Symbolic Math, 30 Years After FLOP

Interface to Maple, programmed in MATLAB

Unless you skipped immediately to this column, you've already seen the lead article in this newsletter announcing MATLAB's new Symbolic Math Toolboxes. These toolboxes make most of the capabilities of Maple, a system for symbolic mathematical computation, available within MATLAB. I'd like to tell you about some of the design and implementation considerations that underlie these new toolboxes. Two of us at The MathWorks, Denise Chen and myself, have been working on this project for several months.

I've been interested in symbolic mathematical computation for a long time. As a summer job in 1962, I worked for Charles Lawson at Caltech's Jet Propulsion Laboratory. Lawson wanted to use computers to manipulate what astronomers call "theories" —huge, multivariate Fourier series that describe the motion of the planets and, especially, the earth's moon. The computer language LISP had just been developed by John McCarthy at MIT.

Lawson's idea was to use LISP-like data structures in Fortran to represent such series. We wrote subroutines to carry out basic operations on symbolic mathematical expressions. We could add, subtract, multiply, and divide them. We could differentiate and, to some extent, integrate them. We could print them out in a pretty, two-dimensional format. We even modified the Fortran II compiler for the IBM 7090 to make Fortran recursive.

But then we ran into a really hard problem—simplification. We had no effective way to decide if two of these expressions represented the same function. In order to conclude that the two expressions

$$t(1-t)$$

and

$$t - t^2$$

represent the same function, you have to convert one to the other.

In fact, you want to convert both of them to the "simplest" form. But which is simplest? That's in the eye of the beholder. The notion of a "simple" symbolic expression is not clearly defined; it depends upon how the expression is going to be used. And it becomes particularly difficult for expressions involving several variables and dozens or hundreds of terms.

The simplification difficulties hit Lawson and me about halfway through the summer. We didn't get much further. Luckily, we had already named the system: Fortran List Oriented Package, or FLOP. As far as I know, this is the first time it has ever been mentioned in print.

Since that experience, I've watched the development of symbolic computation systems over the years — FORMAC, ALPAC, MACSYMA, REDUCE, VAXIMA, MuMath, Derive, AXIOM, SMP, Mathematica, and Maple. In my opinion, Maple is clearly the best available today. It has the strongest mathematical foundation, the best software and language design, and the backing of a professional software organization. (Derive is an impressive alternative for small machines, including hand-held computers.) Maple was developed at the University of Waterloo by Gaston Gonnet, Keith Geddes, and a number of collaborators and students, and is now marketed by Waterloo Maple Software Inc. (WMS), of Waterloo, Ontario.

WMS recently developed a version of Maple known as the OEM Kernel. It is similar to MATLAB's engine; it can be called by any C program. And MATLAB, of course, can call any C program through its "MEX" facility. So, we've made a MEX-file out of Maple. It's actually not a very large MEX-file, because the kernel involves only Maple's parser and interpreter. Most of Maple's mathematical knowledge is embedded in its library, a large, 9-megabyte file containing preparsed M-files. In fact, the MATLAB Symbolic Math Toolboxes access the same library as the stand-alone Maple V, Release 2.

The MEX gateway is really quite simple. It passes strings containing Maple commands to the kernel, receives strings containing results from the kernel, checks error codes, and returns the strings to MATLAB itself. We find the MEX connection preferable to running Maple as a separate process and communicating via files, pipes, or shared memory, because it is more efficient and has better error recovery properties.

The Symbolic Math Toolbox consists of the MEX-ified Maple kernel, the Maple library, and over 50 M-files which provide MATLAB versions of Maple functions for calculus, linear algebra, variable-precision arithmetic, and symbolic equation solving. We've tried to use notation which is a natural extension of MATLAB's notation for numeric expressions.

Any Maple functions which are not used directly by M-files in the toolbox can be accessed by the `maple` function, which sends any string representing a Maple command to the kernel.

For example, the statement

```
f = 'cos(x^3)/(1+x^2)'
```

creates a MATLAB string representing a simple function. The quantity `'x'` is a symbolic free variable. There does not have to be a numeric value for `x` in the MATLAB workspace, because we are not yet actually evaluating `cos(x)`.

The statement

```
ezplot(f)
```

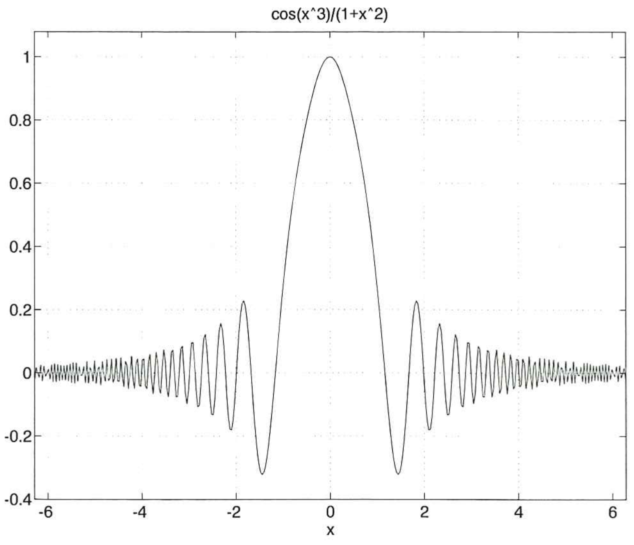
doesn't actually involve Maple. It calls a new MATLAB function to plot a graph of the function defined by the string f. The range of the x axis and the scale of the y axis are determined automatically to show an "interesting" picture of the function, shown at the top of the next page.

The MATLAB statement

```
diff(f)
```

asks for the derivative of `f` with respect to its symbolic variable. This calls the MATLAB M-file which defines `diff`. The M-file determines that the symbolic variable is `'x'`, then sends the Maple command

```
diff(f,x)
```



through the MEX gateway to the Maple kernel. The result,

```
-3*sin(x^3)*x^2/(1+x^2)-2*cos(x^3)/(1+x^2)^2*x
```

is returned from the kernel to `diff`, then back to the MATLAB workspace where it is stored as a string in `ans`. The string is a valid MATLAB expression which can be manipulated and evaluated by subsequent statements. In particular, the statement

```
    pretty(ans)
```

produces the output

```
                    3   2            3
              sin(x ) x        cos(x ) x
          - 3 ---------- - 2 ----------
                     2               2 2
               1 + x            (1 + x )
```

which is easier to read, but not useful for further processing.

This example illustrates one simple, but useful, feature of the toolbox. Several of the functions, including differentiation, integration, and equation solving, automatically determine a free symbolic variable if one is not specified. So

```
    int('a*x^2 + b*x + c')
```

integrates the expression with respect to `'x'`. If you want to integrate with respect to some other variable, you can specify it with a second argument.

Our rule for determining the default variable is a little unorthodox, but we've found it useful. We search the string for all isolated lower-case letters. If there is more than one such letter, we pick the one that is alphabetically closest to `'x'`. So, the default symbolic variable in `'cos(a*t + b)'` is `'t'`.

Except for the MEX gateway, all of the functions in the toolbox are M-files, written in the MATLAB language. Although MATLAB was not originally intended for such tasks, we have found the vector notation and the interpretive environment very convenient for this development. For example, there is a function, `symvar`, which determines the default variable. It uses statements like

```
    k = find(s>='a' & s<='z')
```

to generate the vector of indices of the lower-case letters in the string, `s`. Then

```
    k = k(find(~isletter(s(k-1))& ~isletter(s(k+1))));
```

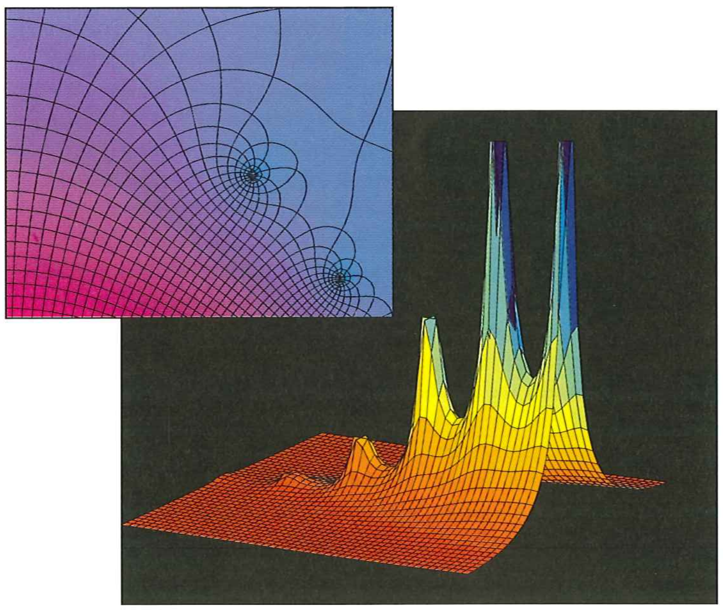
selects the indices of isolated letters which are not part of words. Finally

```
    t = abs(s(k) - 'x')
    v = s(k(find(t == min(t))))
```

selects the letter closest to `'x'`.

Many of the M-files in the toolbox involve this sort string manipulation. Programming the same thing in C or Fortran (or Maple) would not be nearly so convenient.

The Maple library includes functions for evaluating a number of the special functions of classical applied mathematics. These include hypergeometric functions, orthogonal polynomials, sine and cosine integrals, Fresnel integrals, and the Riemann zeta function. There are a number of special functions which Maple can evaluate for complex arguments. Two of them are the error function, `erf(z)`, and the gamma function, `gamma(z)`. These allowed us to produce the two plots shown below.



The contour plot duplicates a portion of a graph from Abramowitz and Stegun, *Handbook of Mathematical Functions*, page 298. It shows the level curves of the polar representation of the complex values of `erf(z)` for a portion of the complex plane. The surface plot shows the poles of the `gamma(z)`, which occur at the negative integers.

We've come a long way since the days of FLOP. I hope you find the Symbolic Math Toolboxes as interesting, as useful, and as much fun as we have.