small s during grid search meant it could only estimate the rough shape of the data.

Finally, plot 3 shows a curve that almost approximates the proportion of short commutes. If algorithm 3 was given more iterations/time to train, it would be as good as algorithm 1.