R Programming Week 3 Assignment

Stephen Holtmann

2022-04-24

## makeCacheMatrix

This is the makeCacheMatrix Function. This function configures a set of functions to set and evaluate the contents of the cache for the matrix input. The set function loads the initial matrix into cache.

makeCacheMatrix <- function(x = matrix()) {  
 ## x: is a square matrix that is dtinvertable  
 ## return: a list containing four(4) functions that  
 ## - set the matrix in cache  
 ## - get the matrix in cache  
 ## - set the inverse matrix in cache  
 ## - get the inverse matrix in cache  
 ## These functions are used as the input to cacheSolve() function.  
   
 inv = NULL  
 set = function(y) {  
 # the `<<-` is used to assign a value to an object in an   
 # environment different from the current environment.   
 x <<- y  
 inv <<- NULL  
 }  
 get = function() x  
 setinv = function(inverse) inv <<- inverse   
 getinv = function() inv  
 list(set=set, get=get, setinv=setinv, getinv=getinv)  
}

## cacheSolve

This is the cacheSolve Function. This function uses the functions defined by makeCacheMatrix to determin if the matrix is in cache. The function uses the “solve” function to determine the inverse matrix.

cacheSolve <- function(x, ...) {  
 ## x: is the input matrix  
 ## inv: inverse of x  
 ## cacheSolve uses the functions created by makeCacheMatrix (get, getinv, setinv) to determine if the matrix inverse is in cache.  
   
 inv = x$getinv()  
   
 # if the inverse has already been calculated  
 if (!is.null(inv)){  
 # get it from the cache and skips the computation.   
 message("using cached data")  
 return(inv)  
 }  
   
 # otherwise, calculates the inverse   
 mat.data = x$get()  
 inv = solve(mat.data, ...)  
   
 # sets the value of the inverse in the cache via the setinv function.  
 x$setinv(inv)  
 return(inv)  
}

## matTestHarness

This is a test harness to evaluate the performance of set/get and recomputing the inverse.

matTestHarness <- function(matIn){  
 ## matIn: an invertible matrix  
 ## matInInv: the inverse of the matIn  
   
 temp = makeCacheMatrix(matIn)  
   
 start.time = Sys.time()  
 matInInv = cacheSolve(temp)  
 lapseTime = Sys.time() - start.time  
 print(lapseTime)  
   
 ## Repeat using cache. What is the time difference?  
 start.time = Sys.time()  
 matInInv = cacheSolve(temp)  
 lapseTime = Sys.time() - start.time  
 print(lapseTime)  
 return(matInInv)  
}

## Sample code to run the test harness and validate the inverse.

Note - If using large matrices, then remove(delete) the print statements.

set.seed(2175712)  
r = sample(1:10000)  
matTest = matrix(r, nrow=10, ncol=10)  
invmatTest = matTestHarness(matTest)

## Time difference of 0.001040936 secs

## using cached data

## Time difference of 0.01603103 secs

matTestInv <- solve(matTest)  
checkAns = matTest %\*% invmatTest  
print(matTest)

## [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]  
## [1,] 6294 8393 49 7088 9239 83 6128 3349 1747 5908  
## [2,] 1587 574 9978 4570 6329 1405 4945 235 1927 1208  
## [3,] 6611 3522 3007 809 5056 897 132 9118 9869 2476  
## [4,] 2564 4290 3879 3925 6200 8142 4201 9488 183 4428  
## [5,] 6501 2113 2525 4243 6613 7528 2725 4935 7680 6476  
## [6,] 2455 4334 4774 533 3579 9207 9809 7850 4361 377  
## [7,] 6377 3292 7318 9938 4912 6721 7179 7912 3777 6686  
## [8,] 4148 9025 3759 3407 7085 2557 976 9782 5765 4129  
## [9,] 8701 3632 6883 1182 6161 6545 7255 5079 5898 3940  
## [10,] 5971 4378 7117 1768 6571 7007 2347 1821 4740 1477

print(invmatTest)

## [,1] [,2] [,3] [,4] [,5]  
## [1,] 5.390285e-05 -1.004849e-04 1.165342e-04 4.989599e-05 -1.303491e-04  
## [2,] -7.675206e-06 -5.274720e-05 -1.243527e-04 -1.074549e-04 -2.406022e-05  
## [3,] -8.405387e-05 5.661839e-05 -6.994288e-05 -2.913032e-05 -3.055188e-05  
## [4,] 5.765782e-05 -2.041734e-05 8.422175e-05 -1.814029e-05 -3.402143e-05  
## [5,] 8.126460e-05 8.493689e-05 9.703148e-05 1.300594e-04 3.099640e-05  
## [6,] -2.038269e-05 -4.936125e-05 -3.254379e-05 2.563602e-05 4.385201e-05  
## [7,] 3.526401e-05 2.986083e-05 -2.230436e-05 -4.114907e-05 6.357298e-06  
## [8,] -4.955646e-06 -8.943506e-06 1.020720e-04 1.006007e-04 -6.674580e-05  
## [9,] -2.321109e-05 2.955116e-05 -1.263665e-08 -1.315984e-04 1.085283e-04  
## [10,] -9.943637e-05 3.923611e-05 -2.339573e-04 -5.767854e-05 1.641766e-04  
## [,6] [,7] [,8] [,9] [,10]  
## [1,] -6.644102e-05 6.241501e-05 -1.532321e-04 4.957358e-05 1.237017e-04  
## [2,] 2.377772e-05 2.020998e-05 1.795218e-04 2.848143e-05 3.455307e-05  
## [3,] -4.658928e-05 2.171108e-05 1.105296e-04 9.676547e-05 -2.505877e-05  
## [4,] 3.166000e-05 1.207065e-04 -1.044465e-04 -1.990565e-04 1.169256e-04  
## [5,] -5.357697e-06 -1.228979e-04 -1.226223e-04 -7.922876e-05 2.883115e-05  
## [6,] 3.197858e-05 9.293000e-06 -2.862650e-05 -5.913211e-05 9.486546e-05  
## [7,] 7.507866e-05 -1.191072e-05 -1.560914e-05 5.185258e-05 -9.252979e-05  
## [8,] -3.450215e-06 1.223849e-05 -4.810265e-05 -1.552533e-05 -3.155227e-05  
## [9,] 7.220684e-05 -4.152550e-06 6.075163e-05 -5.751061e-05 -2.867735e-05  
## [10,] -8.993517e-05 -4.149543e-05 2.502567e-04 2.414904e-04 -2.450198e-04

print(abs(checkAns), digits=0)

## [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]  
## [1,] 1 0 0 0 0 0 0 0 0 0  
## [2,] 0 1 0 0 0 0 0 0 0 0  
## [3,] 0 0 1 0 0 0 0 0 0 0  
## [4,] 0 0 0 1 0 0 0 0 0 0  
## [5,] 0 0 0 0 1 0 0 0 0 0  
## [6,] 0 0 0 0 0 1 0 0 0 0  
## [7,] 0 0 0 0 0 0 1 0 0 0  
## [8,] 0 0 0 0 0 0 0 1 0 0  
## [9,] 0 0 0 0 0 0 0 0 1 0  
## [10,] 0 0 0 0 0 0 0 0 0 1

checkAns = matTest %\*% matTestInv  
print(matTest)

## [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]  
## [1,] 6294 8393 49 7088 9239 83 6128 3349 1747 5908  
## [2,] 1587 574 9978 4570 6329 1405 4945 235 1927 1208  
## [3,] 6611 3522 3007 809 5056 897 132 9118 9869 2476  
## [4,] 2564 4290 3879 3925 6200 8142 4201 9488 183 4428  
## [5,] 6501 2113 2525 4243 6613 7528 2725 4935 7680 6476  
## [6,] 2455 4334 4774 533 3579 9207 9809 7850 4361 377  
## [7,] 6377 3292 7318 9938 4912 6721 7179 7912 3777 6686  
## [8,] 4148 9025 3759 3407 7085 2557 976 9782 5765 4129  
## [9,] 8701 3632 6883 1182 6161 6545 7255 5079 5898 3940  
## [10,] 5971 4378 7117 1768 6571 7007 2347 1821 4740 1477

print(matTestInv)

## [,1] [,2] [,3] [,4] [,5]  
## [1,] 5.390285e-05 -1.004849e-04 1.165342e-04 4.989599e-05 -1.303491e-04  
## [2,] -7.675206e-06 -5.274720e-05 -1.243527e-04 -1.074549e-04 -2.406022e-05  
## [3,] -8.405387e-05 5.661839e-05 -6.994288e-05 -2.913032e-05 -3.055188e-05  
## [4,] 5.765782e-05 -2.041734e-05 8.422175e-05 -1.814029e-05 -3.402143e-05  
## [5,] 8.126460e-05 8.493689e-05 9.703148e-05 1.300594e-04 3.099640e-05  
## [6,] -2.038269e-05 -4.936125e-05 -3.254379e-05 2.563602e-05 4.385201e-05  
## [7,] 3.526401e-05 2.986083e-05 -2.230436e-05 -4.114907e-05 6.357298e-06  
## [8,] -4.955646e-06 -8.943506e-06 1.020720e-04 1.006007e-04 -6.674580e-05  
## [9,] -2.321109e-05 2.955116e-05 -1.263665e-08 -1.315984e-04 1.085283e-04  
## [10,] -9.943637e-05 3.923611e-05 -2.339573e-04 -5.767854e-05 1.641766e-04  
## [,6] [,7] [,8] [,9] [,10]  
## [1,] -6.644102e-05 6.241501e-05 -1.532321e-04 4.957358e-05 1.237017e-04  
## [2,] 2.377772e-05 2.020998e-05 1.795218e-04 2.848143e-05 3.455307e-05  
## [3,] -4.658928e-05 2.171108e-05 1.105296e-04 9.676547e-05 -2.505877e-05  
## [4,] 3.166000e-05 1.207065e-04 -1.044465e-04 -1.990565e-04 1.169256e-04  
## [5,] -5.357697e-06 -1.228979e-04 -1.226223e-04 -7.922876e-05 2.883115e-05  
## [6,] 3.197858e-05 9.293000e-06 -2.862650e-05 -5.913211e-05 9.486546e-05  
## [7,] 7.507866e-05 -1.191072e-05 -1.560914e-05 5.185258e-05 -9.252979e-05  
## [8,] -3.450215e-06 1.223849e-05 -4.810265e-05 -1.552533e-05 -3.155227e-05  
## [9,] 7.220684e-05 -4.152550e-06 6.075163e-05 -5.751061e-05 -2.867735e-05  
## [10,] -8.993517e-05 -4.149543e-05 2.502567e-04 2.414904e-04 -2.450198e-04

print(abs(checkAns), digits=0)

## [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]  
## [1,] 1 0 0 0 0 0 0 0 0 0  
## [2,] 0 1 0 0 0 0 0 0 0 0  
## [3,] 0 0 1 0 0 0 0 0 0 0  
## [4,] 0 0 0 1 0 0 0 0 0 0  
## [5,] 0 0 0 0 1 0 0 0 0 0  
## [6,] 0 0 0 0 0 1 0 0 0 0  
## [7,] 0 0 0 0 0 0 1 0 0 0  
## [8,] 0 0 0 0 0 0 0 1 0 0  
## [9,] 0 0 0 0 0 0 0 0 1 0  
## [10,] 0 0 0 0 0 0 0 0 0 1

## Background Information

These are the makeVector and cacheMean functions from the homework that provided insight to complete the homework assignment.

Function makeVector

makeVector <- function(x = numeric()) {  
 m <- NULL  
 set <- function(y) {  
 x <<- y  
 m <<- NULL  
 }  
 get <- function() x  
 setmean <- function(mean) m <<- mean  
 getmean <- function() m  
 list(set = set, get = get,  
 setmean = setmean,  
 getmean = getmean)  
}

Function cacheMean

cacheMean <- function(x, ...) {  
 m <- x$getmean()  
 if(!is.null(m)) {  
 message("using cached data")  
 return(m)  
 }  
 data <- x$get()  
 m <- mean(data, ...)  
 x$setmean(m)  
 m  
}

Function vecTestHarness

vecTestHarness <- function(tvec){  
 ## @tvec: an invertible matrix  
   
 temp = makeVector(tvec)  
   
 start.time = Sys.time()  
 vecMean=cacheMean(temp)  
 lapseTime = Sys.time() - start.time  
 print(lapseTime)  
   
 start.time = Sys.time()  
 vecMean=cacheMean(temp)  
 lapseTime = Sys.time() - start.time  
 print(lapseTime)  
 return(vecMean)  
}

Running the vecTestHarness *Note needs lots of entries to show value of cache*

set.seed(217712)  
r = sample(1:100000000)  
vec1 = c(r)  
vectorMean=vecTestHarness(vec1)

## Time difference of 0.102371 secs

## using cached data

## Time difference of 0.003558874 secs

print(vectorMean, digits=10)

## [1] 50000000.5