Data 605 Assignment 2

Stephanie Roark

2/10/2019

1. Problem Set 1

(1) Show that A^TA != AA^T in general. (Proof and demonstration.)

If A is an m $\times$ n matrix and A^T is its transpose, then the result of matrix multiplication with these two matrices gives two square matrices: AA^T is m $\times$ m and A^TA is n $\times$ n. Since an mxm matrix cannot be equavilent to an nxn matrix if m!=n, A^TA != AA^T.

```
A <- matrix(c(1,2,3,4,5,6,7,8,9),nrow = 3)
A
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```
A_T <- t(A)
A_T
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
```

```
identical(A, A_T)
```

```
## [1] FALSE
```

```
A %*% A_T == A_T %*% A
```

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

(2) For a special type of square matrix A, we get $A^TA = AA^T$. Under what conditions could this be true?

The Identity matrix I is an example of such a matrix.

```
AI <- matrix(c(1,0,0,0,1,0,0,0,1),nrow = 3)
AI
```

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

```
AI_T <- t(AI)
AI_T
```

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

```r
identical(AI, AI_T)
```

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

```r
AI %*% AI_T == AI_T %*% AI
```

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

This will also be true whenever $\mathbf{A} = \mathbf{A}^{\mathbf{T}}$.

```r
A2 <- matrix(c(1,3,4,3,2,3,4,3,1), nrow = 3)
A2
```

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

```r
A2_T <- t(A2)
A2_T
```

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

```r
identical(A2, A2_T)
```

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

```r
A2 %*% A2_T == A2_T %*% A2
```

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

2. Problem set 2

Matrix factorization is a very important problem. There are supercomputers built just to do matrix factorizations. Every second you are on an airplane, matrices are being factorized. Radars that track flights use a technique called Kalman filtering. At the heart of Kalman Filtering is a Matrix Factorization operation. Kalman Filters are solving linear systems of equations when they track your flight using radars.

Write an R function to factorize a square matrix A into LU or LDU, whichever you prefer.

Factorize A into a product of two matrices: A = LU where U is the matrix that we get at the end of the elimination procedure, it is the Upper Triangular matrix. L is a Lower Triangular matrix and you'll see that the entries of L are the multipliers that we applied to subtract one row from the other.

LU Decomposition

```r
#define a matrix a
a <- matrix(sample (25), 5, 5)
a
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    3   20    4   12    6
## [2,]   25   17    2   16   11
## [3,]    9   21    1   22   18
## [4,]   13    5   24    8    7
## [5,]   14   15   19   10   23
```

```r
#define Gaussian Elimination function to compute matrix with row reductions
Gauss_Elim <- function(a) {
    n <- nrow(a)
    #create an augmented matrix with a and the identity
    a <- cbind(a,diag(n))

    #multiply a row by a constant
    for (i in 1 : n) {
        c <- diag(n)
        c[i, i] <- (1 / a[i, i])
        a <- c %*% a

        #if at the last row, stop
        if (i == n) {
            break ()
        }
        #add a multiple of a row to a different row
        for (j in (i + 1) : n) {
            r <- diag(n)
            r[j, i] <- (- a[j, i])
            a <- r %*% a
        }
    }
    return(a)
}
#call Gauss Elimination function on matrix a
Gauss_Elim(a)
```

```
##      [,1]     [,2]      [,3]       [,4]       [,5]        [,6]
## [1,]    1 6.666667 1.3333333  4.0000000  2.0000000  0.33333333
## [2,]    0 1.000000 0.2093541  0.5612472  0.2605791  0.05567929
## [3,]    0 0.000000 1.0000000 -2.7824038 -3.5844462  0.29222310
## [4,]    0 0.000000 0.0000000  1.0000000  1.2870255 -0.09904286
## [5,]    0 0.000000 0.0000000  0.0000000  1.0000000 -0.04341091
##              [,7]        [,8]        [,9]      [,10]
## [1,]  0.000000000  0.00000000  0.00000000 0.00000000
## [2,] -0.006681514  0.00000000  0.00000000 0.00000000
## [3,]  0.091908877 -0.35271013  0.00000000 0.00000000
## [4,] -0.040169695  0.12334119  0.01471540 0.00000000
## [5,] -0.015083777  0.02269085 -0.03621794 0.05528162
```

```r
#use Gauss_Elim() to compute Upper Triangular Matrix and Lower Triangular Matrix
```

```r
#define LU Decomposition function
LU_decomp <- function(a) {
    n <- nrow (a)
    #compute the U matrix (Upper Triangular)
    g <- Gauss_Elim(a)
    #compute the L matrix (Lower Triangular) with multipliers located below the diagonal
    h <- Gauss_Elim(g[, n + 1 : n])
    return(list(L = h[, n + 1 : n], U = g[, 1 : n]))
}

print(LU <- LU_decomp(a))
```

```
## $L
##      [,1]       [,2]      [,3]     [,4]     [,5]
## [1,]    3    0.00000  0.000000  0.00000  0.00000
## [2,]   25 -149.66667  0.000000  0.00000  0.00000
## [3,]    9  -39.00000 -2.835189  0.00000  0.00000
## [4,]   13  -81.66667 23.763920 67.95601  0.00000
## [5,]   14  -78.33333 16.732739 44.52160 18.08919
##
## $U
##      [,1]     [,2]      [,3]       [,4]       [,5]
## [1,]    1 6.666667 1.3333333  4.0000000  2.0000000
## [2,]    0 1.000000 0.2093541  0.5612472  0.2605791
## [3,]    0 0.000000 1.0000000 -2.7824038 -3.5844462
## [4,]    0 0.000000 0.0000000  1.0000000  1.2870255
## [5,]    0 0.000000 0.0000000  0.0000000  1.0000000
```

```r
#show that the LU decomp is equal to matrix a
LU $ L %*% LU $ U
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    3   20    4   12    6
## [2,]   25   17    2   16   11
## [3,]    9   21    1   22   18
## [4,]   13    5   24    8    7
## [5,]   14   15   19   10   23
```