# Caching the inverse of a matrix

Arin Parsa

6/22/2020

## Contents

### Assignment: Caching the Inverse of a Matrixless

From JHU Data Science Course on Coursera

"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 (there are also alternatives to matrix inversion that we will not discuss here). Your assignment is to write a pair of functions that cache the inverse of a matrix.

Write the following functions:

- makeCacheMatrix: This function creates a special "matrix" object that can cache its inverse.
- cacheSolve: This function computes the inverse of the special "matrix" returned by * makeCacheMatrix above. If the inverse has already been calculated (and the matrix has not changed), then the cachesolve should retrieve the inverse from the cache."

```r
## makeCacheMatrix creates a special "matrix", which is a list
## containing a function to set and get the value of the matrix, and
## set and get the inverse of the matrix

makeCacheMatrix <- function(x = matrix()) {
  i <- NULL
  # set the matrix and clear the cache if any
  set <- function(y) {
    x <<- y    # Set the value
    i <<- NULL # Clear the cache
  }
  # get the matrix
  get <- function() x
  #set inverse in cache
  setInverse <- function(inverse) i <<- inverse
  # return cached inverse
  getInverse <- function() i

  #list of four functions to get and set matrix, get and set cached inverse
  list(set = set, get = get,
       setInverse = setInverse,
       getInverse = getInverse)
}

## cacheSolve calculates the inverse of the special "matrix"
## 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 data (matrix) and sets the value of the inverse
## in the cache via the setInverse function.

cacheSolve <- function(x, ...) {
  i <- x$getInverse() # This fetches the cached value for the inverse
  if(!is.null(i)) { # If the cache was not empty, we can just return it
    message("getting cached data")
    return(i)
  }
  # The cache was empty. We need to calculate it, cache it, and then return it.
  data <- x$get()  # Get value of matrix
  i <- solve(data) # Calculate inverse
  x$setInverse(i)  # Cache the result
  i
}
```

## Calling cacheSolve function twice. Note that on second iteration, we get cached inverse

```r
specialMatrix <- makeCacheMatrix(matrix(c(1, 1, 1, 2), 2, 2))
cacheSolve(specialMatrix)
```

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

```r
cacheSolve(specialMatrix)
```

```
## getting cached data
```

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