# Generating Test Matrices

Daniel Escasa

12/28/2020

Once I'd written the functions, I needed matrices to test them. Two options were available to me, and I hope to others who need random matrices, not just for these functions, but for other purposes.

## Using R's built-in functions

`runif()`, among others, will do the job:

```
# First make the makeCacheMatrix and cacheSolve functions available
source("cachematrix.R")
testMat <- makeCacheMatrix(matrix(runif(25), nrow = 5))
```

Then:

```
testMat$get()
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] 0.66683643 0.9429528 0.8925770 0.5728023 0.8778898
## [2,] 0.09319061 0.3899966 0.7122194 0.7194716 0.3461780
## [3,] 0.98091672 0.5960380 0.4513898 0.1757406 0.6977095
## [4,] 0.24521550 0.5277387 0.9194662 0.9048860 0.1037974
## [5,] 0.21580745 0.2791239 0.3421606 0.1045591 0.1444036
```

Next,

```
testMat$getInv()
```

```
## NULL
```

```
cacheSolve(testMat)
```

```
##           [,1]       [,2]       [,3]      [,4]        [,5]
## [1,] -1.4266742 -0.2914104  1.8877116  0.806113 -0.32827512
## [2,]  4.7587257 -6.2561865 -2.5168896  2.896001 -3.85324540
## [3,] -3.1316872  4.1775018  0.5545721 -2.377410  8.05351236
## [4,]  0.8209799 -0.8930094  0.3370981  1.905823 -5.84892510
## [5,] -0.2402154  3.2764791  0.4857381 -2.549268  0.01621195
```

```
cacheSolve(testMat)
```

```
## getting cached data
```

```
##           [,1]       [,2]       [,3]      [,4]        [,5]
## [1,] -1.4266742 -0.2914104  1.8877116  0.806113 -0.32827512
## [2,]  4.7587257 -6.2561865 -2.5168896  2.896001 -3.85324540
## [3,] -3.1316872  4.1775018  0.5545721 -2.377410  8.05351236
## [4,]  0.8209799 -0.8930094  0.3370981  1.905823 -5.84892510
## [5,] -0.2402154  3.2764791  0.4857381 -2.549268  0.01621195
```

```
testMat$getInv()
```

```
##            [,1]       [,2]       [,3]      [,4]        [,5]
## [1,] -1.4266742 -0.2914104  1.8877116  0.806113 -0.32827512
## [2,]  4.7587257 -6.2561865 -2.5168896  2.896001 -3.85324540
## [3,] -3.1316872  4.1775018  0.5545721 -2.377410  8.05351236
## [4,]  0.8209799 -0.8930094  0.3370981  1.905823 -5.84892510
## [5,] -0.2402154  3.2764791  0.4857381 -2.549268  0.01621195
```

and, for good measure:

```
testMat$get() %*% testMat$getInv()
```

```
##               [,1]          [,2]          [,3]          [,4]          [,5]
## [1,]  1.000000e+00 4.440892e-16 -2.775558e-16 -8.881784e-16 -7.251144e-16
## [2,]  1.942890e-16 1.000000e+00 -8.326673e-17  1.110223e-16  6.791442e-16
## [3,] -2.775558e-16 4.440892e-16  1.000000e+00  0.000000e+00  1.509209e-16
## [4,]  5.551115e-16 3.885781e-16 -1.110223e-16  1.000000e+00 -1.365878e-15
## [5,]  2.220446e-16 0.000000e+00  0.000000e+00  2.220446e-16  1.000000e+00
```

Note that this doesn't look like the identity matrix. If you look more closely, however, the entries in the diagonal are ones, and the others are close enough to zero. In fact, if I had round'ed the matrix, the entries along the diagonal would be exactly one, and the others would be exactly zero.

I was lucky to get a non-singular matrix the first time. If cacheSolve returns NaN, you'll have to try again.

Also, you can of course pass min = and max = parameters to runif.

## An online random matrix generator

I found a handy site for random generation of matrices.

For convenience, set the element and column separators to the comma (","), an option you can find below the Generate Matrix button. That'll make it easy to paste the generated matrix into your R console.

Here's one I got from them, plugged into the makeCacheMatrix function. Note the "byrow = true" parameter to the matrix creation function. I suppose you could omit it, giving you the transpose. Also, it doesn't always generate a non-singular matrix, so you'll have to keep at it until you get one. I used to have a program, written some 40 years ago, to generate non-singular matrices. Given enough time, I could write one again.

```
testMat <- makeCacheMatrix(matrix(c(5, 4, 9, 6, 8, 1, 0, 8, 8, 3, 3, 2, 2, 9, 6, 8, 8, 8, 4, 7, 4, 8, 6
                                    nrow = 5, byrow = TRUE))
```

```
testMat$get()
```

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

```
testMat$getInv()
```

```
## NULL
```

```
cacheSolve(testMat)
```

```
##             [,1]         [,2]        [,3]        [,4]         [,5]
```

```
## [1,] -0.14461980  0.032424677  0.18479197  0.25796270 -0.231850789
## [2,] -0.12453372  0.000143472 -0.11865136  0.02769010  0.189239598
## [3,]  0.05308465  0.093256815 -0.12338594 -0.00143472  0.005738881
## [4,] -0.16241033  0.103873745  0.09641320  0.04763271  0.009469154
## [5,]  0.33974175 -0.203156385  0.01032999 -0.20918221  0.036728838
```

```
testMat$getInv()
```

```
##             [,1]         [,2]        [,3]        [,4]         [,5]
## [1,] -0.14461980  0.032424677  0.18479197  0.25796270 -0.231850789
## [2,] -0.12453372  0.000143472 -0.11865136  0.02769010  0.189239598
## [3,]  0.05308465  0.093256815 -0.12338594 -0.00143472  0.005738881
## [4,] -0.16241033  0.103873745  0.09641320  0.04763271  0.009469154
## [5,]  0.33974175 -0.203156385  0.01032999 -0.20918221  0.036728838
```

```
cacheSolve(testMat)
```

```
## getting cached data
```

```
##             [,1]         [,2]        [,3]        [,4]         [,5]
## [1,] -0.14461980  0.032424677  0.18479197  0.25796270 -0.231850789
## [2,] -0.12453372  0.000143472 -0.11865136  0.02769010  0.189239598
## [3,]  0.05308465  0.093256815 -0.12338594 -0.00143472  0.005738881
## [4,] -0.16241033  0.103873745  0.09641320  0.04763271  0.009469154
## [5,]  0.33974175 -0.203156385  0.01032999 -0.20918221  0.036728838
```

Verifying that this is indeed testMat's inverse is a trivial exercise for the reader.