R Programming Week 3 Assignment

Ken Wood

7/6/2020

For large square matrices, it may take too long to compute the inverse, especially if it has to be computed repeatedly (e.g. in a loop). If the contents of the matrix do not change, it may make sense to cache the matrix inverse so that, when we need it again, it can be looked up in the cache rather than recomputed. In this Programming Assignment, we take advantage of the scoping rules of the R language and observe how they can be manipulated to preserve state inside of an R object.

For this assignment, we introduce the «- operator which can be used to assign a value to an object in an environment that is different from the current environment. Below are two functions that we will use to create a special object that stores a square matrix and caches its inverse.

The first function, makeCacheMatrix creates a special "matrix", which is really a list containing a function to:

- set the elements of the matrix
- get the elements of the matrix
- set the elements of the matrix inverse
- get the elements of the matrix inverse

```r
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)
}
```

The following function calculates the inverse of the special "matrix" created with the above 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 it in the cache via the setinverse function.

```r
cacheinverse <- function(x, ...) {
  inv <- x$getinverse()
  if(!is.null(inv)) {
    message("getting cached data")
    return(inv)
  }
  matrix_to_invert <- x$get()
  inv <- solve(matrix_to_invert, ...)
  x$setinverse(inv)
  inv
}
```

First example to test my code…

```
my_Matrix <- makeCacheMatrix(matrix(1:4, 2, 2))
```

```
my_Matrix$get()
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

```
my_Matrix$getinverse()
```

```
## NULL
```

```
cacheinverse(my_Matrix)
```

```
##      [,1] [,2]
## [1,]   -2  1.5
## [2,]    1 -0.5
```

```
cacheinverse(my_Matrix)
```

```
## getting cached data
```

```
##      [,1] [,2]
## [1,]   -2  1.5
## [2,]    1 -0.5
```

Second example to test my code…

```
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
```

```
cacheinverse(my_Matrix)
```

```
##            [,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
```