CUNY DATA 605 (CompMath) - Assign1

Raphael Nash

2/1/2017

Problem Set 1

(1) Calculate the dont product u.v where u = [0.5;0.5] and v = [3;-4]

```
u <- matrix( c(.5,.5) )
v <- matrix ( c( 3,-4))

t(u) %*%(v)
```

```
##      [,1]
## [1,] -0.5
```

** What are the lengths of u and v? Please note that the mathematical notion of the length of a vector is not the same as a computer science deffnition.**

```
length_u <- sqrt(t(u) %*% u )
length_u
```

```
##           [,1]
## [1,] 0.7071068
```

```
lenght_v <- sqrt(t(v) %*% v )
lenght_v
```

```
##      [,1]
## [1,]    5
```

(2) What is the linear combination 3u-2v?

```
3*u-2*v
```

```
##      [,1]
## [1,] -4.5
## [2,]  9.5
```

Problem Set 2

Set up a system of equations with 3 variables and 3 constraints and solve for x. Please write a function in R that will take two variables (matrix A & constraint vector b) and solve using elimination. Your function should produce the right answer for the system of equations for any 3-variable, 3-equation system. You don't have to worry about degenerate cases and can safely assume that the function will only be tested with a system of equations that has a solution. Please note that you do have to worry about zero pivots, though. Please note that you should not use the built-in function solve to solve this system or use matrix inverses. The approach that you should employ is to construct an Upper Triangular Matrix and then back-substitute to get the solution. Alternatively, you can augment the matrix A with vector b and jointly apply the Gauss Jordan elimination procedure.

Test with:

$$\begin{pmatrix} 1 & 1 & 3 \\ 2 & -1 & 5 \\ -1 & -2 & 4 \end{pmatrix} \begin{pmatrix} X_1 \\ X_2 \\ X_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 6 \end{pmatrix}$$

Should have the soultion of x= [-1.55, -0.32, 0.95]

```
linearEqSolve <- function ( A,C ) {

  #---------------Create upper Triangle Matrix
  aug <- cbind(A,C)
  for ( cur_row in 2: nrow(aug)) {
    for( cur_col in 1: (cur_row -1 ))  {
      denom <- aug[cur_col,cur_col]
      num <- aug [cur_row ,cur_col]
      mult <- num / denom
      aug[cur_row,] <-  aug[cur_row,] - (aug[cur_col,] * mult )

    }

  }
  u <- aug[,1:ncol(aug)-1]
  v <- aug[,ncol(aug)]

  print("Augmented Upper Triangle Matrix:")
  print(aug)


  x <- matrix( , nrow =   nrow(u), ncol = 1 )
  cur_row <-0
  cur_col <-0

  #---------Back Solve
  for (cur_row in ncol(u) : 1) {
    cur_val <-  v[cur_row]
    for( cur_col  in   ncol(u) : cur_row ) {
      offset <- cur_col- cur_row
      if ( offset == 0 ) {
        cur_val <- cur_val  / u[cur_row, cur_row]
        x[cur_row] <- cur_val
      }  else {
        cur_val <- cur_val - u[cur_row, cur_row+offset] * x[cur_row+offset]
      }
    }
  }

x
}
```

```
A <- matrix (c (1,1,3,2,-1,5,-1,-2,4), nrow = 3, ncol=3 , byrow = TRUE)
C <- matrix (c (1,2,6), nrow = 3, ncol=1 , byrow = TRUE)
x<- linearEqSolve(A,C)
```

```
## [1] "Augmented Upper Triangle Matrix:"
##      [,1] [,2]      [,3] [,4]
## [1,]    1    1  3.000000    1
## [2,]    0   -3 -1.000000    0
```

```
## [3,]    0    0  7.333333    7
```

```r
x
```

```
##            [,1]
## [1,] -1.5454545
## [2,] -0.3181818
## [3,]  0.9545455
```