Onionic matrices

Robin K. S. Hankin

Onionic matrices

The onion package allows one to define and manipulate matrices whose elements are quaternions or octonions. Function `romat()` creates a simple `onionmat` object:

```
set.seed(0)
options(digits=2)
romat()
```

```
##    [a,AL] [b,AL]  [c,AL] [d,AL] [e,AL] [a,AK] [b,AK] [c,AK] [d,AK] [e,AK]
## Re   1.26   0.41 -0.0058  -1.15   0.25  -0.22 -0.057 -1.285  -0.43   0.99
## i   -0.33  -1.54  2.4047  -0.29  -0.89   0.38  0.504  0.047  -0.65  -0.43
## j    1.33  -0.93  0.7636  -0.30   0.44   0.13  1.086 -0.236   0.73   1.24
## k    1.27  -0.29 -0.7990  -0.41  -1.24   0.80 -0.691 -0.543   1.15  -0.28
##    [a,AZ] [b,AZ] [c,AZ] [d,AZ] [e,AZ] [a,AR] [b,AR] [c,AR] [d,AR] [e,AR] [a,CA]
## Re   1.76   -1.2   0.83  2.441   0.62  0.359  -0.81  0.248  -0.65   0.14  -0.80
## i    0.56   -1.1  -0.23 -0.795  -0.17 -0.011   0.24  0.065  -0.12  -0.12   1.25
## j   -0.45   -1.6   0.27 -0.055  -2.22 -0.941  -1.43  0.019   0.66  -0.91   0.77
## k   -0.83    1.2  -0.38  0.250  -1.26 -0.116   0.37  0.257   1.10  -1.44  -0.22
##    [b,CA] [c,CA]  [d,CA] [e,CA] [a,CO] [b,CO] [c,CO] [d,CO] [e,CO]
## Re  -0.42   1.26  0.0084  -0.28  0.782  -1.13  -0.50  0.025  -1.12
## i   -0.42   0.65 -0.8809   1.46 -0.777   0.58   1.68  0.027   0.34
## j    1.00   1.30  0.5963   0.23 -0.616  -1.28  -0.41 -1.680   0.49
## k   -0.28  -0.87  0.1197   1.00  0.047   1.63  -0.97  1.054   0.14
##   AL AK AZ AR CA CO
## a  1  6 11 16 21 26
## b  2  7 12 17 22 27
## c  3  8 13 18 23 28
## d  4  9 14 19 24 29
## e  5 10 15 20 25 30
```

This illustrates many features of the package. An `onionmat` object has two slots. The first slot, `x`, is an onion [a vector of quaternions or octonions] and the second, `M`, a matrix, which is used to store attributes such as dimensions and dimnames. The elements of `M` and `d` are in bijective correspondence; thus element `[b,AR]` is number 17 and this is seen to be approximately $-0.81 + 0.24i - 1.43j + 0.37k$. Most R idiom will work with such objects, here is a brief sample.

```
A <- matrix(rquat(21),7,3)   # matrix() calls onion::onionmat()
A
```

```
##    [1,1] [2,1]   [3,1] [4,1] [5,1]  [6,1] [7,1] [1,2]  [2,2]  [3,2] [4,2] [5,2]
## Re -0.12 -1.11 -1.2329  0.80 -1.23  0.741 -1.02  0.47 -0.097 -0.866  0.32  0.78
## i   0.20  1.58 -0.0037 -0.97 -0.96  0.069 -0.77 -1.18  2.370  0.583 -0.49  0.71
## j  -1.07  1.50  1.5117  0.69 -0.87 -0.324 -1.12  1.47  0.891 -0.013  2.66 -0.54
## k  -0.80  0.26 -0.4757 -0.96 -0.91 -1.087 -0.45 -1.31 -0.252 -0.375  1.68  0.89
##    [6,2] [7,2] [1,3] [2,3] [3,3]  [4,3] [5,3]  [6,3] [7,3]
```

```
## Re -0.35 -0.29 -1.52 -0.07  0.53 -0.201  2.02 -1.064 -1.05
## i  -1.01 -0.61 -0.21 -0.43 -0.09  1.102 -0.70  0.018 -0.90
## j   1.88 -0.95 -0.57 -0.59  0.16 -0.017  0.96 -0.390  1.27
## k  -0.93  0.60 -1.39  0.98 -0.74  0.162  1.79 -0.491  0.59
##      [,1] [,2] [,3]
## [1,]    1    8   15
## [2,]    2    9   16
## [3,]    3   10   17
## [4,]    4   11   18
## [5,]    5   12   19
## [6,]    6   13   20
## [7,]    7   14   21
```

See above how object `A` has no rownames or colnames and the defaults are used. We may extract components:

```
A[1,]
```

```
##      [1]   [2]   [3]
## Re -0.12  0.47 -1.52
## i   0.20 -1.18 -0.21
## j  -1.07  1.47 -0.57
## k  -0.80 -1.31 -1.39
```

above, the resulting object is an onion but we may retain the onionmat character using `drop`:

```
A[1,,drop=FALSE]
```

```
##    [1,1] [1,2] [1,3]
## Re -0.12  0.47 -1.52
## i   0.20 -1.18 -0.21
## j  -1.07  1.47 -0.57
## k  -0.80 -1.31 -1.39
##      [,1] [,2] [,3]
## [1,]    1    2    3
```

The extraction methods operate as expected:

```
Re(A)
```

```
##       [,1]   [,2]  [,3]
## [1,] -0.12  0.472 -1.52
## [2,] -1.11 -0.097 -0.07
## [3,] -1.23 -0.866  0.53
## [4,]  0.80  0.318 -0.20
## [5,] -1.23  0.780  2.02
## [6,]  0.74 -0.349 -1.06
## [7,] -1.02 -0.294 -1.05
```

```
k(A)
```

```
##       [,1]  [,2]  [,3]
## [1,] -0.80 -1.31 -1.39
## [2,]  0.26 -0.25  0.98
## [3,] -0.48 -0.37 -0.74
## [4,] -0.96  1.68  0.16
## [5,] -0.91  0.89  1.79
## [6,] -1.09 -0.93 -0.49
## [7,] -0.45  0.60  0.59
```

(above, the matrices returned are numeric). Also replacement methods work as expected:

```
j(A) <- -1
A
```

```
##    [1,1] [2,1]   [3,1] [4,1] [5,1]  [6,1] [7,1] [1,2]  [2,2] [3,2] [4,2] [5,2]
## Re -0.12 -1.11 -1.2329  0.80 -1.23  0.741 -1.02  0.47 -0.097 -0.87  0.32  0.78
## i   0.20  1.58 -0.0037 -0.97 -0.96  0.069 -0.77 -1.18  2.370  0.58 -0.49  0.71
## j  -1.00 -1.00 -1.0000 -1.00 -1.00 -1.000 -1.00 -1.00 -1.000 -1.00 -1.00 -1.00
## k  -0.80  0.26 -0.4757 -0.96 -0.91 -1.087 -0.45 -1.31 -0.252 -0.37  1.68  0.89
##    [6,2] [7,2] [1,3] [2,3] [3,3] [4,3] [5,3]  [6,3] [7,3]
## Re -0.35 -0.29 -1.52 -0.07  0.53 -0.20   2.0 -1.064 -1.05
## i  -1.01 -0.61 -0.21 -0.43 -0.09  1.10  -0.7  0.018 -0.90
## j  -1.00 -1.00 -1.00 -1.00 -1.00 -1.00  -1.0 -1.000 -1.00
## k  -0.93  0.60 -1.39  0.98 -0.74  0.16   1.8 -0.491  0.59
##      [,1] [,2] [,3]
## [1,]    1    8   15
## [2,]    2    9   16
## [3,]    3   10   17
## [4,]    4   11   18
## [5,]    5   12   19
## [6,]    6   13   20
## [7,]    7   14   21
```

Some of the summary methods work:

```
sum(A)
```

```
##      [1]
## Re  -4.6
## i   -1.7
## j  -21.0
## k   -3.2
```

Matrix multiplication

Matrix multiplication is implemented.

```
A <- matrix(rquat(21),3,7)
umbral <- state.abb[1:7]
rownames(A) <- letters[1:3]
colnames(A) <- umbral

B <- matrix(rquat(28),7,4)
rownames(B) <- umbral
colnames(B) <- c("H","He","Li","Be")

A %*% B
```

```
##    [a,H] [b,H] [c,H] [a,He] [b,He] [c,He] [a,Li] [b,Li] [c,Li] [a,Be] [b,Be]
## Re   4.6  0.18  -5.5  -7.29   4.37    4.1    5.1   6.38  -2.68   -6.1    4.6
## i   -0.3  6.00   1.5   9.18   2.88   -3.4    3.1  -0.62  -7.79    2.9    4.4
## j   -1.6  3.39  -2.1   4.62  -7.31   -1.2    1.4  -9.02  -0.91   16.4   -2.2
## k    2.0 -6.30   6.2   0.73   0.86    1.5   -8.1   7.01   0.59    1.1   -8.2
##    [c,Be]
## Re   -0.7
## i   -16.7
```

```
## j    -5.4
## k     2.5
##   H He Li Be
## a 1  4  7 10
## b 2  5  8 11
## c 3  6  9 12
```

However, it is often preferable (but no faster in this case) to use functions such as `cprod()` and `tcprod()`:

```
C <- matrix(rquat(14),7,2)
rownames(C) <- umbral
colnames(C) <- month.abb[1:2]
cprod(B,C)
```

```
##    [H,Jan] [He,Jan] [Li,Jan] [Be,Jan] [H,Feb] [He,Feb] [Li,Feb] [Be,Feb]
## Re     1.1   -0.052     1.14      6.4   3.088    -0.61    -0.18      2.9
## i      3.3    4.925     6.18     -6.1  -0.037    -5.63    -4.12     11.8
## j      6.2   -0.643    -0.55     -4.1   0.604   -14.81    -5.34     -5.8
## k     -8.8   -2.366    -1.97     -2.0   5.096    -1.38    -0.29     -6.3
##    Jan Feb
## H    1   5
## He   2   6
## Li   3   7
## Be   4   8
```

and indeed the single-argument versions work as expected:

```
tcprod(A) - A %*% ht(A)
```

```
##    [a,a] [b,a] [c,a] [a,b] [b,b] [c,b] [a,c] [b,c] [c,c]
## Re     0     0     0     0     0     0     0     0     0
## i      0     0     0     0     0     0     0     0     0
## j      0     0     0     0     0     0     0     0     0
## k      0     0     0     0     0     0     0     0     0
##   a b c
## a 1 4 7
## b 2 5 8
## c 3 6 9
```

Octonions

Consider the following 3×3 octonionic matrices:

```
x <- cprod(matrix(roct(12),4,3))
x
```

```
##      [1,1] [2,1] [3,1] [1,2]    [2,2] [3,2] [1,3] [2,3]    [3,3]
## Re 3.4e+01  -6.0   3.7  -6.0  2.9e+01  -7.0   3.7  -7.0  3.7e+01
## i  0.0e+00   6.9  12.7  -6.9  0.0e+00  -4.7 -12.7   4.7  0.0e+00
## j  0.0e+00  -4.6  -3.8   4.6  1.2e-16   2.1   3.8  -2.1  8.3e-17
## k  4.4e-16   3.5  -4.0  -3.5 -1.5e-16  -5.1   4.0   5.1  1.9e-16
## l  1.1e-16  -7.9   4.5   7.9  5.6e-16 -10.0  -4.5  10.0 -3.1e-16
## il 5.6e-16   3.4  -7.8  -3.4 -3.1e-17  -4.2   7.8   4.2 -1.8e-16
## jl 2.5e-16   4.6   2.1  -4.6  3.5e-18   2.2  -2.1  -2.2 -1.4e-17
## kl 0.0e+00   2.8  -3.8  -2.8 -9.0e-17   6.5   3.8  -6.5  3.3e-16
##      [,1] [,2] [,3]
## [1,]    1    4    7
```

```
## [2,]    2    5    8
## [3,]    3    6    9
```

We see that `x` is Hermitian symmetric:

```
max(Mod(Im(x+t(x))))
```

```
## [1] 1.7e-15
```

[that is, the imaginary components of symmetrically placed elements are mutually conjugate]. We may verify that 3×3 matrices form a Jordan algebra under the composition rule $A \circ B = (AB + BA)/2$ [juxtaposition indicating regular matrix multiplication]; the identity is

$$(xy)(xx) = x(y(xx)).$$

First we define the Jordan product:

```
`%o%` <- function(x,y){(x%*%y + y%*%x)/2}
```

then create a couple of random Hermitian octonionic matrices:

```
x <- cprod(matrix(roct(12),4,3))
y <- cprod(matrix(roct(12),4,3))
```

We first verify numerically that the Jordan product of two Hermitian symmetric matrices is Hermitian:

```
jj <- x %o% y
max(Mod(Im(jj+t(jj))))
```

```
## [1] 1.3e-13
```

then verify the Jordan identity:

```
LHS <- (x %o% y) %o% (x %o% x)
RHS <- x %o% (y %o% (x %o% x))
max(Mod(LHS-RHS))  # zero to numerical precision
```

```
## [1] 1.5e-10
```

showing that the Jordan identity is satisfied, up to a small numerical tolerance.