cachematrix.R

S

Fri Apr 28 13:59:28 2017

```
## Example: Caching the Mean of a Vectorless
## In this example we introduce the <<- operator which can be used to assign a value to an object in an

##The first function, makeVector creates a special "vector", which is really a list containing a functi
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)
}


## Write a short comment describing this function

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

}
#The following function calculates the mean of the special ???vector??? created with the above function
cachemean <- function(x, ...) {
        m <- x$getmean()
        if(!is.null(m)) {
                message("getting cached data")
                return(m)
        }
        data <- x$get()
        m <- mean(data, ...)
        x$setmean(m)
        m
}
# Moving on to the assignment functions
##Matrix inversion is usually a costly computation and there may be some benefit to caching the inverse

## Write the following functions:
        # 1. makeCacheMatrix: This function creates a special ???matrix??? object that can cache its in
        # 2 cacheSolve: This function computes the inverse of the special ???matrix??? returned by make
        # 3. Computing the inverse of a square matrix can be done with the solve function in R. For exa

### For this assignment, assume that the matrix supplied is always invertible.

# cachematrix.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)
}
## Creating the inverse of the above matrix
cacheSolve <- function(x, ...) {
        ## Return a matrix that is the inverse of 'x'
        inv <- x$getInverse()
        if (!is.null(inv)) {
                message("getting cached data")
                return(inv)
        }
        mat <- x$get()
        inv <- solve(mat, ...)
        x$setInverse(inv)
        inv
}
# Testing my functions
test_matrix <- makeCacheMatrix(matrix(1:6, 2, 2))
test_matrix$get()
```

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

```
test_matrix$getInverse()
```

```
## NULL
```

```
cacheSolve(test_matrix)
```

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

```
cacheSolve(test_matrix)
```

```
## getting cached data
```

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

```
test_matrix$getInverse()
```

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

```
test_matrix$set(matrix(c(5, 5, 4, 3), 2, 2))
test_matrix$getInverse()
```

```
## NULL
```

```
cacheSolve(test_matrix)
```

```
##      [,1] [,2]
## [1,] -0.6  0.8
## [2,]  1.0 -1.0
```

```
cacheSolve(test_matrix)
```

```
## getting cached data
```

```
##      [,1] [,2]
## [1,] -0.6  0.8
## [2,]  1.0 -1.0
```

```
test_matrix$getInverse()
```

```
##      [,1] [,2]
## [1,] -0.6  0.8
## [2,]  1.0 -1.0
```