## makeCacheMatrix stores a matrix and returns a list of functions,

## cacheSolve tests for and return the cached inverse of x or

## solves for and returns if the cache is empty

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

## define m inside the function

m <- NULL

## assign function output to set and assign values in parent env

set <- function(y) {

# assign the input to x in the parent env

# define m in the parent environment

x <<- y

m <<- NULL

}

## assign function with output type x to get

get <- function() x

## assign function of parameter and type inverse(matrix) to setinverse

setinverse <- function(inverse) m <<- inverse

## assign function of type m to getinverse

getinverse <- function() m

## make list of previous assignments

return(list(set = set,

get = get,

setinverse = setinverse,

getinverse = getinverse))

}

## cacheSolve tests if there is a value in m

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

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

## assign the getinverse element of x to m

m <- x$getinverse()

## if m is not null, return the contents of m

if(!is.null(m)) {

message("getting cached data")

return(m)

}

## assign makeCacheMatrix$get to data

data <- x$get()

## assign solve(data) to m

m <- solve(data, ...)

## calculate function (inverse)

x$setinverse(m)

return(m)

}

a <- matrix(c(1,0,5,2,1,6,3,4,0),nrow=3,ncol=3)

anothermatrix <- makeCacheMatrix(a)

invMatrix <- cacheSolve(anothermatrix)

invMatrix

[,1] [,2] [,3]

[1,] -24 18 5

[2,] 20 -15 -4

[3,] -5 4 1

cacheSolve(anothermatrix)

[,1] [,2] [,3]

[1,] -24 18 5

[2,] 20 -15 -4

[3,] -5 4 1

round(a %\*% invMatrix,1)

[,1] [,2] [,3]

[1,] 1 0 0

[2,] 0 1 0

[3,] 0 0 1