Implementing the MM algorithm for a linear model

Floris Holstege, Ruoying Dai,

## Data preparation

## first, we load the air quality data  
  
# load the air quality data  
load("Data/Airq\_numeric.Rdata")  
  
# set to dataframe  
dfAirQ <- data.frame(Airq)  
  
# select dependent variable of air quality  
Yair = dfAirQ$airq  
  
# select all other variables as independent variables  
Xair = dfAirQ[,-1]  
  
# scale the independent variables, and add an intercept to these  
XairScaled <- scale(Xair)  
XairIntercept <- cbind(intercept = 1, XairScaled)  
  
  
## second, we follow a similar procedure for the advertising data  
  
# load the advertising data  
load("Data/Advertising.Rdata")  
  
# set the advertising to dataframe  
dfAdv <- data.frame(Advertising)  
  
# select dependent variable of sales  
YAdv = dfAdv$Sales  
  
# select independent variable - for this example we use TV  
XAdv = data.frame(dfAdv$TV)  
  
# scale the independent variables, and add an intercept to these  
XAdvscaled <- scale(XAdv)  
XAdvintercept <- cbind(intercept = 1, XAdvscaled)  
colnames(XAdvintercept) <- c("intercept", "TV")

## Compare the linear model with MM to the standard lm() function

# set seed to ensure stability of results  
set.seed(0)  
  
# set e small  
e <- 0.0001  
  
# calculate the model using the MM algorithm  
modelMM <- calcModelMM(XairIntercept, Yair, e)  
  
# calc the model using the standard R library  
modelTest <- lm(airq ~ XairScaled,dfAirQ)  
  
# set notation of numbers  
options(scipen = 12)  
  
  
# Compare the sum of squared errors  
ResiMM <- modelMM$RSS  
ResiTest <- sum(resid(modelTest)^2)  
dfResi <- dfCompare(ResiMM, ResiTest, c("RSS with MM", "RS with lm()"))  
dfResi

## RSS with MM RS with lm()  
## 1 14071.6 14058.45

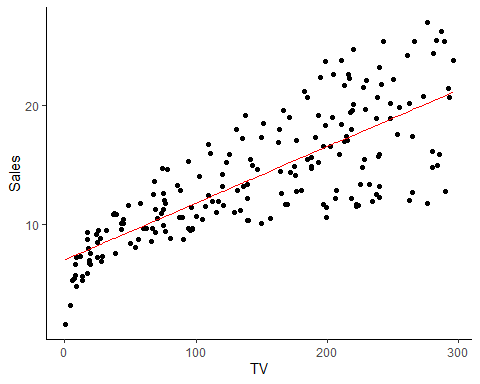
# Compare R^2   
RsquaredMM <- modelMM$Rsquared  
RsquaredStandard <- summary(modelTest)$r.squared  
dfRsquared <-dfCompare(RsquaredMM, RsquaredStandard, c("R^2 with MM", "R^2 with lm()"))  
dfRsquared

## R^2 with MM R^2 with lm()  
## 1 0.3823472 0.3829221

# Compare the beta's  
BetaMM <- modelMM$Beta  
BetaStandard <- as.double(modelTest$coefficients)  
dfBetaCompare <- dfCompare(BetaMM, BetaStandard, c("Beta with MM", "Beta with lm()"))  
dfBetaCompare

## Beta with MM Beta with lm()  
## intercept 104.692508 104.700000  
## vala 5.603005 4.090287  
## rain 3.410146 3.381518  
## coasyes -15.285871 -15.566666  
## dens -2.962245 -3.035229  
## medi 5.375968 6.930640

## A different example: Bivariate regression with advertising data



## Appendix: Helper functions that generated the model

# calcRSS  
# Calculates the residual squared errors for a multiple regression of the form Y = XBeta + e  
#   
# Parameters:   
# X: Dataframe of n x p (n = observations, p = independent variables)  
# Y: Dataframe of n x 1 dependent variables (n = observations)  
# Beta: p x 1 double with coefficients of said model  
#   
# Output:  
# ESquared: float, residual squared errors  
#   
  
calcRSS <- function(X, Y, Beta){  
   
 # set the dataframes to matrices  
 mX = as.matrix(X)  
 mY = as.matrix(Y)  
 mBeta = as.matrix(Beta)  
   
  
 # calculate the errors  
 mE <- mY - mX %\*% mBeta  
   
 # get errors squared  
 ESquared <- t(mE) %\*% mE  
   
 # return the residual sum of squared errors  
 return(ESquared)  
   
}  
  
# calcLargestEigen  
# Calculates the largest eigenvalue of an matrix of independent variables  
#   
# Parameters:   
# X: Dataframe of n x p (n = observations, p = independent variables)  
#   
# Output:  
# LargestEigenval: float, largest eigenvalue of said matrix  
#   
  
calcLargestEigen <- function(X){  
   
 # get matrix of squared X  
 mX = as.matrix(X)  
 XSquared <- t(mX) %\*% mX  
   
 # get the eigenvalues of X squared  
 EigenValXSquared <- eigen(XSquared)$values  
   
 # from these eigenvalues, get the largest one  
 LargestEigenVal <- max(EigenValXSquared, na.rm = TRUE)  
   
 return(LargestEigenVal)  
   
}  
  
#CalcBetaK  
# Calculates the kth beta in an MM algorithm  
#   
# Parameters:  
# prevBeta: double, k-1th beta  
# Lambda: double, largest eigenvalue of X (independent variables) squared  
# X: Dataframe of n x p (n = observations, p = independent variables)  
# Y: Dataframe of n x 1 dependent variables (n = observations)  
#   
# Output:   
# BetaK; matrix of new set of coefficients  
  
calcBetaK <- function(prevBeta,Lambda, X, Y){  
   
 # get matrix of squared X  
 mX = as.matrix(X)  
 XSquared <- t(mX) %\*% mX  
   
 # turn Y into matrix  
 mY = as.matrix(Y)  
   
 # calculate the Kth   
 BetaK = prevBeta - ((1/Lambda) \* XSquared %\*% prevBeta) + ((1/Lambda) \* t(mX) %\*% mY )  
   
 return(BetaK)  
   
   
   
}  
  
# CalcStepScore  
# Calculates the % improvement between the k-1th and kth set of beta's  
#   
# Parameters:  
# prevBeta: double, k-1th beta  
# currbeta: double, kth beta  
# X: Dataframe of n x p (n = observations, p = independent variables)  
# Y: Dataframe of n x 1 dependent variables (n = observations)  
#   
# Output:   
# StepScore; double, % improvement between the RSS of the two sets of beta's  
  
calcStepScore <- function(X,Y, prevBeta, currBeta){  
   
 # difference in RSS between previous and current set of beta's  
 diffRSS <- (calcRSS(X,Y,prevBeta) - calcRSS(X,Y,currBeta))  
   
 # divide difference with previous score to get % change  
 StepScore <- diffRSS /calcRSS(X,Y,prevBeta)  
   
 return(StepScore)  
   
}  
  
# getB0  
# Set the initial set of Beta's, randomly, between 0 and 1  
#   
# Parameters:  
# X: Dataframe of n x p (n = observations, p = independent variables)  
#   
# Output:   
# B0: double, set of Beta's between 0 and 1  
  
getB0 <- function(X){  
   
 # determine the number of independent variables, generate as many random beta's  
 nIndVar = ncol(X)  
 Beta0 <- runif(nIndVar, min=0, max=1)  
   
 return(Beta0)  
   
}  
  
# calcYest  
# Calculates the predicted Y, based on the X and est. Beta's of a linear model  
#   
# Parameters:  
# X: Dataframe of n x p (n = observations, p = independent variables)  
# BetaEst: Estimated Beta's, px1 vector,   
#  
# Output:  
# Yestdf: Dataframe, predicted Y  
#  
  
calcYest <- function(X,BetaEst){  
   
 # turn X and Beta's (est.) into matrix  
 mBetaEst <- as.matrix(BetaEst)  
 mX <- as.matrix(X)  
   
 # multiply X with Beta (est.) to get predicted Y  
 Yest <- mX %\*% mBetaEst  
   
 # turn into dataframe  
 Yestdf <- as.data.frame(Yest)  
 colnames(Yestdf) <- c("Yest")  
   
 return(Yestdf)  
   
}  
  
# calcRsquared  
# Calculates the r-squared  
#  
# Parameters:  
# Y: dataframe, the true dependent variable   
# Yest: dataframe, the predicted dependent variable  
#   
# Output:  
# Rsquared: double, the Rsquared for a linear model  
  
calcRsquared <- function(Y, Yest){  
   
 # standardize Y, and Yest (mean of 0)  
 standardY = Y - mean(Y)  
 standardYest = Yest - mean(Yest$Yest)  
  
 # turn into matrix to perform multiplication  
 mY <- as.matrix(standardY)  
 mYest <- as.matrix(standardYest)  
   
   
 # calculate Rsquared  
 numerator <- (t(mY) %\*% mYest)^2  
 denominator <- (t(mY) %\*% Y) %\*% (t(mYest) %\*% mYest)  
 Rsquared <- (numerator/denominator)  
   
 return(Rsquared)  
   
   
}  
  
  
# calcModelMM  
# Calculates a linear model, using the majorization in minimization (MM) algorithm  
#  
# Parameters:  
# X: Dataframe of n x p (n = observations, p = independent variables)  
# Y: Dataframe of n x 1 dependent variables (n = observations)  
#  
#  
# Output:  
# model: dataframe, with the following attributes  
# - Beta: dataframe, the calculated Beta's  
# - RSS: double, Sum of squared residuals  
# - Yest: dataframe, the predicted Y  
# - Rsquared: double, R^2 for the predicted Y  
# - Residuals: dataframe, Y - Yest.  
#  
  
  
calcModelMM <- function(X,Y,e){  
   
 # set the previous beta to initial, random beta's  
 prevBeta <- getB0(X)  
   
 # get largest eigenvalue for the square of independent variables  
 Lambda <- calcLargestEigen(X)  
   
 # set initial stepscore to 0, k to 1.   
 StepScore <- 0  
 k <- 1  
   
   
 # run while, either if k is equal to 1, or the improvement between k-1th and kth set of beta's is smaller than the parameter e  
 while (k == 1 | StepScore > e ){  
   
 # step to next k  
 k <- k + 1  
   
 # calculate beta's for this k  
 BetaK <- calcBetaK(prevBeta, Lambda, X,Y)  
  
 # new stepscore, % difference in RSS between new Beta's and previous beta's  
 StepScore <- calcStepScore(X,Y,prevBeta,BetaK)  
  
 # assign current beta's to prevBeta variable for next iteration  
 prevBeta <- BetaK  
   
 }  
   
   
 # calculate several attributes of the linear model, put in dataframes or doubles  
 BetaFinal <- data.frame(BetaK)  
 RSSBetaK <- calcRSS(X,Y, BetaK)  
 Yest <- calcYest(X, BetaFinal)  
 Rsquared <- calcRsquared(Y, Yest)  
 Resi <- data.frame(residuals = Y - Yest$Yest)  
  
 # add these attributes together as a list to make it easily accessible  
 results <- list(Beta = BetaFinal, RSS = RSSBetaK, Yest = Yest, Rsquared = Rsquared, Residuals = Resi)  
   
  
 return(results)  
   
}