

Understanding Calculations Through Experiments With Functions

Jerzy A. Piotrowski

School of Computing & IT, University of Western Sydney – Nepean, Australia

Abstract *First year students often use a functional language in their introductory programming classes today. Experiences gained in this way allows them to build mathematical concepts quickly and properly.*

On the other hand, modern algebra systems, such as MATLAB[†], accelerate advanced calculation and offer an excellent platform for experimentation, but they do not facilitate understanding of fundamental concepts.

It is proposed to use jointly the functional approach and a modern algebra system (in this instance, the MATLAB software). This would reduce a danger of creating a clutter of unrelated procedures in place of a coherent system of concepts.

The first computer algebra software which became available in the late 50's is very popular now and is often used in education.

There is a real possibility, however, that an approach based solely on such packages does not develop sufficient understanding of concepts. Though a fluent operational knowledge contributes towards the professionalism of an engineer, it cannot replace its solid theoretical foundations.

The first functional languages emerged at the same time as computer algebra packages. Functional languages such as Haskell or Miranda[‡] are now widely used in education and major industrial projects. Their features are: a language syntax resembling a conventional, mathematical notation, they are strongly typed, and expressions are evaluated on-demand only (*lazy evaluation*).

Strong typing requires a user to think about domains, when objects and their transformations are to be defined.

Laziness means that calculations are performed only when explicitly requested. This allows computers to deal with infinity and undefinedness in a way people do so.

Experiments with functions can be conducted safely in an environment provided by these languages. Such experimentation can properly explain how methods work in principle. Only then more sophisticated, faster algorithms can be used with some degree of assurance.

For example, one may begin a process of building of abstract objects with a notion of a complex number.

In MATLAB, complex numbers are introduced by special functions `i` and `j` and, for instance, a complex number `z` can be defined as: `z = 3 + 4*i`. This is consistent with conventions used in mathematics. However, when the name `i` is needed for some other purpose, the use of a definition `ii = sqrt(-1)` is suggested. By so doing, the standard name `i` is released.

Such defining formulae are in fact statements in a programming language, but they focus on getting results quickly rather than on formal soundness.

By contrast, the complex numbers are introduced in Miranda as a properly defined abstract data type, in exactly the same way they are introduced in mathematics. The type `complex == (num,num)` may have the following operations associated with it:

```
cAdd (a,b) (c,d) = (a+c,b+d)
```

```
cMul (a,b) (c,d) = (a*c-b*d,a*d+b*c)
```

Such type definitions are usually contained in a separate file and can be later included as a library module.

The problem of finding eigenvalues of a linear operator is interesting because it involves matrices, polynomials, and complex numbers. A matrix of numbers, which represents the linear operator, is transformed into the matrix of polynomials, whose determinant is a polynomial, and roots of this polynomial are complex numbers.

Apart from the `complex`, two more ADTs are needed now: `matrix` and `polynomial`. All three can be used in combination, preparing students for writing, with good understanding, the MATLAB expression `eig([0 1;-1 0])` and allowing them to verify the correctness of results.

In conclusion, the algebra systems, such as MATLAB, represent a “tool-box philosophy” and allow only simple programming. While getting an answer to a direct question is easy, building a system of equations with many dependencies is not. On the other hand, functional languages provide a sound environment within which expressions are evaluated. Hence the use of a functional approach, in combination with a system such as MATLAB, can provide an appropriate mathematical foundation for an engineering student.

[†]MATLAB is a trademark of The MathWorks, Inc.

[‡]Miranda is a trademark of Research Software Ltd