Objectively Speaking

OOPS is not an apology

by Cleve Moler

Here is a MATLAB one-liner that produces the graphs accompanying this article:

```
plot(eig(randn(1024, 1024*p)), '.')
```

This example computes and plots the eigenvalues of a 1024-by-1024 matrix with normally distributed random elements. You can see that the eigenvalues lie in a disc in the complex plane with radius about 32, which is the square root of 1024. If you issue the command repeatedly, you will get different random eigenvalues, but they will always be in nearly the same disc.

Here's another MATLAB one-liner, which, at first glance, has no relation to our first one:

```
convert(units('microcentury'), 'minutes')
```

The output is:

```
ans =
   52.5949 minutes
```

This tells us that a microcentury is a little less than an hour. It's the optimum length for the lecture in a Monday, Wednesday, Friday class.

What is the point of these examples? Look at them again. What is that lone `p` lurking in the first one? Where did that `minutes` in the output for the second one come from? They came from *objects*. We introduced objects with MATLAB 5 two years ago. We haven't made a big deal out of them. You haven't seen any ads proclaiming, "MATLAB is now *object oriented*." But MATLAB objects are proving to be quite powerful and interesting.

Objects allow anyone to add new data types to MATLAB. By writing a handful of M-files, you can have your MATLAB do operations we never dreamed of at The MathWorks.

When you get your next release of MATLAB and its various toolboxes, you'll see extensive use of objects. Objects are defined by constructor functions in directories whose names begin with an @ sign. The full Release 11 will have dozens of @ directories.

Our first example, the one with the isolated `p`, comes from a research project at MIT lead by Professor Alan Edelman and involving grad students Parry Husbands and Charles Isbell. They plan to call their system MATLAB*P, for "MATLAB in Parallel." (For details, see *http://math.mit.edu/~edelman*, where you can also find out why those random eigenvalues live in a disc.) There is an object named `p` implemented in a directory named `@p`, which includes a constructor function `p.m`. The expression `1024*p` returns the value `1024`, stored in a structure that tags the result to be of class `p`. This may be the only place where `p` appears in the entire program.

When functions like `randn` and `eig` are invoked on a `p` object, the MATLAB system looks for functions, or *methods*, in `@p` which *overload* the traditional functions with those names. This allows the matrices to be constructed and stored, and the computations carried out, on an external parallel computer system. The matrix never exists on the front-end computer. The actual eigenvalue computation is done by SCALAPACK, a distributed memory version of LAPACK. An eigenvalue computation of this size can be done on a system with eight processors in slightly more than one-eighth of the time required by a single processor. More importantly, because large matrices can be distributed across multiple memories, huge problems can be tackled.

In the MIT system, MATLAB is serving as a convenient and familiar interface to a parallel computing environment. MATLAB*P is not a general-purpose parallel MATLAB, but it is an interesting use of the object technology.

The `units` object in our second example is another experimental project. The idea is to have MATLAB do arithmetic operations and conversions on quantities involving units of measurement. The constructor, `@units/units.m`, does most of the work. It recognizes several dozen key words, including meter, kg, second, milli, foot, volt and lightyear. The resulting structure has three fields: a matrix containing the numerical values, a cell array of key words denoting units, and a vector of powers of those units. For example, the acceleration of gravity in MKS metric units is

```
g = units('g')
```

The numerical value of g is 9.8066. The basis array consists of `'meter'` for length, `'second'` for time, and six other key words for mass, temperature, current, molarity, light, and rotation. The powers vector includes 1, –2 and six zeros. This allows the display functions in `@units` to output

```
g =
    9.8066 m/s^2
```

To convert g to traditional English units,

```
convert(g, 'English')
ans =
    386.0886 in/s^2
```

To add disparate units,

```
g + 100*units('yard/nanocentury^2')
ans =
  18.9889 m/s^2
```

This last statement uses the overloading provided by `@units/plus.m`, which is called to evaluate `A + B` whenever either `A` or `B` is a units object.

```
function C = plus(A, B)
A = units(A);
B = units(B);
if ~isequal(A.bas, B.bas)
  B = (B, A);
end
if ~isequal(A.pow, B.pow)
  error('Incompatible for addition.')
end
C = A;
C.val = A.val + B.val;
```

We first ensure that both input arguments are units objects. We then check that they use the same basis. If not, the second argument is converted to the basis used by the first argument. We then check that the powers are the same. If not, they can't be added. Finally, after conversion, both arguments have the same basis and powers, so their numerical values can simply be added.

The units object is not part of Release 11, but the code is available from our Web site at *www.mathworks.com/publications/newsletter/winter99.cleve.shtml.* Please give it a try and let me know if you find it useful.

Overloading is used by both the MIT parallel object and our units object to extend the definition of MATLAB operations to new data types. *Inheritance* is another important aspect of object-oriented programming. The LTI object, which is used by several of the control toolboxes to manipulate Linear Time-Invariant systems of ordinary differential equations, provides an example of inheritance. The LTI object itself is known as the *base class.* It carries out operations that are independent of any particular representation of the system. Bode plots and Nichols charts are LTI methods. Detailed operations that depend upon a particular representation are done by the *derived classes.* These include "tf" for transfer function, "ss" for state space, "zpk" for zero-pole-gain and "frd" for frequency response data. Each of the derived classes has a plus method, which corresponds to connecting the systems in parallel.

This is just a glimpse at the world of MATLAB objects. MATLAB 5.3 has over a dozen objects, including `uint8`, `char`, `inline` and `cell`. The Symbolic Toolbox is now based on the `sym` object. The Financial Toolbox has a `financial time series` object. The new Database Toolbox described on page 4 has `database` and `cursor` objects. The list goes on. You can make it even longer. ■

Cleve Moler is chairman and co-founder of The MathWorks. His e-mail address is moler@mathworks.com.