How Do You Want Them Numbers?

Sometimes you want symbolic, sometimes you don't

I go into the MATLAB all-night diner off of Old Cochituate Road, west of Natick. "I see you've got a special on Hilbert matrices," I tell the redheaded waitperson. "I'll have one of the 3-by-3s. I'll start with the determinant, then you can bring me the inverse, and I'll finish up with a few eigenvalues."

"OK, buddy," the waitperson replies. "How do you want them numbers?"

Good question. How do you want *your* numbers? Until recently with MATLAB, you really didn't have much choice. You could have them any way, as long as they were floating-point. Sure, we could display them in several different formats, and sometimes even make them look like integers, but there was really only one big pot of them in the kitchen.

Now, with the addition of the Symbolic Math Toolbox, there is a wider range of numeric representations and arithmetic. This toolbox, which uses the kernel and library of Maple V® to do much of its computation, employs a combination of unlimited precision integer and rational arithmetic, fixed-precision hardware floating-point arithmetic, variable precision-software floating-point arithmetic, and symbolic computer algebra.

My own experience over the years with both numeric and symbolic computation has led me to appreciate a mixture of all of these representations. Symbolic computation is terrific for problems like matrix determinants and inverses, which involve only rational arithmetic operations. But for "irrational" problems, like matrix eigenvalues and solutions of nonlinear equations, I usually prefer the approximate, floating-point representations.

Consider the number "4.2". Mathematically, this is the point one fifth of the way from 4 to 5 on the real number line. In the text processor where I'm typing this column, and in the input text for MATLAB, this number is represented as a character string—the character "4", followed by the character ".", followed by the character "2". In IEEE floating-point format, this number is represented by some bits which print out as `4010CCCCCCCCCCCD`. (OK, with roundoff error, that's not exactly 4.2. But, for a change, I'm not worried about that now.)

Here is list of possible representations of this one number:

```
4.2
4010CCCCCCCCCCCD
42/10
21/5
4200000000000000000000000.0E-24
sqrt(17.64)
(2+sqrt(.2)*i)*(2-sqrt(.2)*i)
RootOf(105*Z^2-416*Z-105)
```

All of these can now occur in MATLAB with the Symbolic Math Toolbox. And each of them may be advantageous in a particular situation.

For matrix problems larger than about 10-by-10, there is usually no choice. MATLAB's traditional floating-point computation is usually the only practical approach. But even for small problems, the exact results produced by symbolic computation are sometimes too complicated to be really useful.

Let's see how symbolic computation affects one of our simplest test matrices, the Hilbert matrix. The MATLAB statement

```
H = hilb(3)
```

produces the floating-point representation

```
H =

   1.0000    0.5000    0.3333
   0.5000    0.3333    0.2500
   0.3333    0.2500    0.2000
```

Of course, the internal representation of `H` involves the floating-point approximations to 1, 1/2, 1/3, 1/4 and 1/5. The symbolic representation of `H` can be computed by

```
S = sym(H)
```

The `sym` function is able to determine that the elements of `H` are the ratios of small integers, and so produces

```
S =

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

The determinant of the Hilbert matrix can be computed from either representation. The floating point, approximate, result is

```
4.6296e-04
```

while the symbolic, exact, result is

```
1/2160
```

Which do you prefer? I suspect almost all of us would vote for the symbolic result. It's prettier; it's somehow more basic; and, it's "exact."

The same is true of matrix inversion. Maple V does matrix inversion in a finite number of exact, rational operations and so, in this case, is able to produce the exact result, which turns out to have integer entries.

```
[   9,  -36,   30]
[ -36,  192,  180]
[  30, -180,  180]
```

It's hard to argue with perfection. The floating-point computation done by the statement

```
inv(H)
```

would only produce values near these integers.

The characteristic polynomial? Again, a finite number of exact, rational operations produces the linear form

```
x^3-23/15*x^2+127/720*x-1/2160
```

which is suitable for further manipulation, or the "prettyprint" form:

```
 3     23   2    127
x   - ---- x   + --- x - 1/2160
        15       720
```

which is easier to look at. In contrast, MATLAB's traditional

```
poly(H)
```

produces

```
1.0000   -1.5333   0.1764  -0.0005
```

For polynomials of modest degree with rational coefficients, the symbolic representation is certainly preferable.

I hope you guessed that the next step is the eigenvalues. Now the scales begin to tip away from the exact answer. The traditional

```
eig(H)
```

produces the floating-point results

```
0.12232706585391
0.00268734035577
1.40831892712365
```

The new toolbox gives an intimidating result,

```
[(129287/1458000+1/14400*i*29933^(1/2)*3^
 (1/2)^(1/3)+6559/32400/(129287/1458000+1/
 14400*i*29933^(1/2)*3^(1/2))^(1/3)+23/45]

[-1/2*(129287/1458000+1/14400*i*29933^
 (1/2)*3^(1/2))^(1/3)-6559/64800/(129287/
 1458000+1/14400*i*29933^(1/2)*3^(1/2))^
 (1/3)+23/45+1/2*i*3^(1/2)*((129287/14580
 00+1/14400*i*29933^(1/2)*3^(1/2))^(1/3)65
 59/32400/(129287/1458000+1/14400*i*29933^
 (1/2)*3^(1/2))^(1/3))]

[-1/2*(129287/1458000+1/14400*i*29933^
 (1/2)*3^(1/2))^(1/3)6559/64800/(129287/1
 458000+1/14400*i*29933^(1/2)*3^(1/2))^(1
 /3)+23/45-1/2*i*3^(1/2)*((129287/1458000
 +1/14400*i*29933 ^(1/2)*3^(1/2))^(1/3)
 6559/32400/(129287/1458000+1/14400*i*2993
 3^(1/2)*3^(1/2))^(1/3))]
```

This is the exact answer. It results from a classic formula known as Cardano's solution, which gives the roots of a cubic polynomial.

The symbolic eigenvalues are stored as a MATLAB string containing 786 characters which represents a vector whose three elements are the eigenvalues. You can see both square roots and cube roots in the result. And, although all three roots are real (since the matrix is symmetric), their symbolic representation involves `i`, the imaginary unit. The presence of `i`, or something equivalent, is unavoidable; it is not possible to represent these real quantities without using square roots of negative numbers or referencing trig functions. (Historically, this is where complex numbers first appeared. Cardano realized they were necessary in his formula for the roots of a cubic, even though the roots themselves were real.)

The singular values are even worse. Maple computes `svd(A)` from

```
sqrt(eig(A'*A))
```

which is perfectly OK when there are no roundoff errors. But for the 3-by-3 Hilbert matrix, this leads to a symbolic representation which is even more complicated then the eigenvalues of `A` itself, a string containing 837 characters, complex numbers, and sixth roots. Let's save a little space by printing only the third singular value; the other two are similar.

```
[1/1200*(1440000*(18282673/64000000+3/
 160000*i*89799^(1/2))^(2/3)+624681+
 959200*(18282673/64000000+3/160000*i*
 89799^(1/2))^(1/3))^(1/2)/(18282673/
 64000000+3/160000*i*89799^(1/2))^(1/6)]
```

However, the Hilbert matrix is symmetric and positive definite. So its eigenvalues and its singular values are equal. The third singular value is actually equal to one of the three eigenvalues; it's just represented differently. But it isn't clear which one. And it turns out to be a very difficult symbolic computation to verify that the symbolic representation of the eigenvalues and the singular values are, in fact, the same quantities.

These symbolic results for the eigenvalues and singular values of the 3-by-3 Hilbert matrix are pretty typical. If the characteristic polynomial doesn't factor nicely into linear and quadratic factors with small, integer coefficients, the symbolic representations of its roots are too cumbersome and complicated to be much use.

So, we're pleased to see "symbolic" added to the menu of ways that MATLAB can represent numbers. For rational operations on small matrices, it's great to get the exact answer. But, like fresh broccoli or low-fat ice cream, we have no intention of making it our steady diet.

Good question. How do you want your *numbers? Until recently with MATLAB, you really didn't have much choice. You could have them any way, as long as they were floating-point.*