Functions of Matrices

You can't always get back to where you started

You might already know about the matrix exponential and its role in the solution of systems of ordinary differential equations. But have you ever wondered about the logarithm of a matrix, or the square root of a matrix? These functions also turn out to be interesting and useful.

In MATLAB, the functions

```
expm(A), logm(A), sqrtm(A)
```

compute the matrix exponential, matrix logarithm and matrix square root. (The "m" in their names distinguishes them from the ones which compute the corresponding element-by-element functions.) The *p*th power of a matrix doesn't need a named function; it's just

```
A^p
```

We can study these functions by checking identities involving the functional inverses. Does

```
sqrtm(A^2) = A ?
sqrtm(A)^2 = A ?
expm(logm(A)) = A ?
logm(expm(A)) = A ?
(A^p)^(1/p) = A ?
```

Let's start with a few hopefully surprising examples. For `A = [1 2; 3 4]`, we find that `sqrtm(A^2)` is not equal to A. In fact,

```
A =
      1      2
      3      4

A^2 =
      7     10
     15     22

sqrtm(A^2) =
    2.6458    3.1623
    3.8730    4.6904
```

For `A = [0 4; -4 0]`, we find that `logm(expm(A))` is not equal to A. In fact,

```
A =
    0      4
   -4      0

expm(A) =
   -0.6536   -0.7568
    0.7568   -0.6536

logm(expm(A)) =
    0.0000   -2.2143
    2.2143    0.0000
```

My favorite examples are Nick Higham's variants of the Pascal matrices, obtained with `pascal(n,2)`. The entries are binomial coefficients, with curious sign patterns, stored in a "flipped" triangular matrix. Here is `n = 4`.

```
P = pascal(4,2)
P =
  0     0     0    -1
  0     0    -1     3
  0     1     2    -3
  1     1     1     1
```

Squaring this matrix gives

```
P^2 =
 -1    -1    -1     1
  3     2     1     0
 -3    -1     0     0
 -1     0     0     0
```

and `sqrtm(P^2)` is nowhere near P. Even more interesting, the cube of P is the identity.

```
P^3 =
  1     0     0     0
  0     1     0     0
  0     0     1     0
  0     0     0     1
```

The cube root, `(P^3)^(1/3)`, is still the identity; we don't get P back.

What's going on here? There are two distinct issues, one "purely" mathematical and one computational. And, as you certainly have anticipated, the analysis involves eigenvalues.

Let's start with the simplest situation, the functions `z^2` and `sqrt(z)` applied to 1-by-1 matrices. Here we can leave off the "m" at the end of the function name. There is no question about the definition of `z^2`—just multiply z by itself. But notice that both z and -z have the same square. This leads to our first problem. What do we mean by `sqrt(z)`? The equation

```
w^2 = z
```

has two solutions; one is the negative of the other. So MATLAB, and most other mathematical computing environments, choose one of them for you—the one with the positive real part. If z is real and negative, then both solutions have zero real part, so the one with the positive imaginary part is chosen.

This choice cuts the complex plane in half. Let `C(1/2)` denote the right half of the complex plane, including the positive imaginary axis and the origin, but excluding the negative imaginary axis. Then the function `z^2` maps `C(1/2)` onto the entire complex plane and, conversely, the function `sqrt(z)` has been chosen to map the entire complex plane onto `C(1/2)`. Furthermore,

```
sqrt(z)^2 == z
```

for all z, but

```
sqrt(z^2) == z
```

only if z is in `C(1/2)`

Now consider `exp(z)` and `log(z)`, again for "monoelemental" matrices. As with `z^2`, the definition of `exp(z)` is not a problem. Although it is unsatisfactory in the presence of roundoff error, the infinite series

```
exp(z) = 1 + z + z^2/2 + z^3/6 + ...
```

does converge for any finite, complex `z` and provides a rigorous definition. But since

```
exp(z+2*pi*i) = exp(2*pi*i)*exp(z) = exp(z)
```

we see that infinitely many values of `z`, which differ by integer multiples of `2*pi*i`, have the same value of `exp(z)`. So, what do we mean by `log(z)`? The equation

```
exp(w) = z
```

has infinitely many solutions. MATLAB, and many (but not all) other systems, chose the one that has an imaginary part in the range

```
-pi < imag(log(z)) <= pi
```

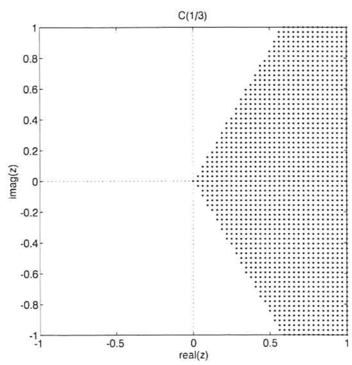
This defines another region in the complex plane; let's call it `S` (for "strip"). Then `exp(z)` maps `S` onto the entire plane and, conversely, `log(z)` has been chosen to map the entire plane onto `S`.

Furthermore,

`exp(log(z))` always equals `z`

but

`log(exp(z))` equals `z` only if `z` is in `S`.



The last 1-by-1 case involves the function `z^p`, which, by reciprocating `p`, can serve as its own inverse. The definition of `z^p` relies on the definitions of `exp` and `log`,

```
z^p = exp(p*log(z))
```

The corresponding region in the complex plane is a pie-shaped slice containing the positive real axis, namely the set of `z`, whose polar angle satisfies

```
-p*pi < angle(z) <= p*pi
```

Let's call this `C(p)`, generalizing our definition of `C(1/2)`. The figure shows `C(1/3)`. For `p >= 1`, `C(p)` is the whole plane, but as `p` gets smaller, so does `C(p)`. `C(2/3)` is two-thirds of the plane, `C(1/4)` is one-fourth, etc. For `p < 1`, the function `z^p` maps the whole plane onto `C(p)` and the inverse function `z^(1/p)` maps `C(p)` onto the whole plane.

This implies that for `p <= 1`,

`(z^p)^(1/p)` always equals `z`

but for `p > 1`,

`(z^p)^(1/p)` equals `z` only if `z` is in `C(1/p)`.

Now to matrices larger than 1-by-1, at least most of them. A matrix `A` is "similar to a diagonal matrix" if, ignoring roundoff error, the statement

```
[V,D] = eig(A)
```

would produce an eigenvector matrix, `V`, which is nonsingular, and an eigenvalue matrix, `D`, which is diagonal. Such matrices can be reconstructed from their eigenvalues and eigenvectors by

```
A = V*D/V
```

Moreover, matrix functions `F(A)` can be defined by

```
F(A) = V*F(D)/V
```

with `F(D)` defined by applying `F(z)` element-wise to the eigenvalues on its diagonal.

If `A` is similar to a diagonal matrix, then, still ignoring roundoff,

```
sqrtm(A) = V*diag(sqrt(diag(D)))/V,
logm(A) = V*diag(log(diag(D)))/V, etc.
```

Consequently

`sqrtm(A^2) == A` only if all `eig(A)` are in `C(1/2)`,

`logm(expm(A)) == A` only if all `eig(A)` are in `S`,

`(A^p)^(1/p) == A` only if all `eig(A)` are in `C(1/p)`.

The three examples at the beginning of the article were interesting because: one of the eigenvalues of `[1,2; 3,4]` is negative, and so is not in `C(1/2)`; both of the eigenvalues of `[0 4; -4 0]` have imaginary parts greater than `pi`, and so are not in `S`; some of the eigenvalues of `pascal(n,2)` are in the left half plane and so are not in `C(1/3)`.

Not all matrices are similar to diagonal matrices, and the situation gets even more complicated. The prototype of such matrices is the 2-by-2 "Jordan block"

```
J=
    0    1
    0    0
```

When complex numbers are allowed, all 1-by-1 matrices have square roots and all of them except 0 have finite logarithms. But this matrix does not have a square root or a logarithm. The equations

```
X^2 = J
```

and

```
expm(X) = J
```

do not have any solutions, real or complex. If you ask MATLAB to compute

```
sqrtm(J)
```

or

```
logm(J)
```

you will get serious warning messages. This is not a bug or an algorithm failure—it is just impossible to compute something that doesn't exist.

If a matrix is not similar to a diagonal matrix, it still might have a square root or logarithm. For example, `E = [1 1; 0 1]`

```
E =
    1    1
    0    1
```

is not similar to a diagonal matrix, but has `J` as one possible logarithm.

So far, we've been ignoring roundoff error. Nothing we've said involves approximations, scaling, finite precision arithmetic or condition numbers. It's all been "pure" mathematics, expressed in MATLAB notation. But, we've run out of room here. The numerical analysis—the really good stuff—will have to wait for another time.