Assignment 4

Vishal Arora

September 20, 2019

Problem Set 1

In this problem, we'll verify using R that SVD and Eigenvalues are related as worked out in the weekly module. Given a 3 2 matrix A

write code in R to compute X = AAT and Y = ATA. Then, compute the eigenvalues and eigenvectors of X and Y using the built-in commans in R.Then, compute the left-singular, singular values, and right-singular vectors of A using the svd command. Examine the two sets of singular vectors and show that they are indeed eigenvectors of X and Y. In addition, the two non-zero eigenvalues (the 3rd value will be very close to zero, if not zero) of both X and Y are the same and are squares of the non-zero singular values of A. Your code should compute all these vectors and scalars and store them in variables.Please add enough comments in your code to show me how to interpret your steps.

```
# creating the matrix and then computing  X= AAT and Y = ATA
A <- matrix(c(1, 2, 3, -1, 0, 4), nrow = 2, byrow = T)
A
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]   -1    0    4
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
X <- A %*% t(A)
X
```

```
##      [,1] [,2]
## [1,]   14   11
## [2,]   11   17
```

```
Y <- t(A) %*% A
Y
```

```
##      [,1] [,2] [,3]
## [1,]    2    2   -1
## [2,]    2    4    6
## [3,]   -1    6   25
```

```
#creating function for calculating the eigen value & vectors
calEigen_Val_Vec <- function(B){
    eigenB <- eigen(B)
   return (list(eigenB$values,eigenB$vectors))
}

# Passing the value to function for eigen values & vectors will returned by calling the function calEigen_Val_Vec
eigen_x <- calEigen_Val_Vec(X)
eigen_y <- calEigen_Val_Vec(Y)

#Eigen values & Vectors for X
eigen_x
```

```
## [[1]]
## [1] 26.601802  4.398198
##
## [[2]]
##           [,1]       [,2]
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
## [1,] 0.6576043 -0.7533635
## [2,] 0.7533635  0.6576043
```

```r
#Eigen values & vectors for Y
eigen_y
```

```
## [[1]]
## [1] 2.660180e+01 4.398198e+00 1.058982e-16
##
## [[2]]
##             [,1]       [,2]       [,3]
## [1,] -0.01856629 -0.6727903  0.7396003
## [2,]  0.25499937 -0.7184510 -0.6471502
## [3,]  0.96676296  0.1765824  0.1849001
```

```r
# Now we will compute Singular Vector Decomposition(SVD) of  matrix A
# A = uE(v)T
# Where u & v are orthogonal matrices  and  E is a daigonal matrices

svd_A <- svd(A)
svd_A
```

```
## $d
## [1] 5.157693 2.097188
##
## $u
##            [,1]       [,2]
## [1,] -0.6576043 -0.7533635
## [2,] -0.7533635  0.6576043
##
## $v
##             [,1]       [,2]
## [1,]  0.01856629 -0.6727903
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
## [2,] -0.25499937 -0.7184510
## [3,] -0.96676296  0.1765824
```

```
# Comparing the left singular value to eigen_x which came from finding the eigen vectors for A * T(A)

comp_lsv <-  list(eigen_x[[2]] ,svd_A$u)
comp_lsv
```

```
## [[1]]
##           [,1]       [,2]
## [1,] 0.6576043 -0.7533635
## [2,] 0.7533635  0.6576043
##
## [[2]]
##            [,1]       [,2]
## [1,] -0.6576043 -0.7533635
## [2,] -0.7533635  0.6576043
```

```
round(svd(A)$u,4) == round(eigen_x[[2]],4)
```

```
##       [,1] [,2]
## [1,] FALSE TRUE
## [2,] FALSE TRUE
```

```
urev <- round(svd(A)$u,4)
urev[,1] = -urev[,1]
round(urev,4) == round(eigen_x[[2]],4)
```

```
##      [,1] [,2]
## [1,] TRUE TRUE
## [2,] TRUE TRUE
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
# Comparing the right singular value to eigen_y  which in turn came from finding the eigen vectors for T(A)*A
comp_rsv <- list(eigen_y[[2]], svd_A$v)
comp_rsv
```

```
## [[1]]
##             [,1]       [,2]       [,3]
## [1,] -0.01856629 -0.6727903  0.7396003
## [2,]  0.25499937 -0.7184510 -0.6471502
## [3,]  0.96676296  0.1765824  0.1849001
##
## [[2]]
##             [,1]       [,2]
## [1,]  0.01856629 -0.6727903
## [2,] -0.25499937 -0.7184510
## [3,] -0.96676296  0.1765824
```

```
round(svd(A)$v,4) == round(eigen_y[[2]][,-3],4)
```

```
##       [,1] [,2]
## [1,] FALSE TRUE
## [2,] FALSE TRUE
## [3,] FALSE TRUE
```

```
vrev <- round(svd(A)$v,4)
vrev[,1] = -vrev[,1]
round(vrev,4) == round(eigen_y[[2]][,-3],4)
```

```
##      [,1] [,2]
## [1,] TRUE TRUE
## [2,] TRUE TRUE
## [3,] TRUE TRUE
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```r
# Comparing the sq of diagnoal matrix for svd(A) with the eigenvalues of X & Y .
round(eigen_x[[1]],4) == round(svd_A$d^2,4)
```

```
## [1] TRUE TRUE
```

```r
round(eigen_y[[1]][1:2],4) == round(svd_A$d^2,4)
```

```
## [1] TRUE TRUE
```

Problem Set 2

Using the procedure outlined in section 1 of the weekly handout, write a function to compute the inverse of a well-conditioned full-rank square matrix using co-factors. In order to compute the co-factors, you may use built-in commands to compute the determinant.Your function should have the following signature: B = myinverse(A)

where A is a matrix and B is its inverse and AB = I. The o-diagonal elements of I should be close to zero, if not zero. Likewise, the diagonal elements should be close to 1, if not 1. Small numerical precision errors are acceptable but the function myinverse should be correct and must use co-factors and determinant of A to compute the inverse.

```r
# Function myinverse takes a matrix and first checkes whether the matrix fullfills conditions for inversing the matrix i.e. the matrix should be a square matrix and full- rank matrix. Then if the matrix fullfills the conditions it returns the inverse of the matrix.
myinverse <- function(mat) {
  # checking if the inbound matrix is square and full-rank
  if (nrow(mat) == ncol(mat) & qr(A)$rank== dim(A)[1]) {
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```r
    sq_fr <- TRUE
  } else {sq_fr <- FALSE}

  if (sq_fr == TRUE) {
    size <- nrow(mat)
    #creating an empty co-factor matrix
    C <- matrix(nrow = size, ncol = size)
    for (i in 1:size) {
      for (j in 1:size) {
        # M is mat with row i & column j excluded
        M <- mat[-i, -j]
        # Co-factors matrix populated
        C[i, j] <- (-1)^(i + j) * det(M)
      }
    }
    # returning the inverse of matrix
    B <- t(C) / det(mat)
  } else {B <- "As the matrix is not a square matrix & full rank hence matrix is not invertible."}
  B
}

# Showing that myinverse checks whether a incoming matrix fullfills both conditions (i.e. full-rank & square matr
ix)
A <- matrix(c(1, 2, 3, -1, 0, 4), nrow = 2, byrow = T)
B <- myinverse(A)
B
```

```
## [1] "As the matrix is not a square matrix & full rank hence matrix is not invertible."
```

```r
# creating a new matrix which is square & full-rank matrix.
A <- matrix(c(1,3,0,4,1,5,2,0,7), nrow = 3, byrow = TRUE)
B <- myinverse(A)
B
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
##               [,1]       [,2]       [,3]
## [1,] -0.14893617  0.4468085 -0.3191489
## [2,]  0.38297872 -0.1489362  0.1063830
## [3,]  0.04255319 -0.1276596  0.2340426
```

```
# As A␂B = I , so we will now prove that our method works fine by multiplying A with the inverse of A i.e. B whi
ch should give us an Identity Matrix.
round(B %*% A,14)
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
```

```
# Hence proved that our method is working
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD