Week 4 - Linear Transformations & Representations

Christian Thieme

9/17/2020

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.

```
A <- matrix(c(1,-1,2,0,3,4), nrow = 2)

# A is a 2 x 3
# Transpose will be a 3 x 2
#-----------------------------------------
X <- A %*% t(A)  ## A * Transpose(A)

#calculating the eigenvalues of X
x_eigen_values <- eigen(X)$values

#Calculating the eigenvectors of X
x_eigen_vectors <- eigen(X)$vectors

#-----------------------------------------
Y <-  t(A) %*% A ## Transpose(A) * A

#calculating the eigenvalues of Y
y_eigen_values <- eigen(Y)$values

#Calculating the eigenvectors of Y
y_eigen_vectors <- eigen(Y)$vectors

#-----------------------------------------

## SVD of A
svd_A <- svd(A)

left_singular_vectors <- svd_A$u #This will be the equivalent of A * A^T's eigenvectors

right_singular_vectors <- svd_A$v #This will be the equivalent of A^T * A's eigenvectors (only the columns that h
ave a non-zero singular value)

singular_values <- svd_A$d # This is the equivalent of the square roots of the eigenvalues of A^T * A and A * A^T
```

Now we will examine if the two sets of singular matrices are the eigenvectors of X and Y. First, let's investigate the eigenvectors of X.

```
x_eigen_vectors
```

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

Next, we'll look at the left singular values of A:

```
left_singular_vectors
```

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

You'll notice that the eigenvectors from X and the left singular vectors are the same, but with some sign changes. This Stackoverflow explains in detail why that can happen, but in summary these sign changes can happen due to random changes in the signs of the eigen vecotrs from using the eigen() function, since the eigenvectors can be scaled by -1.

Now let's look at the eigenvectors of Y:

```
y_eigen_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
```

Next, we'll look at the right singular values of A:

```
right_singular_vectors
```

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

First, you'll notice that the first two vectors are both the same, but with some sign changes as discussed above. However, there is also a third vector when looking at Y's eigenvectors. This is jumping ahead a little bit, but there are only two non-zero singular values which relate to the first to vectors, so we ignore the third vector. I'll remove that vector and you'll see that it is identical (except for sign changes) to the right singular vectors.

```
y_eigen_vectors[,1:2]
```

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

Finally, let's take a look at the singular values of A. If we've done this right, the singular values of A should be the square roots of the eigenvalues of X and Y, so if we square the singular values, they should be equivalent to the eigenvalues. The singular values of A are:

```
singular_values
```

```
## [1] 5.157693 2.097188
```

The eigenvalues are (both X and Y will have the same non-zero eigenvalues):

```
x_eigen_values
```

```
## [1] 26.601802  4.398198
```

Now lets see if we get these numbers when we square the singular values of A:

```
singular_values ** 2
```

```
## [1] 26.601802  4.398198
```

Its a perfect match!

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.

Please submit PS1 and PS2 in an R-markdown document with your first initial and last name.

The below function `myinverse` will calculate the inverse for any size of square matrix that is passed to it.

```
myinverse <- function(A) {
  if (dim(A)[1] != dim(A)[2] | det(A) == 0 ) {
    print("Please enter a square matrix that is invertible.")
  } else {

    list_for_matrix = c()
    determinant_list = c()
    indexes <- seq(from = 1, to = dim(A)[1])

    inverse_matrix_row_counter <- 1
    inverse_matrix_col_counter <- 1

    cur_row <- 1
    cur_column <- 1

    moving_row <- 1
    moving_col <- 1

    while (cur_column <= dim(A)[1]) {

      if (cur_row <= dim(A)[1]) {

          for (num in A) {

            if (moving_row == cur_row | moving_col == cur_column) {
              if ( moving_row < dim(A)[1] ) {
                moving_row <- moving_row + 1
              } else {
                    moving_row <- 1
                    if (moving_col < dim(A)[1]) {
                      moving_col <- moving_col + 1
                    }
              }
            } else {
                number <- A[moving_row, moving_col]
                list_for_matrix <- c(list_for_matrix, number)
              if ( moving_row < dim(A)[1] ) {
                moving_row <- moving_row + 1
              } else {
                    moving_row <- 1
                    if (moving_col < dim(A)[1]) {
                      moving_col <- moving_col + 1
                    }
              }
            }
          new_matrix <- matrix(list_for_matrix, nrow = (dim(A)[1])-1)
          list_for_matrix <- c()
          moving_row <- 1
          moving_col <- 1
          determinant <- det(new_matrix)
          if ((inverse_matrix_row_counter + inverse_matrix_col_counter)%%2 == 0){
            determinant_list <- c(determinant_list, determinant)
          } else {
            determinant_list <- c(determinant_list, -1 * determinant)
          }
          if (inverse_matrix_row_counter < dim(A)[1]) {
            inverse_matrix_row_counter <- inverse_matrix_row_counter + 1
          } else {
            inverse_matrix_row_counter <- 1
            inverse_matrix_col_counter <- inverse_matrix_col_counter + 1
          }
          cur_row <- cur_row + 1
      } else {
        cur_row <- 1
        cur_column <- cur_column + 1
      }
    }

    cofactor_matrix <- matrix(determinant_list, nrow = dim(A)[1])
    transposed_cofactor_matrix <- t(cofactor_matrix)
    determinant_of_a <- det(A)

    (1/determinant_of_a) * transposed_cofactor_matrix
  }
}
```

Now let's give our function a run on a 3x3 matrix to see if we can correctly calculate the inverse as well as get all the signs right as well.

```
A <- matrix(c(-1,2,3,-2,1,4,2,1,5), nrow = 3)
B <- myinverse(A)
B
```

```
##           [,1]       [,2]       [,3]
## [1,]  0.04347826  0.78260870 -0.1739130
## [2,] -0.30434783 -0.47826087  0.2173913
## [3,]  0.21739130 -0.08695652  0.1304348
```

To see if we've calculated the inverse correctly, we can multiply A * B and we should get the identity matrix:

```
round(A %*% B,0)
```

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

VOILA! Perfect. Now let's try our hand on 4x4 matrix:

```
AA <- matrix(c(4,0,0,1,0,0,1,0,0,2,2,0,0,0,0,1), nrow = 4)
BB <- myinverse(AA)
BB
```

```
##      [,1] [,2] [,3] [,4]
## [1,]  0.25  0.0    0    0
## [2,]  0.00 -1.0    1    0
## [3,]  0.00  0.5    0    0
## [4,] -0.25  0.0    0    1
```

Again, to see if we've calculated the inverse correctly, if we multiply AA * BB, we should get the identity matrix as well.

```
round(AA %*% BB, 0)
```

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

Our function is working perfectly!