A Matrix Operator in APL

Bob Smith
Sudley Place Software
bsmith@sudleyplace.com
Originally Written
30 Jul 2017
Updated
10 Sep 2017

PRELIMINARY

Introduction

Normally in APL when applying a scalar function to a matrix, the matrix is viewed as a container of its scalar elements and the scalar function applies to the individual elements. Matrix functions are different – in this case, the scalar function applies to the matrix as a whole[2], that is they treat the matrix as a new datatype. This concept is identical to that of the Multiset[3] operator where functions such as IndexOf (`L⍳⍦R`) and MemberOf (`L∊⍦R`) take on new meaning in that their arguments are treated as a new datatype, that is, sets with repeated elements where the repetitions play an integral role when calculating the result. Similarly, the Transpose, Matrix Inverse, and Matrix Divide functions and Inner Product and Determinant operators all apply to the matrix as a whole.

In the same manner, the (monadic) Matrix operator applies its (scalar function) operand to its matrix (right) argument as a new datatype to which the function (left operand) is applied. Not all scalar functions have been extended to matrices, but a large number have and are in use in various scientific fields. For the nonce, this operator applies its function operand to **diagonalizable**[5] **fixed-precision Real** matrices only.

Matrix functions date back to at least 1858 when A. Cayley wrote about matrix Square Roots. Their applications are many and varied. Here’s a list of the section headings in Nicholas J. Higham’s “Functions of Matrices”[11], Chapter 2, Applications:

- Differential Equations
- Exponential Integrators
- Nuclear Magnetic Resonance
- Markov Models
- Control Theory
- The Nonsymmetric Eigenvalue Problem
- Orthogonalization and the Orthogonal Procrustes Problem
- Theoretical Particle Physics
- Other Matrix Functions
- Nonlinear Matrix Equations
- Geometric Mean
- Pseudospectra
- Algebras
- Sensitivity Analysis

Properties of a Matrix Function

In order to be worthy of study, a matrix function must have some useful and interesting properties where `f` is a function defined on square Complex matrices and `X` is a non-singular conformable square matrix:

`f(A)` commutes with `A`: `A+.×f⌼A ←→ (f⌼A)+.×A`
`f⌼⍉A ←→ ⍉f⌼A`
`f⌼X+.×A+.×⌹X ←→ X+.×(f⌼A)+.×⌹X`

and many more.

Definition of a Matrix Function

There are multiple ways to define how a matrix function works. One way are to split into cases of diagonalizable and non-diagonalizable matrices. The latter case is not treated here.

Diagonalizable Matrices

A matrix `A` is diagonalizable[5] if there exists an invertible unitary matrix `P` and a diagonal matrix `D` such that `A≡P+.×D+.×⌹P`. Showing their sense of humor, mathematicians call a non-diagonalizable matrix **defective**. The process of finding the matrices `P` and `D` above involves computing the **Eigenvalues** and **Eigenvectors** of the matrix `A`.

Eigenvalues and Eigenvectors

These concepts[1] from Linear Algebra and Matrix Theory define the characteristic values and vectors of the linear transformation represented by a matrix. Every square simple Real numeric matrix has (possibly Complex) Eigenvalues and Eigenvectors. To calculate these objects in NARS2000, use the Variant operator with a left operand of the Domino function and a right operand of an integer scalar as described in the following table:

`Z←(⌹⍠1) R`	`Z` is a Complex floating point vector of the Eigenvalues
`Z←(⌹⍠2) R`	`Z` is a Complex floating point matrix of the Eigenvectors one per column
`Z←(⌹⍠3) R`	`Z` is a Nested two-element vector with a Complex floating point vector of the Eigenvalues in the first element and a Complex floating point matrix of the Eigenvectors in the second
`Z←(⌹⍠4) R`	`Z` is a Nested three-element vector with a Complex floating point vector of the Eigenvalues in the first element, a Complex floating point matrix of the Eigenvectors in the second, and a Real matrix of Schur vectors one per column in the third

In particular, if the matrix `A` is diagonalizable, then the Eigenvalues and Eigenvectors of `A` are exactly the components of its diagonalizable representation using the helper function `Diag`:

```
      Diag←{
➥A←(2⍴⍴⍵)⍴0
➥⋄ A[,¨⍨⍳⍴⍵]←⍵
➥⋄ A
➥}
      Diag 1 2 3
1 0 0
0 2 0
0 0 3
      (Eval Evec)←⌹⍠3 A
      A≡Evec+.×(Diag Eval)+.×⌹Evec
1
```

This latter form then yields a definition of a function `f` applied to matrix as a Matrix Function:

```
      (Eval Evec)←⌹⍠3 A
      Evec+.×(Diag f Eval)+.×⌹Evec
```

Structured as a Matrix Operator, this becomes

```
      MatOpr←{
➥⍺←⊢                         ⍝ Optional left argument
➥⋄ (Eval Evec)←⌹⍠3 ⍵ ⍝ Eigenvalues/vectors
➥⋄ Evec+.×(Diag ⍺ ⍺⍺ Eval)+.×⌹Evec
➥}
```

With this enhancement, many more functions can be extended to work on matrices as a whole. For example, we can use this to calculate the factorial of a diagonalizable matrix:

```
      M←2 2⍴1 3 2 1
      M
1 3
2 1
      !MatOpr M
3.6274 8.8423
5.8949 3.6274
      *MatOpr M
 15.86  19.136
 12.758 15.86
      ⍟MatOpr M
0.80472J1.5708  0.53094J¯1.9238
0.35396J¯1.2825 0.80472J1.5708
      ⍟MatOpr *MatOpr M
1 3
2 1
```

The above examples show how various functions such as Exponentiation and Logarithm can be extended to work on square Real matrices as a whole matrix. The last example demonstrates that these

Matrix functions work as expected with their inverse.

In turn, this operator can be used to calculate the Factorial of a Complex number:

```
      CplexFact←{
➡⎕IO←1
➡⋄ (a b)←>ω          ⍝ Coefficients of a Complex #
➡⋄ M←2 2⍴a (-b) b a  ⍝ Matrix rep of a ...
➡⋄ F←!MatOpr M       ⍝ Matrix factorial of ...
➡⋄ <9○1⌷[2] F        ⍝ Factorial of a Complex #
➡}
      CplexFact 1J2
 0.11229J0.32361
      !1J2
 0.11229J0.32361
```

where the latter expression uses the Gnu Scientific Library Complex number routines to calculate the Factorial of a Complex number.

In a similar manner, using the appropriate `4×4` matrix representation of a Quaternion, its Factorial can be calculated which is the algorithm used in the implementation:

```
      !<⍳4
0.0060975i¯0.0010787j¯0.001618k¯0.0021573
```

A Matrix Operator

The `MatOpr` operator has been implemented as a primitive operator in the Alpha version of NARS2000. The symbol for the Matrix Operator is QuadJot (⌻, U+233B) (Alt-'F' or Ctrl-'F' depending upon your keyboard layout). The left operand is either a Jot (`∘`) or the scalar function to be applied to the entire matrix. For example,

```
      M
1 3
2 1
      !⌼M
3.6274 8.8423
5.8949 3.6274
      *⌼M
 15.86  19.136
 12.758 15.86
      ⍟⌼M
0.80472J1.5708  0.53094J¯1.9238
0.35396J¯1.2825 0.80472J1.5708
      ⍟⌼*⌼M
1 3
2 1
      A←(?3 3⍴10)÷100
      A≡¯1○⌼1○⌼A
1
```

Squaring a matrix can be done using the Commute operator as in `2*⍨⌼M` which is logically the same as `M+.×M`., or `+.×⍨M`.

Moreover, with a left operand of a Jot, the Matrix Operator provides two new functions:

Diagonal Matrix

If the right argument to the derived function `∘⌼` is a vector,

```
      ∘⌼⍳4
1 0 0 0
0 2 0 0
0 0 3 0
```

```
0 0 0 4
```

produces a diagonal matrix, which is also a new way to produce an identity matrix: `∘⌻Nρ1`. This function also produces block diagonal matrices from a nested vector of matrices, vectors, and scalars:

```
      ∘⌻(2 2ρι4x)(3 3ρι9v)10(20 30)
1 2 0 0 0  0  0  0
3 4 0 0 0  0  0  0
0 0 1 2 3  0  0  0
0 0 4 5 6  0  0  0
0 0 7 8 9  0  0  0
0 0 0 0 0 10  0  0
0 0 0 0 0  0 20  0
0 0 0 0 0  0  0 30
```

Matrix Representation

If the right argument to the derived function `∘⌻` is a Hypercomplex scalar,

```
      ∘⌻123456789
123456789
      ∘⌻<ι2
1 ¯2
2  1
      ∘⌻<ι4
1 ¯2 ¯3 ¯4
2  1 ¯4  3
3  4  1 ¯2
4 ¯3  2  1
```

produces one of the several equivalent matrix representations for Real, Complex[9], and Quaternions[10] right arguments only.

These two derived functions for Diagonal Matrix and Matrix Representation seem to conflict on a simple Real scalar versus a simple one-element Real vector, however in fact they are identical as they have the same value and shape:

```
      (∘⌻N)≡∘⌻,N←123456789
1
```

In general, a Hypercomplex number has a matrix representation if the matrix representation is both additive and multiplicative. That is, for Hypercomplex `a` and `b`, if

```
(∘⌻a+b)≡(∘⌻a)+   ∘⌻b
(∘⌻a×b)≡(∘⌻a)+.×∘⌻b
```

Unfortunately, because Octonions are not associative, no matrix representation of Octonions satisfies the multiplicative property. However, an excellent paper by Yongge Tian[8] shows how to represent Octonions in a left and right matrix form with several restrictions on how the matrix representations behave. In the Alpha version of NARS2000, the left and right matrix representations are distinguished by making the matrix representation function (`∘⌻`) on Octonions sensitive to the system variable `⎕DQ` which may also be specified through the Variant operator as follows:

```
      om←{∘⌻⍠'l' ω} ⍝ Left matrix representation
      nu←{∘⌻⍠'r' ω} ⍝ Right ...

      om <⍳8
1 ¯2 ¯3 ¯4 ¯5 ¯6 ¯7 ¯8
2  1 ¯4  3 ¯6  5  8 ¯7
3  4  1 ¯2 ¯7 ¯8  5  6
4 ¯3  2  1 ¯8  7 ¯6  5
5  6  7  8  1 ¯2 ¯3 ¯4
6 ¯5  8 ¯7  2  1  4 ¯3
7 ¯8 ¯5  6  3 ¯4  1  2
8  7 ¯6 ¯5  4  3 ¯2  1
```

```
      nu ⊂⍳8
1 ¯2 ¯3 ¯4 ¯5 ¯6 ¯7 ¯8
2  1  4 ¯3  6 ¯5 ¯8  7
3 ¯4  1  2  7  8 ¯5 ¯6
4  3 ¯2  1  8 ¯7  6 ¯5
5 ¯6 ¯7 ¯8  1  2  3  4
6  5 ¯8  7 ¯2  1 ¯4  3
7  8  5 ¯6 ¯3  4  1 ¯2
8 ¯7  6  5 ¯4 ¯3  2  1
```

Note that neither the left (written as ω(a) in Tian's paper) nor right (ν(a)) matrix representation of Octonions is multiplicative in that the matrix representation of the product of two Octonions is **not** the product of their matrix representations. However, it is the case for Octonions a and b that a more complicated relationship exists[8] such that

```
(om a×b)≡(nu a)+.×((om a)+.×om b)+.×⌹nu a
(nu a×b)≡(om b)+.×((nu b)+.×nu a)+.×⌹om b
```

or alternatively

```
(om a×b)≡((om a)+.×om b)+((om a)+.×nu b)-(nu b)+.×om a
(nu a×b)≡((nu b)+.×nu a)+((om b)+.×nu a)-(nu a)+.×om b
```

and that the left and right matrix representations are related by

```
      K8←∘⌼1,7⍴¯1
      (nu a)≡K8+.×(⍉om a)+.×K8
```

Note that the above definition of K8 differs from the one in Tian's paper where it is defined as K8←∘⌼K4 I4.

Simplifying

The above functions Diag, MatOpr, and CplexFact may be simplified in the light of the new operator and its derived functions.

```
      Diag←∘⌼
      MatOpr←⌼
```

```
      HyperFact←{⍝ Hypercomplex version of CplexFact
➥⎕IO←1
➥⋄ M←∘⌹⍵       ⍝ Matrix rep of a Hypercomplex #
➥⋄ F←!⌹M       ⍝ Matrix factorial of ...
➥⋄ <9∘1⌷[2] F  ⍝ Factorial of a Hypercomplex #
➥}
```

or more simply and generally as a shallow depth (≤1) scalar function:

```
      HyperFact←{⍝ Hypercomplex version of CplexFact
➥⎕IO←1 ⋄ <¨9∘1⌷[2]¨!⌹¨∘⌹¨⍵
➥}
      HyperFact 1J2 3J4
 0.11229J0.32361 0.70586J¯0.49674
      !1J2 3J4
 0.11229J0.32361 0.70586J¯0.49674
      HyperFact <2 1 4⍴⍳8
0.0060975i¯0.0010787j¯0.001618k¯0.0021573
 0.010645i0.0039216j0.0045752k0.0052288
      !<2 1 4⍴⍳8
0.0060975i¯0.0010787j¯0.001618k¯0.0021573
 0.010645i0.0039216j0.0045752k0.0052288
```

Restrictions

For the moment, only diagonalizable[5] matrices are in the domain of this operator's derived functions (i.e., `M≡⊢⌹M`). The **Spectral Radius** of a matrix `R` is defined to be largest of the absolute values of the Eigenvalues of the matrix, i.e., `⌈/|(⌹⌹1) R`.

Certain functions have additional restrictions **[More examples needed]**:

A matrix has a logarithm if and only if it is invertible, that is, `0≠-.×M`.

Matrix square roots are multi-valued. For example, “the matrix `2 2⍴33 24 48 57` has square roots `2 2⍴1 4 8 5` and `2 2⍴5 2 4 7`, as well as their additive inverses”[4], so that the identity `M≡2*⍨⌹√⌹M` is more likely to hold than is `M≡√⌹2*⍨⌹M`.

Matrix trigonometric functions[6] whose range on scalars is `[¯1, 1]` are valid iff the Spectral Radius of the matrix is `≤1`, that is all of the Eigenvalues are in the same range as for the scalar functions.

...

Inverse Functions

[More examples needed]

`M≡¯1○⌹1○⌹M`

...

Identities

[More examples needed]

Subject to the above restrictions which is the matrix operator analog of $1 = e^{-x} \times e^{x}$:

```
(∘⌹(≢M)⍴1)≡(*⌹-M)+.×*⌹M
```

The following identity is the matrix operator form of the Pythagorean Trigonometric Identity ($\sin^2\theta + \cos^2\theta = 1$):

```
((2*⍨⌹1○⌹M)+2*⍨⌹2○⌹M)≡∘⌹(≢M)⍴1
```

...

Disclaimers

Some of the above matches (e.g., A≡B) against an identity matrix might fail because the off-diagonal elements of the identity matrix are all zero, but the corresponding elements on the other side of the match are close to but not equal to zero. Modulo this difference, the matches are correct.

Some of the above examples (e.g., !<⍳4) might run as indicated in the 32-bit version of NARS2000, but not the 64-bit version (and possibly vice versa). I've traced this down to a difference in the result of a GSL function between the 32- and 64-bit version of GSL. I'm still exploring this bug.

Online Version

This paper is an ongoing effort and can be out-of-date the next day. To find the most recent version, go to http://sudleyplace.com/APL/ and look for the title of this paper on that page.

References

1. Wikipedia, "Eigenvalues and Eigenvectors", https://en.wikipedia.org/wiki/Eigenvalues_and_eigenvectors
2. Wikipedia, "Matrix Function", https://en.wikipedia.org/wiki/Matrix_function
3. "Multisets", http://wiki.nars2000.org/index.php/Multisets
4. Wikipedia, "Square Root of a Matrix", https://en.wikipedia.org/wiki/Square_root_of_a_matrix#Properties
5. Wikipedia, "Diagonalizable Matrix", https://en.wikipedia.org/wiki/Diagonalizable_matrix
6. Wikipedia, "Trigonometric functions os matrices", https://en.wikipedia.org/wiki/Trigonometric_functions_of_matrices

7. Wikipedia, “Pythagorean Trigonometric Identity”, https://en.wikipedia.org/wiki/Pythagorean_trigonometric_identity
8. Tian, Yongge, “Matrix Representations of Octonions and Their Applications”, arXiv:math/0003166 [math.RA]
9. Wikipedia, “Complex Number Matrix Representation”, https://en.wikipedia.org/wiki/Complex_number#Matrix_representation_of_complex_numbers
10. Wikipedia, “Quaternion Matrix Representation”, https://en.wikipedia.org/wiki/Quaternion#Matrix_representations
11. Higham, Nicholas J. (2008). *Functions of matrices theory and computation*. Philadelphia: Society for Industrial and Applied Mathematics. ISBN 9780898717778, http://epubs.siam.org/doi/abs/10.1137/1.9780898717778