605-HW01-GaussJordan

Michael Y

September 1, 2019

Contents

Setup

```r
knitr::opts_chunk$set(echo = TRUE)
directory = "C:/Users/Michael/Dropbox/priv/CUNY/MSDS/201909-Fall/DATA605_Larry/20190901_Week01/"
knitr::opts_knit$set(root.dir = directory)

### Make the output wide enough
options(scipen = 999, digits=6, width=150)

### Load some libraries
library(tidyr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(kableExtra)
```

```
##
## Attaching package: 'kableExtra'

## The following object is masked from 'package:dplyr':
##
##     group_rows
```

```r
library(pracma)
```

Function to write Matrix, courtesy of Vinayak Patel :

```r
writeMatrix <- function(x) {
  begin <- "\\begin{bmatrix}"
  end   <- "\\end{bmatrix}"
  X     <-    apply(x, 1, function(x) {
      paste(
          paste(x, collapse = "&"),
          "\\\\"
      )
    }
  )
  paste(c(begin, X, end),
        collapse = "")
  }
```

Part 1 - DotProduct, Norm, Cosine

You can think of vectors representing many dimensions of related information.

For instance, Netflix might store all the ratings a user gives to movies in a vector.

This is clearly a vector of very large dimensions (in the millions) and very sparse as the user might have rated only a few movies.

Similarly, Amazon might store the items purchased by a user in a vector, with each slot or dimension representing a unique product and the value of the slot, the number of such items the user bought.

One task that is frequently done in these settings is to find similarities between users.

And, we can use dot-product between vectors to do just that.

As you know, the dot-product is proportional to the length of two vectors and to the angle between them.

In fact, the dot-product between two vectors, normalized by their lengths is called as the cosine distance and is frequently used in recommendation engines.

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

```
u = c(0.5, 0.5)
v = c(3  , -4 )
dotproduct_uv = dot(u,v)        # dot is in library pracma
dotproduct_uv
```

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

$$u \cdot v = -0.5$$

(2) 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 definition.

```
len_u = sqrt(dot(u,u))
len_u
```

```
## [1] 0.707107
```

```
len_v = sqrt(dot(v,v))
len_v
```

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

$$\|u\| = 0.707107$$

$$\|v\| = 5$$

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

```
result = 3*u - 2*v
as.matrix(result)
```

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

$$3u - 2v = 3\begin{bmatrix}0.5\\0.5\end{bmatrix} - 2\begin{bmatrix}3\\-4\end{bmatrix} = \begin{bmatrix}-4.5\\9.5\end{bmatrix}$$

(4) What is the angle between u and v?

```
rad2deg <-function(rad) rad/pi * 180

deg2rad <-function(deg) pi*deg / 180

cos_theta = dotproduct_uv / (len_u * len_v)
cos_theta
```

```
## [1] -0.141421
```

```
theta_rads = acos(cos_theta)
theta_rads
```

```
## [1] 1.71269
```

```
theta_degs = rad2deg(theta_rads)
theta_degs
```

```
## [1] 98.1301
```

$$cos\left(\theta_{uv}\right) = \frac{u \cdot v}{\|u\|\,\|v\|} = \frac{-0.5}{0.707107 * 5} = -0.141421$$

$$cos^{-1}\left(-0.141421\right) = 1.712693\ (in\ radians) = 98.130102^{\circ}\ (in\ degrees)$$

Part 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.

Please test it with the system below and it should produce a solution

x = [-1.55 ; -0.32 ; 0.95]

```
### Define the matrices specified.
### For equation display, also define the "x" column vector

A = matrix(c( 1, 1, 3,
               2,-1, 5,
              -1,-2, 4),
            nrow=3,byrow = T)
x <- matrix(c("x1","x2","x3"),
            nrow=3,byrow = T)
b <- matrix(c(1,2,6),
           nrow=3,byrow = T)
```

Display the equation given in the assignment:

$$\begin{bmatrix} 1 & 1 & 3 \\ 2 & -1 & 5 \\ -1 & -2 & 4 \end{bmatrix} \begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 6 \end{bmatrix}$$

Function to perform Gauss-Jordan elimination

If successful, returns the identity matrix followed by the result column

```
MYGaussJordan <- function(A, b){
    # A: coefficient matrix of coefficients
    # b: right-hand side

    rowCount <- nrow(A)
    colCount <- ncol(A)
    epsilon=1e-7                     # checking for zero pivot
    determinant <- 1                 # keep track of the determinant that has been divided out
    pivotList <- rep(0, rowCount)    # e.g., (0 0 0)

    b <- as.matrix(b)                # ensure that b is a matrix (rather than a vector)
    Combined <- cbind(A, b)          # append the columns of b onto A

    i <- 1
```

```
j <- 1

while (i <= rowCount && j <= colCount){
    while (j <= colCount){
        # select column j
        thisColumn <- Combined[,j]

        # replace the values at or above the diagonal with zeroes
        thisColumn[1:rowCount < i] <- 0

        # find the maximum pivot in thisColumn at or below current row
        whichRow <- which.max(abs(thisColumn))  # which.max gets the index
                                                # of the maximum item in the vector
        pivot <- thisColumn[whichRow]     # select the element in the column for pivoting
        determinant <- determinant*pivot  # because the matrix determinant will be
                                          # divided by the pivot
        pivotList[i] <- pivot

        # check for zero pivot!
        if (abs(pivot) <= epsilon) {      # check for zero pivot!
            j <- j + 1                    # skip to the next column,
                                          # because there is a zero here!
            next                          # flow goes back to while (j <= colCount)
        }

        if (whichRow > i) {
            Combined[c(whichRow,i),] <- Combined[c(i,whichRow),] # switch rows
            determinant <- -determinant   # switching rows negates the determinant
        }
        Combined[i,] <- Combined[i,] / pivot    # multiply row by pivot
        for (k in 1:rowCount){
            if (k == i) next              # skip the diagonal
            if (abs(Combined[k, j]) < epsilon) next # there is a zero here, so skip to next k
            Combined[k,] <- Combined[k,] - Combined[k, j] * Combined[i,]
        }
        j <- j + 1
        break                             # quit the inner loop because pivot completed
    }
```

```
        i <- i + 1
    }

    # move rows filled with zeros to the bottom
    whichZeros <- which(apply(Combined[,1:colCount], 1, function(x) max(abs(x)) <= epsilon))
    if (length(whichZeros) > 0){
        # move rows of all zeros (inconsistent system) to the bottom
        Combined <- rbind(Combined[-whichZeros,],
                          Combined[whichZeros,])
    }
    # return the combined matrix
    Combined
}
```

Compute via MYGaussJordan :

```
MYGJ = MYGaussJordan(A,b)
```

Display the full result (with identity at left)

```
MYGJ
```

```
##      [,1] [,2] [,3]      [,4]
## [1,]    1    0    0 -1.545455
## [2,]    0    1    0 -0.318182
## [3,]    0    0    1  0.954545
```

$$Result_{MYGJ} = \begin{bmatrix} 1 & 0 & 0 & -1.545454545454545454 \\ 0 & 1 & 0 & -0.3181818181818181819 \\ 0 & 0 & 1 & 0.9545454545454545454 \end{bmatrix}$$

Compute the number of rows and columns, to trim the desired answer from the full MYGJ result

```
numrows <- nrow(MYGJ)
colsA <- ncol(A)
colsb <- ncol(b)
colsMYGJ <- ncol(MYGJ)
```

Result for x (e.g., final column)

```
xMYGJ <- MYGJ[,-(1:colsA)]
xMYGJ
```

```
## [1] -1.545455 -0.318182  0.954545
```

If the above is a single vector, it will not be stored as a matrix, so make it a matrix

```
xMYGJmat <- matrix(xMYGJ, nrow=numrows)
xMYGJmat
```

```
##           [,1]
## [1,] -1.545455
## [2,] -0.318182
## [3,]  0.954545
```

Display the result for x_{MYGJ}

$$x_{MYGJ} = \begin{bmatrix} -1.54545454545454 \\ -0.318181818181819 \\ 0.954545454545454 \end{bmatrix}$$

Confirm that the MY Gauss Jordan result is correct:

```r
### Compute Ax
A %*% xMYGJmat
```

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

```r
### Check that Ax-b is close to zero (it may differ by a tiny epsilon due to numerical precision)
A %*% xMYGJmat - b
```

```
##                          [,1]
## [1,] -0.000000000000000888178
## [2,]  0.000000000000000444089
## [3,]  0.000000000000000000000
```

Final equation for MYGJ:

$$\begin{bmatrix} 1 & 1 & 3 \\ 2 & -1 & 5 \\ -1 & -2 & 4 \end{bmatrix} \begin{bmatrix} -1.545454545454545454 \\ -0.318181818181819 \\ 0.954545454545454 \end{bmatrix} = \begin{bmatrix} 0.9999999999999999 \\ 2 \\ 6 \end{bmatrix} \approx \begin{bmatrix} 1 \\ 2 \\ 6 \end{bmatrix}$$

Gauss-Jordan Elimination, using matlib::gaussianElimination() :

```r
# A is 3x3 coefficient matrix,
# B are the contraints
# function solves for vector x such that Ax=b

library(matlib)
```

```
##
## Attaching package: 'matlib'
## The following objects are masked from 'package:pracma':
##
##     angle, inv
```

```
gaussJordan <- function(A, b) {
  x = gaussianElimination(A,b,verbose=TRUE)
  return(x)
}
```

Compute via Gauss-Jordan Elimination (package matlib) :

```
GJ <- gaussJordan(A,b)
```

```
##
## Initial matrix:
##      [,1] [,2] [,3] [,4]
## [1,]    1    1    3    1
## [2,]    2   -1    5    2
## [3,]   -1   -2    4    6
##
## row: 1
##
##  exchange rows 1 and 2
##      [,1] [,2] [,3] [,4]
## [1,]    2   -1    5    2
## [2,]    1    1    3    1
## [3,]   -1   -2    4    6
##
##  multiply row 1 by 0.5
##      [,1] [,2] [,3] [,4]
## [1,]    1 -0.5  2.5    1
## [2,]    1  1.0  3.0    1
## [3,]   -1 -2.0  4.0    6
##
##  subtract row 1 from row 2
##      [,1] [,2] [,3] [,4]
## [1,]    1 -0.5  2.5    1
## [2,]    0  1.5  0.5    0
## [3,]   -1 -2.0  4.0    6
##
##  multiply row 1 by 1 and add to row 3
##      [,1] [,2] [,3] [,4]
## [1,]    1 -0.5  2.5    1
## [2,]    0  1.5  0.5    0
## [3,]    0 -2.5  6.5    7
##
## row: 2
##
##  exchange rows 2 and 3
##      [,1] [,2] [,3] [,4]
## [1,]    1 -0.5  2.5    1
## [2,]    0 -2.5  6.5    7
## [3,]    0  1.5  0.5    0
##
##  multiply row 2 by -0.4
##      [,1] [,2] [,3] [,4]
## [1,]    1 -0.5  2.5  1.0
## [2,]    0  1.0 -2.6 -2.8
## [3,]    0  1.5  0.5  0.0
##
##  multiply row 2 by 0.5 and add to row 1
##      [,1] [,2] [,3] [,4]
## [1,]    1  0.0  1.2 -0.4
## [2,]    0  1.0 -2.6 -2.8
## [3,]    0  1.5  0.5  0.0
##
##  multiply row 2 by 1.5 and subtract from row 3
##      [,1] [,2] [,3] [,4]
## [1,]    1    0  1.2 -0.4
## [2,]    0    1 -2.6 -2.8
## [3,]    0    0  4.4  4.2
##
## row: 3
##
##  multiply row 3 by 0.227273
##      [,1] [,2] [,3]      [,4]
## [1,]    1    0  1.2 -0.400000
## [2,]    0    1 -2.6 -2.800000
## [3,]    0    0  1.0  0.954545
## 
##  multiply row 3 by 1.2 and subtract from row 1
##      [,1] [,2] [,3]      [,4]
## [1,]    1    0  0.0 -1.545455
## [2,]    0    1 -2.6 -2.800000
## [3,]    0    0  1.0  0.954545
## 
##  multiply row 3 by 2.6 and add to row 2
##      [,1] [,2] [,3]      [,4]
## [1,]    1    0    0 -1.545455
## [2,]    0    1    0 -0.318182
## [3,]    0    0    1  0.954545
```

Display the full result (with identity at left)

```r
GJ
```

```
##      [,1] [,2] [,3]      [,4]
## [1,]    1    0    0 -1.545455
## [2,]    0    1    0 -0.318182
## [3,]    0    0    1  0.954545
```

$$Result_{GJ} = \begin{bmatrix} 1 & 0 & 0 & -1.54545455 \\ 0 & 1 & 0 & -0.31818182 \\ 0 & 0 & 1 & 0.95454545 \end{bmatrix}$$

Compute the number of rows and columns, to trim the desired answer from the full GJ result

```r
numrows <- nrow(GJ)
colsA <- ncol(A)
```

```
colsb <- ncol(b)
colsGJ <- ncol(GJ)
```

Result for x (e.g., final column)

```
xGJ <- GJ[,-(1:colsA)]
xGJ
```

```
## [1] -1.545455 -0.318182  0.954545
```

If the above is a single vector, it will not be stored as a matrix, so make it a matrix

```
xGJmat <- matrix(xGJ, nrow=numrows)
xGJmat
```

```
##           [,1]
## [1,] -1.545455
## [2,] -0.318182
## [3,]  0.954545
```

Display the result for x_{GJ}

$$x_{GJ} = \begin{bmatrix} -1.54545455 \\ -0.31818182 \\ 0.95454545 \end{bmatrix}$$

Confirm that the Gauss-Jordan result is correct:

```
### Compute Ax
A %*% xGJmat
```

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

```
### Check that Ax-b is close to zero (it may differ by a tiny epsilon due to numerical precision)
A %*% xGJmat - b
```

```
##              [,1]
## [1,] -0.00000002
## [2,] -0.00000003
## [3,] -0.00000001
```

Final equation for Gauss-Jordan:

$$\begin{bmatrix} 1 & 1 & 3 \\ 2 & -1 & 5 \\ -1 & -2 & 4 \end{bmatrix} \begin{bmatrix} -1.54545455 \\ -0.31818182 \\ 0.95454545 \end{bmatrix} = \begin{bmatrix} 0.99999998 \\ 1.99999997 \\ 5.99999999 \end{bmatrix} \approx \begin{bmatrix} 1 \\ 2 \\ 6 \end{bmatrix}$$

Row-Reduced Echelon Form, using pracma::rref()

```
# A is 3x3 coefficient matrix,
# B are the contraints
# function solves for vector x such that Ax=b

library(pracma)
RowReducedEchelonForm <- function(A, b) {
  x = rref(cbind(A,b))
  return(x)
}
```

Compute via Row Reduced Echelon Form (rref) :

```r
RREF <- RowReducedEchelonForm(A,b)
```

Display the full result (with identity at left)

```r
RREF
```

```
##      [,1] [,2] [,3]      [,4]
## [1,]    1    0    0 -1.545455
## [2,]    0    1    0 -0.318182
## [3,]    0    0    1  0.954545
```

$$Result_{RREF} = \begin{bmatrix} 1 & 0 & 0 & -1.545454545454545454 \\ 0 & 1 & 0 & -0.318181818181819 \\ 0 & 0 & 1 & 0.954545454545454 \end{bmatrix}$$

Compute the number of rows and columns, to trim the desired answer from the full RREF result

```r
numrows <- nrow(RREF)
colsA <- ncol(A)
colsb <- ncol(b)
colsRREF <- ncol(RREF)
```

Result for x (e.g., final column)

```r
xRREF <- RREF[,-(1:colsA)]
xRREF
```

```
## [1] -1.545455 -0.318182  0.954545
```

If the above is a single vector, it will not be stored as a matrix, so make it a matrix

```r
xRREFmat <- matrix(xRREF, nrow=numrows)
xRREFmat
```

```
##            [,1]
## [1,] -1.545455
## [2,] -0.318182
## [3,]  0.954545
```

Display the result for x_{RREF}

$$x_{RREF} = \begin{bmatrix} -1.54545454545454 \\ -0.318181818181819 \\ 0.954545454545454 \end{bmatrix}$$

Confirm that the Row Reduced Echelon Form result is correct:

```r
### Compute Ax
A %*% xRREFmat
```

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

```r
### Check that Ax-b is close to zero (it may differ by a tiny epsilon due to numerical precision)
A %*% xRREFmat - b
```

```
##                           [,1]
## [1,] -0.000000000000000888178
## [2,]  0.000000000000000444089
## [3,]  0.000000000000000000000
```

Final equation for RREF:

$$\begin{bmatrix} 1 & 1 & 3 \\ 2 & -1 & 5 \\ -1 & -2 & 4 \end{bmatrix} \begin{bmatrix} -1.545454545454545 \\ -0.318181818181819 \\ 0.954545454545454 \end{bmatrix} = \begin{bmatrix} 0.999999999999999 \\ 2 \\ 6 \end{bmatrix} \approx \begin{bmatrix} 1 \\ 2 \\ 6 \end{bmatrix}$$