## Caching the Inverse of a Matrix:

## Matrix inversion is usually a costly computation and there may be some

## benefit to caching the inverse of a matrix rather than compute it repeatedly.

## Below are a pair of functions that are used to create a special object that

## stores a matrix and caches its inverse.

## This function creates a special "matrix" object that can cache its inverse.

makeCacheMatrix <- function(x = matrix()) {

inv <- NULL

set <- function(y) {

x <<- y

inv <<- NULL

}

get <- function() x

setInverse <- function(inverse) inv <<- inverse

getInverse <- function() inv

list(set = set,

get = get,

setInverse = setInverse,

getInverse = getInverse)

}

## This function computes the inverse of the special "matrix" created by

## makeCacheMatrix above. If the inverse has already been calculated (and the

## matrix has not changed), then it should retrieve the inverse from the cache.

cacheSolve <- function(x, ...) {

## Return a matrix that is the inverse of 'x'

inv <- x$getInverse()

if (!is.null(inv)) {

message("getting cached data")

return(inv)

}

mat <- x$get()

inv <- solve(mat, ...)

x$setInverse(inv)

inv

}

> source("ProgrammingAssignment2/cachematrix.R")  
> my\_matrix <- makeCacheMatrix(matrix(1:4, 2, 2))  
> my\_matrix$get()  
 [,1] [,2]  
[1,] 1 3  
[2,] 2 4  
> my\_matrix$getInverse()  
NULL  
> cacheSolve(my\_matrix)  
 [,1] [,2]  
[1,] -2 1.5  
[2,] 1 -0.5  
> cacheSolve(my\_matrix)  
getting cached data  
 [,1] [,2]  
[1,] -2 1.5  
[2,] 1 -0.5  
> my\_matrix$getInverse()  
 [,1] [,2]  
[1,] -2 1.5  
[2,] 1 -0.5  
> my\_matrix$set(matrix(c(2, 2, 1, 4), 2, 2))  
> my\_matrix$get()  
 [,1] [,2]  
[1,] 2 1  
[2,] 2 4  
> my\_matrix$getInverse()  
NULL  
> cacheSolve(my\_matrix)  
 [,1] [,2]  
[1,] 0.6666667 -0.1666667  
[2,] -0.3333333 0.3333333  
> cacheSolve(my\_matrix)  
getting cached data  
 [,1] [,2]  
[1,] 0.6666667 -0.1666667  
[2,] -0.3333333 0.3333333  
> my\_matrix$getInverse()  
 [,1] [,2]  
[1,] 0.6666667 -0.1666667  
[2,] -0.3333333 0.3333333