Data605-Week4-HomeWork4-kamath

Vinayak Kamath

09/20/2020

Problem set 1

1. 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 $\mathbf{A}$

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ -1 & 0 & 4 \end{bmatrix} \tag{1}$$

write code in R to compute $\mathbf{X} = \mathbf{A}\mathbf{A}^\mathbf{T}$ and $\mathbf{Y} = \mathbf{A}^\mathbf{T}\mathbf{A}$. Then, compute the eigenvalues and eigenvectors of $\mathbf{X}$ and $\mathbf{Y}$ using the built-in commans in R.

Then, compute the left-singular, singular values, and right-singular vectors of $\mathbf{A}$ using the *svd* command. Examine the two sets of singular vectors and show that they are indeed eigenvectors of $\mathbf{X}$ and $\mathbf{Y}$. In addition, the two non-zero eigenvalues (the 3rd value will be very close to zero, if not zero) of both $\mathbf{X}$ and $\mathbf{Y}$ are the same and are squares of the non-zero singular values of $\mathbf{A}$.

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.

Figure 1: .

```r
#defining the matrix:
A <- matrix(c(1, -1, 2, 0, 3, 4), 2, 3)
A
```

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

```r
# Compute X and Y using built-in commands
X <- A%*%t(A)
X
```

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

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

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

```r
# compute eigenvalues and eigenvectors
eigen_X <- eigen(X)
eigen_X
```

```
## eigen() decomposition
## $values
## [1] 26.601802  4.398198
##
## $vectors
##           [,1]       [,2]
## [1,] 0.6576043 -0.7533635
## [2,] 0.7533635  0.6576043
```

```r
eigen_Y <- eigen(Y)
eigen_Y
```

```
## eigen() decomposition
## $values
## [1] 2.660180e+01 4.398198e+00 1.058982e-16
##
## $vectors
##             [,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
# compute the left-singular, singular values, and right-singular vectors of A using the 'svd()' command
# v = right, u = left
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
## [2,] -0.25499937 -0.7184510
## [3,] -0.96676296  0.1765824
```

```r
# Compare the left singular vector u to the eigenvectors of x
eigen_X$vectors
```

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

```r
svd_A$u
```

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

```r
round(abs(eigen_X$vectors)) == round(abs(svd_A$u))
```

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

==> We can see that the vectors are same; except that one of the vectors is multiplied by a scalar of -1. Thus the left singular vectors are the eigenvectors of X.

```r
# Compare the right singular vector v to the eigenvectors of x
eigen_Y$vectors
```

```
##             [,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
svd_A$v
```

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

```r
round(abs(eigen_Y$vectors[1:3, 1:2])) == round(abs(svd_A$v))  #Comparing frist two column data only
```

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

==> We can see that the vectors are same; Thus the right singular vectors are the eigenvectors of Y.

```r
# all.equal -- compare R objects x and y testing 'near equality'
all.equal(eigen_X$values, (svd_A$d)^2)
```

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

```r
all.equal(eigen_Y$values[1:2], (svd_A$d)^2)   #Comparing frist two data only
```

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

Problem set 2

2. 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 $\mathbf{A}$ is a matrix and $\mathbf{B}$ is its inverse and $\mathbf{A}\times\mathbf{B} = \mathbf{I}$. The off-diagonal elements of $\mathbf{I}$ 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 $\mathbf{A}$ to compute the inverse.

Figure 2: .

```
#reference : https://www.mathsisfun.com/algebra/matrix-inverse-minors-cofactors-adjugate.html
myinverse <- function(A) {

  #Check to see if the matrix is invertible:
  if (det(A) == 0) {
    stop('This matrix is non-invertible. Try another!')
  }

  inverse_A <- diag(0, nrow=nrow(A), ncol=ncol(A))


  #Step 1: Matrix of Minors:
    #For each element of the matrix:
    #  1. ignore the values on the current row and column
    #  2. calculate the determinant of the remaining values
    #Put those determinants into a matrix (the "Matrix of Minors")

    for(i in 1:nrow(A))
    {
      for(j in 1:ncol(A))
      {
        inverse_A[i,j] <- det(A[-i,-j]) # fill the rows and columns
      }
    }


  #Step 2: Matrix of Cofactors:
    #Temperory copy of inverse_A
    temp_A <- inverse_A

    #Starting  the sign for frist row with +1
    sign_multiple_row = +1
    for(i in 1:nrow(A))
    {
      #setting sign for first column in row
```

```
      sign_multiple_col = sign_multiple_row
      for(j in 1:ncol(A))
      {
        inverse_A[i,j] <- sign_multiple_col*(temp_A[i,j]) # fill the rows and columns
        #flipping sign for next column in row
        sign_multiple_col = sign_multiple_col * -1
      }
      #flipping the sign for next row
      sign_multiple_row = sign_multiple_row * -1
    }


  #Step 3: Adjugate (also called Adjoint or Transpose):
    inverse_A = t(inverse_A)


  #Step 4: Multiply by 1/Determinant:
    inverse_A = inverse_A * (1/det(A))


  return(inverse_A)

}
```

==> **We can test it as below:**

```
# Try it out
A = matrix(c(3, 2, 0, 0,0, 1, 2, -2, 1), 3, 3)
A
```

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

```
B <- myinverse(A)
B
```

```
##      [,1] [,2] [,3]
## [1,]  0.2  0.2    0
## [2,] -0.2  0.3    1
## [3,]  0.2 -0.3    0
```

```
# We can crosscheck the answer using solve():
s_A <- solve(A)
round(s_A,2) == round(B,2)
```

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