Annex

Problem 1

```
# Problem 1

# Clear memory, include libraries
rm(list=ls())
require("ggplot2")

# Read Data
data = read.csv(file="MLR.csv", head=FALSE, sep =",")

#Prepare Data:
# number of dimensions and observations
n = dim(data)[1]
p = dim(data)[2]-1

# Define Design Matrix X and Response
X = as.matrix(data[,1:p])
Y = as.matrix(data[,p+1])
data = as.matrix(data)

# implement feature scaling
#X.scaled <- scale(X, center = TRUE, scale = TRUE)
#Y.scaled <- scale(Y, center = TRUE, scale = TRUE)
#data.scaled <- scale(data, center = TRUE, scale = TRUE)
X.scaled <- X
Y.scaled <- Y

##### Gradient descent (with fixed step length) #####

#----ALGORITHM INPUTS-----
# differentiable function f()
f = function (X.scaled, Y.scaled, beta ){
  return (sum((Y.scaled-X.scaled %*% beta)^2)/(2*n))
}
# fixed step length
L = ((norm(X.scaled,type="2"))^2)/ n
alpha = 1/L
# gradient function
gradient = function (X.scaled, Y.scaled, beta ){
  return ( 1/n * t(X.scaled) %*% (X.scaled %*% beta - Y.scaled))
}
#initial point beta_0
beta = matrix(0, nrow = p, ncol = 1)
#----BEGIN ALGORITHM-----
threshold = 5.053251e-20
max_i = 50
FunctionValues = rep(0,max_i)
FunctionValues[1] = f(X.scaled,Y.scaled,beta)
```

```
plotseq = seq(1:max_i)
best_beta = 0
i = 2
while (i <= max_i){
  g = gradient(X.scaled,Y.scaled,beta)
  beta = beta - alpha * g
  FunctionValues[i] = f(X.scaled,Y.scaled,beta)
  #Convergence Check
  if((norm(g, type ="2") < threshold)
     && (FunctionValues[i] - FunctionValues[i-1] < threshold) ){
      best_beta = beta
      print("break")
      break
  }
  i=i+1
  if(i <= max_i) {best_beta = beta}
}
best_beta

#----CREATE PLOT-----
P1d_Plot <- ggplot(data = as.data.frame(cbind(plotseq,FunctionValues)),
                   aes(x = plotseq, y = FunctionValues)) +
  geom_point() +
  xlab("Iteration Number") +
  ylab("f(beta)") +
  ggtitle(bquote(list("f(beta) versus number of iterations")))
print(P1d_Plot)
ggsave(P1d_Plot,filename=paste("P1d_Plot.png",sep=""),width = 4, height = 2)

#----CHECK MSE-----
true_beta = as.matrix(read.csv(file ="True_Beta.csv", head=FALSE ,sep =","))
MSE = sum((best_beta - true_beta) ^ 2) / n
MSE
```

Problem 2

```
# Problem 2

# Clear memory, include libraries
rm(list=ls())
require("ggplot2")

# Read Data
data = read.csv(file="MLR.csv", head=FALSE, sep =",")

#Prepare Data:
# number of dimensions and observations
n = dim(data)[1]
p = dim(data)[2]-1

# Define Design Matrix X and Response
X = as.matrix(data[,1:p])
Y = as.matrix(data[,p+1])
data = as.matrix(data)

# implement feature scaling
#X.scaled <- scale(X, center = TRUE, scale = TRUE)
#Y.scaled <- scale(Y, center = TRUE, scale = TRUE)
#data.scaled <- scale(data, center = TRUE, scale = TRUE)
X.scaled <- X
Y.scaled <- Y

##### Stochastic Gradient descent  #####

#----ALGORITHM INPUTS-----
#differentiable function g()
g = function (X.scaled, Y.scaled, beta){
  return (sum((Y.scaled-X.scaled %*% beta)^2)/(2*n))
}
#Lipschitz
L = ((norm(X.scaled,type="2"))^2)/ n
# gradient function
avg_gradient = function (X.scaled, Y.scaled, beta , b){
  minibatch = sample (1:n, b, replace = TRUE)
  minigrad = matrix(0, nrow = p, ncol = 1)
  for(k in minibatch){
    minigrad = minigrad - (X.scaled[k,]%*%beta - Y.scaled[k])*X.scaled[k,]
  }
  return (- 1/b * minigrad )
}

#----BEGIN ALGORITHM per b-----
for(bb in c(10,25,100,1)){
      #fixed step length
      alpha = bb/(n*L)

      #initial point beta_0
```

```
    beta = matrix(0, nrow = p, ncol = 1)
    #----BEGIN ALGORITHM-----
    threshold = 0.0000001
    max_i = 1000
    FunctionValues = rep(0,max_i)
    FunctionValues[1] = g(X.scaled,Y.scaled,beta)
    plotseq = seq(1:max_i)
    best_beta = 0
    i = 2
    while (i <= max_i){
      grad = avg_gradient(X.scaled,Y.scaled,beta, bb)
      beta = beta - alpha * grad
      FunctionValues[i] = g(X.scaled,Y.scaled,beta)
      #Convergence Check
      if((norm(grad, type ="2") < threshold)
         && (FunctionValues[i] - FunctionValues[i-1] < threshold) ){
        best_beta = beta
        print("break")
        break
      }
      i=i+1
      if(i <= max_i) {best_beta = beta}
    }

    #----CREATE PLOT-----
    Plot <- ggplot(data = as.data.frame(cbind(plotseq,FunctionValues)),
                   aes(x = plotseq, y = FunctionValues)) +
      geom_point() +
      xlab("Iteration Number") +
      ylab("f(beta)") +
      ggtitle(bquote(list("g(beta) versus number of iterations. b", .(bb))))
    print(Plot)
    ggsave(Plot,filename=paste("P2_Plot",bb,".png",sep=""),width = 4, height = 2)

    #----CHECK MSE-----
    true_beta = as.matrix(read.csv(file ="True_Beta.csv", head=FALSE ,sep =","))
    MSE = sum((best_beta - true_beta) ^ 2) / n
    print(MSE)

}
```

Problem 3

```
# Problem 3

# Clear memory, include libraries
rm(list=ls())
require("ggplot2")

# Read Data
data = read.csv(file="OPCA.csv", head=FALSE, sep =",")
totalrows = dim(data)[1]
d = dim(data)[2]
n = totalrows/d

# Create list of A[i]
A = lapply(1:n, function (x) as.matrix(data[((x-1)*d+1):( x*d),]))

# Get True Eigenvector
true_eigenvector = as.matrix(read.csv(file="True_eigvector.csv", head=FALSE, sep =","))

#Define Step Function
step = function (i, q){
  return (1/(100 + i*q))
}

#Visualization stuff
nlist = c(0:n)
part = c("a)","b)")
part[1]

# Oja's algorithm
Oja = function (q){
  # Initialization: w0
  w = matrix(1/sqrt(d), nrow=d, ncol = 1)
  #Simmilarity for Plot
  similarity = rep(0,n+1)
  similarity[1] = 1 - (t(w)%*%true_eigenvector)^2
  #Iterate through n
  for(i in 1:n){
    w = w + step(i,q)* A[[i]]%*%w
    w = w/norm(w, type = c("2"))
    similarity[i+1] = 1 - (t(w)%*%true_eigenvector)^2
  }
  #----CREATE PLOT-----
  p = part[q+1]
  Plot <- ggplot(data = as.data.frame(cbind(nlist,similarity)),
                 aes(x = nlist, y = similarity)) +
    geom_point(size = 0.7, alpha = 1/3) +
    xlab("Iteration Number") +
    ylab("dist(wi; v)") +
    ggtitle(bquote(list("Measure of similarity v/s iteration. Problem", .(p))))
  print(Plot)
  ggsave(Plot,filename=paste("P3_Plot",p,".png",sep=""),width = 5, height = 2.5)
```

```
  print(similarity[length(similarity)])
}

Oja(0)
Oja(1)
```