# The makeCacheMatrix function creates a special "matrix",

# which is really a list containing a function to

# 1. set the value of the matrix

# 2. get the value of the matrix

# 3. set the value of the inverse of the matrix

# 4. get the value of the inverse of the matrix

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

i <- NULL

set <- function(y) {

x <<- y

i <<- NULL

}

get <- function() x

setinverse <- function(inverse) i <<- inverse

getinverse <- function() i

list(set = set,

get = get,

setinverse = setinverse,

getinverse = getinverse)

}

# The cacheSolve function calculates the inverse of the special "matrix"

# the special "matrix" which created with the makeCacheMatrix function.

# However, it first checks to see if the inverse has already been calculated.

# If so, it gets the inverse from the cache and skips the computation.

# Otherwise, it calculates the inverse of the matrix and sets the value of the inverse

# in the cache via the setinverse function.

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

i <- x$getinverse()

if (!is.null(i)) {

message("getting cached data")

return(i)

}

data <- x$get()

i <- solve(data, ...)

x$setinverse(i)

i

}

# Running makeCacheMatrix.R & cacheSolve.R in the R Console

# The steps and results as displayed belows;

R version 3.4.1 (2017-06-30) -- "Single Candle"

Copyright (C) 2017 The R Foundation for Statistical Computing

Platform: x86\_64-apple-darwin15.6.0 (64-bit)

> B <- matrix(c(1,2,3,4),2,2)

> B1 <- makeCacheMatrix(B)

> cacheSolve(B1)

[,1] [,2]

[1,] -2 1.5

[2,] 1 -0.5