DATA 605 Week 2 Homework

Ilya Kats

September 10, 2017

Problem Set 1

Part (1)

Show that $A^TA \neq AA^T$ in general. (Proof and demonstration.)

Solution: Consider matrix $A_{m\times n}$ where $m \neq n$. Then A^T will be the size of $n \times m$. Then AA^T will be a square matrix of size $m \times m$ and A^TA will be a square matrix of size $n \times n$. Since $m \neq n$, these two matrices will not be equal. This will hold for any non-square matrix, but may be true for some square matrices.

Demonstration:

```r
A <- matrix(c(1,2,3,4,5,6), nrow=2, byrow=TRUE)
At <- t(A)

# Consider matrix A
A
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
```

```r
# and its transpose
At
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```

```r
# Multiplying A by its transpose we get 2x2
A %*% At
```

```
##      [,1] [,2]
## [1,]   14   32
## [2,]   32   77
```

```r
# Multiplying transposed A by A we get 3x3
At %*% A
```

```
##      [,1] [,2] [,3]
## [1,]   17   22   27
## [2,]   22   29   36
## [3,]   27   36   45
```

Part (2)

For a special type of square matrix A, we get $A^TA = AA^T$. Under what conditions could this be true? (Hint: The Identity matrix I is an example of such a matrix).

Solution: This condition is true if $A = A^T$. Transposing a matrix switches columns into rows or simply flips it along the diagonal. If a matrix is symmetrical along the diagonal, $A^T = A$ and the condition holds.

Demonstration:

```
A <- matrix(c(1,2,0,2,1,0,0,0,5), nrow=3, byrow=TRUE)

# Consider matrix A
A
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    0
## [2,]    2    1    0
## [3,]    0    0    5
```

```
# its transpose is the same matrix
t(A)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    0
## [2,]    2    1    0
## [3,]    0    0    5
```

```
A == t(A)
```

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

```
# so
(A %*% t(A)) == (t(A) %*% A)
```

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

Problem Set 2

Write an R function to factorize a square matrix A into LU.

Solution

The code is based on the algorithm presented by *James Sousa* for *Mathispowerf4u* (https://www.youtube.com/watch?v=UlWcofkUDDU).

```
factorizeLU <- function(A) {
  # Check that A is square
  if (dim(A)[1]!=dim(A)[2]) {
    return(NA)
  }

  U <- A
  n <- dim(A)[1]
  L <- diag(n)
  # If dimension is 1, then U=A and L=[1]
  if (n==1) {
    return(list(L,U))
  }

  # Loop through the lower triangle (by rows and columns)
  # Determine multiplier for each position and add it to L
  for(i in 2:n) {
    for(j in 1:(i-1)) {
      multiplier <- -U[i,j] / U[j,j]
      U[i, ] <- multiplier * U[j, ] + U[i, ]
      L[i,j] <- -multiplier
    }
  }
  return(list(L,U))
}
```

Demonstration

Example 1

```
A <- matrix(c(1,4,-3,-2,8,5,3,4,7), nrow=3, byrow=TRUE)
LU <- factorizeLU(A)
L<-LU[[1]]
U<-LU[[2]]

A
```

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

```
L
```

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

```
U
```

```
##      [,1] [,2] [,3]
## [1,]    1    4 -3.0
## [2,]    0   16 -1.0
## [3,]    0    0 15.5
```

```
A == L %*% U
```

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

Example 2

```r
A <- matrix(c(2,1,6,8), nrow=2, byrow=TRUE)
LU <- factorizeLU(A)
L<-LU[[1]]
U<-LU[[2]]

A
```

```
##      [,1] [,2]
## [1,]    2    1
## [2,]    6    8
```

```r
L
```

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

```r
U
```

```
##      [,1] [,2]
## [1,]    2    1
## [2,]    0    5
```

```r
A == L %*% U
```

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

Notes

The function fails if it encounters any zeros on the diagonal. This will require pivoting and per stated problem, we do not need to account for permutating rows of A. Researching the LU decomposition, I've noticed that some approaches store L and U in the same matrix and in fact overwrite A with combined LU as the algorithm moves forward. There may be some interesting posibilities here.